A Class of Rate-Based Real-Time Scheduling Algorithms

Tei-Wei Kuo, *Member*, *IEEE Computer Society*, Wang-Ru Yang, and Kwei-Jay Lin, *Member*, *IEEE Computer Society*

Abstract—This paper investigates a class of rate-based real-time scheduling algorithms based on the idea of general processor sharing (GPS). We extend the GPS framework in [18] for periodic and sporadic process scheduling and show the optimality of GPS-based scheduling. In particular, we propose the Earliest-Completion-Time GPS (EGPS) scheduling algorithm to simulate the GPS algorithm with much lower run-time overheads. The schedulability of each process is enforced by a guaranteed CPU service rate, independent of the demands of other processes. We provide a theoretical foundation to assign proper CPU service rates to processes to satisfy their individual stringent response time requirements. We also propose a GPS-based scheduling mechanism for jitter control. Finally, the performance of the proposed algorithms is studied using a generic avionics platform example and simulation experiments on jitter control and mixed soft and hard real-time process scheduling.

Index Terms—Generalized processor sharing, real-time process scheduling, service rate adjustment, jitter control, sporadic process scheduling, soft real-time process scheduling.

1 Introduction

The real-time resource allocation problem has been an active research topic in the past decades. Optimal scheduling algorithms, such as the rate monotonic scheduling algorithm, the earliest deadline first algorithm, and the least slack first algorithm [14], [16], were proposed in different contexts. As the limitations and strength in priority-driven scheduling are better understood, researchers have begun studying scheduling based on different disciplines such as time-driven scheduling [8], [9] and rate-based scheduling [4], [5], [24], [25], [26], [27].

This paper investigates a class of rate-based real-time scheduling algorithms based on the idea of generalized processor sharing (GPS) which was proposed by Parekh and Gallager [18] in the context of rate-based flow and congestion control at network gateway nodes. The idea of GPS-based scheduling is very different from common disciplines used in real-time systems such as priority-driven scheduling [1], [14], [16], [21], [22], [23] and time-driven scheduling [8], [9]. The GPS-based scheduling is a work-conserving scheduling mechanism. The schedulability of each process in GPS-based systems is guaranteed with an assigned CPU service rate, independent of the demands of other processes. The enforcement of a guaranteed CPU service rate for a process must rely on certain admission control mechanisms to manage the total workload of the system [24].

Past work on GPS-based process scheduling, e.g., [25], [26], [27], mainly focused on how to emulate GPS scheduling such that the difference between the service time that a process should receive under GPS and the time it actually receives is minimized. The time complexity of such algorithms is usually very high. Some researchers focused on handling sporadic processes [24], as well as the issues in scheduling real-time and non-real-time applications in a two-level hierarchical scheduling scheme [4], [5]. Moreover, most past research assumes that the guaranteed CPU service rate of a process is equal to its utilization factor. The goal of this paper is to study the GPS framework [18] for practical systems. We first investigate an efficient implementation of GPS, called Earliest-Completion-Time GPS (EGPS), and prove its desirable properties. The schedulability of each process is guaranteed with an assigned CPU service rate, independent of the demands of other processes. We provide a sufficient condition and prove theorems to adjust CPU service rates for processes to guarantee their individual stringent response time requirements. We then extend EGPS for practical system issues, such as efficient scheduling of periodic and sporadic processes and jitter control. Using our results, application engineers can not only implement GPS-based scheduling for practical real-time applications with stringent deadline requirements, but also have a way to adjust or bound the completion time of any selected process in the system. We demonstrate the feasibility of the rate adjustment mechanism by a case study on the generic avionics platform example [13]. Our experimental results show that the proposed GPS-based scheduling algorithms have good performance in mixing the scheduling of soft and hard real-time processes and in largely reducing process jitters.

- *T.-W. Kuo is with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan 106, ROC. E-mail: ktw@csie.ntu.edu.tw.*
- *W.-R. Yang is with First International Corp., Taipei, Taiwan, ROC.*
- *K.-J. Lin is with the Department of Electrical and Computer Engineering, University of California, Irvine, Irvine, CA 92697. E-mail: klin@ece.uci.edu.*

Manuscript received 24 Aug. 1998; revised 24 Oct. 2000; accepted 2 May 2001.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 107304.

The contribution of this work is as follows: 1) We extend the GPS framework [18] for scheduling processes with different timing characteristics and response requirements. The idea of CPU bandwidth reservation is used successfully in jitter control, sporadic process scheduling, and mixed scheduling of hard and soft real-time processes. 2) A rate adjustment framework has been developed to meet the response time requirement of each individual periodic or sporadic process. Theorems are provided that a sufficient condition is to guarantee the schedulability of processes.

The rest of the paper is organized as follows: Section 2 provides an overview of related research on GPS-based scheduling. Section 3 presents the basic scheduling algorithm which simulates the GPS algorithm without the impractical assumptions of the GPS algorithm. We then propose our methodologies for service rate adjustment, jitter-control, and sporadic process scheduling based on the basic scheduling algorithm. Section 4 presents a case study and experimental results using the proposed algorithms. The paper is concluded in Section 5.

2 GPS-Based Scheduling and Related Work

2.1 Related Work

Recently, the idea of generalized processor sharing (GPS) has received a great deal of attention in the context of process scheduling and packet transmission [4], [5], [18], [24], [25], [26], [27]. The idea of GPS-based scheduling is very different from common disciplines used in real-time systems such as priority-driven scheduling [1], [14], [16], [21], [22], [23] and time-driven scheduling [8], [9].

The GPS-based scheduling is a work-conserving scheduling mechanism. The schedulability of each process is guaranteed with an assigned CPU service rate, independent of the demands of other processes. On the other hand, the idea of priority-driven scheduling relies on an effective method for priority assignment. The interactions among processes with different priorities are often found to be very difficult to analyze or even quantify. Although priority-driven scheduling problems have been analyzed for different architectural assumptions and contexts in the past decades, e.g., [1], [14], [16], [21], [22], [23], numerous challenges still remain.

Time-driven scheduling is a classical but useful scheduling paradigm which can have good predictability and low runtime overheads. It is shown to perform well in jitter control and scheduling in distributed systems [8], [9]. Concerns on the inflexibility of system maintenance and development do exist if the system is not implemented carefully [9]. Time-driven scheduling is similar to GPS-based scheduling in the sense that they all depend on the service rate of each process. The time-driven schedule usually needs to have period (or rate) transformation to leave every operation or process into a time slot of a major cycle. The transformation often results in an increase of the system workload. On the other hand, GPS-based scheduling provides a simple work-conserving mechanism. The schedulability of each process is simply guaranteed with an assigned CPU service rate. The price paid for GPS-based scheduling is runtime overheads in scheduling, context switches, process synchronization, etc.

Parekh and Gallager [18] were the first to use the idea of GPS for rate-based flow and congestion control at network gateway nodes. Waldspurger and Weihl [26], [27] were the first to develop a process scheduling algorithm based on the idea of GPS. Their algorithm allocates a time quantum to the process which has the worst progress in the system according to its resource reservation. They showed that the *service time lag* of a process at any time can be reduced to $O(\log n)$, where the service time lag is the difference between the service time that a process should receive under GPS and the time it actually receives. Stoica et al. [25] proposed and analyzed another GPS-based process scheduling algorithm, where CPU time is allocated in discrete-sized time quanta such that a process, regardless of whether it is a real-time or non-real-time process, makes progress at a uniform rate. They were the first to implement and to test a GPS-based process scheduling algorithm which guarantees constant lag bounds.

Spuri et al. [24] proposed an effective mechanism, called *Total Bandwidth (TB) server*, to service sporadic processes. The capacity and performance of a TB server are guaranteed by preserving the CPU bandwidth. The schedulability of a TB server and other processes in the system is guaranteed under the framework of the EDF scheduling. A rejection mechanism is also proposed for overload handling of a TB server. Deng et al. [4], [5] proposed a two-level hierarchical scheme for scheduling real-time and non-real-time (multithreaded) applications. The schedulability of each application is guaranteed independently of other applications by an underlying OS scheduler which adopts the idea of GPS. Different scheduling algorithms may be used for different applications. They considered different contexts, such as the existence of global resource sharing and nonpreemptable critical sections among applications.

Our proposed scheduling algorithm, EGPS, is closely related to the process scheduling algorithms proposed in [4], [5], [24], [25], [26], [27]. The advanced techniques from the above-mentioned work, such as lag management, quantum-based scheduling, TB server, and two-level hierarchical scheduling scheme, are orthogonal to our work and can be made to complement each other. For example, each TB server or real-time application can be considered as a periodic (or sporadic) process in the EGPS algorithms. Various techniques, such as service rate adjustment, developed in this paper can be applied to the performance and/or capacity management of TB servers or real-time applications in an open environment [4], [5], [24].

2.2 General Processor Sharing

The idea of the Generalized Processor Sharing (GPS) was proposed earlier by Parekh et al. [3], [18] in the context of rate-based flow and congestion control at network gateway nodes. When combined with leaky bucket admission control, the worst-case performance on throughput and delay can be guaranteed.

GPS is an idealized multiplexing discipline based on the concept of the reservation ratio of processor computation time. It assumes that traffic is infinitely divisible such that the server can serve multiple traffic sessions

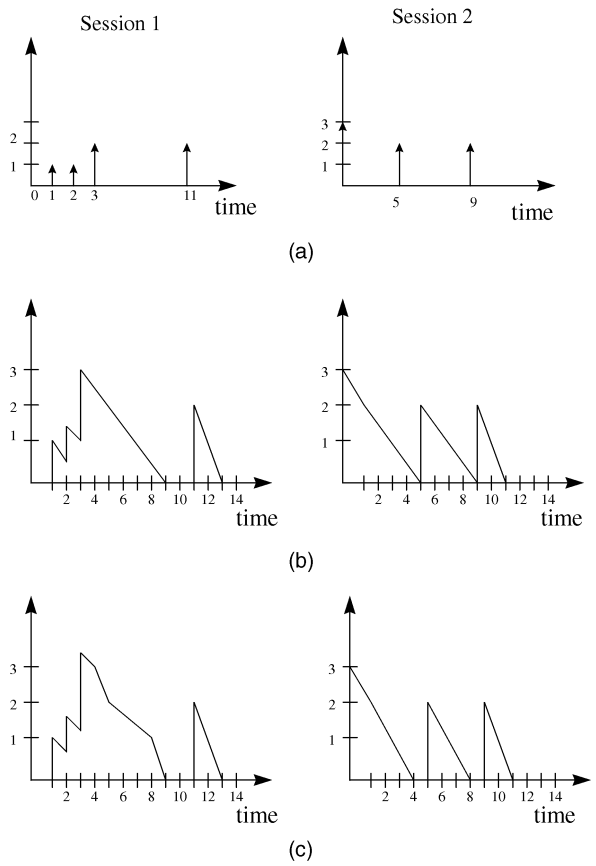


Fig. 1. GPS schedules. (a) Packet arrivals. (b) $\phi 1 = \phi 2$. (c) $2\phi 1 = \phi 2$.

simultaneously. Suppose a GPS server executes at a fixed rate r and each traffic session i is characterized by a positive real number θ_i, $i = 1, 2, \cdots, N$ (θ_i is called the *reservation ratio* of session i). Each session i is guaranteed to be served at a rate of

$$g_i = \frac{\theta_i}{\sum_j \theta_j} r.$$

Suppose that $S_i(t_1, t_2)$ is the amount of traffic session i served in the interval $[t_1, t_2]$. For any session i backlogged in the interval $[t_1, t_2]$,

$$\frac{S_i(t_1, t_2)}{S_j(t_1, t_2)} \geq \frac{\theta_i}{\theta_j}, j = 1, 2, \cdots, N.$$

Example 1. *A GPS schedule:* Fig. 1 illustrates the services received by two sessions under the GPS algorithm. Let a GPS server operate at a fixed rate $r = 1$. Fig. 1a describes the arrival times and sizes of packets for sessions 1 and 2. Fig. 1b shows the services of the two sessions when $\theta_1 = \theta_2$. When $\theta_1 = \theta_2$ and both sessions are backlogged, each session is served at rate $\frac{1}{2}$. For example, the packet of session 2 which arrives at time 0 is served at the full speed of the GPS server at time 0 since session 2 is the only active session in the system. At time 1, when the first packet of session 1 arrives, sessions 1 and 2 are both backlogged and the packet of session 2 is served at rate $\frac{1}{2}$. The packet is transmitted at time 5 and the amount of accumulated packet size under transmission for session 1 at time 5 is 2. Fig. 1c shows the services of sessions 1 and 2 when $2\theta_1 = \theta_2$. When $2\theta_1 = \theta_2$, and both sessions are backlogged, session 1 is served at rate $\frac{1}{3}$ and session 2 is served at rate $\frac{2}{3}$.

3 Earliest-Completion-Time GPS (EGPS) Scheduling

Since GPS assumes that processes are infinitely divisible such that the server can serve them simultaneously, it is impractical for actual computer systems. In this section, we first propose a basic scheduling algorithm similar to the PGPS algorithm [18] and show the optimality of the scheduling algorithm. We then use the algorithm as a framework for our methodologies in service rate adjustment, sporadic process scheduling, and jitter control.

3.1 The EGPS Scheduling Framework

3.1.1 The EGPS Protocol

The Earliest-Completion-Time GPS (EGPS) scheduling algorithm is designed to simulate the GPS algorithm with much lower runtime overheads. In this section, we assume that all processes are periodic. This constraint will be relaxed in Section 3.3.

We now show how to model a static set of processes as a collection of traffic sessions for EGPS scheduling. Let c_i and p_i be the computation time and the period of a periodic process τ_i, respectively, for $i = 1, 2, \cdots, N$. Each process τ_i is modeled as a traffic session i in which a packet of size c_i arrives at a regular interval p_i, and the reservation ratio θ_i of process τ_i is set to its utilization factor $u_i = \frac{c_i}{p_i}$.

The EGPS algorithm: EGPS uses a preemptive priority-driven scheduling approach in which processes are scheduled in the order of their GPS completion times. Any ready process request with the earliest GPS completion time is executed first. Processes with the same GPS completion time are executed on a FCFS basis.

The GPS completion time for each process is computed based on the total load at the time. Since the total system load changes with the arrival or the completion of process requests, all GPS completion times must be updated accordingly. Therefore, the implementation of EGPS depends on the existence of an efficient way to track the system load. The implementation of EGPS will be discussed in the Appendix.

Example 2. *An EGPS schedule:* We illustrate EGPS scheduling by an example. Suppose there are two periodic processes, τ_1 and τ_2, in a single processor environment. τ_1 and τ_2 have computation time requirements 2 and 3, respectively, and they have periods 6 and 9, respectively. The reservation ratios θ_1 and θ_2 of τ_1 and τ_2 are equal to their utilization factors $\frac{1}{3}$ and $\frac{1}{3}$, respectively. The GPS completion times of process requests are shown in Table 1.

At time 0, the first request of τ_1 arrives. Since it is the only ready process, τ_1 executes and finishes its execution at time 2. At time 6, the second request of τ_1 and the first request of τ_2 arrive. Since the GPS completion time of the

TABLE 1
GPS Completion Time of Example 1

	1st period	2nd period	3rd period
τ_1	2	10	14
τ_2	11	18	29

first request of τ_2 is later than the GPS completion time of the second request of τ_1, τ_1 is assigned the processor. At time 8, τ_1 finishes its execution and τ_2 starts execution. At time 11, τ_2 finishes its execution. The rest of the schedule can be found in Fig. 2.

Using EGPS, the reservation ratio θ_i of each process τ_i is set to its utilization factor $u_i = \frac{c_i}{p_i}$. Processes are also assumed to be periodic. We shall later extend the EGPS scheduling framework in the following ways: 1) The reservation ratio θ_i of each process τ_i may be different from its utilization factor u_i to allow the adjustment of the response time of process τ_i. 2) The response time of each sporadic process is considered in the same framework for scheduling periodic processes. 3) The ready time of a process in each period may be adjusted to minimize the variation of the completion times of a process in consecutive periods.

3.1.2 Properties

In this section, we shall further extend the idea of EGPS by considering the case when the reservation ratio of a process is different from its utilization factor and then show the optimality of the GPS and EGPS algorithms. (A sufficient condition will be presented in Section 3.2 to verify the schedulability of real-time processes more precisely.) We will also show that the overhead in implementing EGPS is very low and the schedulability of each process under EGPS is guaranteed with an assigned CPU service rate, independent of the demands of other processes.

Suppose that $T = \{\tau_1, \tau_2, \cdots, \tau_N\}$ is a static set of independent periodic processes. Let $u_i = \frac{c_i}{p_i}$ be the utilization factor of process τ_i and $U = \sum_{i=1,N} u_i \leq 1$. The reservation ratio θ_i of process τ_i is a positive real number which may or may not be equal to u_i. We assume that a process request which misses its deadline remains active (as assumed in GPS scheduling of network packets [18]). We can show the following theorems:

Claim 1. *Under GPS scheduling, process τ_i is guaranteed a service rate of $g_i = \frac{\theta_i}{\sum_j \theta_j}$.*

Proof. The proof directly follows from the definitions of the GPS algorithm. □

Lemma 1. *Under GPS scheduling, process τ_i is guaranteed to finish its execution in every period no later than $\frac{c_i}{g_i}$ time units after the request time if $\frac{c_i}{g_i} \leq p_i$. When $\theta_i = u_i$ for all processes and $U = \sum_j u_j \leq 1$, process τ_i is guaranteed to finish its execution in every period no later than $U * p_i$ time units after the request time.*

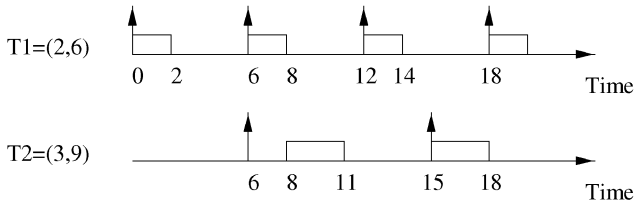


Fig. 2. An EGPS schedule of real-time periodic processes.

Proof. Claim 1 shows that process τ_i is guaranteed a service rate of g_i. Therefore, τ_i is guaranteed to finish its execution in every period no later than $\frac{c_i}{g_i}$ time units after the request time. When $\theta_i = u_i$ for all processes and $U = \sum_j u_j \leq 1$, τ_i is guaranteed to finish its execution in every period no later than $\frac{c_i}{g_i} = \frac{c_i}{\frac{u_i}{U}} = c_i * U * \frac{p_i}{c_i} = U * p_i$ time units after the request time. □

Theorem 1. *The achievable utilization factor of the GPS algorithm is 100 percent. In other words, the GPS algorithm can schedule any process set which is not overloaded.*

Proof. The proof directly follows from Lemma 1. □

Denote the *real* times at which the jth request of process τ_i completes its execution under GPS and EGPS as $GPS\_CT_{i,j}$ and $EGPS\_CT_{i,j}$, respectively. We shall show that the completion time of a process under EGPS scheduling will be no later than that under GPS scheduling and EGPS is an optimal scheduling algorithm with the achievable utilization factor equal to 100 percent. We will then show that the overhead in implementing EGPS is low.

Lemma 1. $EGPS\_CT_{i,j} \leq GPS\_CT_{i,j}$ *for any* jth *request of any process* τ_i.

Proof. Let a GPS scheduler multiplex the processor among ready processes in a unit of ϵ time units, where ϵ is any sufficiently small positive real number. We shall show the correctness of this lemma by transforming a schedule produced by a GPS scheduler for the interval $[0, n\epsilon]$ (for any integer $n > 0$) to one that is produced by an EGPS scheduler such that $EGPS\_CT_{i,j} \leq GPS\_CT_{i,j}$ for any jth request of any process τ_i in the interval. The proof is by induction on n.

The statement is trivially true at time $0 = 0\epsilon$ for $n = 0$. Suppose that the induction hypothesis holds for interval $[0, n\epsilon]$ and the kth request, i.e., $\tau_{p,k}$, of process τ_p is scheduled in the interval $[n\epsilon, (n+1)\epsilon]$ while there is another ready process request $\tau_{q,m}$ with the smallest completion time $GPS\_CT_{q,m} < GPS\_CT_{p,k}$ among all of the ready process requests. Notice that both of the process requests $\tau_{q,m}$ and $\tau_{p,k}$ must finish execution before their GPS completion time $GPS\_CT_{q,m}$ and $GPS\_CT_{p,k}$, respectively, according to Lemma 1. Thus, we can simply schedule the mth request, i.e., $\tau_{q,m}$, of process τ_q in the interval $[n\epsilon, (n+1)\epsilon]$ and schedule the kth request of process τ_p in the first unit interval occupied by the mth request of process τ_q after $(n+1)\epsilon$ and before $GPS\_CT_{q,m}$. Because $GPS\_CT_{q,m} < GPS\_CT_{p,k}$, both process requests still complete before their GPS completion times and the resulted schedule for the interval $[0, (n+1)\epsilon]$ is an $EGPS$ schedule. □

Lemma 3. *Under EGPS scheduling, process τ_i is guaranteed to finish its execution in every period no later than Up_i time units after the request time, where $\theta_i = u_i$ for all processes and $U = \sum_j u_j \leq 100\%$.*

Proof. The proof directly follows from Lemmas 1 and 2. □

Theorem 2. *The achievable utilization factor of the EGPS algorithm is 100 percent. In other words, the EGPS algorithm can schedule any process set which does not overload the system.*

Proof. The proof follows directly from Lemma 3. □

Theorem 2 guarantees that each process under EGPS will always meet its deadline if the total system utilization factor $U = \sum_j u_j$ is no more than 100 percent. That is, once a process τ_i is given a reservation ratio $u_i = \frac{c_i}{p_i}$, the process will always finish its execution in every period no later than $U * p_i$ time units after the request time, regardless of any bad behavior of other processes (see Lemmas 1 and 2). The enforcement of a guaranteed CPU service rate for a process must rely on a certain admission control mechanism to manage the total workload of the system [24]. As a result, EGPS, which provides a work-conserving scheduling mechanism, is very different from common disciplines such as priority-driven scheduling [1], [14], [16], [21], [22], [23] and time-driven scheduling [8], [9]. In the following sections, we provide an intuitive way for application engineers to verify the schedulability of real-time processes better, especially when the reservation ratio of a process is different from its utilization factor. We will further extend EGPS to address other important scheduling issues such as preperiod deadlines and jitter.

Corollary 1. *The schedulability of each process is enforced by a guaranteed CPU service rate, independent of the demands of other processes, if the system is not overloaded.*

Proof. The proof directly follows from Theorem 2. □

Lemma 4. *Under EGPS scheduling, the maximum number of context switching per process request is two.*

Proof. Since the priority order of ready processes under EGPS remains fixed during process executions and EGPS follows a stack discipline in scheduling processes, the maximum number of context switching per process request is two. One counts for the preemption of the executing process and the other counts for the completion of the process. □

Lemma 5. *If the number of process requests before time t is M, then the calculations of virtual time $V()$ and $Next()$ for the predictions of GPS completion times in EGPS scheduling are no more than $2M$ times.*

Proof. The proof directly follows from the fact that $V()$ and $Next()$ need to be calculated only when a GPS event occurs. There are at most $2M$ GPS events for M process requests. (The implementation of EGPS scheduling, including the calculations of $V()$ and $Next()$, can be found in the Appendix.) □

3.2 A Sufficient Schedulability Condition for EGPS with Service Rate Adjustment

Past research in GPS-based process scheduling [24] often assumes that the reservation ratio of a process is equal to its utilization factor, which is defined as the ratio of its computation requirements and its minimum separation time. However, processes in many real-time applications have preperiod or even postperiod deadlines. GPS-based scheduling, which assumes that the reservation ratio and the utilization factor are equal, cannot be applied to such applications. For example, when playing an MPEG movie, the interval between the display of two consecutive frames must remain nearly a constant. Therefore, the display of a frame in each period of an MPEG movie has a stringent jitter requirement. Finishing the display of a frame before its next period is not good enough to smoothly play an MPEG movie. This observation motivates us to develop a methodology to guarantee the individual response time requirements of processes whose reservation ratios may or may not be equal to their utilization factors.

In this section, we provide an intuitive way for application engineers to verify the schedulability of real-time processes better, especially when the reservation ratio of a process is different from its utilization factor. We relate the concept of blocking time in real-time process scheduling to the rate assignment mechanism. A sufficient condition is provided to verify the schedulability of real-time processes. The research results also provide a convenient way to manage the jitter problem of critical processes. We shall show the practicality of this work by a case study on the generic avionics platform example [13] in Section 4.1.

3.2.1 Theorems for Schedulability Analysis

Suppose the reservation ratio θ_i of a process τ_i is a positive real number which may or may not be equal to u_i. Process τ_i is guaranteed to be served at a rate of $g_i = \frac{\theta_i}{\sum_j \theta_j}$ under GPS. As astute readers may notice, Lemmas 1 and 2 are very conservative to estimate the completion time of a process with a small reservation ratio. We shall show that a process τ_i with a large reservation ratio (and, thus, a potentially large g_i) may not result in much delay on the completion time of a process τ_j with a small reservation ratio (and a potentially small g_j) because the computation time c_i of process τ_i does not increase in proportion to the value of its reservation ratio.

Let process τ_i be the only process with the reservation ratio θ_i larger than u_i, and $U = \sum_{1 \leq k \leq N} u_k \leq 100\%$. Define $I_i = (\frac{c_i(U-u_i+\theta_i)}{\theta_i})$. I_i is not only the bound of the completion time of process τ_i but also the (maximum) bound of an interval in each period of τ_i such that other active processes τ_j have a small guaranteed service rate g_j (because of a large value for θ_i). We can show the following theorems for the EGPS algorithm:

Theorem 3. *Each request of process τ_j $(j \neq i)$ is guaranteed to complete before its next period if $c_j \leq p_j \frac{u_j}{U-u_i+\theta_i}$, where*

$p_j \le I_i$. In other words, process τ_j is guaranteed to finish its execution in every period no later than $p_j(U - u_i + \theta_i)$ time units after the request time.

Proof. Since $p_j \le I_i$, the entire period of a request of process τ_j may overlap the I_i intervals of some request of process τ_i. Thus, the request of process τ_j may only receive a total amount of $p_j \frac{u_j}{U-u_i+\theta_i}$ time units for execution under GPS. In other words, each request of process τ_j is guaranteed to complete before its next period under GPS if $c_j \le p_j \frac{u_j}{U-u_i+\theta_i}$. Since Lemma 2 shows that the EGPS completion time of a process request is no later than its GPS completion time, each request of process τ_j is guaranteed to complete before its next period under EGPS if $c_j \le p_j \frac{u_j}{U-u_i+\theta_i}$. In other words, process τ_j is guaranteed to finish its execution in every period no later than $\frac{c_j(U-u_i+\theta_i)}{u_j} = p_j(U - u_i + \theta_i)$ time units after the request time. □

Theorem 4. *Each request of process τ_j $(j \ne i)$ will complete before its next period if $c_j \le (p_j - \lceil \frac{p_j}{p_i} \rceil I_i) \frac{u_j}{U-u_i} + \lceil \frac{p_j}{p_i} \rceil I_i \frac{u_j}{U-u_i+\theta_i}$, where $p_j > I_i$.*

Proof. Since the number of requests from process τ_i within each period of process τ_j is at most $\lceil \frac{p_j}{p_i} \rceil$, there can be at most $\lceil \frac{p_j}{p_i} \rceil$ number of I_i intervals in which process τ_j has a small guaranteed service rate g_j under GPS (in the worst case). Within each such I_i interval, process τ_j receives at most $I_i \frac{u_j}{U-u_i+\theta_i}$ units of processor time under GPS. Besides these I_i intervals, process τ_j may receive an amount of $(p_j - \lceil \frac{p_j}{p_i} \rceil I_i) \frac{u_j}{U-u_i}$ time units for execution under GPS. Thus, process τ_j may receive a total amount of $(p_j - \lceil \frac{p_j}{p_i} \rceil I_i) \frac{u_j}{U-u_i} + \lceil \frac{p_j}{p_i} \rceil I_i \frac{u_j}{U-u_i+\theta_i}$ time units for execution in each period under GPS. Obviously, if c_j is no larger than this amount, each request of τ_j will complete before its next period under GPS. Since Lemma 2 shows that the EGPS completion time of a process request is no later than its GPS completion time, each request of τ_j will also complete before its next period under EGPS. □

Corollary 2. *Each request of process τ_j $(j \ne i)$ will complete before its next period if $p_j \ge \lceil \frac{p_j}{p_i} \rceil c_i + (U - u_i)p_j$, where $p_j > I_i$ and $\theta_i >> u_i$.*

Proof. When $\theta_i >> u_i$, Theorem 4 implies that process τ_i may receive up to $\lceil \frac{p_j}{p_i} \rceil c_i$ units of CPU time in every period of process τ_j because $I_i \simeq c_i$. In other words, process τ_j may have to share only $(p_j - \lceil \frac{p_j}{p_i} \rceil c_i)$ units of CPU time with other processes (besides τ_i) in its period under GPS. According to the definition of the GPS algorithm, process τ_j will receive a guaranteed amount $\frac{u_j}{U-u_i}(p_j - \lceil \frac{p_j}{p_i} \rceil c_i)$ of CPU time in its period. Obviously, if $c_j \le \frac{u_j}{U-u_i}(p_j - \lceil \frac{p_j}{p_i} \rceil c_i)$, then process τ_j will complete before its next period under GPS. Since $c_j \le (p_j - \lceil \frac{p_j}{p_i} \rceil c_i) \frac{u_j}{U-u_i}$ implies that $(U - u_i)p_j \le (p_j - \lceil \frac{p_j}{p_i} \rceil c_i)$ because $u_j = \frac{c_j}{p_j}$. $p_j \ge \lceil \frac{p_j}{p_i} \rceil c_i + (U - u_i)p_j$. Since Lemma 2 shows that the EGPS completion time of a process request is no later than its GPS completion time, each request of τ_j will also complete before its next period under EGPS if the above condition is true. □

Note that, when $p_i > p_j$, Corollary 2 implies that process τ_j is guaranteed to finish its execution no later than $(U - u_i)p_j + c_i$ time units after the request time. Process τ_i introduces "blocking time" of c_i time units to process τ_j. The idea of Corollary 2 can be further extended in the following way:

Let processes in a process set T be partitioned into two exclusive process sets T_1 and T_2, where each process τ_i in T_1 has a reservation ratio $\theta_i >> u_i$ and each process τ_j in T_2 has a reservation ratio $\theta_j = u_j$. Let I_i be the maximum length of an interval in a period of τ_i such that another active process τ_j has a small guaranteed service rate g_j (because of a large value for θ_i).

We can show the following corollary:

Corollary 3. *Each request of process τ_j in T_2 will complete before its next period if $p_j \ge (\sum_{i \in T_1} \lceil \frac{p_j}{p_i} \rceil c_i) + U_2 p_j$, where $p_j > I_i$ for any $i \in T_1$ and $j \in T_2$, and $U_2 = \sum_{i \in T_2} u_i$.*

Proof. Similarly to the argument in Corollary 2, process τ_j may have to share only $(p_j - (\sum_{i \in T_1} \lceil \frac{p_j}{p_i} \rceil c_i))$ units of CPU time with other processes in T_2 in its period under GPS. According to the definition of the GPS algorithm, process τ_j will receive a guaranteed amount $\frac{u_j}{U_2}(p_j - (\sum_{i \in T_1} \lceil \frac{p_j}{p_i} \rceil c_i))$ of CPU time in its period. Obviously, if $c_j \le \frac{u_j}{U_2}(p_j - (\sum_{i \in T_1} \lceil \frac{p_j}{p_i} \rceil c_i))$, then process τ_j will complete before its next period under GPS. Since $c_j \le \frac{u_j}{U_2}(p_j - (\sum_{i \in T_1} \lceil \frac{p_j}{p_i} \rceil c_i))$ implies that $U_2 p_j \le (p_j - (\sum_{i \in T_1} \lceil \frac{p_j}{p_i} \rceil c_i))$ (because $u_j = \frac{c_j}{p_j}$), $p_j \ge (\sum_{i \in T_1} \lceil \frac{p_j}{p_i} \rceil c_i) + U_2 p_j$. Since Lemma 2 shows that the EGPS completion time of a process request is no later than its GPS completion time, each request of τ_j will also complete before its next period under EGPS if the above condition is true. □

Corollary 3 can be further extended when each process τ_j in T_2 has a reservation ratio $\theta_j \le u_j$:

Corollary 4. *The response time of each request of process τ_j in T_2 will complete before its next period if $p_j \ge (\sum_{i \in T_1} \lceil \frac{p_j}{p_i} \rceil c_i) + \frac{U_2' c_j}{\theta_j}$, where $p_j > I_i$ for any $i \in T_1$ and $j \in T_2$, and $U_2' = \sum_{i \in T_2} \theta_i$.*

Proof. This lemma can be proven in a way similar to that in Corollary 3. □

The theorems provided in this section can be used to guarantee the schedulability of periodic processes. For example, Lemmas 1 and 2 can be used to estimate the worst-case completion time of periodic processes with a large reservation ratio (i.e., processes with stringent response time requirements). Corollary 4 was used to guarantee the schedulability of periodic processes with a reservation ratio no larger than the process utilization factor. The schedulability tests in this section are to provide an intuitive and sufficient condition to test the schedulability of real-time processes. We relate the concept of blocking time in real-time process scheduling to the rate assignment mechanism in rate-based scheduling. The

results in this paper can be further extended by incorporating the concept of feasible ordering and service curves, originally proposed by Parekh and Gallager in [18]. The concept of feasible ordering and service curves (which considers a sequence of intervals in which different sessions are backlogged) is closely related to the Rate Monotonic Analysis (RMA) [23], where RMA considers the computation times requested by higher-priority real-time processes when a real-time process under schedulability tests has the worst-case response time [14]. Although RMA offers a more precise schedulability test, it is a pseudopolynomial-time schedulability test. The results in this section can be further extended into RMA-like pseudo-polynomial-time tests based on the concept of feasible ordering and service curves. It can be done by considering the periodic occurrences of real-time processes and their accumulated computation times under different busy intervals, where the corresponding sessions of real-time processes are backlogged (similar to the sequence of busy intervals under the concept of feasible ordering and service curves). Due to the real-time process model, the resulted schedulability tests are also pseudo-polynomial-time tests. We refer interested readers to [18], [19], [23] for details.

3.3 Sporadic Process Scheduling

In the past decades, researchers have proposed various methodologies to schedule sporadic processes. In particular, Mok proposed a simple and conservative methodology to reserve CPU cycles for each sporadic process [16]. The required CPU cycles of each sporadic process are guaranteed by using a periodic process. Sprunt proposed the idea of the Sporadic Server in the context of fixed priority scheduling, which not only utilizes the CPU cycles better, but also provides an improved response time for sporadic processes [21]. The *Total Bandwidth (TB) server* proposed by Spuri et al. [24] is the first sporadic process scheduling mechanism based on the idea of rate-based scheduling, although the TB server is scheduled with other processes under the earliest deadline first (EDF) scheduling [14]. The TB server is the first simple and effective mechanism for sporadic process scheduling in the context of dynamic priority scheduling. The reservation ratio of a TB server is set as the expected utilization factor of the processor for the server.

This section extends EGPS scheduling further for sporadic process scheduling. A sporadic process τ_i can be characterized by three timing parameters: a minimum separation time p_i, a worst-case computation time c_i, and a deadline d_i. The deadline of a sporadic process can be larger or smaller than its minimum separation time. In particular, we are interested in sporadic processes whose deadline is no larger than their individual minimum separation time in this paper. We shall discuss the relaxation of this condition later.

We allow the reservation ratio of a sporadic process τ_i to be different from its utilization factor, i.e., $\frac{c_i}{p_i}$. Theorems are shown to guarantee the schedulability of sporadic processes which may coexist with periodic processes in a system where an arbitrary assignment of reservation ratio for processes is possible. The goal of this work is not only to extend the EGPS scheduling for mixed scheduling of periodic and sporadic processes with arbitrary reservation ratios, but also to provide a sound theoretical foundation for applications engineers to assign proper reservation ratios for TB servers and other processes in the system.

3.3.1 Theorems for Schedulability Analysis

We now investigate how we can handle sporadic processes. Let process τ_j be a sporadic process with a minimum separation time p_j, a worst-case computation time c_j, a deadline d_j, and a guaranteed service rate $g_j = \frac{\theta_j}{\sum_{1 \le i \le N} \theta_i}$, respectively. Denote the *real* times at which the jth request of process τ_i completes its execution under GPS and EGPS as $GPS\_CT_{i,j}$ and $EGPS\_CT_{i,j}$, respectively. We can show the following theorem using EGPS:

Lemma 6. *$EGPS\_CT_{i,j} \le GPS\_CT_{i,j}$ for any jth request of any periodic (or sporadic) process τ_i with θ_i being any positive real number.*

Proof. This lemma can be proven in the same way adopted in Lemma 2, regardless of whether a periodic or sporadic process request misses its deadline or not, since EGPS and GPS both consider a process request as a scheduling unit and do not consider the periodic nature of any process. □

Theorem 5. *The response time of each request of process τ_j is no more than $\frac{c_j}{g_j}$, where $\frac{c_j}{g_j} \le p_j$.*

Proof. Requests from τ_j are guaranteed to be serviced at the rate g_j under GPS. If $\frac{c_j}{g_j} \le p_j$, then each process request can complete its execution no later than $\frac{c_j}{g_j}$ under GPS. Since Lemma 6 shows that the EGPS completion time of a process request is no later than its GPS completion time, each process request completes its execution no later than $\frac{c_j}{g_j}$ under EGPS. □

Theorem 5 can be further extended to consider the "blocking time" of process τ_j which is introduced by periodic processes τ_i with $\theta_i >> u_i$ (see Corollary 2). Let processes excluding τ_j in a process set T be partitioned into two exclusive process sets T_1 and T_2, where each process τ_i in T_1 has a reservation ratio $\theta_i >> u_i$ and each process τ_j in T_2 has a reservation ratio $\theta_j \le u_j$. Suppose that T_1 consists of periodic processes only. We can show the following corollary:

Corollary 5. *The response time of each request of process τ_j is no more than $(\sum_{i \in T_1} \lceil \frac{d_j}{p_i} \rceil c_i) + \frac{c_j}{g'_j}$, where d_j is the deadline of process τ_j, $g'_j = \frac{\theta_j}{\theta_j + \sum_{k \in T_2} \theta_k}$, and $(\sum_{i \in T_1} \lceil \frac{d_j}{p_i} \rceil c_i) + \frac{c_j}{g'_j}$ is no more than the minimum separation time p_j of process τ_j.*

Proof. This lemma can be proven in a way similar to that in Corollary 4. □

Theorem 5 does not consider the cases where $\frac{c_j}{g_j} > p_j$ (or $d_j > p_j$), when some sporadic processes arrive with certain distributions, and when more than one sporadic process share a service rate g_j. When $\frac{c_j}{g_j} > p_j$ (or $d_j > p_j$), the service of a request of a sporadic process τ_j may not be completed

when a new request arrives because of an insufficient reservation ratio g_j. The accumulation of pending requests might also happen when the arrivals and the computation time requirements of sporadic processes follow some distributions. Such situations can be analyzed and resolved by the service curve approach originally proposed by Parekh and Gallager [18], where the arrival of a traffic stream is bounded by a curve. Note that deterministic service guarantee, a more general concept of rate guarantee, has been well-developed in the networking area, e.g., [2], [18], [20]. The traffic aggregation approach developed in [2] can resolve the case when more than one sporadic process share a service rate g_j. In particular, the proposed SCED+ algorithm provides statistical multiplexing between the best-effort and "guaranteed" services. We refer interested readers to [2], [18], [20] for details.

3.4 Jitter-Control EGPS (JEGPS)

In many real-time applications, the temporal distance between consecutive completions of a process must be controlled to reduce jitters. Many real-time applications, such as multimedia systems and avionics software, often have stringent jitter requirements. Jitter control in traditional priority-driven scheduling often relies on proper priority assignments for processes. Such an approach often results in violations of timing constraints for less critical processes and degradation of overall system utilization. Jitter control is trivial in time-driven scheduling [8], [9]. As computations are properly assigned to time slots of a major cycle, no jitter will occur. The cost paid for time-driven scheduling is the difficulty in system maintenance and the increased system workload to fit computations into the major cycle.

The goal of this section is to propose a GPS-based mechanism to alleviate the jitter problem. Distinct from the methodology for service rate adjustment proposed in the previous section, the jitter-control mechanism proposed in this section aims at providing heuristics to improve the EGPS algorithm in jitter control.

Let $x_i = CT_{i,j} - CT_{i,j-1}$ denote the time difference between the completion times of process τ_i in the jth and $(j-1)$th periods. The *jitter* of process τ_i is defined as the variance of x_i divided by the period, i.e., $Variance(x_i)/p_i$. The idea for the jitter-control version of the EGPS algorithm is as follows: When a process produces the result too late in a period, it is better not to produce the result too early in the next period. Having results produced too close to each other from the same process may cause buffer overflows for multimedia tasks and even more jitters for future executions.

3.4.1 Jitter-Control Mechanism

The Jitter-Control EGPS (JEGPS) is a generalization of the EGPS algorithm to minimize the variation of the completion times of process requests in consecutive periods. If the request of a process completes its execution late in the current period, the request of the same process in the next period should not complete its execution too early. The JEGPS algorithm is defined as follows:

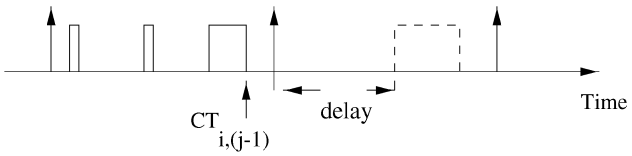


Fig. 3. The delay time of the jth request of process τ_i to reduce jitter.

Let $r_{i,j}$ be the *real* time at which the jth request of process τ_i arrives and $CT_{i,(j-1)}$ be the *real* time at which the $(j-1)$th request of process τ_i finished its execution. Suppose that the ready time of a process request is equal to its request time and we will use the terms "ready time" and "request time" interchangeably for the rest of this paper. The jth request time of process τ_i is set as follows:

$$r'_{i,j} = r_{i,j} + \min\{(p_i - Up_i), (CT_{i,(j-1)} - r_{i,(j-1)} - c_i)\},$$

where $(p_i - Up_i)$ is the maximum amount of slack time that the jth request of process τ_i has and $(CT_{i,(j-1)} - r_{i,(j-1)} - c_i)$ is the amount of time that the jth request of process τ_i wishes to delay its execution, as shown in Fig. 3. The GPS completion time of the jth request of process τ_i is calculated based on the new request time. Processes are scheduled in the order of their GPS completion times.

3.4.2 Properties

Suppose that $U(=\sum_{i=1,N} u_i) \leq 100\%$. Let the reservation ratio θ_i of each process τ_i be equal to its utilization factor $u_i = \frac{c_i}{p_i}$. Denote the *real* times at which the jth request of process τ_i completes its execution under GPS and JEGPS as $GPS\_CT_{i,j}$ and $JEGPS\_CT_{i,j}$, respectively. We can prove the following theorems:

Lemma 7. *Under the GPS algorithm, process τ_i with an adjusted ready time is guaranteed to finish its execution in every period no later than p_i time units after the (original) request time.*

Proof. Since the difference between the new request time $r'_{i,j}$ and the original request time $r_{i,j}$ is no more than $(p_i - Up_i)$, the correctness of this proof directly follows from Lemma 1. □

Lemma 8. *$JEGPS\_CT_{i,j} \leq GPS\_CT_{i,j}$ for any jth request of any process τ_i, where the process request times in a GPS schedule are based on the process request times in the corresponding JEGPS schedule.*

Proof. This lemma can be proven in a similar way as in Lemma 2. Note that the completion time of a process request is simply a function of its guaranteed CPU service rate and required computation time, regardless of whether the process request time is adjusted or not. □

Theorem 6. *The achievable utilization factor of the JEGPS algorithm is 100 percent. In other words, the JEGPS algorithm can schedule any process set which does not overload the system.*

Proof. The proof directly follows from Lemma 8. □

Obviously, JEGPS is also an optimal scheduling algorithm. It is trivial to show that other properties of EGPS

TABLE 2
Generic Avionics Platform

	period (ms)	execution time (ms)	reservation ratio	bound of worst-case completion time (ms)	u_i %
Timer Interrupt	1	0.051	12.048567	1	5.1
Weapon Release	200	3	141.74786	5	1.5
Radar Tracking Filter	25	2	8	24.8875	8
RWR Contact Mgmt	25	5	20	24.8875	20
Data Bus Poll Device	40	1	2.5	38.02	2.5
Weapon Aiming	50	3	6	46.775	6
Radar Target Update	50	5	10	46.775	10
Nav Update	59	8	12.5	58.777	13.56
Display Graphic	80	9	11.25	73.04	11.25
Display Hook Update	80	2	2.5	73.04	2.5
Tracking Target Update	100	5	5	90.55	5
Weapon Protocol	A	1	0.5	178.1	
Nav Steering Cmds	200	3	1.5	178.1	1.5
Display Stores Update	200	1	0.5	178.1	0.5
Display Ketset	200	1	0.5	178.1	0.5
Display Stat Update	200	3	1.5	178.1	1.5
BET E Status Update	1000	1	0.1	878.5	0.1
Nav Status	1000	1	0.1	878.5	0.1
Total Utilization	89.61%				

shown in the previous section remain here. We shall study the performance of JEGPS in Section 4.2.

4 Performance Evaluation

The performance of the proposed algorithms was verified by a case study on the generic avionics platform example [13] and a series of simulation experiments. A number of simulations experiments were done to compare the performance of the EGPS algorithms (EGPS and JEGPS) with other well-known real-time scheduling algorithms such as the rate monotonic scheduling (RMS) algorithm, the earliest deadline first (EDF) algorithm, the least slack first (LSF) algorithm, and the first-in-first-out (FIFO) algorithm. The LSF algorithm used a fixed slack (equal to the period minus the computation time) for each process.

The primary performance metrics used in this paper are the jitter of process executions in consecutive periods, referred to as the *Jitter*, and the ratio of requests that miss deadlines, referred to as the *Miss Ratio*. Let num_i and $miss_i$ be the total number of process requests and deadline violations during an experiment, respectively. *Miss Ratio* is calculated as $\frac{miss_i}{num_i}$. Let $x_j = f_j - f_{j-1}$ denote the time difference between the completion times of process τ_i in the jth and $(j-1)$th periods. The *Jitter* of process τ_i is defined as the variance of x_j divided by the period, i.e., $\frac{Variance(x_j)}{p_i}$.

4.1 Generic Avionics Platform

Table 2 shows the assignment of a reservation ratio for each process in the generic avionics platform example [13] to meet their respective jitter or deadline requirements. The assignment of a reservation ratio for each process was done as follows: Initially, each process was assigned a reservation ratio equal to its utilization factor. Then, we started to reassign reservation ratios to processes which have stringent response time requirements and processes which are affected by the reassignments.

Since process Weapon_Release has a stringent $5ms$ jitter requirement, the reservation ratios x and y of processes Timer_Interrupt and Weapon_Release were reassigned by solving the following equations: Note that the reservation ratio of process Timer_Interrupt had to be considered because the assignment of a large-valued reservation ratio for process Weapon_Release would seriously affect the schedulability of process Timer_Interrupt.

$$\frac{x}{82.45 + x + y} = 0.051,$$
$$\frac{y}{82.45 + x + y} = \frac{3}{5},$$

where 82.45 is the total reservation ratio of other processes. The above two equations were written according to Lemmas 1 and 2. Note that Lemmas 1 and 2 were used to estimate the worst-case completion time of periodic processes with a large reservation ratio (i.e., processes with stringent response time requirements). Corollary 4 was used to guarantee the schedulability of periodic processes with a reservation ratio no larger than the process utilization factor. Process Nav_Update was assigned a reservation ratio less than its utilization factor simply to demonstrate Corollary 4.

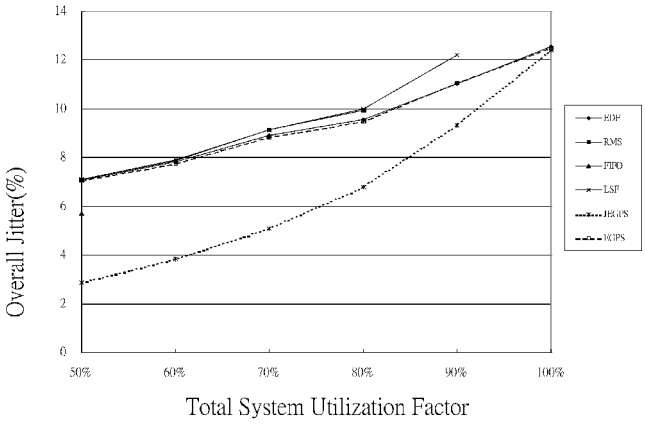


Fig. 4. Jitters of processes.

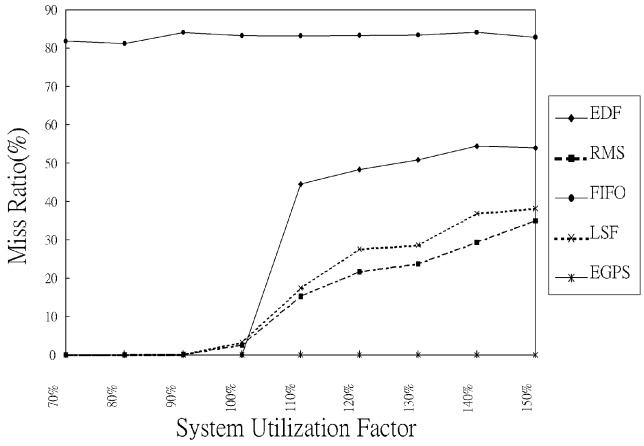


Fig. 5. Miss ratios of hard real-time processes.

The response time of sporadic processes, e.g., Weapon Protocol, was guaranteed by Corollary 5. We assumed that the minimum separation time of sporadic process Weapon_Protocol is $200ms$, i.e., the period of Display_ Keyset. The reservation ratio of process *Nav_Update* was set as 12.5 to satisfy the deadlines of processes *Radar_ Tracking_Filter* and *RWR_Contact_Mgmt*.

4.2 Jitter Control

In this section, we demonstrate the capability of the JEGPS algorithm on jitter control in a uniprocessor environment. Process sets in the simulation experiments were generated by a random number generator. The number of processes in a process set ranged from 10 to 20. The utilization factor of a process ranged from 2 percent to 30 percent and the period of a process ranged from 10 to $1,000$. Ten process sets were simulated for each system utilization factor and the simulation results were averaged. Each process set was simulated for all scheduling algorithms from time 0 to time $2,000,000$. The simulation experiments started from the system utilization factor equal to 50 percent to that equal to 100 percent.

Fig. 4 shows that JEGPS greatly outperformed EDF, RMS, FIFO, LSF, and EGPS in reducing the jitters of processes. The simulation results of process sets with system utilization factors equal to 60 percent and 90 percent and larger than 90 percent were not shown in Fig. 4 for FIFO, RMS, and LSF, respectively, because of some deadline violations for the algorithms. The lower the system utilization factor, the better the jitter control for JEGPS because of more flexibility in adjusting the ready time of processes. When the system utilization factor approached 100 percent, the advantage of JEGPS diminished because of less flexibility in adjusting the ready time of a process. Note that, when the system utilization factor is 100 percent, EGPS and JEGPS had exactly the same schedule as EDF.

4.3 Scheduling of Mixed Process Sets

This section is meant to demonstrate the capability of the EGPS algorithm in mixed scheduling of hard real-time and soft real-time processes. Process sets in the simulation experiments were generated by a random number generator. The number of processes in a process set ranged from 10 to 20. The utilization factor of a process ranged from 2 percent to 30 percent and the period of a process ranged from 10 to $1,000$. More than 10 process sets were simulated for each system utilization factor and the simulation results were averaged. Each process set was simulated for all scheduling algorithms from time 0 to time $2,000,000$. The simulation experiments started from the system utilization factor equal to 50 percent to that equal to 150 percent. The ratio of the total utilization factors of hard real-time and soft real-time processes is $1:2$. That is, when the total system utilization factor was 150 percent, the total utilization factors of hard real-time and soft real-time processes were 50 percent and 100 percent, respectively. The priority assignment of processes is according to that defined in each simulated algorithm. Note that the total utilization factor of hard real-time processes is no more than 100 percent.

Fig. 5 shows that the miss ratio of hard real-time processes scheduled by EDF, RMS, FIFO, LSF, and EGPS, respectively. EGPS greatly outperformed other scheduling algorithms and the schedulability of hard real-time processes was fully guaranteed. It is also interesting to see that the miss ratio of soft real-time processes scheduled by EGPS was very low, as shown in Fig. 6. In fact, it was about the same as the miss ratio of soft real-time processes scheduled

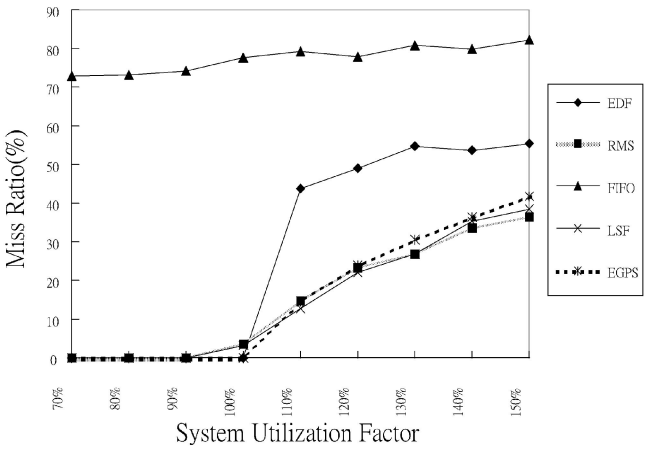


Fig. 6. Miss ratios of soft real-time processes.

by RMS and LSF. The low miss ratio of soft real-time processes scheduled by EGPS was partially because the EGPS reservation ratio for the processor was given to soft real-time processes in inverse order of their periods in the experiments. The miss ratio of JEGPS was the same as that of EGPS because the only difference between JEGPS and EGPS was on the adjustment of ready times of processes. Note that, since the process requests that missed their deadlines were aborted, the increasing rate of the miss ratio for each scheduling algorithm was much lower than that for each scheduling algorithm when process requests that missed their deadlines could be delayed to the next period. That also explained why the increasing rate of the miss ratio for EDF was very low.

5 Conclusions

This paper investigates a class of optimal real-time scheduling algorithms, called EGPS, based on the idea of generalized processor sharing. The schedulability of each process is guaranteed by an assigned CPU service rate, independent of the demands of other processes. We have presented a GPS-based scheduling framework for periodic and sporadic process scheduling, jitter control, service rate adjustment, and mixed scheduling of soft and hard real-time processes. In particular, we propose a methodology and theorems to adjust CPU service rates for processes to guarantee their individual stringent response-time requirements. The performance of the proposed algorithms is verified by a case study on the generic avionics example [13] and a series of simulation experiments. Techniques proposed by previous work, such as lag management, quantum-based scheduling, TB server, and two-level hierarchical scheduling scheme [4], [5], [24], [25], [26], [27], are really orthogonal to our work and can be made to complement each other. For example, each TB server or real-time application can be considered as a periodic (or sporadic) process in the EGPS scheduling. Various techniques, such as service rate adjustment, considered in this paper can be applied to the performance and/or capacity management of TB servers or real-time applications in an open environment [4], [5], [24].

Past research on process scheduling, e.g., [14], [16], has concentrated on the schedulability problem or the feasibility problem of a process system. Researchers proposed various "optimal" scheduling algorithms with a good achievable utilization factor, but ignored properties such as jitter management which may be important to some commercial time-critical systems such as Video-on-Demand systems [10]. We believe that more research in exploiting the application semantics and the needs of application systems may provide the best reward in solving the resource allocation problem. For future research, we plan to extend the EGPS algorithms to distributed real-time systems. With the homogeneity of the EGPS algorithms and network scheduling algorithms such as the Packet-by-Packet GPS algorithm [18], we expect that the EGPS algorithms will be good in handling processes with end-to-end delay requirements in a distributed real-time system [6], [7], [17]. The work reported in [19] for multinode GPS scheduling provides a good direction for extending the EGPS scheduling framework. We shall also explore more precise schedulability tests based on the concept of feasible ordering and service curves [18].

Appendix

The Implementation of EGPS

The practical implementation of EGPS depends on the existence of an efficient way to track the progress of GPS. A simple mechanism based on the idea of virtual time [18] is included in this appendix to track the progress of GPS. We refer interested readers to [18] for details.

Let each arrival or departure, i.e., service completion, of a packet from the GPS server be an event and t_i be the time at which the ith event occurs. B_i is the set of process requests that are busy in the interval (t_{i-1}, t_i). Suppose that $t_1 = 0$ denotes the arrival time of the first packet, i.e., process request. *Virtual time* $V(t)$ is defined as follows:

$$V(0) = 0$$

$$V(t_{i-1} + \delta) = V(t_{i-1}) + \frac{\delta}{\sum_{m \in B_i} \theta_m}, \delta \leq (t_i - t_{i-1}), i = 2, 3, \cdots$$

Virtual time $V(t)$ can be interpreted as being increased at a marginal rate

$$\frac{1}{\sum_{m \in B_i} \theta_m}$$

(at which backlogged sessions receive service). Let $V(t) = 0$ if the processor is idle at time t.

Let the jth request of process τ_k arrive at *real* time $a_{k,j}$. $S_{k,j}$ and $F_{k,j}$ denote the virtual times at which the system begins and completes $\tau_{k,j}$ under the GPS algorithm, respectively. If the total utilization factor of the process set is no larger than 100 percent, we have

$$S_{k,j} = V(a_{k,j}),$$
$$F_{k,j} = S_{k,j} + \frac{c_k}{\theta_k}.$$

Note that the virtual time of the GPS completion time of a process request is determined at the process arrival time. The EGPS algorithm simply schedules processes in order of the virtual times of the GPS completion times of the processes. The only overhead here is to keep track of B_i. The virtual time of a system is only updated when an event occurs.

The arrivals of events which correspond to new process requests can be observed during the system operation. The main difficulty in tracking GPS events lies in the calculation of the *real* time $Next(t)$ at which the next process request will complete service under the GPS algorithm (after *real* time t). Let F_{min} be the smallest virtual time of the GPS completion times of packets in the system at *real* time t. If there are no arrivals of packets in the *real* time interval $[t, Next(t)]$, $Next(t)$ can be calculated as follows (based on the way that virtual time is calculated):

$$Next(t) = t + (F_{min} - V(t)) \sum_{m \in B_i} \theta_m.$$

Thus, when a process request arrives at *real* time t, the virtual GPS completion time, i.e., $F_{k,j}$, of the process request

is calculated. The *real* time $Next(t)$ at which the next process request will complete under the GPS algorithm after *real* time t is also calculated. If some process request arrives at time t' and $t < t' < Next(t)$, then the calculations of the virtual time of the new process request and the *real* time $Next(t')$ are done again at time t'. If there is, indeed, no arrival of process requests between the *real* time interval $[t, Next(t)]$, the virtual time $V()$ and the *real* time $Next()$ are calculated again at time $Next(t)$. The EGPS algorithm schedules processes in an increasing order of their virtual GPS completion times. The EGPS algorithm is formally defined as follows:

Algorithm EGPS_scheduler()

- When the system is initiated, $i = 0$, $t_0 = 0$, $V(t_0) = 0$, $B_1 = \phi$, and $Next(t_0) = \infty$.
- If no process request arrives before *real* time $Next(t_i)$, then
  - $i = i + 1$, and $t_i = Next(t_{i-1})$.
  - Let process $\tau_{k,j}$ be the process in B_i with the minimum GPS-completion time F_{min}; $B_{i+1} = B_i - \{\tau_{k,j}\}$.
  - $V(t_i) = V(t_{i-1}) + \frac{t_i - t_{i-1}}{\sum_{m \in B_i} \theta_m}$;
  - If $B_{i+1} == \phi$, then $Next(t_i) = \infty$; otherwise, let F_{min} be the minimum GPS-completion time $F_{n,l}$ of processes in B_{i+1} and $Next(t_i) = t_i + (F_{min} - V(t_i)) \sum_{m \in B_i} u_m$.
- If process request $\tau_{k,j}$ arrives at *real* time t and t is earlier than $Next(t_i)$, then[1]
  - $i = i + 1$, $t_i = t$, and $B_{i+1} = B_i \cup \{\tau_{k,j}\}$.
  - If $B_i == \phi$, then $V(t_i) = 0$, and $S_{k,j} = 0$. Otherwise, $V(t_i) = V(t_{i-1}) + \frac{t_i - t_{i-1}}{\sum_{m \in B_i} \theta_m}$ and $S_{k,j} = V(t_i)$.
  - $F_{k,j} = S_{k,j} + \frac{c_k}{\theta_k}$.
  - Let F_{min} be the minimum GPS-completion time $F_{n,l}$ of processes in B_{i+1};

$$Next(t_i) = t_i + (F_{min} - V(t_i)) \sum_{m \in B_i} u_m.$$

  - Schedule the process with the minimum GPS-completion time F_{min}.
- If process request $\tau_{k,j}$ completes its execution, then schedule the process with the minimum GPS-completion time F_{min}.

The complexity of the EGPS algorithm comes from the maintenance of virtual time $V()$, which has time complexity $O(1)$ per calculation and the dispatching of the process with the earliest GPS completion time. Note that EGPS scheduling is different from the earliest-deadline-first (EDF) scheduling since the GPS completion time of a process request is sensitive to the reservation ratio of each process. The reservation ratio of a process may be very different from the utilization factor of a process (please see Section 3.2).

1. An optimized algorithm and Example 3 will consider the simultaneous arrivals of multiple process requests.

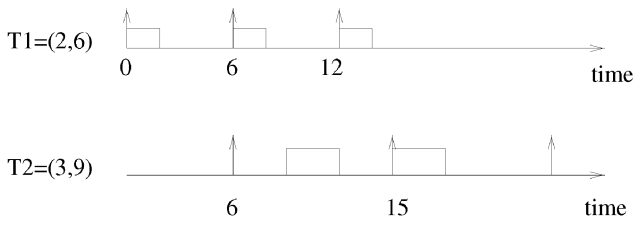


Fig. 7. An EGPS schedule of hard real-time periodic processes.

Example 3. *An EGPS schedule:* We now revisit Example 2 to illustrate the calculations of virtual GPS completion time. At time 0, the first request of τ_1 arrives. Since it is the only ready process, τ_1 executes and finishes its execution at time 2. ($V(0) = 0$, $S_{1,1} = 0$, $F_{1,1} = S_{1,1} + \frac{2}{1/3} = 6$, $B_2 = \{\tau_{1,1}\}$, and $Next(0) = 2$.) At time 6, the second request of τ_1 and the first request of τ_2 arrive. Since the CPU is idle before time 6, virtual time $V(6)$ is reset to zero and $S_{1,2} = V(6) = 0, F_{1,2} = S_{1,2} + \frac{2}{1/3} = 6, S_{2,1} = V(6) = 0, F_{2,1} = S_{2,1} + \frac{3}{1/3} = 9$, $B_3 = \{\tau_{1,2}, \tau_{2,1}\}$, and

$$Next(6) = 6 + (F_{min}(6) - V(6)) \left(\sum_{\{\tau_{1,2}, \tau_{2,1}\}} \theta_i \right)$$
$$= 6 + (6 - 0)\left(\frac{2}{3}\right) = 10.$$

Since the virtual GPS completion time of the first request of τ_2 is larger than the virtual GPS completion time of the second request of τ_1, τ_1 is scheduled. At time 8, τ_1 finishes its execution and τ_2 starts execution. At time $t = Next(6) = 10$, the algorithm sets virtual time $V(10) = 6$, $B_4 = \{\tau_{2,1}\}$, and *real* time $Next(10) = 11$. At time 11, τ_2 finishes its execution and virtual time $V(11)$ is again reset to zero ($B_5 = \phi$). The rest of the schedule can be found in Fig. 7.

ACKNOWLEDGMENTS

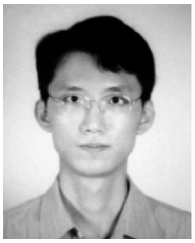
The authors would like to thank Professor R.H. Hwang at National Chung-Cheng University and Professor Al Mok at the University of Texas at Austin for their valuable comments on this work. Special thanks also go to the reviewers for their valuable comments. This research was supported in part by a research grant from the National Science Council under Grants NSC85-2213-E-194-008 and NSC87-2213-E-194-002. This paper is an extended version of a paper in the 10th Euromicro Workshop on Real-Time Systems. Wang-Ru Yang graduated from the Department of Computer Science and Information Engineering, National Chung Cheng University, under the supervision of Tei-Wei Kuo.

REFERENCES

[1] T.P. Baker, "A Stack-Based Resource Allocation Policy for Real Time Processes," *Proc. IEEE 11th Real-Time Systems Symp.*, Dec. 1990.

[2] R.L. Cruz, "SCED+: Efficient Management of Quality of Service Guarantees," *Proc. IEEE INFOCOM 98,* pp. 625-634, 1998.
[3] A. Demers, S. Keshav, and S. Shenkar, "Analysis and Simulation of a Fair Queueing Algorithm," *Proc. SIGCOMM,* pp. 1-12, 1989.
[4] Z. Deng, J.W.-S. Liu, and J. Sun, "A Scheme for Scheduling Hard Real-Time Applications in Open System Environment," *Proc. Ninth Euromicro Workshop Real-Time Systems,* pp. 191-199, June 1997.
[5] Z. Deng and J.W.-S. Liu, "Scheduling Real-Time Applications in an Open Environment," *Proc. IEEE Real-Time Systems Symp.,* Dec. 1997.
[6] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes," *Proc. IEEE Real-Time Systems Symp.,* Dec. 1994.
[7] C.-W. Hsueh, K.-J. Lin, and N. Fan, "Distributed Pinwheel Scheduling with End-to-End Timing Constraints," *Proc. IEEE Real-Time Systems Symp.,* Dec. 1995.
[8] C.-W. Hsueh and K.-J. Lin, "An Optimal Pinwheel Scheduler Using the Single-Number Reduction Technique," *Proc. IEEE Real-Time Systems Symp.,* pp. 196-205, Dec. 1996.
[9] C-W. Hsueh and K.-J. Lin, "On-Line Schedulers for Pinwheel Tasks Using the Time-Driven Approach," *Proc. 10th Euromicro Real-Time Systems,* pp. 180-187, June 1998.
[10] R.-H. Hwang, S.-L. Lee, T.-W. Kuo, T.-F. Chen, R.-F. Chang, and J.-J. Leou, "A Hierarchical Video-on-Demand System on ATM Networks," *Proc. 1996 Workshop Comm. Network,* pp. 311-332, 1996.
[11] T.-W. Kuo and A.K. Mok, "Load Adjustment in Adaptive Real-Time Systems," *Proc. IEEE 12th Real-Time Systems Symp.,* Dec. 1991.
[12] T.-W. Kuo, W.-R. Yang, and K.-J. Lin, "EGPS: A Class of Real-Time Scheduling Algorithms Based on Processor Sharing," *Proc. 10th Euromicro Workshop Real-Time Systems,* pp. 27-34, June 1998.
[13] C.D. Locke, D.R. Vogel, and T.J. Mesler, "Building a Predictable Avionics Platform in Ada: A Case Study," *Proc. IEEE 12th Real-Time Systems Symp.,* Dec. 1991.
[14] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM,* vol. 20, no. 1, pp. 46-61, Jan. 1973.
[15] M.-C. Liang, T.-W. Kuo, and L. Shu, "BAP: A Class of Abort-Oriented Protocols Based on the Notion of Compatibility," *Proc. Third Int'l Workshop Real-Time Computing Systems and Applications,* pp. 118-127, Oct. 1996.
[16] A.K. Mok, "Fundamental Design Problems for the Hard Real-Time Environment," PhD dissertation, Massachusetts Inst. of Technology, Cambridge, 1983.
[17] M.D. Natale and J.A. Stankovic, "Dynamic End-to-End Guarantees in Distributed Real-Time Systems," *Proc. IEEE Real-Time Systems Symp.,* Dec. 1994.
[18] A.K. Parekh and R.G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case," *IEEE/ACM Trans. Networking,* vol. 4, no. 1, pp. 344-357, Feb. 1993.
[19] A.K. Parekh and R.G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case," *IEEE/ACM Trans. Networking,* vol. 2, no. 2, pp. 137-150, Apr. 1994.
[20] H. Sariowan, R.L. Cruz, and G.C. Ployzos, "Scheduling for Quality of Service Guarantees via Service Curves," *Proc. Int'l Conf. Computer Comm. and Networks,* pp. 512-520, 1995.
[21] B. Sprunt, "Aperiodic Task Scheduling for Real-Time Systems," PhD dissertation, Dept. of Electrical and Computer Eng., Carnegie Mellon Univ., 1990.
[22] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. Computers,* vol. 39, no. 9, Sept. 1990.
[23] L. Sha, "Distributed Real-Time System Design Using Generalized Rate Monotonic Theory," lecture note, Software Eng. Inst., Carnegie Mellon Univ., 1992.
[24] M. Spuri, G. Buttazzo, and ? Sensini, "Scheduling Aperiodic Tasks in Dynamic Scheduling Environment," *Proc. IEEE Real-Time Systems Symp.,* 1995.
[25] I. Stoica, H. Abdel-Wahab, K. Jeffay, S.K. Baruah, J.E. Gehrke, and C.G. Plaxton, "A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems," *Proc. IEEE Real-Time Systems Symp.,* pp. 288-299, 1996.
[26] C.A. Waldspurger and W.E. Weihl, "Stride Scheduling Deterministic Proportional Share Resource Management," Technical Memorandum, MIT/LCS/TM-528, Laboratory for Computer Science, Massachusetts Inst. of Technolgy, July 1995.
[27] C.A. Waldspurger, "Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management," PhD thesis, Technical Report, MIT/LCS/TR-667, Laboratory for Computer Science, Massachusetts Inst. of Technology, Sept. 1995.

Tei-Wei Kuo received the BSE degree in computer science and information engineering from National Taiwan University in Taipei, Taiwan, in 1986. He received the MS and PhD degrees in computer sciences from the University of Texas at Austin in 1990 and 1994, respectively. He is currently an associate professor in the Department of Computer Science and Information Engineering of the National Taiwan University, Taiwan, Republic of China (ROC). He was an associate professor in the Department of Computer Science and Information Engineering of the National Chung Cheng University, Taiwan, ROC, from August 1994 to July 2000. His research interests include real-time databases, real-time process scheduling, real-time operating systems, and control systems. He was the program cochair of IEEE Seventh Real-Time Technology and Applications Symposium, 2001, and has consulted for government and industry on problems in various real-time systems design. Dr. Kuo is a member of the IEEE Computer Society.



Wang-Ru Yang received the BSE degree in computer science and information engineering from Tung Hai University in Taichung, Taiwan, in 1995. He received the MS degree in computer sciences from the National Chung Cheng University in 1997. He currently serves as a principal engineer at the First International Coporation, Taipei, Taiwan, Republic of China. His research interests include real-time systems, real-time operating systems, and embedded systems.

Kwei-Jay Lin received the BS degree in electrical engineering from the National Taiwan University in 1976 and the MS and PhD degrees in computer science from the University of Maryland, College Park, in 1980 and 1985, respectively. He is currently a professor in the Department of Electrical and Computer Engineering at the University of California, Irvine. From 1991 to 1993, he was an associate professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign, which he joined in 1985. His research interests include real-time systems, scheduling theory, e-commerce, operating systems, and embedded systems. He has published more than 100 papers in academic journals and conference proceedings. Dr. Lin was an associate editor of the *IEEE Transactions on Computers*. He was the symposium chair of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998, and Workshop cochair of the International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems, Santa Clara, California, 8-9 April 1999. He is the founding cochair of the IEEE Task Force on E-Commerce. He is a member of the IEEE Computer Society.