Hindawi Publishing Corporation
Modelling and Simulation in Engineering
Volume 2014, Article ID 206364, 10 pages
http://dx.doi.org/10.1155/2014/206364

Research Article

Schedulability Analysis for Rate Monotonic Algorithm-Shortest Job First Using UML-RT

S. Ewins Pon Pushpa[1] and Manamalli Devasikamani[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 7 February 2014; Revised 17 June 2014; Accepted 17 June 2014; Published 23 July 2014

Academic Editor: Luis Carlos Rabelo

System modelling with a unified modelling language (UML) is an active research area for developing real-time system development. UML is widely used modelling language in software engineering community, to specify the requirement, and analyse the target system successfully. UML can be used to provide multiple views of the system under design with the help of a variety of structural and behavioural diagrams at an early stage. UML-RT (unified modelling language-real time) is a language used to build an unambiguous executable specification of a real-time system based on UML concepts. This paper presents a unified modeling approach for a newly proposed rate monotonic scheduling algorithm-shortest job first (RMA-SJF) for partitioned, semipartitioned and global scheduling strategies in multiprocessor architecture using UML-RT for different system loads. As a technical contribution, effective processor utilization of individual processors and success ratio are analyzed for various scheduling principles and compared with EDF and D_EDF to validate our proposal.

1. Introduction

The integration of object modelling and design methods and real-time scheduling theory is the key to successful use of object technology for real-time software. Surprisingly, many past approaches to integrate the two either restrict the object models or do not allow sophisticated schedulability analysis techniques [1]. Visual notations and model abstractions aid object-oriented designer to understand the problem space at an early stage of development cycle. Nowadays, embedded real-time systems are implemented as hardware and software configurations, where the software components have become key for a successful system [2]. The dominance of software in real-time embedded systems design caused the interest in methodologies with widely accepted notations in the software community, such as the unified modeling language (UML). UML-RT (unified modelling language-real time) is a language used to build an unambiguous executable specification of a real-time system based on UML concepts [3]. The integration of schedulability analysis with the industry standard unified modelling language-real time (UML-RT) allows real-time developers to detect and prevent costly design mistakes at an early stage of development.

Technological trends on high performance computational systems are moving towards execution platforms made up of multiple programmable and dedicated processing elements implemented on a single chip known as multiprocessor system-on-chip (MPSoC). In recent years, model based system level design has gained considerable attention in multiprocessor system-on-chip, since it simplifies the application behaviour and reveals the top-level structure of the behaviour, abstracting out the low-level details [4]. A critical issue for MPSoC design is to evaluate the expected performance, early in the design process before hardware implementation [5] and hence real-time scheduling and schedulability analysis for multiprocessor systems have become an important research area [6].

Algorithms for multiprocessor scheduling can be classified as partitioned scheduling and global scheduling. Tasks which are assigned statically to processors and task migration

between processors are not allowed in partitioned scheduling, whereas, in later approach, ready tasks are enqueued in a global queue and tasks are assigned dynamically to available processors. All fixed-priority algorithms based on partitioned scheduling [7–10] and global scheduling [11, 12] have lower utilization bounds. Based on fixed priority, there is a little advantage of global scheduling over partitioned scheduling and rate monotonic algorithm is extensively researched and implemented successfully in conjunction with UML profile for schedulability analysis [13].

When scheduling single-processor systems, nonpreemptive has been considered inferior because of its poor responsiveness. However, in multiprocessor systems, high-priority tasks will still have a chance to execute on available processors to meet deadlines. Moreover, nonpreemptive scheduling algorithms are easier to implement and have lower runtime overhead [14]. In [14], under many parameter settings, experimental simulation surprisingly shows that, in multiprocessor environment, global nonpreemptive fixed-priority scheduling (NP-FP) outperforms global preemptive fixed-priority scheduling (P-NP).

In this paper, a MPSoC system is modelled using UML-RT and schedulability of nonpreemptive rate monotonic algorithm-shortest job first (RMA-SJF) for periodic task sets on three homogeneous multiprocessors is analysed for various load conditions. As a technical contribution, we present our simulation experiments comparing the success ratio and effective processor utilization for global, semipartitioned, and partitioned techniques to validate our work.

This paper improves upon previous publications in the following aspects.

(1) Our result shows better performance than conventional EDF algorithm and D_EDF [15].

(2) In [15] periodic tasks are preemptive whereas our contribution considers nonpreemptive tasks, which were considered inferior to preemptiveness, and still achieves good schedulability for overloaded conditions.

(3) Runtime overhead will be much more reduced in our proposal because EDF is dynamic whereas rate monotonic algorithm is a static algorithm.

The paper is constructed as follows. Overview of UML-RT is presented in Section 2. We present MPSoC model using UML-RT in Section 3. Rate monotonic algorithm (nonpreemptive) for global, partitioned and semi-partitioned scheduling are analyzed in Section 4. Simulation experiments and performance evaluation are done in Section 5. Section 6 concludes the paper.

2. Object-Oriented Models

Object-oriented programming upgrades procedural programming in the areas of adaptability, understandability, and code reusability. Real-time object-oriented model must be expressed unambiguously and explicitly to represent synchronization and concurrency among processes. Currently available object-oriented real-time models are weak in specifying temporal and behavioural requirements and also lack schedulability analysis [16]. RTSO-RAC model [17] does not explore timeliness and schedulability aspects of the model. OPM/T [18] does not describe the effects of priority assignments and concurrency of the model. In TMO [16] (time triggered and message triggered objects), the model success depends on designers' knowledge of underlying hardware platform. CHAOS [19] does not specify timeliness aspects of real-time system and seems reasonable only for soft-real-time system. Some of the popular tools including Rhapsody, ObjectTime developer, and IBM Rational Rose Real Time provide a framework for analysing the problem space in real-time domain.

Object-oriented models like ADARTS (ADA based design approach for real time), CODARTS (concurrent design approach for real time), HRT-HOOD (hard-time hierarchical object-oriented design) [20], UML-RT [21], and ROOM (real-time object-oriented modelling) [22] use object-oriented notations to capture temporal properties of a real-time system [23]. Limitations of ADARTS and CODARTS are that they are designed mainly for ADA and use limited number of views. With the introduction of UML, ObjectTime has cooperated with Rational software to develop UML-RT, which uses UML's in-built extensibility mechanisms to integrate ROOM concepts within UML. UML-RT and the code generation technology of ObjectTime Developer have been integrated into Rational Rose in the new product Rational Rose Real Time [1]. UML-RT includes all the modelling capabilities of ROOM. The recently standardized UML profile for modelling and analysis of real-time and embedded systems (UML MARTE profile) has been provided by OMG group. But, still when it comes to scheduling, UML-RT profile introduces a set of common scheduling annotations which are fairly sufficient to perform schedulability analysis [24].

2.1. UML and UML-RT. Rational Rose Real time is a software development environment tailored to the demands of real-time software [9]. Developers use Rational Rose Real Time to create models of the software system based on the UML constructs, generate the implementation code, compile, and then run and debug the application. Rational Rose Real Time can be used through all phases of the software development lifecycle, from initial requirements analysis through design, implementation, test, and final deployment.

The tool, named UML-RT for real time, developed by the Rational Corporation, uses UML to express the original ROOM (real-time object-oriented modelling) concepts and their extensions. It includes constructs for modelling both the structure and behaviour of event-driven real-time systems.

Rational Rose Real Time includes features for

(i) creating UML-RT models using the elements and diagrams defined in the UML-RT;

(ii) generating complete code implementations (applications) for those models;

(iii) executing, testing, and debugging models at the modelling language level using visual observation tools.

In UML-RT the three principle constructs for modelling structure are capsules, ports, and connectors. UML-RT is a profile that extends UML with stereotyped active objects, called capsules, to represent system components. The internal behavior of a capsule is defined using statecharts; its interaction with other capsules takes place by means of protocols that define the sequence of signals exchanged through stereotyped objects called ports. The UML-RT profile defines a model with precise execution semantics; hence it is suitable to capture system behavior and support simulation or synthesis tools (e.g., Rose RT).

2.1.1. Capsules. The fundamental modelling constructs of UML-RT are capsules.

(i) They are possibly distributed architectural active objects that interact with other capsules exclusively through one or more ports.

(ii) The behaviour 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.

2.1.2. Ports. Messages are sent to and received from capsule instances through objects known as ports. Ports connected to a state machine of a capsule (end port) can handle messages sent to them.

2.1.3. Connectors. The key communication relationships between capsule roles are captured by connectors. They interconnect capsule roles that have similar public interfaces through ports.

2.1.4. Protocols

(i) They define a set of messages exchanged between a set of capsules.

(ii) Messages are defined from the perspective of both the receiver and the sender.

3. Multiprocessor Model Using UML-RT

The behaviour of a real-time model is composed of periodic independent tasks with three homogeneous processors. A real-time system must be analysed to ensure that all the tasks are schedulable. A multiprocessor system is modelled with active objects called capsules in UML-RT and shown in Figure 1. Two capsules are created, one for generating tasks named "gentask" and the other named "scheduler" to schedule the generated tasks. Using the property of inheritance, three capsule instances are created for the three homogeneous processors where the tasks are executed and named as "processor1," "processor2," "processor3." The ports used in each capsule are shown in Table 1.

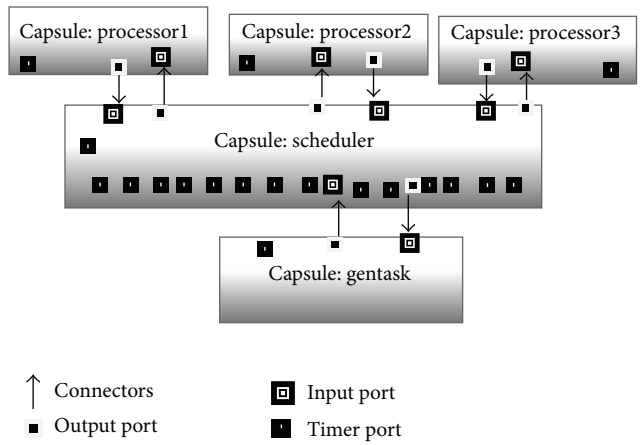

Figure 1: Multiprocessor system modelled in UML-RT.

3.1. "gentask" Capsule. This is the task activation model where task sets are generated. The tasks to be scheduled are triggered in the "gentask" capsule. The output port "gout" of the "gentask" capsule is connected to the input port "sin" of the "scheduler" capsule. The input port "gin" of the "gentask" capsule is connected to the output port "sout" of the "scheduler" capsule.

3.2. "scheduler" Capsule. In "scheduler" capsule, priorities are assigned to each task and tasks are scheduled as per the algorithm.

3.3. "processor1" Capsule. The output port "p1out" of the "scheduler" capsule is connected to the input port "pin11" of the "processor1" capsule. Likewise, the output port "pout11" of the "processor1" capsule is connected to the input port "p1in" of the "scheduler" capsule.

3.4. "processor2" Capsule. The output port "p2out" of the "scheduler" capsule is connected to the input port "pin22" of the "processor2" capsule. Likewise, the output port "pout22" of the "processor2" capsule is connected to the input port "p2in" of the "scheduler" capsule.

3.5. "processor3" Capsule. The output port "p3out" of the "scheduler" capsule is connected to the input port "pin33" of the "processor3" capsule. Likewise, the output port "pout33" of the "processor3" capsule is connected to the input port "p3in" of the "scheduler" capsule.

4. System Description

For a periodic task system, let task set τ consist of a set of "m" tasks ($\tau = \tau_1, \tau_2, \tau_3, \ldots, \tau_m$). Each task is defined by two parameters: computation time C and interrelease time or period T. So, a task set can be denoted by $\tau = \{C_1T_1, C_2T_2, \ldots C_kT_k, \ldots, C_mT_m\}$, where "$\tau_k$" is the kth task considered. For a task τ_k, the task utilization is $U_k = C_k/T_k$, and let the system load $U(\tau) = \sum_{\tau_i \in \tau} U_i$. A set of all higher

Table 1: Ports and capsules used in modelling multiprocessor architecture.

Capsule	Input ports	Output ports
gentask	gin	gout
scheduler	p1in, p2in, p3in, and sin	p1out, p2out, p3out, and sout
processor1	pin11	pout11
processor2	pin22	pout22
processor3	pin33	pout33

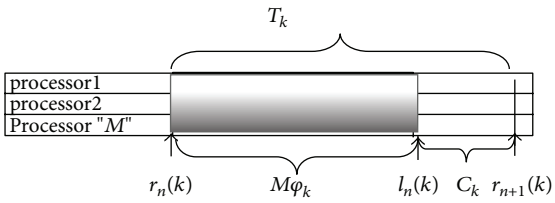

Figure 2: Work area in multiprocessor architecture.

priority tasks of τ_k are denoted by $hp(k)$; similarly, a set of all lower priority tasks of τ_k are denoted by $lp(k)$.

The scheduling strategy of a task to be schedulable in an "M" identical nonpreemptive multiprocessor architecture is analyzed. Depending on the number of processors "M" in the system, the total tasks in the system are split into higher priority tasks τ_1 to τ_M, intermediate tasks τ_{M+1} to τ_{m-M}, and lower priority tasks τ_{m-M+1} to τ_m. Therefore, there must be $m \geq 2M + 1$ tasks in the system. The work area $M\varphi_k = M(T_k - C_k)$ is shown in Figure 2. For a task τ_k, $r_n(k)$ is the nth release time and it is an integer multiple of T_k. The condition for a task τ_k to be schedulable is that the processor must start to execute C_k at least at $l_n(k)$ or the total work load $\dot{W}(k)$ contributed by other tasks in work area $M\varphi_k$ must be less than or equal to $M\varphi_k$, so that the processor is available to execute C_k on or before $l_n(k)$. Therefore, the contribution of total work load $\dot{W}(k)$ of the other tasks in the work area $M\varphi_k$ is analyzed under pessimistic conditions.

The work load in $M\varphi_k$ can be categorized into three parts as reported by Guan et al. [6], shown in Figure 3.

The total work contributed in the work area $M\varphi_k$ can be classified into three works: initial, intermediate, and final work.

Initial Work. Any task will contribute the initial work, if its execution started before $r_n(k)$ and finishes in the work area.

Final Work. Any task will contribute the final work, if its execution started in work area and finishes later to $l_n(k)$.

Intermediate Work. The contribution of any task started on or later to $r_n(k)$ and finishes on or before $l_n(k)$.

Theorems and lemmas are analyzed considering that the entire task contributes maximum in the work area, and task τ_k, released at $r_{n-1}(k)$, is always scheduled before $r_n(k)$. The given computation times are arranged in nondecreasing

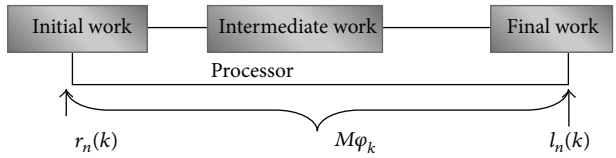

Figure 3: Work load of other tasks in work area φ_k.

order $C_1 \leq C_2 \leq C_3 \cdots \leq C_m$. From the known computation time, let

$$\begin{aligned} C_{\max 1} &= C_m \\ C_{\max 2} &= C_{m-1} \\ C_{\max M} &= C_{m-M+1}. \end{aligned} \tag{1}$$

The maximum computation times are assigned as $C_{\max 1}, C_{\max 2}, \ldots, C_{\max M}$ shown from (1):

$$\dot{W}(k) = \sum_{hp(k),lp(k)} (\text{initial work} + \text{intermediate work} + \text{final work}). \tag{2}$$

A task τ_k will be schedulable if the following is satisfied:

$$M\varphi_k \geq \dot{W}(k). \tag{3}$$

Equation (2) gives the total work load contributed by the other tasks in the work area. Equation (3) is the schedulability condition for a task to be schedulable on an "M" identical nonpreemptive multiprocessor architecture. Equation (3) can be written as follows as stated by Guan et al. [6]:

$$\varphi_k M \geq C_{\max k}^{\text{initial}} + \sum_{hp(k)} \frac{\varphi_k}{T_i} C_i + C^{\text{final}}, \tag{4}$$

where $C_{\max k}^{\text{initial}}$, $\sum_{hp(k)}(\varphi_k/T_i)C_i$, and C^{final} are the initial, intermediate, and final work contribution by other tasks, respectively.

The final work contributions are bounded by the second term in the RHS of (4). Therefore, (4) can be written as follows:

$$\varphi_k M \geq C_{\max k}^{\text{initial}} + \varphi_k \sum_{hp(k)} \frac{C_i}{T_i}. \tag{5}$$

Recall that

$$\sum_{hp(k)} \frac{C_i}{T_i} = U_{hp}(k). \tag{6}$$

Equation (6) is substituted in (5) to obtain (7) and (8):

$$\varphi_k M \geq C_{\max k}^{\text{initial}} + \varphi_k U_{hp}(k), \tag{7}$$

$$\varphi_k \left[M - U_{hp}(k)\right] \geq C_{\max k}^{\text{initial}}, \tag{8}$$

$$\varphi_k \geq \frac{C_{\max k}^{\text{initial}}}{\left[M - U_{hp}(k)\right]}. \tag{9}$$

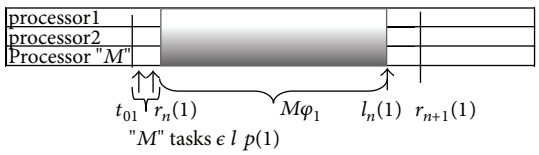

FIGURE 4: Work area $M\varphi_1$ of τ_1.

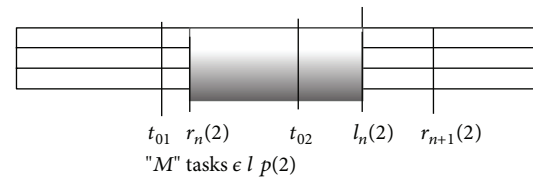

FIGURE 5: "M" tasks $\in lp(2)$ released at $t \in [r_n(2), t_{02}]$.

Equation (9) must be satisfied for the task τ_k to be schedulable.

Lemma 1. *An optimum condition for the highest priority task τ_1 to be schedulable by a work conserving nonpreemptive algorithm is given by*

$$\varphi_1 M \geq C_{\max 1}^{initial}. \tag{10}$$

Proof. To prove (10), consider the time duration given below:

(i) $t \in [t_{01}, r_n(1)]$.

Tasks in execution before $r_n(1)$ in "M" processors will be $lp(1)$ because τ_1 is the highest priority task considered. For "M" processor system, "M" initial works contributed by "M" lower priority tasks (with maximum computation time) under pessimistic conditions are considered. Figure 4 shows that there are "M" tasks $\in lp(1)$, which are in execution during $t \in [t_{01}, r_n(1)]$.

Therefore, initial work contributed by $lp(1)$ is obtained from (1) and summation of "M" initial work is considered as total initial work, $C_{\max 1}^{\text{initial}}$ for task τ_1, and it is shown in:

$$C_{\max 1}^{\text{initial}} = \sum_{i=m-M+1}^{m} C_{\max i}. \tag{11}$$

To Prove by contradiction, if highest priority task τ_1 satisfies (10) but is not schedulable, then task τ_1 misses its deadline. For τ_1 to miss its deadline, "M" processors must be continuously busy in the work area $M\varphi_1$, which means that $M\varphi_1 < C_{\max 1}^{\text{initial}}$. This contradicts the assumption that τ_1 satisfies (10). Therefore, (10) is the optimum condition for task τ_1 to be schedulable. □

Lemma 2. *Maximum waiting time $W\tau_1$ of highest priority task τ_1 for a "M" processor system is shown in*

$$W\tau_1 = \min\left[C_{\max 1}, C_{\max 2}, \ldots, C_{\max M}\right]. \tag{12}$$

Proof. To prove consider, consider

(i) $t \in [t_{01}, r_n(1)]$.

Tasks in execution before $r_n(1)$ in "M" processors will be $lp(1)$ because τ_1 is the highest priority task considered. In worst case condition, "M" tasks $\in$ $lp(1)$ having maximum computation time will contribute initial work to τ_1 as proved in Lemma 1. Consider

(ii) $t \in [r_n(1), t_{02}]$.

Task τ_1 is the highest priority task, released at $r_n(1)$, and is waiting to be executed in any of the "M" processors. The task that finishes first will be the task having minimum execution time during $t \in [r_n(1), t_{02}]$. Therefore, during $t \in [r_n(1), t_{02}]$ when any one processor is free, τ_1 starts to execute. It will be the minimum time of all initial works as given in (12). □

Lemma 3. *The maximum waiting time of periodic task τ_2 for an "M" processor system is given in*

$$W\tau_2 = \min\left[\left(W\tau_1 + C_1\right), \min\left(C_{\max 1}, C_{\max 2}, \ldots, C_{\max M-1}\right)\right], \tag{13}$$

where $W\tau_2$ is considered as initial work contribution to τ_2.

Proof. By Lemma 2 it is proved that the maximum waiting time for the task τ_1 is $W\tau_1 = \min[C_{\max 1}, C_{\max 2}, \ldots, C_{\max M}]$; therefore, τ_2 has to wait for τ_1 to finish execution and also for the other $lp(1)$, executing on "$M-1$" processors.

To prove consider, consider:

(i) $t \in [t_{01}, r_n(2)]$.

The same is considered as in Lemma 2:

(ii) $t \in [r_n(2), t_{02}]$.

Consider Figure 5; assume that the task τ_1 executes on one of the processors, and the other "$M-1$" task $\in lp(2)$ executes on "$M-1$" processors. Task τ_2 will start to execute on a processor, which becomes free first. During $t \in [r_n(2), t_{02}]$, assume that one processor becomes free; therefore, it will be the minimum time as shown in (13). To generalize, for τ_1 to τ_M tasks,

$i = 1$;

$$W\tau_i = \min\left[C_{\max 1}, C_{\max 2}, \ldots, C_{\max M}\right]$$

$$W\tau_{i+1} = \min\left[\left(W\tau_i + C_i\right), C_{\max 1}, \ldots, C_{\max M-1}\right]$$

$$W\tau_{i+2} = \min\left[\left(W\tau_{i+1} + C_{i+1}\right), \min\left(C_{\max 1}, \ldots, C_{\max M-2}, \left(W\tau_i + C_i\right)\right)\right]$$

$$\vdots$$

$$W\tau_M = \min\left[\left(W\tau_{M-1} + C_{M-1}\right), \min\left(C_{\max 1}, \left(W\tau_{M-2} + C_{M-2}\right), \ldots, \left(W\tau_1 + C_1\right)\right)\right]. \quad (14)$$

□

Lemma 4. *Consider a task τ_k; if $k + M \leq m$, then "M" tasks $\in lp(k)$ will contribute "M" initial work under pessimistic condition.*

Proof. To prove consider the time duration as below:

(i) $t \in [t_{01}, r_n(k)]$.

The tasks in execution before $r_n(k)$ in "M" processors will be either $lp(k)$ or $hp(k)$ because task τ_k is not yet released.

Under pessimistic condition, it is considered that "M" tasks $\in lp(k)$ are in execution during $t \in [t_{01}, r_n(k)]$ because all $lp(k)$ have greater computation time compared to $hp(k)$. Since, $k + M \leq m$, there are "M" tasks $\in lp(k)$ contributing initial work to τ_k, as shown in Figure 6. Therefore, the initial work contribution to task τ_k, if $k + M \leq m$, is given in

$$C_{\max k}^{\text{initial}} = \sum_{i=m-M+1}^{m} C_{\max i}. \quad (15)$$

□

Lemma 5. *For $k+M > m$, "$m-k$" tasks $\in lp(k)$ will contribute a worst case of "$m-k$" initial work, and "$M-(m-k)$" tasks $\in hp(k)$ will contribute the remaining initial work.*

Proof. To prove consider the time duration as below:

(i) $t \in [t_{01}, r_n(k)]$.

Tasks in execution before $r_n(k)$ in "M" processors will be either $lp(k)$ or $hp(k)$. Since a pessimistic condition is analyzed, there is only "$m-k$" task $\in lp(k)$ for the above condition, contributing the initial work and "$M-(m-k)$" task $\in hp(k)$ having greater computation time will contribute the remaining initial work. Therefore,

$$C_{\max k}^{\text{initial}} = \sum_{m-(m-k)+1}^{m} C_{\max i} + \sum_{m-M}^{m-(m-k)-1} C_i. \quad (16)$$

Therefore, (16) shows the initial work for the task τ_k if $k+M \leq m$. □

5. RMA-SJF Algorithm

With known computation times, the aim is to design a work conserving task system to utilize the available processing capacity using RMA-SJF priority scheme. A task set is derived, which satisfies RMA-SJF from the known computation time; that is, the higher priority task possesses lesser computation time and lesser interrelease time. It is named as work area analysis (WAA), which is as follows.

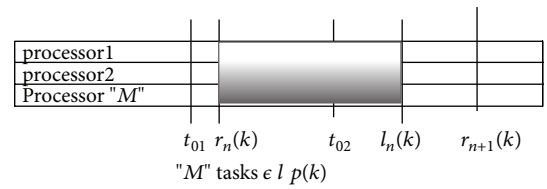

Figure 6: "M" tasks $\in lp(k)$ starting to execute at $t \in [t_{01}, r_n(k)]$.

The known computation timings are arranged in nondecreasing order: $C_1 \leq C_2 \leq C_3 \leq \cdots \leq C_k \leq \cdots \leq C_m$. It is considered that, if there are "M" processors in the system, then there must be minimum of "$2M + 1$" tasks for the proposed algorithm to work. Therefore, the task set can be divided into three categories according to priority, as given below:

(i) higher priority tasks τ_1 to τ_M;

(ii) intermediate tasks τ_{M+1} to τ_{m-M};

(iii) lower priority tasks τ_{m-M+1} to τ_m.

Case 1 (higher priority tasks (τ_1 to τ_M)). Consider

$$\text{for} \quad (k = 1; k \leq M, k++), \quad C_{\max k}^{\text{initial}} = W\tau_k, \quad T_k = C_{\max k}^{\text{initial}} + C_k. \quad (17)$$

From (17), the interrelease times for each higher priority tasks are found using Lemmas 1, 2, and 3.

Case 2 (intermediate tasks (τ_{M+1} to τ_{m-M})). Considering (9), $C_{\max k}^{\text{initial}}$ is obtained from (15) and (16) depending on the kth value of the task for intermediate tasks:

$$C_{\max ik}^{\text{initial}} = C_{\max k}^{\text{initial}} \quad (18)$$

$$\text{for} \quad (k = M + 1; k \leq m - M, k++) \quad (19)$$

$$\varphi_k \geq \frac{C_{\max ik}^{\text{initial}}}{\left[M - U_{hp}(k)\right]} \quad (20)$$

$$T_k = \varphi_k + C_k. \quad (21)$$

From (19) to (21), interrelease times of intermediate tasks are found.

Case 3 (lower priority tasks (τ_{m-M+1} to τ_m)). Consider

$$\text{for} \quad (k = m - M + 1,\ k \leq m - 1, k++) \quad (22)$$

$$C_{\max lk}^{\text{initial}} = C_{\max ik}^{\text{initial}} - C_k \quad (23)$$

```
Step 1. begin
Step 2. for τ_i = τ_1 to τ_m (m periodic task)
Step 3.     while there is a free processor M_f and an unassigned tasks do
Step 4.         pick higher priority task
Step 5.             assign (τ_i, M_f)
Step 6.             if task executed within deadline
Step 7.             return "success"
Step 8.             else
Step 9.             return "failure"
Step 10.            endif
Step 11.    endwhile
Step 12. endfor
```

Pseudocode 1

```
Step 1. begin
Step 2. for τ_i = τ_1 to τ_m (m periodic task)
Step 3.     while there is a free processor M_f and an unassigned tasks do
Step 4.         pick higher priority task
Step 5.             assign (τ_i, M_f) (pre-assigned)
Step 6.         if task executed within deadline
Step 7.             return "success"
Step 8.         else
Step 9.             return "failure"
Step 10.            endif
Step 11.    endwhile
Step 12. endfor
```

Pseudocode 2

$$\varphi_k \geq \frac{C_{\max lk}^{\text{initial}}}{\left[M - U_{hp}(k)\right]} \tag{24}$$

$$T_k = \varphi_k + C_k \tag{25}$$

$$\text{for} \quad k = m \tag{26}$$

$$U_{hp}(k) + \frac{C_m}{T_m} \leq M. \tag{27}$$

Initially, the interrelease times are derived for tasks τ_{m-M+1} to τ_{m-1} as shown from (22) to (25). In (23), C_k is subtracted because it does not form the initial work. It is the computation time to be analyzed for schedulability. For the least priority task, $k = m$ in (26); the interrelease time T_m is to be found. It is the only unknown in (27); it is found using condition for schedulability. The condition for a set of tasks to be feasibly scheduled on a multiprocessor system is that its system load should be less than or equal to M.

6. Global Scheduling

For randomly generated computation times, interrelease times for tasks are derived using WAA analysis. The derived task set possesses the RMA-SJF priority scheme; that is, higher priority tasks will have lesser computation time and lesser interrelease time. The task set is analyzed for global, partitioned, and semipartitioned scheduling strategies and success ratio and effective processor utilization of tasks are analyzed.

Initially, task set is analyzed for global scheduling where tasks are assigned dynamically to the available processors.

The pseudocode for global scheduling is as shown in Pseudocode 1.

7. Partitioned Scheduling

In partitioned scheduling, tasks are first assigned to specific processors and executed without migrations. The pseudocode for partitioned scheduling is as shown in Pseudocode 2.

8. Semipartitioned Scheduling

In semipartitioned scheduling, some tasks are global and others are partitioned. The pseudocode for partitioned scheduling is as shown in Pseudocode 3.

9. Simulation and Performance Evaluation

Simulation work is carried out for various load values. Figures 7, 8, and 9 show the total number of released tasks and total number of scheduled tasks for the load of 2.94 for partitioned, semipartitioned, and global scheduling,

```
Step 1. begin
Step 2. for τ_i = τ_1 to τ_m (m periodic task)
Step 3.     while there is a free processor M_f and an unassigned tasks do
Step 4.         pick higher priority task
Step 5.         assign (τ_i, M_f) (pre-assigned or free processor)
Step 6.         if task executed within deadline
Step 7.             return "success"
Step 8.             else
Step 9.             return "failure"
Step 10.            endif
Step 11.    endwhile
Step 12. endfor
```

Pseudocode 3

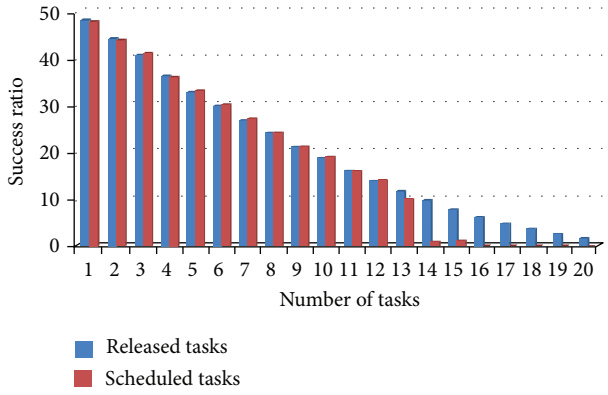

Figure 7: Success ratio of partitioned scheduling for the system load 2.94.

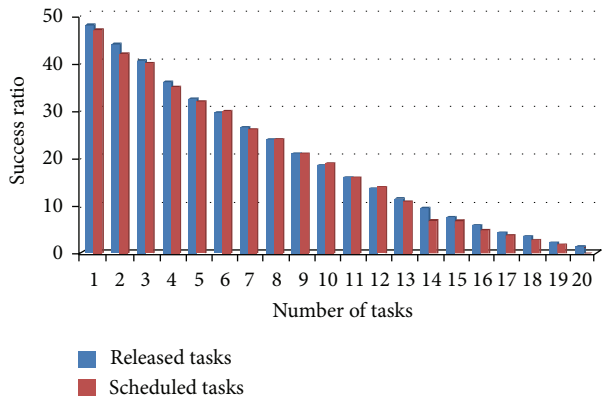

Figure 8: Success ratio of semipartitioned scheduling for the system load of 2.94.

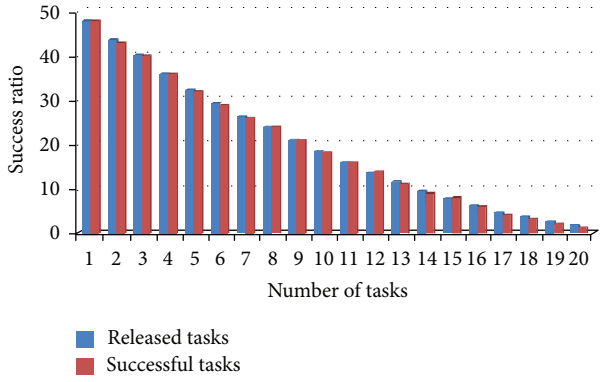

Figure 9: Success ratio of global scheduling for the system load of 2.94.

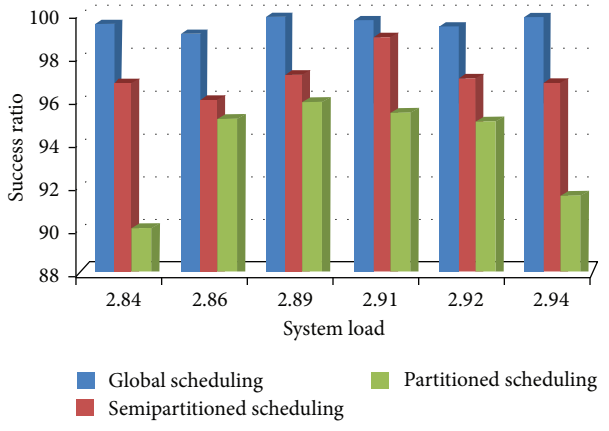

Figure 10: Comparison of success ratio for different scheduling techniques.

respectively. From the results, it is observed that the global scheduling utilises processors a little more efficiently when compared with partitioned and semipartitioned scheduling, thus increasing the schedulability. The success ratios of the three proposed scheduling methods are compared in Figure 10; it is inferred that the success ratio is comparatively more in global scheduling than in partitioned and semipartitioned scheduling. Effective processor utilization for partitioned, semipartitioned, and global scheduling is calculated for various loads and shown in Figure 11.

Global scheduling utilizes processor more efficiently when compared with partitioned and semipartitioned scheduling as analysed from Figure 11. Guan et al. [6] conducted simulation experiments empirically comparing

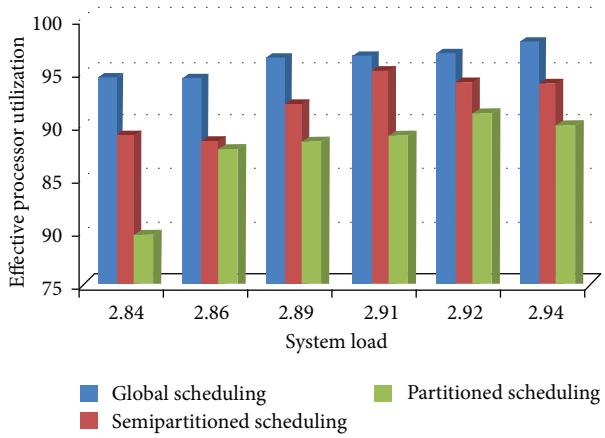

Figure 11: Comparison of effective processor utilization for different scheduling techniques.

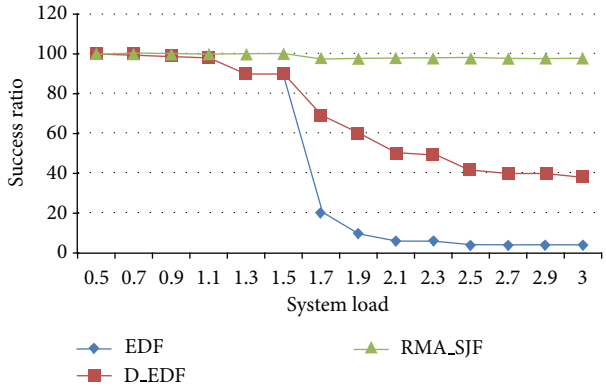

Figure 12: Success ratio analyzed with EDF, D_EDF, and RMA-SJF.

the real-time performance of preemptive and nonpreemptive global fixed-priority scheduling, by which they obtained interesting results suggesting that, for a considerably part of applications on multiprocessor platforms, nonpreemptive scheduling is actually a better choice than preemptive scheduling regarding the real-time performance.

Therefore, obtained results are also compared with the success ratio for the same load by Thakor and Shah [15], and they show that the global RMA_SJF outperforms the global preemptive EDF (earliest deadline first) and D_EDF (deadline monotonic_earliest deadline first) in schedulability for the same load. Figure 12 shows success ratio analyzed with EDF, D_EDF, and RMA-SJF and Figure 13 shows the effective processor utilization is analyzed with EDF, D_EDF, and RMA-SJF. From the analysis, it is inferred that the RMA-SJF outperforms EDF and D_EDF in success ratio by effectively utilizing the processor.

10. Conclusion

In single processor scheduling, nonpreemptiveness leads to poor task responsiveness because higher priority tasks are blocked by lower priority tasks, but, in the case of

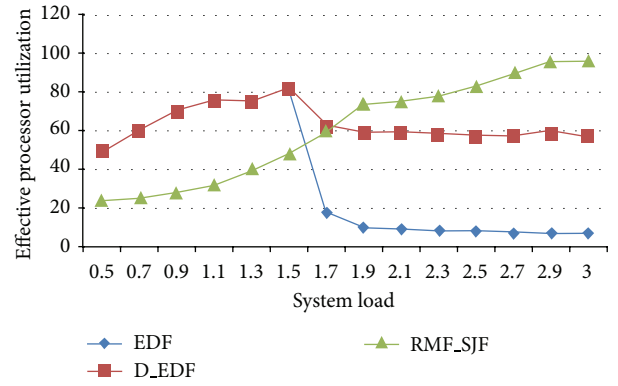

Figure 13: Effective processor utilization analyzed with EDF, D_EDF, and RMA-SJF.

multiprocessor environment, higher priority tasks still have chance to execute in available processors. Moreover, nonpreemptiveness enjoys benefits like lower implementation complexity and lower runtime overhead [6]. Our contribution considers nonpreemptive periodic tasks, scheduled using rate monotonic algorithm-shortest job first (RMA-SJF) on multiprocessor environment and modelled using modelling language UML-RT. In this newly proposed algorithm, the interrelease time for each task is derived from the known computation times and schedulability conditions. RMA-SJF is analysed for various scheduling principles such as global, semipartitioned, and partitioned scheduling for various system loads. Our result shows that the global scheduling utilizes processors little more efficiently when compared to partitioned and semipartitioned scheduling, thus improving schedulability. When compared with the success ratio and effective processor utilization for the same load in [15], RMA-SJF is analysed for global, semi-partitioned and partitioned scheduling strategies for various system loads.

Notations

$M\varphi_k = M(T_k - C_k)$: Problem area $M\varphi_k$ 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 $M\varphi_k$

$hp(k)$: Set of all higher priority tasks of τ_k

$C_{\max 1}^{\text{initial}}$: Initial work for τ_1

$C_{\max k}^{\text{initial}}$: Initial work for τ_k

$U_{hp}(k)$: Summation of all the task utilization of $hp(k)$

$\sum_{hp(k)}(\varphi_k/T_i)C_i$: Intermediate work

$r_n(k)$: 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 executing

$\dot{W}(k)$: Total work done by other tasks in the problem area of φ_k

C^{final}: Final work

$lp(k)$: Set of all lower priority tasks of τ_k.

Conflict of Interests

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

References

[1] M. Saksena and P. Karvelas, *Designing for Schedulability Integrating Schedulability Analysis with Object-Oriented Design*, IEEE, 2000.

[2] M. A. Wehrmeister, L. B. Becker, F. R. Wagner, and C. E. Pereira, "An object-oriented platform-based design process for embedded real-time systems," in *Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '05)*, pp. 125–128, May 2005.

[3] A. Chureau, Y. Savaria, and E. M. Aboulhamid, "Interface-based design of systems-on-chip using UML-RT," in *Proceedings of the 4th IEEE International Workshop on System-on-Chip for Real-Time Applications (IWSOC '04)*, pp. 39–44, July 2004.

[4] Y. Wang, X. Zhou, B. Zhou, L. Liang, and C. Peng, "A MDA based SoC modeling approach using UML and SystemC," in *Proceedings of the 6th IEEE International Conference on Computer and Information Technology (CIT '06)*, p. 245, Seoul, Republic of Korea, September 2006.

[5] H. Yang, S. Kim, and S. Ha, "An MILP-based performance analysis technique for non-preemptive multitasking MPSoC," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 10, pp. 1600–1613, 2010.

[6] 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.

[7] B. Andersson, "Global static-priority preemptive multiprocessor scheduling with utilization bound 38%," in *Proceedings of the 12th International Conference on Principles of Distributed Systems (OPODIS '08)*, pp. 73–88, Luxor, Egypt, 2008.

[8] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 193–202, December 2001.

[9] J. M. López, J. L. Díaz, and D. F. García, "Minimum and maximum utilization bounds for multiprocessor rate monotonic scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 7, pp. 642–653, 2004.

[10] J. M. López, M. García, J. L. Díaz, and D. F. García, "Utilization bounds for multiprocessor rate-monotonic scheduling," *Real-Time Systems*, vol. 24, no. 1, pp. 5–28, 2003.

[11] D. Oh and T. P. Baker, "Utilization bounds for N-processor rate monotone scheduling with static processor assignment," *Real-Time Systems*, vol. 15, no. 2, pp. 183–192, 1998.

[12] Y. Oh and S. H. Son, "Allocating fixed-priority periodic tasks on multiprocessor systems," *Real-Time Systems*, vol. 9, no. 3, pp. 207–239, 1995.

[13] H. Saiedian and S. Raguraman, *Using UML-Based Rate Monotonic Analysis to Predict Schedulability*, IEEE Computer Society, 2004.

[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] D. Thakor and A. Shah, "D-EDF: an efficient scheduling algorithm for real-time multiprocessor system," in *Proceedings of the World Congress on Information and Communication Technologies (WICT '11)*, pp. 1044–1049, IEEE, Mumbai, India, December 2011.

[16] K. H. Kim, "Object structures for real-lime systems and simulators," *IEEE Computer*, vol. 30, no. 8, pp. 62–70, 1997.

[17] L. Cingiser and L. Ma, "A UML package for specifying real-time objects," Tech. Rep., Department of Computer Science, The University of Rhode Island, Kingston, RI, USA, 1998.

[18] M. Peleg and D. Dori, "Extending the object-process methodology to handle real-time systems," *Journal of Object-Oriented Programming*, vol. 11, no. 8, pp. 53–58, 1999.

[19] T. Bihari and P. Gopinath, "Object-oriented real-time systems: concepts and examples," *IEEE Computer*, vol. 25, no. 12, pp. 25–32, 1992.

[20] A. Burns and A. Wellings, "Real-time systems and programming: Ada95," in *Real-Time Java and Real-Time POSIX*, Addison-Wesley, Reading, Mass, USA, 3rd edition, 2001.

[21] R. Grosu, M. Broy, B. Selic, and G. Stefanescu, "Towards a calculus for UML-RT specifications," in *Proceedings of the 7th OOPSLA Workshop on Behavioral Semantics of OO Business and System Specifications*, H. Kilov, B. Rumpe, and I. Simmonds, Eds., Vancouver, Canada, 1998.

[22] B. Selic, G. Gullekson, and P. Ward, *Real-Time Object Oriented Modelling*, John Wiley & Sons, London, UK, 1994.

[23] P. Kumarakulasingam and H. Saiedian, "A framework for evaluating the effectiveness of real-time object-oriented models," *Information and Software Technology*, vol. 44, no. 7, pp. 395–404, 2002.

[24] 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.