SCHEDULING PERIODIC AND APERIODIC TASKS IN HARD REAL-TIME COMPUTING SYSTEMS

Tein-Hsiang Lin and Wernhuar Tarng

Department of Electrical and Computer Engineering
State University of New York at Buffalo, Buffalo, New York

ABSTRACT

Scheduling periodic and aperiodic tasks to meet their time constraints has been an important issue in the design of real-time computing systems. Usually, the task scheduling algorithms in such systems must satisfy the deadlines of periodic tasks and provide fast response times for aperiodic tasks. A simple and efficient approach to scheduling real-time tasks is the use of a periodic server in a static preemptive scheduling algorithm. Periodic tasks, including the server, are scheduled *at priori* to meet their deadlines according to the knowledge of their periods and computation times. The scheduling of aperiodic tasks is then managed by the periodic server during its service time. In this paper, a new scheduling algorithm is proposed. The new algorithm creates a periodic server which will have the highest priority but not necessarily the shortest period. The server is suspended to reduce the overhead if there are no aperiodic tasks waiting, and is activated immediately upon the arrival of the next aperiodic task. After activated, the server performs its duty periodically until all waiting aperiodic tasks are completed. For a set of tasks scheduled by this algorithm, the deadlines of periodic tasks are guaranteed by a deterministic feasibility check, and the mean response time of aperiodic tasks are estimated using a queueing model. Based on the analytical results, we can determine the period and service time of the server producing the minimum mean response time for aperiodic tasks. The analytical results are compared with simulation results to demonstrate the correctness of our model.

Index Terms — Scheduling algorithm, real-time, hard deadline, static, preemptive, periodic server, periodic tasks, aperiodic tasks, feasibility checking, queueing model.

1 INTRODUCTION

The use of computers in life-critical applications such as the control of aircrafts and nuclear reactors usually requires very high reliability and responsiveness. Most tasks in these applications have time constraints, or deadlines, to be satisfied. Computing systems containing tasks with deadlines are called real-time computing systems. A deadline is *hard* if missing it can cause a catastrophic result, and is *soft* if it can be satisfied in statistic sense. Generally, real-time tasks are categorized as periodic and aperiodic. Periodic tasks are initiated and executed repeatedly by the same time intervals called periods, and they appear mostly in monitoring and control-loop subroutines. Aperiodic tasks usually result from operator's commands and exception handling subroutines, and thus occur randomly.

Scheduling periodic and aperiodic tasks to satisfy their deadlines has been an important topic in the study of real-time computing. A scheduling algorithm is *preemptive* if the running task can be interrupted when another task with higher priority arrives, and is *non-preemptive* if each task is executed continuously until completion. A scheduling algorithm is *static* if the priorities assigned to all tasks are fixed throughout the mission's life time, and is *dynamic* if the priorities of tasks may change during their executions. The rate-monotonic (RM) scheduling algorithm is the optimal static preemptive scheduling algorithm for scheduling periodic tasks on a single processor [1]. Let the utilization factor of CPU time on a single-processor system be denoted by U_f. For a set of m tasks, all their deadlines are guaranteed under the RM scheduling algorithm if U_f is not greater than $m(2^{1/m} - 1)$, which approaches 69.3% for very large m.

Algorithms built upon the RM scheduling algorithm for scheduling aperiodic tasks are presented in [2, 3]. In these algorithms, a high-priority periodic server is used to service aperiodic tasks. The use of a periodic server embedded in the RM scheduling algorithm has the advantages that scheduling of periodic tasks is simple and the mean response time of aperiodic tasks can be statistically controlled. Since the RM scheduling algorithm assigns priorities to tasks according to their periods, the periodic server must have a short period to receive a high priority. However, given the same U_f, a periodic server with a short period will result in more overhead in context switching and is thus not suit-

able for aperiodic tasks with low arrival rates. If we prolong the period of the server, its priority will be lower and the response times of aperiodic tasks will also become longer. One way for the periodic server to maintain its high priority with a long period is to relax the rate-monotonic constraint. In other words, the server can have the highest priority while its period is not necessarily the shortest. This gives us more flexibility in scheduling aperiodic tasks with various workloads.

In this paper, we will study how to achieve high processing time usage and fast response time in scheduling real-time tasks. It is assumed that periodic tasks have hard deadlines and aperiodic tasks do not have deadlines but must be completed as soon as possible. A static preemptive scheduling algorithm is developed to achieve high responsiveness for aperiodic tasks. In this algorithm, periodic tasks are scheduled rate-monotonically. A periodic server is then created with the highest priority to handle the execution of aperiodic tasks. The server operates in two states, active or suspended, depending on the current condition of the aperiodic-task queue. When the aperiodic-task queue is not empty, the server is initiated periodically to serve aperiodic tasks. After all aperiodic tasks in the queue are completed, the server uses the remaining service time to serve the highest-priority periodic task or becomes idle if there are no periodic tasks waiting. At the beginning of the next service period, the server is suspended if aperiodic-task queue is empty, and it will not be initiated until becoming active again. The suspended server is activated by the arrival of an aperiodic task. Since the server has the highest priority, it can start serving aperiodic tasks as soon as it becomes active. We call this algorithm the immediate server (IS) scheduling algorithm.

Based on a static analysis, we find some combinations of service time and period for the server which all result in very high U_f, and the period is not necessarily the shortest. To optimize the performance of the periodic server, a queueing model is developed under some general assumptions about the distributions of inter-arrival time and execution time of aperiodic tasks. The optimal service time and period combination of the server is determined by minimizing the mean response time of aperiodic tasks. Since the IS scheduling algorithm is not completely rate-monotonic, the formula proposed in [4] can not be applied to check whether the deadlines of all periodic tasks are satisfied. A feasibility checking (FC) algorithm is thus proposed in Section 2 to determine if all deadlines are guaranteed under any specific priority assignment. The FC algorithm is an essential part in developing the IS scheduling algorithm. It can also be combined with a search algorithm to determine the optimal server. In Section 3, we present the IS scheduling algorithm and derive a set of candidate servers by the FC algorithm and a binary search algorithm. All these candidate servers result in very high total U_f's (about 98%). A queueing model is developed in Section 4 to evaluate the performance of these servers. Simulation results are provided for comparison with analytical results. The paper is concluded by Section 5.

2 A FEASIBILITY CHECKING ALGORITHM

The main objective of task scheduling is meeting the deadline. A scheduling algorithm is *feasible* for a set of tasks if the deadlines of all tasks can be satisfied. Since a static preemptive scheduling algorithm will be used to schedule periodic tasks, we must first answer the question "Given a set of periodic tasks, can the deadlines of all periodic tasks be guaranteed under a specific priority assignment?". Let the response time of a periodic task be defined as the time span between the initiation and the completion of the task. A *critical instant* for a task is defined as a starting time which will result in the longest response time. Liu and Layland [1] proved the following theorem in finding the critical instant for a periodic task.

Theorem 1

> A *critical instant* for a task occurs whenever the task is initiated simultaneously with all higher priority tasks.

Since the deadline of a periodic task is assumed to be identical to its period, the priority assignment is feasible for a set of periodic tasks if every periodic task initiated at its critical instant can be completed before the next iteration is initiated. Clearly, if the first iteration can meet its deadline, later iterations will also meet their deadlines. A feasibility checking (FC) algorithm will be developed based on this principle to determine whether a priority assignment is feasible.

Let $\tau_1, \tau_2, \cdots, \tau_m$ denote a set of periodic tasks. The computation time and period of τ_i are denoted by C_i and T_i, respectively. Without loss of generality, the priority of τ_i is assumed to be higher than the priority of τ_j if $i < j$. For the convenience of checking, the initiation times of the first iterations of all tasks are set to the same instant, which becomes their common critical instant. Different iterations of the same task will be distinguished by their initiation time. When a task is initiated, it is executed immediately if there are no other tasks with higher priority waiting; otherwise, it has to wait until all higher-priority tasks are completed. The FC algorithm simulates the scheduling of periodic tasks by the decreasing order of their priorities. Only one iteration for each task has to be checked. If the deadline of the first iteration for each task is satisfied, the priority assignment is feasible; otherwise, the assignment is unfeasible.

Let $t = 0$ be the common critical instant and let $T = \max(T_1, \cdots, T_m)$ be the longest period among the m periodic tasks. We define a *slack interval* $[x, y]$ as a time slot such that the processor is busy immediately before x and after y but idle between x and y. In a static preemptive scheduling algorithm, the computation time available for a periodic task is the slack intervals left after the scheduling of all higher-priority tasks. The FC algorithm always records the slack intervals after the scheduling of each task which

starts from the highest-priority task. Initially, the slack interval available for τ_1 is set to $\{[0,T]\}$. Since τ_1 will be initiated at $t = kT_1$, $k \geq 0$, the slack intervals left after the scheduling of τ_1 are $\{[kT_1 + C_1, (k+1)T_1], k \geq 0\}$. Let S_i represent the set of slack intervals available for τ_i, then

$$S_i = \{[x_k, y_k] \mid x_k < y_k < x_{k+1}, 1 \leq k \leq N_i\}$$

where N_i is the number of slack intervals in S_i. Because the deadline of the first iteration of τ_i is set at T_i, the total slack intervals available for τ_i within $[0, T_i]$ must be greater than or equal to C_i so as to satisfy its deadline; otherwise, τ_i will miss its deadline and the algorithm is stopped. The derivation of S_{i+1} from S_i is performed by filling a computation time C_i into the slack intervals inside every period $[lT_i, (l+1)T_i]$, $0 \leq l \leq \lceil T/T_i \rceil$. When there are more than one slack intervals to be filled, the slack interval $[x_k, y_k]$ in S_i with the smallest k will be filled first. A slack interval is deleted if it is completely filled, shortened if it is partially filled, or split into two slack intervals if a partial filling has to start in the middle. The checking algorithm is expressed explicitly as follows.

The FC algorithm

1. Set $i := 1$ and initialize S_i to $\{[0,T]\}$.
2. Add τ_i to the set of scheduled tasks.
3. If τ_i misses its deadline, stop. The priority assignment is unfeasible.
4. If $i = m$, stop. The priority assignment is feasible.
5. Set $i := i + 1$ and derive S_{i+1}. Go to **Step 2**.

Example 1:

Task	Comp. Time	Period	Priority	Pattern
τ_1	1	4	1st	
τ_2	1	3	2nd	
τ_3	3	8	3rd	

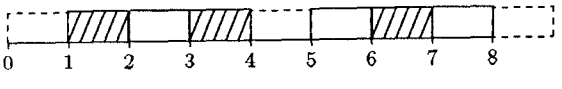


Figure 1: An example of the FC algorithm

Consider a set of three periodic tasks to be scheduled by the priority assignment specified in Figure 1. To check the feasibility using the FC algorithm, we initiate all tasks simultaneously at $t = 0$. Because $T = \max(4, 3, 8) = 8$, the slack interval available for τ_1 is $S_1 = \{[0, 8]\}$. The first iteration of τ_1 is completed at $t = 1$, which is well ahead of its deadline at $t = 4$. Since two iterations of τ_1 occur during $[0, 8]$, S_2 is derived as $\{[1, 4], [5, 8]\}$ after filling $[0, 1]$ and $[4, 5]$ into $[0, 8]$. For τ_2, the first iteration is completed at $t = 2$, which also meets its deadline at $t = 3$. The set of slack intervals available for τ_3 is then derived as $S_3 = \{[2, 3], [5, 6], [7, 8]\}$. The first iteration of τ_3 will be completed at $t = 8$ which just meet its deadline. The FC algorithm ends here since all tasks have been checked and their deadlines are all satisfied. Consequently, the priority assignment is determined as feasible. The total U_f by this assignment equals $\frac{1}{4} + \frac{1}{3} + \frac{3}{8} \approx 96\%$.

For many periodic tasks, the priority assignment is not very important as long as their deadlines are satisfied. However, if the periodic task is a server for aperiodic tasks, we may want to assign it a higher priority to provide faster response times. It is also preferable that the service time of the server can mostly cover the computation time of aperiodic tasks such that they don't have to wait until the next service period. If we want to determine the service time and period of the server according to the characteristics of aperiodic tasks, we may require that the server with a long period can still have the highest priority. In that case, the priority assignment may not be rate-monotonic. Fortunately, the FC algorithm provides us with the capability of checking all static preemptive algorithms including the RM scheduling algorithm. When combined with a search algorithm, the FC algorithm can also be used to determine the maximal service time for the periodic server if the computation times and periods of the remaining periodic tasks are known.

3 IMMEDIATE SERVER SCHEDULING

Most scheduling algorithms for real-time systems consider the scheduling of periodic tasks first, because the arrivals and computation times of periodic tasks are predictable while that of aperiodic tasks are not. The methods to schedule aperiodic tasks vary with their requirements for responsiveness. If response time is not important for aperiodic tasks, the background-job method is adequate which simply assigns the lowest priority to all arriving aperiodic tasks. Faster response time for aperiodic tasks can be achieved by the use of a periodic server, which is a periodic task with a high priority designated to serve only aperiodic tasks. However, the "pure" periodic-server method also has some drawbacks. For example, if no aperiodic tasks are waiting during service period, the server becomes idle and its service time is wasted. Likewise, if an aperiodic task arrives after the service period, it has to wait until the next service period even though the server was idle previously. The IS scheduling algorithm proposed below can remedy these drawbacks by using a periodic server called the immediate server.

The IS scheduling algorithm is built upon a static preemptive scheduling algorithm in which the periodic server for serving aperiodic tasks has the highest priority and the remaining periodic tasks are rate-monotonically scheduled.

It is not necessary that the period of the server is the shortest, which enables us to select the server from a wider range of period and computation time combinations. However, if the period of the server happens to be the shortest, the scheduling becomes rate-monotonic. The server always serves arriving aperiodic tasks in a first-come-first-serve (FCFS) manner, and it operates in two states, active or suspended, depending on the current condition of aperiodic-task queue. The server performs its duty to serve aperiodic tasks when it is active. However, if all arrived aperiodic tasks are completed in the middle of any service period, the remaining service time is used to serve the waiting periodic tasks according to their priorities or wasted when there are no periodic tasks waiting. However, any aperiodic task arrives during the remaining service period will still be served immediately. The server is suspended at the beginning of its service period if there are no aperiodic tasks waiting for service. When the server is suspended, the execution of the current iteration is stopped and so does the initiation of the next iteration. The suspended server is activated immediately upon the arrival of any aperiodic task, and, again, it has the whole service time to serve the arrived aperiodic task. Since the server has the highest priority, it can start to serve aperiodic tasks as soon as activated. Obviously, the satisfaction of deadline for all periodic tasks is not affected by the suspension and activation of the server, because the period of the server is extended after a suspension. Actually, most periodic tasks will have shorter response times when the server becomes suspended. Also, if the arrival rate of aperiodic tasks are low compared to the server's service rate, the probability that aperiodic tasks receive immediate service is very high.

Example 2:

Task	Comp. Time	Period	Priority	Pattern
τ_1	1	4	1st	
τ_2	1	3	2nd	
τ_3	3	8	3rd	
τ_a	1.6	—	—	

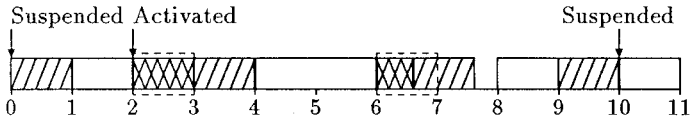


Figure 2: The server in the IS scheduling algorithm

Figure 2 shows the scheduling of tasks using the IS scheduling algorithm. The immediate server τ_1 and two periodic tasks, τ_2, τ_3, are all initiated at $t = 0$. Since no aperiodic tasks arrived before $t = 0$, the server is suspended and the second highest priority task, τ_2, is executed. After τ_2 is completed, τ_3 starts its execution at $t = 1$ and is completed at $t = 2$. The aperiodic task τ_a arrives at $t = 2$ with computation time equal to 1.6 and the server is activated immediately to serve it. Since τ_a can not be completed during the first iteration of the server, the second iteration is again initiated at $t = 6$ to complete the execution of τ_a at $t = 6.6$. The remaining computation time of the server is used to execute τ_2 because no other aperiodic tasks arrive. However, any aperiodic task arrives between $t = 6.6$ and $t = 7$ can still be served by the server immediately. The second iteration of τ_3 starts at $t = 8$, but is preempted at $t = 9$ by τ_2 which is completed at $t = 10$. If any aperiodic task arrives between $t = 7$ and $t = 10$, it will be served by the third iteration of the server beginning at $t = 10$. Since no aperiodic tasks arrive, the server is suspended at the beginning of the third iteration at $t = 10$ and τ_3 then continues its execution.

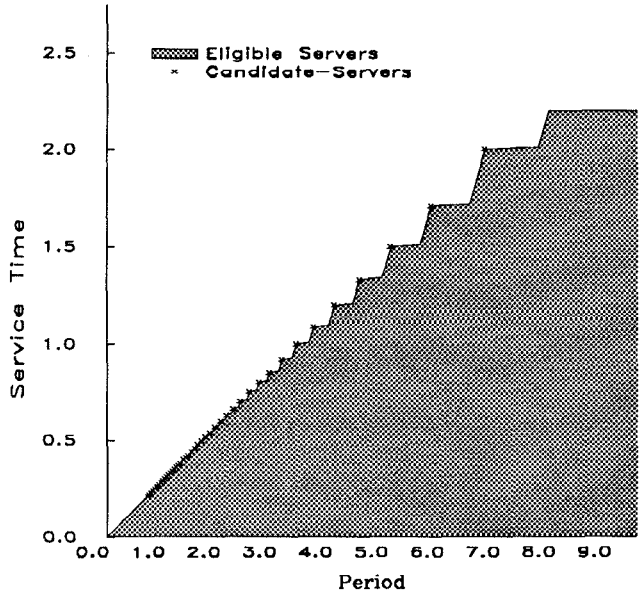
For the rest of this paper, we assume that τ_1 is the immediate server and its service time and period are denoted by C_1 and T_1 respectively. The efficiency of the server depends greatly upon its $U_f = C_1/T_1$. Many eligible combinations of C_1 and T_1 exist for the immediate server which all result in high U_f's. For periodic servers with the same U_f, it is obvious that if C_1 is smaller then T_1 is also smaller. The procedure described below can be used to find out a set of eligible servers called the *candidate-servers* for the IS scheduling algorithm to produce very high total U_f, and they are determined by searching the minimal period T_1 for a given service time C_1. Let $\tau_2, \tau_3, \ldots, \tau_m$ denote a feasible set of periodic tasks under the RM scheduling algorithm. Again, we assume that $T_i \leq T_j$ if $i < j$ such that the priority of τ_i is higher than the priority of τ_j if $i < j$, and the immediate server, τ_1, has the highest priority. The procedure starts by finding the upper bound of C_1. Conceptually, the upper bound of C_1 is the longest time that the periodic tasks $\tau_2, \tau_3, \cdots, \tau_m$ can be delayed from their common critical instant without missing any deadlines. An acceptable delay at the critical instant will certainly be acceptable at any other instant. It is easy to derived that the maximal delay time is not greater than

$$D = \min(T_2 - C_2, \cdots, T_m - C_m).$$

Whether a particular delay d is accepted can be checked by the FC algorithm with the initial slack interval changed from $[0, T_m]$ to $[d, T_m]$. Hence, the upper bound of C_1 can be determined by a binary search between 0 and D and then justified by the FC algorithm. When the upper bound of C_1 is found, it also guarantees that the minimal T_1 for any given C_1 within this bound is not greater than T_m. Because the server is initiated once during during $[0, T_m]$, and it can be initiated again at $t = T_m$ without causing any other periodic tasks to miss their deadlines. Since the total U_f of all tasks can not exceed 1, the value of T_1 is bounded below by

$$\frac{C_1}{(1 - \sum_{i=2}^{m} \frac{C_i}{T_i})} \leq T_1.$$

For any ligitimate value of C_1, the minimal T_1 can be determined by the FC algorithm via another binary search between $C_1/\left(1 - \sum_{i=2}^{m} \frac{C_i}{T_i}\right)$ and T_m.



	τ_2	τ_3	τ_4	τ_5	τ_6
C_i	1.2	0.5	0.3	0.6	1.4
T_i	3.4	5.2	8.3	15.2	26

	τ_7	τ_8	τ_9	τ_{10}	τ_{11}
C_i	0.4	2.5	0.8	4	2
T_i	26	40.6	42.1	240	1000

Figure 3: The candidate-servers derived by the FC algorithm.

Example 3: Consider the ten periodic tasks listed in Figure 3. The maximal delay time is upper bounded by

$$D = \min(T_2 - C_2, \cdots, T_{11} - C_{11}) = 2.2.$$

Since the FC algorithm confirms that $d = 2.2$ is an acceptable delay for all periodic tasks $\tau_2, \cdots, \tau_{11}$, C_1 is bounded by $0 \leq C_1 \leq 2.2$. For each C_1, the minimal T_1 is obtained as a ladder-like curve in the figure and the shaded region represents all possible combinations of C_1 and T_1. The points with "x" marks on the curve are chosen as candidate-servers, because their U_f's are about the same (≈ 0.3) and higher than the other points. The curve becomes a straight line when C_1 and T_1 are very small, and the slope also approaches 0.3. However, a server with very short period is not suitable when the overhead of context-switching is considered. In this example, an average of total $U_f = 98\%$ can be achieved by using a candidate server in the IS scheduling algorithm.

4 A QUEUEING MODEL

In this section, a queueing model is developed to analyze the performance of the server in the IS scheduling algorithm with the following assumptions:

1. Arrivals of aperiodic tasks form a Poisson process with rate λ, i.e., the CDF of inter-arrival time S of aperiodic tasks is
$$F_S(s) = 1 - e^{-\lambda s}, \qquad s \geq 0. \tag{4.1}$$

2. The computation times of aperiodic tasks are exponentially distributed with mean $1/\mu$, i.e., the computation time X of an aperiodic task is random with the CDF
$$F_X(x) = 1 - e^{-\mu x}, \qquad x > 0. \tag{4.2}$$

3. Let Z be the first service period that an aperiodic task receives from the immediate server. Apparently, Z is a random variable taking value between zero and C_1. For an aperiodic task which activates the server, Z is equal to C_1. The probability that a server is busy can be approximated by $\frac{T_1\lambda}{C_1\mu}$ using an M/M/1 queueing model with a modified service rate of $(C_1/T_1)\mu$. If all arrived aperiodic tasks in task queue are completed and the next aperiodic task does not arrive before the end of the service period, the next arriving aperiodic task will receive the full service time C_1. Because the probability that no aperiodic tasks arrive within a service period C_1 equals $e^{-\lambda C_1}$, the probability that $Z = C_1$ is approximately $(1 - \frac{T_1\lambda}{C_1\mu})e^{-\lambda C_1}$. The distribution of Z within $[0, C_1]$ is then assumed to be uniform. Thus, the CDF of Z is expressed as
$$F_Z(z) = \begin{cases} \frac{1-(1-\frac{T_1\lambda}{C_1\mu})e^{-\lambda C_1}}{C_1} z, & 0 \leq Z < C_1 \\ (1 - \frac{T_1\lambda}{C_1\mu})e^{-\lambda C_1}, & Z = C_1. \end{cases} \tag{4.3}$$

The mean response time of aperiodic tasks can easily be solved using an M/M/1 queueing model if they are served by a full-time server. When aperiodic tasks are served by the immediate server, the *actual service time* X', defined as the time between the beginning and the completion of the execution of the task, is no longer an exponentially distributed variable since

$$X' = \begin{cases} X, & \text{if } X \leq Z \\ Z + (X - Z) + \left\lceil \frac{X-Z}{C_1} \right\rceil (T_1 - C_1), & \text{if } X > Z, \end{cases}$$

or equivalently,

$$X' = X + \left\lceil \frac{X - Z}{C_1} \right\rceil (T_1 - C_1). \tag{4.4}$$

An M/G/1 queueing model is required to compute the mean response time R_m for aperiodic tasks. From [5],

$$R_m = E[X'] + \frac{\lambda E[X'^2]}{2(1 - \lambda E[X'])} \tag{4.5}$$

From Eq. (4.4),

$$\begin{aligned} E[X'] &= E\left[X + \left\lceil \frac{X - Z}{C_1} \right\rceil (T_1 - C_1)\right] \\ &= E[X] + (T_1 - C_1) E\left[\left\lceil \frac{X - Z}{C_1} \right\rceil\right] \\ &= \frac{1}{\mu} + (T_1 - C_1) E\left[\left\lceil \frac{X - Z}{C_1} \right\rceil\right]. \end{aligned} \tag{4.6}$$

We then compute

$$E\left[\left\lceil\frac{X-Z}{C_1}\right\rceil\right] = \int_0^{C_1}\int_0^{\infty}\left\lceil\frac{x-z}{C_1}\right\rceil f_X(x)f_Z(z)dxdz$$
$$= \int_0^{C_1}\left(\sum_{n=0}^{\infty}\int_{nC_1+z}^{(n+1)C_1+z}(n+1)f_X(x)dx\right)f_Z(z)dz$$
$$= \int_0^{C_1}\left(\sum_{n=0}^{\infty}e^{-\mu(nC_1+z)}\right)f_Z(z)dz$$
$$= \int_0^{C_1}\frac{e^{-\mu z}}{1-e^{-\mu C_1}}f_Z(z)dz$$
$$= \frac{K_1}{1-e^{-\mu C_1}} \quad (4.7)$$

where

$$K_1 = \int_0^{C_1}e^{-\mu z}f_Z(z)dz$$
$$= \frac{(1-e^{-\mu C_1})}{\mu C_1}\left(1-(1-\frac{T_1\lambda}{\mu C_1})e^{-\lambda C_1}\right)$$
$$+(1-\frac{T_1\lambda}{\mu C_1})e^{-(\mu+\lambda)C_1}. \quad (4.8)$$

Therefore,

$$E[X'] = \frac{1}{\mu}+(T_1-C_1)\frac{K_1}{1-e^{-\mu C_1}}. \quad (4.9)$$

Because $E[X^2]=\frac{2}{\mu^2}$, we can derive that

$$E[X'^2] = E\left[\left(X+(T_1-C_1)\left\lceil\frac{X-Z}{C_1}\right\rceil\right)^2\right]$$
$$= \frac{2}{\mu^2}+(T_1-C_1)^2E\left[\left\lceil\frac{X-Z}{C_1}\right\rceil^2\right]$$
$$+2(T_1-C_1)E\left[\left\lceil\frac{X-Z}{C_1}\right\rceil X\right]. \quad (4.10)$$

Since

$$E\left[\left\lceil\frac{X-Z}{C_1}\right\rceil^2\right] = \int_0^{C_1}\int_0^{\infty}\left\lceil\frac{x-z}{C_1}\right\rceil^2 f_X(x)f_Z(z)dxdz$$
$$= \int_0^{C_1}\left(\sum_{n=0}^{\infty}\int_{nC_1+z}^{(n+1)C_1+z}(n+1)^2f_X(x)dx\right)f_Z(z)dz$$
$$= \int_0^{C_1}\left(e^{-\mu z}\sum_{n=0}^{\infty}(2n+1)e^{-(n+1)\mu C_1}\right)f_Z(z)dz$$
$$= \int_0^{C_1}\frac{1+e^{-\mu C_1}}{(1-e^{-\mu C_1})^2}e^{-\mu z}f_Z(z)dz$$
$$= \frac{1+e^{-\mu C_1}}{(1-e^{-\mu C_1})^2}K_1 \quad (4.11)$$

and

$$E\left[\left\lceil\frac{X-Z}{C_1}\right\rceil X\right] = \int_0^{C_1}\int_0^{\infty}\left\lceil\frac{x-z}{C_1}\right\rceil xf_X(x)f_Z(z)dxdz$$
$$= \int_0^{C_1}\left(\frac{1+(\mu C_1-1)e^{-\mu C_1}}{\mu(1-e^{-\mu C_1})^2}e^{-\mu z}+\frac{ze^{-\mu z}}{1-e^{-\mu C_1}}\right)f_Z(z)dz$$
$$= \frac{1+(\mu C_1-1)e^{-\mu C_1}}{\mu(1-e^{-\mu C_1})^2}K_1+\frac{K_2}{1-e^{-\mu C_1}} \quad (4.12)$$

where

$$K_2 = \int_0^{C_1}ze^{-\mu z}f_zdz$$
$$= \frac{(1-(\mu C_1+1)e^{-\mu C_1})}{\mu^2 C_1}\left(1-(1-\frac{T_1\lambda}{C_1\mu})e^{-\lambda C_1}\right)$$
$$+C_1(1-\frac{T_1\lambda}{C_1\mu})e^{-(\mu+\lambda)C_1}, \quad (4.13)$$

we further derive that

$$E[X'^2] = \frac{2}{\mu^2}+(T_1-C_1)^2\left(\frac{1+e^{-\mu C_1}}{(1-e^{-\mu C_1})^2}K_1\right)$$
$$+2(T_1-C_1)\left(\frac{1+(\mu C_1-1)e^{-\mu C_1}}{\mu(1-e^{-\mu C_1})^2}K_1+\frac{K_2}{1-e^{-\mu C_1}}\right) \quad (4.14)$$

Combining Eq. (4.5), Eq.(4.9) and Eq.(4.14), we get

$$R_m = \frac{1}{\mu}+\frac{(T_1-C_1)K_1}{1-e^{-\mu C_1}}$$
$$+\frac{\lambda\left(\frac{2}{\mu^2}+(T_1-C_1)^2\left(\frac{1+e^{-\mu C_1}}{(1-e^{-\mu C_1})^2}K_1\right)\right)}{2\left(1-\lambda\left(\frac{1}{\mu}+\frac{(T_1-C_1)K_1}{1-e^{-\mu C_1}}\right)\right)} \quad (4.15)$$
$$+\frac{\lambda(T_1-C_1)\left(\frac{1+(\mu C_1-1)e^{-\mu C_1}}{\mu(1-e^{-\mu C_1})^2}K_1+\frac{K_2}{1-e^{-\mu C_1}}\right)}{\left(1-\lambda\left(\frac{1}{\mu}+\frac{(T_1-C_1)K_1}{1-e^{-\mu C_1}}\right)\right)}.$$

To justify the result in Eq.(4.15), we use different combinations of λ and μ to compute R_m for a subset of the candidate-servers chosen in Example 3. Thirty candidate-servers are selected with their periods between 1 and 7. The analytical results are then compared with simulation results, which are shown in Figure 4 to Figure 7.

In Figure 4, the mean computation time of aperiodic tasks is short but the arrival rate is high, and we can see that a server with shorter period and service time can provides better performance. Also, the mean response time increases about linearly with the period of the candidate server. Hence, the optimal server, in this case, is the one with the shortest period. In Figure 5 and 6, the mean computation time and arrival rate of aperiodic tasks are more close to the service times and periods of the candidate-servers. The mean response time in both figures decreases as the period or service time of the server increases, because a longer service time can provides higher probabilities to complete aperiodic tasks within one service period. However, the mean response time in Figure 6 is shorter because the arrival rate is lower. In Figure 7, the arrival rate of aperiodic tasks is low and the mean computation time is long, and it shows that candidate-servers with longer periods can provide shorter response times. Based on these results, we can easily select the optimal server for the IS scheduling algorithm under different workloads.

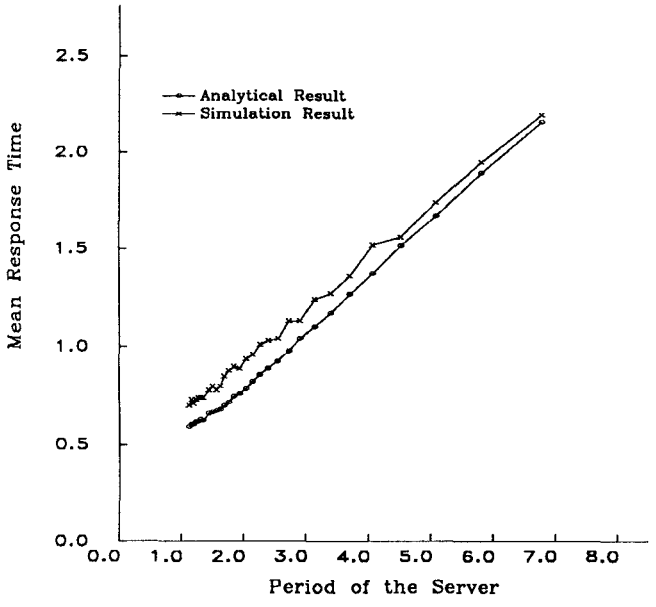


Figure 4: The mean response time for aperiodic tasks with $\lambda = 1.5$ and $\mu = 10$.

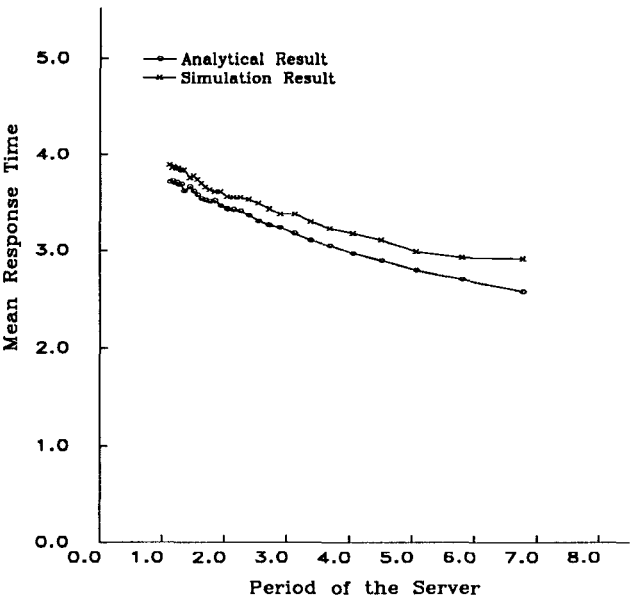


Figure 6: The mean response time for aperiodic tasks with $\lambda = 0.05$ and $\mu = 1$.

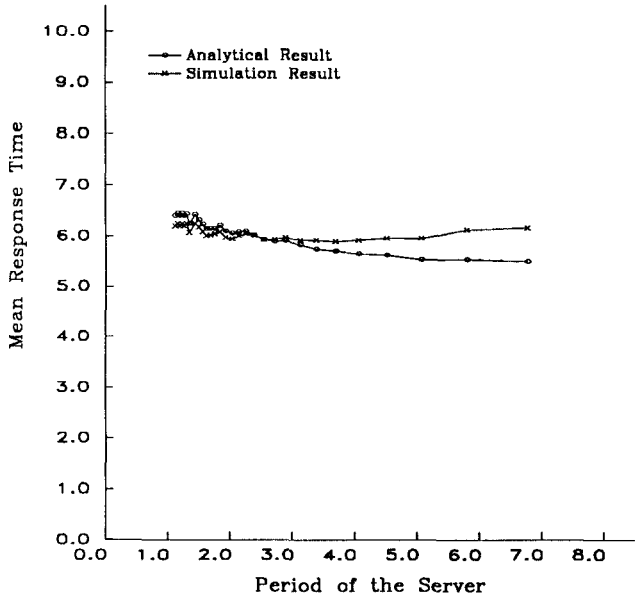


Figure 5: The mean response time for aperiodic tasks with $\lambda = 0.15$ and $\mu = 1$.

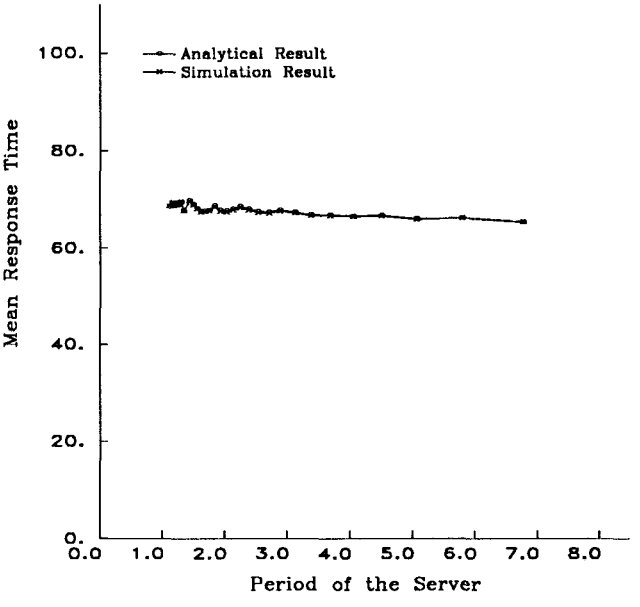


Figure 7: The mean response time for aperiodic tasks with $\lambda = 0.015$ and $\mu = 0.1$.

5 CONCLUSION

In this paper, we have studied the problem of scheduling periodic and aperiodic tasks in real-time computing systems. A feasibility checking (FC) algorithm is proposed to check whether a set of periodic tasks can be scheduled by a static preemptive algorithm to satisfy their deadlines. The IS scheduling algorithm is then developed which assigns the highest priority to the periodic server in scheduling aperiodic tasks. The periodic server can be activated immediately to provide fast response time for aperiodic tasks after a suspension. For a set of periodic tasks, we can apply the FC algorithm and a search algorithm to find the set of candidate-servers which result in very high percentages in using CPU time. Since the response times of aperiodic tasks are mostly determined by the service time and period of the server, a queueing model is proposed to analyze the performance of the candidate-servers to find out the one providing the minimal mean response time for aperiodic tasks under different workloads. The analytical and simulation results both shows that the server with a longer period is usually the optimal when the arrival rate of aperiodic tasks is low. In contrast, if the arrival rate is high, a server with a shorter period is more desirable. It is also suggested that the service time of the server be greater than the mean computation time of aperiodic tasks for better performance. The extensions of our current research include adaptive task scheduling and task scheduling for real-time distributed systems.

References

[1] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. ACM*, vol. 20, Jan. 1973.

[2] J. Lehoczky, L. Sha, and J. Strosnider, "Enhanced aperiodic responsiveness in hard real-time environments," *Proc. 1987 Real-Time System Symp.*, 1987.

[3] B. Sprunt, J. Lehoczky, and L. Sha, "Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm," *Proc. 1988 Real-Time System Symp.*, 1988.

[4] Lehoczky, Sha, and Ding, "The rate monotonic scheduling algorithm - exact characterization and average case behavior," *Technical Report Dept. Statistics, Carnegie-Mellon U.*, Mar. 1988.

[5] L. Kleinrock, "Queueing systems," *Volume 1: Theory, Wiley, New York*, 1975.