JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 23, 1349-1365 (2007)

Event-Driven Dynamic Workload Scaling for Uniprocessor Real-Time Embedded Systems*

LI-PIN CHANG AND YA-SHU CHEN+
Department of Computer Science
National Chiao Tung University
Hsinchu, 300 Taiwan
+*Department of Electrical Engineering*
National Taiwan University of Science and Technology
Taipei, 106 Taiwan

Many embedded systems are designed to take timely reactions to the occurrences of particular scenarios. Such systems could sometimes experience transient overloads because of workload bursts or hardware malfunctions. Thus a mechanism to focus limited resources on the processing of urgent events is a key to retain system validity under stressing workloads. In this paper, we propose a new approach for workload scaling in uniprocessor real-time embedded systems. The idea is to view the system as a black box, and workload scaling for overload management can be done via very intuitive primitives, *i.e.*, how hardware events are selectively fed into the system. Such a new approach removes the need for the adjustments of task periods and task phasing, which is important for many workload-scaling techniques. The proposed approach is implemented in a real-time surveillance system. Experimental results show that the system still delivers good accuracy and high responsiveness for visual-object tracking under the presence of overloads.

Keywords: embedded systems, real-time systems, adaptive applications, overload management, real-time surveillance

1. INTRODUCTION

Many embedded systems are designed to take timely reactions to the occurrences of particular scenarios. For example, an automobile controlling system should simultaneously control cruising, traction, the brake system, and the engine. A real-time surveillance system in a shopping mall could track a number of moving people and evaluate pre-defined rules to detect thieving. These systems might sometimes experience transient workload bursts due to hardware malfunctions (*e.g.*, one processor fails in a dual-processor system) or workload bursts (*e.g.*, a number of people suddenly rush into monitored areas). To prevent the systems from being overloaded, the system could smartly allocate available computing power among tasks so as to pay more attention to those urgent events and, at the same time, to slow down the processing of inactive events.

Consider an embedded system which deals with periodically recurring external (hardware) events. Intuitively, proportional period adjustment for periodic tasks could be

Received November 15, 2006; accepted February 15, 2007.
Communicated by Sung Shin and Tei-Wei Kuo.
* This work was partly supported by the National Science Council of Taiwan, R.O.C., under grant No. 95-2221-E-009-063. This paper is an extended version of the paper in [21].

useful to slow down or to speed up the processing of events. However, this simple approach has some major drawbacks: First, even though the period of a task could be arbitrarily adjusted, the recurring rate of a hardware event are usually not. Consider a thermal sensor connected to a system via USB and its reading is sampled every 1 ms. Because the minimal time frame for USB bus traffic is 1 ms, change the sampling rate to 1.5 ms is unlikely applicable. Second, mismatches of hardware-event recurring periods and task periods could sometimes result in unpredictable arrivals of events due to buffer overflows. Not being aware of which event is sampled and when it is sampled could damage the usefulness of timing-sensitive algorithms such as Kalman filters. Third, responsiveness for detecting pre-defined scenarios would largly be retarded due to long propagation delays introduced by improper task periods and task phasing. All these drawbacks of proportional period adjustment will be shown in our experiments.

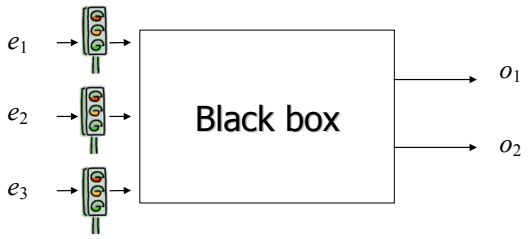


Fig. 1. A "black box" that receives periodic hardware events and then delivers outputs.

In this paper, we propose a new approach for workload scaling in uniprocessor real-time embedded systems. Let a system be viewed as a "black box" (as shown in Fig. 1), and the system requires no intervention when workload scaling is conducted. When transient overloads are experienced, the feeding of events (*i.e.*, e_1, e_2, and e_3 in Fig. 1) could be controlled by some mechanism (*i.e.*, the traffic lights in Fig. 1) to protect the system against timing violations. Our objectives are summarized as follows: (1) System workloads should be adjusted by means of very simple and intuitive primitives. (2) Tasks in the system should automatically react to new settings for workload scaling without intervention from the scheduler. (3) Tasks in the system should have deterministic timing behaviors. (4) An on-line admission control policy is needed to examine whether any change to system workloads can be made.

The rest of this paper is organized as follows: Section 2 provides past work related to the issues we considered. The system model and terminologies are introduced in section 3. System timing analysis and an efficient on-line admission control algorithm are presented in section 4. Performance evaluation and comparison are provided in section 5, and this paper is concluded in section 6.

2. RELATED WORK

As modern embedded software is component-based, event-driven paradigms are one of many software architectures to formulate precedence constraints among components. Tindell [13] considered an iterative approach to calculate response time for distributed

event-driven tasks. The proposed analysis was extended to handle complex precedence constraints in hard-real-time distributed systems [14]. The model is applicable to many types of applications such as parallel and distributed systems [15] and embedded systems [16]. Precedence constraints among tasks could also be formulated as producer-consumer relation in uniprocessor systems [1]. Recently, operating systems based on event-driven models are also considered in the implementation of sensor nodes [8].

Dynamic workloads could transiently overload a real-time system. Prior work proposed many excellent workload-scaling approaches for overload management: In particular, Koren and Shasha [4] proposed to schedule tasks while task jobs could be skipped in a controllable way. Hamdaoui and Ramanathan [5, 6] proposed the (m, k)-firm task model to complete at least m task jobs out of any k consecutive ones. Window-constrained scheduling [2, 3] is to fulfill m task jobs out of every window of k successive task jobs, while windows do not overlap one another. Imprecise computation [9] defines that any job is of a mandatory portion and an optional portion. Mandatory portions must be successfully completed before their deadlines, while the more CPU execution are contributed to optional portions the more reward is attained. Besides to skip jobs, task-period adjustment is also a commonly adopted technique [17, 18].

Besides considering new task models, an alternative approach for overload handling is to consider how "fresh" a piece of data is. Data freshness can be defined over time domain or value domain: A piece of fresh data stands for either one that has just been recently updated, or one which's value has not been changed a lot. When systems are overloaded, either the sampling rates of those inactive events could be decreased [20], or only updates (*i.e.*, jobs) to those "stale" data should be scheduled for execution [19]. However, there is no direct mapping from the definition of data freshness into how workload is scaled down. The algorithm may need to try a number of different settings before the system gets rid of being overloaded.

Different from prior work, this paper aims at a new approach for dynamic workload scaling. It employs no adjustment and alignment of task periods. With a proposed on-line admission control policy, workload adjustment can be done on the fly.

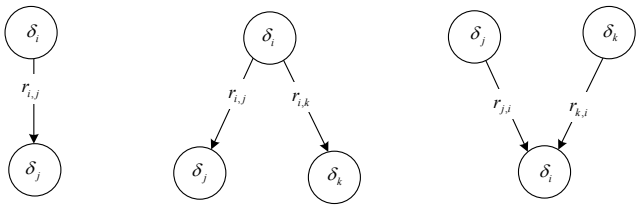
3. THE SYSTEM MODEL

Let tasks be preemptively scheduled with fixed priorities over a single processor. An event-driven task δ_i is a template of jobs, and job $\delta_{i,n}$ stands for the nth task job. Let each job of task δ_i require c_i units of processor time, and a static priority Ω_i be assigned to all jobs of task δ_i. A *result* is a piece of data that a task job prepares for another task, and an *event* is sent to the destination task when a predefined number of results are ready. A task job is released and becomes ready when the task receives an event. A task could either (1) receive one (and only one) hardware event, or (2) receive one or many events from other tasks. An event sent from task δ_i to task δ_j is denoted by $e_{i,j}$, and a hardware event that task τ_i receives is denoted by $e_{0,i}$. No tasks could receive both hardware events and events from other tasks. Let all hardware events be periodic, and p_i stands for the recurring period of hardware event $e_{0,i}$.

If δ_i sends events to task δ_j, then δ_i and δ_j are referred to as the *producer* and the *consumer* of each other. The relation is denoted by $\delta_i \prec \delta_j$. An event $e_{i,j}$ is composed by

$r_{i,j}$ cumulated results ($r_{i,j} \in Z^+$). Upon every $r_{i,j}$th result is ready, δ_i composes an event and sent to consumer δ_j. At the same time, a ready job of consumer δ_j is released. For any $n \in Z^+$, job $\delta_{i,n \cdot r_{i,j}}$ is referred to as an *effective job* of $\delta_i \prec \delta_j$ since it triggers the execution of job $\delta_{j,n}$. Note that each arrival of hardware event $e_{0,j}$ triggers the execution of a job of task δ_j because $r_{0,j} = 1$.

Task jobs are assigned to no explicit deadlines. Instead, a task job must complete before the next task job arrives. A task is referred to as a *pump* if it receives a hardware event. A taskchain $\Delta = \{\delta_1, \delta_2, \ldots, \delta_n\}$ is a collection of tasks, in which δ_1 is a pump and $\delta_i \prec \delta_{i+1}$ for any $\{\delta_i, \delta_{i+1}\} \subseteq \Delta$. As shown in Fig. 2, three basic structures are considered: pipes, forks, and syncs. Note that, in a sync, a consumer job becomes ready as long as the consumer receives at least one event from each producer of the sync.



(a) pipe. (b) fork. (c) sync.

Fig. 2. Three basic structures in our event-driven system model.

For comparison and later use in this paper, a time-driven periodic system is defined as follows: A periodic task $\tau_i = (o_i, c_i, p_i)$ is a template of jobs, and job $\tau_{i,n}$ denotes the nth task job. The first job of τ_i arrives at time o_i and then a job arrives every p_i units of time. In other words, job $\tau_{i,m}$ is released at time $o_i + (m - 1)p_i$ for $m \in Z^+$. Computation requirement of each job of task τ_i is no more than c_i units of time, and the relative deadline of a task job is the task period. A fixed priority Ω_i is assigned to all jobs of task τ_i for preemptive scheduling.

4. WORKLOAD SCALING FOR EVENT-DRIVEN SYSTEMS

4.1 Timing Analysis for Pipes and Forks

This section aims at timing analysis of tasks in pipes and forks with or without event skips.

4.1.1 Scheduling without event skips

In pipes and forks, every consumer has only one producer. First we shall determine the optimal priority assignment for tasks in a taskchain:

Lemma 1 Given a taskchain $\Delta = \{\delta_1, \delta_2, \ldots, \delta_n\}$ and an arbitrary priority assignment $A = \{\Omega_1, \Omega_2, \ldots, \Omega_n\}$. If Δ is schedulable with A then Δ is also schedulable with a priority assignment $A' = \{\Omega_1', \Omega_2', \ldots, \Omega_n'\}$, in which Ω'_i is higher than Ω'_{i+1} for any i.

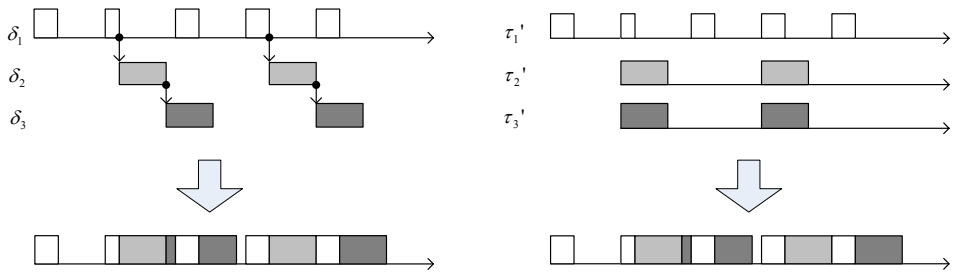
Proof Sketch: It can be proved by repeatedly swapping priorities Ω_i and Ω_{i+1} until Ω_i is higher than Ω_{i+1} for any i. After each priority swap taskchain Δ is still schedulable. ❑

Under the optimal priority assignment, the following theorem shows that event-driven tasks in a taskchain can be equivalently modeled as a collection of independent periodic tasks in terms of timing behaviors:

Theorem 1 Given a taskchain $\Delta = \{\delta_1, \delta_2, \ldots, \delta_n\}$. Let T be a set of independent periodic tasks $\{\tau_1 = (0, c_1, p_1), \tau_2 = (p_1(r_{1,2} - 1), c_2, p_1 \cdot r_{1,2}), \ldots, \tau_n = (p_1 \cdot ((\prod_{x=1}^{n} r_{x-1,x}) - 1), c_n, p_1 \cdot \prod_{x=1}^{n} r_{x-1,x})\}$. Under a priority assignment $A = \{\Omega_1, \Omega_2, \ldots, \Omega_n\}$ in which Ω_i is higher than Ω_{i+1} for any i, Δ and T produce the same schedule.

Proof: We shall show the correctness of this theorem by induction on n the number of tasks:

I.B.: The base case is trivial.
I.H.: Suppose that the theorem is true when for n tasks.
I.S.: Consider the case $\{\delta_1, \delta_2, \ldots, \delta_n\} \cup \{\delta_{n+1}\}$. By I.H., δ_n is equivalent to a periodic task $\tau_n = (p_1 \cdot ((\prod_{x=1}^{n} r_{x-1,x}) - 1), c_n, p_1 \cdot \prod_{x=1}^{n} r_{x-1,x})$ in terms of timing behaviors. Because Ω_n is higher than Ω_{n+1}, job $\delta_{n+1,i}$ could be treated as one which arrives simultaneously with job $\tau_{n,i \cdot r_{n,n+1}}$ for any i. The inter-arrival time of jobs of task δ_{n+1} could then be $r_{n,n+1} \cdot p_n$. Because any job must complete before the next job of the same task arrives, δ_{n+1} could be treated as a periodic task $(r_{n+1,n} \cdot p_1 \cdot ((\prod_{x=1}^{n} r_{x-1,x}) - 1), c_{n+1}, r_{n+1,n} \cdot p_1 \cdot \prod_{x=1}^{n} r_{x-1,x})$ in terms of timing behaviors. ❑



(a) Event-driven tasks. (b) Periodic tasks.

Fig. 3. Modeling a collection of event-drive tasks as a collection of purely periodic tasks.

Fig. 3 shows a schedule fragment resulted by three event-driven task $\Delta = \{\delta_1, \delta_2, \delta_3\}$, where $r_{1,2} = 2$ and $r_{2,3} = 1$. Suppose $c_1 = 1$, $c_2 = 2$, and $c_3 = 2$. The pump δ_1 is driven by a hardware event e_1 every 3 units of time. On the other hand, a collection of independent periodic tasks $T = \{\tau_1 = (0, 1, 3), \tau_2 = (3, 2, 6), \tau_3 = (3, 2, 6)\}$ is considered. For all i, Ω_i is assigned both to δ_i and to τ_i. Ω_1 is the highest priority and Ω_3 is the lowest priority. As shown in Fig. 3, Δ and T result in the same schedule, no matter how long the actual execution times of jobs are.

Because a fork could be considered as a structure that "joins" one or more taskchains, Lemma 1 and Theorem 1 is still applicable to taskchains in a fork. Tasks in different taskchains could be optimally assigned to priorities which are inversely proportional to the corresponding "periods" from Theorem 1.

4.1.2 Scheduling with event skips

With event skips, this section considers how tasks in pipes and forks behave. A firm-real-time scheduling technique is first introduced, and timing analysis is then presented.

1. (m, k)-Firm Scheduling

We choose to adopt (m, k)-firm scheduling [5, 6] as the policy for event selection so as to achieve workload scaling. A (m, k)-firm task is a generalization of a periodic task. A task being subject to (m, k)-firm constraint must successfully complete m jobs before their relative deadlines (*i.e.*, the period) out of any k consecutive jobs. A job is classified as being *mandatory* if it has to be completed in time, or it is classified as being *optional*.

Let all jobs of a task be labeled with their ordinal numbers and the smallest number be *zero*. The following equation classifies which job should be classified as being mandatory:

$$\left\lfloor \left\lceil \frac{i \cdot m}{k} \right\rceil \frac{k}{m} \right\rfloor, \; i \in Z. \tag{1}$$

For example, a task being subject to (3, 5)-firm constraint must successfully complete the jobs labeled with 0, 1, 3, and so on. An alternative form representing (m, k)-firm constraint is an array of k binary elements: A job labeled with i is a mandatory job if the i%kth element[1] is 1, otherwise the job is an optional job. For example, (3, 5)-firm constraint can be represented as 11010. The array is referred to as the *activation pattern* of (3, 5)-firm constraint.

Derived from Eq. (1), two functions can be derived: (1) $f(n)$ that calculates the smallest ordinal number of a job until which there are n jobs had been classified as being mandatory[2], and (2) $h(n)$ that calculates the number of jobs classified as being mandatory from the job labeled with zero to the job labeled with n:

$$f(n) = \left\lfloor \frac{(n-1)k}{m} \right\rfloor, \text{ and } h(n) = \left\lceil \frac{(n+1) \cdot m}{k} \right\rceil. \tag{2}$$

2. Scheduling Event-Driven Tasks with (m, k)-firm Constraint

When (m, k)-firm constraints are applied to pumps (*i.e.*, tasks that receive hardware events) to skip some certain events, the question that what are the timing behaviors of consumer tasks behave in taskchains needs to be answered.

Consider the example shown in Fig. 4, where producer δ_i is subject to (4, 7)-firm

[1] % is the modulus operator and the array elements are labeled from zero.
[2] $f(n)$ is an abbreviation of $f_m^k(n)$.

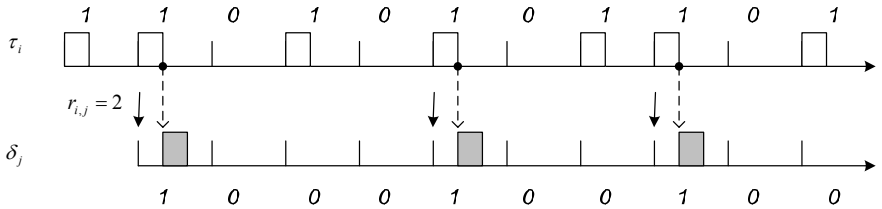


Fig. 4. A producer-consumer pair of task δ_i and task δ_j, where δ_i is a pump being subject to (4, 7)-firm constraint.

constraint. Based on the same arguments in the proving of Theorem 1, consumer δ_j could be treated as a periodic task which inherits task period p_i from producer δ_i and is subject to an activation pattern {1000100}. It is conjectured that the pattern {1000100} is the result of rotating the activation pattern of (4/$r_{i,j}$, 7)-firm (*i.e.*, (2, 7)-firm) constraint by three elements leftward (*i.e.*, {1001000} $\rightarrow$ {1000100}).

To verify our conjecture, the notion of pattern rotations is introduced: Let the activation pattern of (m, k, ε)-firm constraint be the result of rotating the pattern of (m, k)-firm constraint by ε elements leftward. Let us be focused on $\left(m, k, \left\lfloor \frac{\rho \cdot k}{m} \right\rfloor\right)$-firm constraint, where $\rho \in Z$. Function $f()$ in Eq. (2) could be generalized as follows:

$$f'(n) = \left\lfloor \frac{((n-1)+\rho)k}{m} \right\rfloor - \left\lfloor \frac{\rho \cdot k}{m} \right\rfloor. \tag{3}$$

Consider $\delta_i \prec \delta_j$, where δ_i is subject to (m, k)-firm constraint. Based on the same arguments in the proving of Theorem 1 and $f()$ in Eq. (2), the arrival times of δ_j's jobs are:

$$\left\lfloor \frac{(r_{i,j}n-1)k}{m} \right\rfloor - \left\lfloor \frac{(r_{i,j}-1)k}{m} \right\rfloor.$$

Suppose that $m = r_{i,j} \cdot m'$ for some $m' \in Z$. By some modulus algebra the above equation could be rewritten as

$$\left\lfloor \frac{((n-1)+(\pi+1))k}{m'} \right\rfloor - \left\lfloor \frac{(\pi+1)k}{m'} \right\rfloor \tag{4}$$

for some $\pi \in$ Z. Comparing Eq. (3) with Eq. (4), obviously the consumer is subject to $\left(m', k, \left\lfloor \frac{(\pi+1)k}{m'} \right\rfloor\right)$-firm constraint. Let a firm-real-time task τ is denoted by ((o, c, p), (m, k, ε)), where o, c, and p denote the initial arrival time, the computation demand, and the period of task τ respectively. τ is subject to (m, k, ε)-firm constraint. Now Theorem 1 can be generalized as follows:

Corollary 1 Given a taskchain $\Delta = \{\delta_1, \delta_2, \ldots, \delta_n\}$, where pump δ_1 is subjected to (m_1, k_1, ε_1)-firm constraint and $\prod_{i=1}^{x} r_{i-1,i}$ divides m_1 for every $x \leqq n$. Let T be a set of independent firm-real-time tasks

$$\{((0, c_1, p_1), (m_1, k_1, \varepsilon_1)), ((p_1 f_{m_1}^{k_1}(r_{1,2}), c_2, p_1), (\frac{m_1}{r_{1,2}}, k_1, \varepsilon_2)), \ldots,$$
$$\tau_n = ((p_1 f_{m_1}^{k_1}(\prod_{i=1}^{n} r_{i-1,i}), c_n, p_1), (\frac{m_1}{\prod_{i=1}^{n} r_{i-1,i}}, k_1, \varepsilon_n))\}.$$

Systems Δ and T produce same schedule under a priority assignment in which Ω_i is higher than Ω_{i+1} for any i.

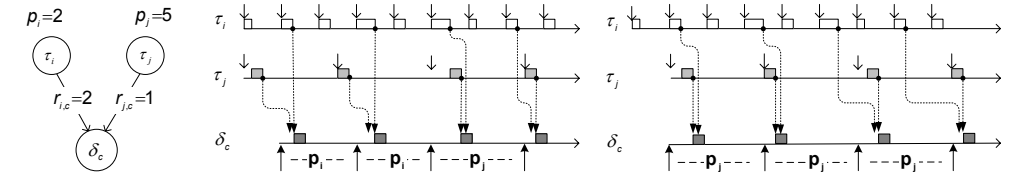
Proof: It directly follows Theorem 1 and the above discussion. ❑

4.2 Timing Analysis for Syncs

Now let our attention be focused on timing analysis for tasks involved in sync structures. As we did in the previous sections, the discussions first begin with scheduling event-driven tasks in syncs, where no event skips are allowed.

Let the consumer δ_c of a sync be an event-driven task, and the producers of the sync be independent periodic tasks. Let $\vdash\delta_c = \{\tau_i \mid \tau_i \prec \delta_c\}$ be a collection of tasks including all producers of consumer δ_c. For any producer $\tau \in \vdash\delta_c$, the inter-arrival time of the effective jobs of $\tau \prec \delta_c$ is referred to as the *effective cycle* of producer τ. Consumer job $\delta_{c,x}$ becomes ready as soon as all effective jobs in $\{\tau_{i,x \cdot r_{i,c}} \mid \forall i : \tau_i \in \vdash\delta_c\}$ send an event to consumer δ_c. Priorities assigned to producers are inversely proportional to their recurring periods, and the consumer is assigned to the lowest priority (as shown in Lemma 1).

Fig. 5 depicts a scenario in which two periodic producers τ_i and τ_j conjunctively trigger the execution of the event-driven consumer δ_c. As Fig. 5 (b) shows, even though the producer τ_i and τ_j periodically issue events to consumer δ_c, jobs of δ_c do not arrive in a regular period. On the other hand, as Fig. 5 (c) shows, if the first effective job of τ_j (*i.e.*, $\tau_{j,1}$) arrives no earlier than the effective first job of τ_i (*i.e.*, $\tau_{i,2}$) does, then all jobs δ_c will be released every $r_{j,c} \cdot p_j$ units of time as if δ_c were a periodic task. The adjustments of task arrival times could be formulated as the following corollary:



(a) A sync structure. (b) Timing δ_c when τ_i and τ_j arrive simultaneously. (c) Timing δ_c when τ_j arrives 2 units of time later than τ_i does.

Fig. 5. Arrivals of consumer jobs in a sync.

Corollary 2 Suppose that task δ_c is conjunctively driven by events sent from a collection of independent periodic task $\Gamma = \vdash\delta_c$. Let be

$$\exists \tau_i \in \Gamma, \forall \tau_j \in \Gamma - \{\tau_i\} :$$
$$r_{i,c} \cdot p_i \geq r_{j,c} \cdot p_j, o_i + (r_{i,c} - 1)p_i \geq o_j + (r_{j,c} - 1)p_j.$$

Consider a periodic task $\tau_c = (o_i + (r_{i,c} - 1)p_i, c_c, r_{i,c}p_i)$. Systems $\Gamma \cup \{\tau_c\}$ and $\Gamma \cup \{\delta_c\}$ produce the same schedule.

Proof: For any job $\delta_{c,x}$ of task δ_c, the job becomes ready as soon as every task $\tau_j \in \Gamma$ sends x events to task δ_c. Because the arrival of job $\tau_{i,x \cdot r_{i,c}}$ is always the latest among those of jobs $\tau_{j,x \cdot r_{i,c}}$ and task δ_c is assigned to the lowest priority, the processor would not be available to job $\delta_{c,x}$ until job $\tau_{i,x \cdot r_{i,c}}$ completes. Job $\delta_{c,x}$ could then be treated as one that arrives simultaneously with job $\tau_{i,x \cdot r_{i,c}}$, and the inter-arrival time of jobs of δ_c is $p_i \cdot r_{i,c}$. Obviously, $\Gamma \cup \{\tau_c\}$ and $\Gamma \cup \{\delta_c\}$ produce the same schedule. ❑

A simple algorithm based on topological sort could be adopted to adjust pumps' initial arrival times to comply with the condition stated in Corollary 2. Corollary 2 can be revised accordingly to analyze tasks in sync with event skips.

4.3 On-Line Admission Control

As previous sections show, all event-driven tasks could be exactly modeled as independent periodic tasks. This section is then focused on providing an efficient schedulability test for a collection of periodic tasks being subject to (m, k, ε)-firm tasks.

Let a periodic task being subject to (m, k, ε)-firm constraint be referred to as a (m, k, ε)-*firm task* for conciseness. Consider that a collection of event-driven tasks $\{\delta_1, \delta_2, \ldots, \delta_n\}$ had been successfully modeled as a collection T_n of (m, k, ε)-firm tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$. The critical instance of T occurs when all (m, k, ε)-firm tasks are in phase and $\varepsilon_i = 0$ for all $\tau \in T_n$. Because each task job must complete before the next task job arrives (please see section 3), the schedulability test based on worst-case response time analysis is as follows:

Lemma 2 [5] A collection of (m, k, ε)-firm tasks $T_n = \{\tau_1, \tau_2, \ldots, \tau_n\}$ with arbitrary arrival times is schedulable if and only if

$$\forall i \in \{1, \ldots, n\}, \exists t_i : 0 \le t_i \le \left\lfloor \frac{k_i}{m_i} \right\rfloor p_i, t_i \ge \sum_{j=1}^{i} c_j \left\lceil \frac{\lceil t_i / p_j \rceil m_j}{k_j} \right\rceil.$$

In the above test, the term $\left\lfloor \frac{k_i}{m_i} \right\rfloor p_i$ stands for the arrival time of the second job of task τ_i (please refer to Eq. (2)), and the term $c_j \left\lceil \frac{\lceil t_i / p_j \rceil m_j}{k_j} \right\rceil$ calculates the cumulative computation demand of task τ_j in the interval $[0, t_i)$ (please refer to Eq. (2)). The computational cost depends on task periods and the constraints that tasks are subject to. Because workload scaling should be performed on the fly to handle transient bursts, an efficient on-line schedulability test is needed.

In the rest of this section, a sufficient condition for the schedulability of a collection of (m, k, ε)-firm tasks is introduced. Our basic idea is to develop a systematic method that transforms a collection of (m, k, ε)-firm tasks into a collection of purely periodic tasks. *The transformation guarantees that if the given collection of* (m, k, ε)*-firm tasks is unschedulable then after the transformation the resultant collection of purely periodic tasks is unschedulable*. So, conversely, if the resulted collection of purely periodic tasks

is schedulable, then the given collection of (m, k, ε)-firm tasks is schedulable. The technical question is how to prevent the transformation from being overly pessimistic.

Let $U(n)$ be the utilization bound of the Liu-and-Layland schedulability test [7], we have the following test:

Theorem 2 A collection of (m, k, ε)-firm tasks $T_n = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is schedulable if

$$\forall i \in \{1, \ldots, n\} : \sum_{j=1}^{i-1} \frac{c_j^*}{p_j} + \frac{c_i + \Delta_i}{p_i^*} \le U(n), \text{ where } c_i^* = c_i \frac{m_i}{k_i}, \; p_i^* = p_i \left\lfloor \frac{k_i}{m_i} \right\rfloor,$$

$$\text{and } \Delta_i = c_k \sum_{k=1}^{i-1} \left(\left\lceil \frac{\lceil p_i^* / p_k \rceil m_k}{k_k} \right\rceil - \frac{\lceil p_i^* / p_k \rceil m_k}{k_k} \right).$$

Proof: Suppose that $T_{n\text{-}1} = \{\tau_1, \tau_2, \ldots, \tau_{n\text{-}1}\}$ is schedulable but $T_n = T_{n\text{-}1} \cup \{\tau_n\}$ is not. Now we are going to transform T_n into a collection of purely periodic tasks which is guaranteed to be unschedulable.

Let $T_{n\text{-}1}$ first be transformed into T'_{n-1}, where any task $\tau_i = ((0, c_i, p_i), (m_i, k_i, 0))$ in $T_{n\text{-}1}$ is converted to a purely periodic task $\tau'_i = ((0, c_i \frac{m_i}{k_i}, p_i), (1, 1, 0))$ in T'_{n-1}. Based on Lemma 2, at any time $t \geqq 0$, the cumulative processor time demand of task τ'_i would be $c_i \frac{\lceil t/p_i \rceil m_i}{k_i}$, which is no larger than that of task τ_i (*i.e.*, $c_i \left\lceil \frac{\lceil t/p_i \rceil m_i}{k_i} \right\rceil$). Therefore if $T_{n\text{-}1}$ is schedulable then T'_{n-1} is schedulable.

Because there is some loss of cumulative processor time demand during the transforming from $T_{n\text{-}1}$ into T'_{n-1}, so even though T_n is unschedulable it is not guaranteed that $T'_{n-1} \cup \{\tau_n\}$ is unschedulable. For this purpose, we shall transform τ_n into $\tau''_n = ((0, c'', p''), (1, 1, 0))$, where $p'' = p_n \left\lfloor \frac{k_n}{m_n} \right\rfloor$ and $c'' = c_n + \Delta_n$. Δ_n denotes the loss of processor time demand in the time interval $[0, p'')$ when transforming $T_{n\text{-}1}$ into T'_{n-1}. If Δ_n is "reclaimed" and added to c_n'' then it is straightforward to show that $T'_{n-1} \cup \{\tau''\}$ is surely unschedulable. By using Eq. (2) it can be derived that $\Delta_n = c_k \sum_{k=1}^{n-1} \left(\left\lceil \frac{\lceil p_i'' / p_k \rceil m_k}{k_k} \right\rceil - \frac{\lceil p_i'' / p_k \rceil m_k}{k_k} \right)$. It is then concluded that if T_n is unschedulable then $T'_{n-1} \cup \{\tau''_n\}$ is unschedulable. Conversely, if $T'_{n-1} \cup \{\tau''_n\}$ can be admitted by any schedulability test for purely period tasks (*e.g.*, the Liu-and-Layland test) then T_n is schedulable. ❑

The time complexity of the proposed test is $O(n^2)$, which is efficient enough for on-line implementations.

5. EXPERIMENTAL RESULTS

5.1 Overview

The usefulness of the proposed workload-scaling approach is demonstrated by conducting a series of experiments on a real-time surveillance system. We built a surveillance

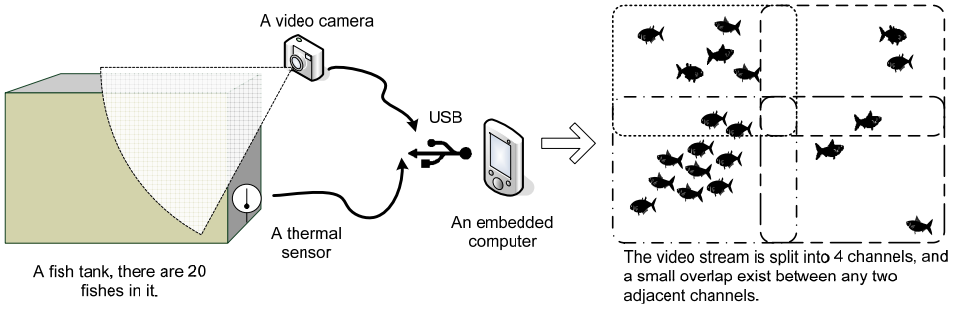


Fig. 6. A real-time surveillance system monitoring fishes in a fish tank.

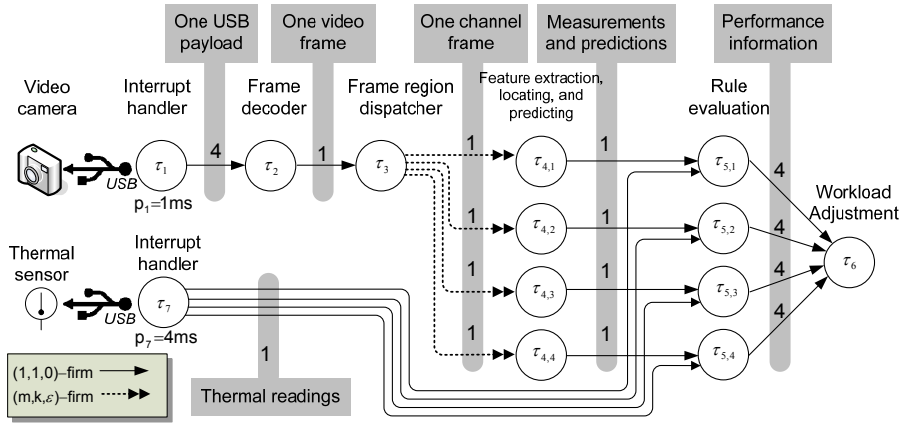


Fig. 7. Interactions of tasks in the surveillance system for demonstration.

system to monitor fishes in a fish tank. The application was a part of a biological research project, in which dynamics of a group of fishes were analyzed based on real-time data collection. The system was built over an embedded computing platform, on which an ARM processor and 16 MB of RAM were adopted. The processor was normally rated at 200 MHZ. As shown in Fig. 6, a video camera and a thermal sensor were connected to the system via USB. The resolution and frame-rate of the video stream delivered by the camera were 352 × 288 pixels and 25 fps, respectively. We chose to modify and port the OpenCV package [10] onto the target platform. The tracking of visual objects in video streams was based on the Lucas-Kanade optical-flow algorithm [11], and Kalman filters [12] were used to estimate the trajectories of moving objects. An instance of the Kalman filter was created for each visual object and the prediction provided by the filter was used to refine the result of the optical-flow algorithm so as to reduce trajectory losses.

There were twenty guppies in the fish tank. Normally the processor has been already overloaded if all twenty fishes need to be tracked. Thus we proposed to split the video stream into four *channels* by partitioning every frame into four equal-sized regions, as shown in the right-hand side of Fig. 6. The intention is to flexibly allocate processor time among the channels so as to pay more attention to those active objects and, at the same time, to slow down the monitoring of inactive objects. As shown in Fig. 7, the de-

livery of channel frames from task τ_3 to instances of task τ_4 was subject to different (m, k, ε)-firm constraints for workload scaling. To conduct experiments, a pre-recorded 20-minute video was projected onto a small white screen and the vision was then captured by the USB camera.

Two performance metrics were adopted. The first is the average of the Root-Mean-Square errors between actual positions and predicted positions of visual objects. The lower the errors, the more accurate the tracking is. The other metric is responsiveness, which is the time between the ground-truth time of an occurrence of a particular scenario and the time the scenario is detected by the system. The shorter the time, the higher the responsiveness is.

Let the proposed firm-real-time-based event-driven approach be referred to as *FE* approach. An alternative approach was developed for performance comparison: Let all tasks in the system be independent and periodic. For workload adjustment, the periods of channel-frame delivery from task τ_3 to instances of task τ_4 were proportionally scaled. Let the approach be referred to as *PP* approach. More details on PP approach would be included in the later sections.

5.2 Visual-Object Tracking Accuracy

Let first only one of the four channels is enabled, and all the other three channels are ignored by the processor. In other words, the flow of $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \rightarrow \tau_{4,1} \rightarrow \tau_{5,1} \rightarrow \tau_6$ is enabled. Processor time available to the system was controlled by a parameter *workload-scaling factor* for both FE approach and PP approach. A workload-scaling factor reflected a percentage of available processor utilization. For FE approach, with respect to workload-scaling factor x/y, the delivery of channel frames from task τ_3 to task $\tau_{4,1}$ was subject to (x, y, 0)-firm constraint. On the other hand, for PP approach, periods of task $\tau_{4,1}$ and task $\tau_{5,1}$ were proportionally enlarged by multiplying their periods by y/x. In the experiments, seven workload-scaling factors were considered: 7/7 = 100%, 6/7 = 86%, 5/7 = 71%, 4/7 = 57%, 3/7 = 43%, 2/7 = 29%, and 1/7 = 14%. For example, with respect to workload-scaling factor 57%, under FE approach, frame delivery from task τ_3 to task $\tau_{4,1}$ would be subject to (4, 7, 0)-firm constraint. Under PP approach, the periods of task $\tau_{4,1}$ and task $\tau_{5,1}$ would both be enlarged from 4 ms to 4 × 7/4 = 7ms. Because 7 is a prime number, under anyone of the seven work-scaling factors (except 7/7 and 1/7), the inter-arrival times of jobs of task $\tau_{4,1}$ under PP approach and under FE approach would not be the same (and so were those of jobs of task $\tau_{5,1}$).

The experimental results for the only one enabled channel were shown in Fig. 8, in which the X-axis denoted the workload-scaling factors and the Y-axis denoted the average RMS errors. The figure showed that, even when the workload-scaling factor was decreased to 57%, FE approach could still kept the average RMS errors as small as if there were no frames skipped (*i.e.*, 100%). The average RMS error significantly grew under a very small workload-scaling factor because the tracking of visual objects was ineffective when a lot of frames were skipped. A similar phenomenon was observed for PP approach, however it imposed a relatively larger average RMS errors than the FE approach did when the workload-scaling factor was between 86% and 29%. The rationale would be explained later. Note that PP approach and FE approach resulted in the same average RMS errors when the workload-scaling factor were 7/7 = 100% and 1/7 =

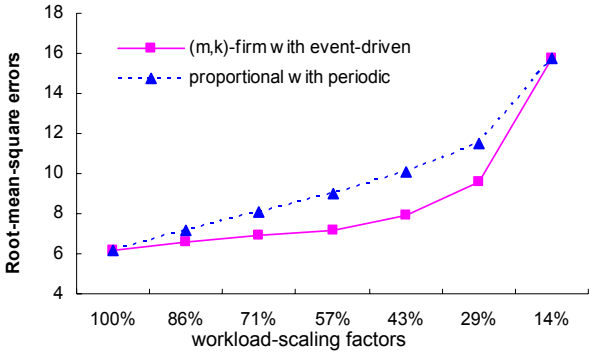


Fig. 8. RMS errors resulted by different workload-scaling factors under the proposed event-driven paradigm and the purely periodic system.

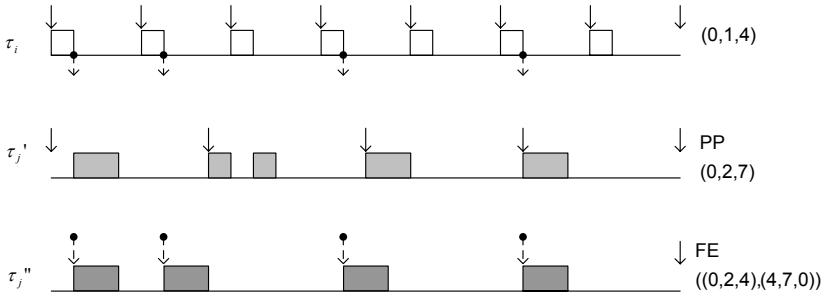


Fig. 9. A scenario for workload scaling, where task $\tau_i = (0, 1, 4)$ and $\tau_j = (0, 2, 4)$ are a pump and the consumer (not shown), respectively. Let the workload-scaling factor be 4/7. Under PP approach and under FE approach we have $\tau_j' = (0, 2, 7)$ and $\tau_j'' = ((0, 2, 4), (4, 7, 0))$, respectively.

14%. That was because, with respect to 7/7 or 1/7, the inter-arrival times of channel frames under the two approaches were the same.

One advantage of FE approach over PP approach is that the "absolute" inter-arrival time of any two successive events is exactly known. Consider the scenario shown in Fig. 9: Producer $\tau_i = (0, 1, 4)$ handles a periodic hardware event and send events to consumer $\tau_j = (0, 2, 4)$ with $r_{i,j} = 1$. Now let the workload-scaling factor for the delivery of events be 4/7 = 57%. Under PP approach consumer τ_j became $\tau_j' = (0, 2, 7)$ and under FE approach it became $\tau_j'' = ((0, 2, 4), (4, 7, 0))$. As shown in Fig. 9, τ_j' assumed that the inter-arrival time of events was $p_j = 7$. However, it can be seen that the inter-arrival times of the first four events were actually $0p_i$, $1p_i$, $3p_i$, and $5p_i$. Conversely, under FE approach, the inter-arrival times of events could be calculated for τ_j'' based on Eq. (2) and Corollary 1. To exactly know the inter-arrival times of events is very important to timing-sensitive algorithms such as Kalman filters. With the absolute time of the current event, Kalman filters need to know when the prior event arrived and when the next event will arrive for refinement and prediction, respectively.

With multiple channels, this part of experiments was focused on evaluating whether or not processor time could be effectively allocated among channels for accurate tracking.

Now all four channels are enabled for experiments. A naive workload adjustment policy (*i.e.*, task τ_6) was adopted for both FE approach and PP approach. The following illustration was based on FE approach: Let the delivery of channel frames from task τ_3 to tasks $\tau_{4,1}$, $\tau_{4,2}$, $\tau_{4,3}$, and $\tau_{4,4}$ be subject to (m, k, ε)-firm constraint chosen from {(1, 7, 0), (2, 7, 0), (3, 7, 0), (4, 7, 0), (5, 7, 0), (6, 7, 0), (7, 7, 0)}. Each time when task τ_6 was invoked, it picked the channel which was of the largest average RMS error. Ties were broken arbitrarily. Suppose task $\tau_{4,i}$ corresponded to the chosen channel. Task τ_6 tried to promote (m_i, k_i, ε_i) (*e.g.*, from (3, 7, 0) to (4, 7, 0)). Let TH_{RMS} be a predefined system parameter for the threshold of the maximum difference between the average RMS errors of any two channels. If the workload could not be admitted by the test in Theorem 2, some procedures were taken: Let $\tau_{4,j}$ correspond the channel which had the smallest average RMS error. τ_6 tried to demote (m_j, k_j, ε_j) (*e.g.*, from (5, 7, 0) to (4, 7, 0)). The demotion was repeated until the resulted workload was schedulable. If all possible adjustments to workloads were not admitted, all changes were reverted. Note that, based on proportional period adjustment; workload adjustment for PP approach was done similarly. But the difference is that PP adjusts task periods.

The experimental results were presented in Fig. 10, where the X-axis denoted different settings of the threshold TH_{RMS} and the Y-axis denoted the average RMS error over the four channels. As we can see, FE approach significantly outperformed PP approach no matter what TH_{RMS} was. Because PP approach was less effective in tracking objects than FE approach was (as shown in the previous section), for one single channel to reduce it's average RMS error PP approach could require more CPU utilization than FE approach did, and consequently processor utilization available to the monitoring of other channels became small. We must point out that a very small setting of TH_{RMS} could result in frequent workload adjustments (*i.e.*, when TH_{RMS} = 1), which usually incorrectly slowed down the monitoring of some channels which needed a lot of attentions.

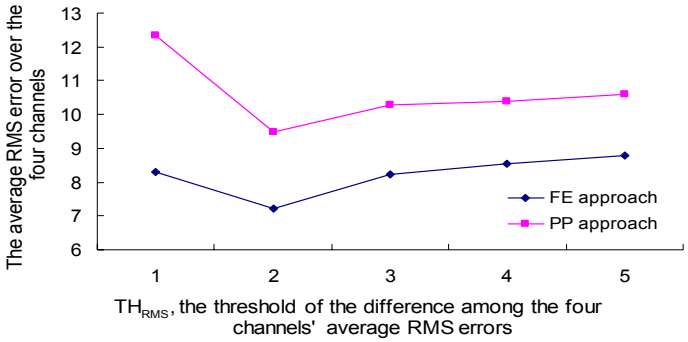


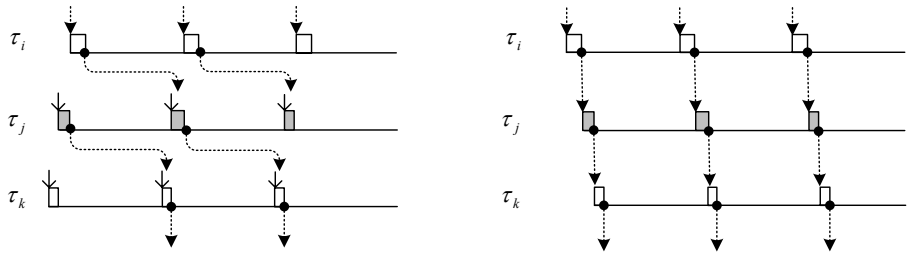
Fig. 10. The overall average RMS error resulted by the FE approach and the PP approach under different settings of the difference threshold of average RMS errors of channels (*i.e.*, TH_{RMS}).

5.3 Responsiveness

This section provides evaluations of the two approaches' responsiveness. Responsiveness was evaluated in terms of the time interval between the ground-truth time of a

scenario's occurrence and the time when the occurrence was detected. The time interval is referred to as the *response time* hereafter. The ground-truth time of the occurrences of a scenario was first measured from the video stream by off-line analysis, and during run-time the scenario was defined as a rule to be evaluated by task τ_5. From the off-line analysis, the ground-truth number of scenario occurrences was 32.

The experimental results were shown in Table 1. Compared to the results of PP approach, FE approach showed significantly reduced average response time. The phenomenon was due to misalignments of task jobs of PP approach: Consider the taskchain with $r_{i,j} = r_{j,k} = 1$ shown Figs. 11 (a) and (b). Because under PP approach any task job needed only to complete before its task period, a lengthy delay could be resulted as events were propagated among task jobs if they were not aligned. Such a phenomenon would be exaggerated when the periods of consumers and producers were relatively prime to each other because most task jobs were misaligned. The response time of the scenario shown in Fig. 11 (a) could potentially go up to $p_i + p_j + p_k$. On the other hand, with respect to FE approach shown in Fig. 11 (b), a job of task τ_k must complete before the upcoming job of task τ_i arrived, and consequently the worst-case response time was bounded by p_i. In addition, as showed in Table 1, four scenario occurrences were not detected by PP approach due to ineffective tracking. The number of undetected occurrences was only two under FE approach.



(a) PP approach. (b) FE approach.

Fig. 11. Scenarios of FE approach and PP approach in responding hardware events, where $r_{i,j} = r_{j,k} = 1$.

Table 1. The average response time of PP approach and FE approach.

(The ground-truth number of scenario occurrences was 32 in the pre-recorded video)

	The average response time (ms)	The number of scenario occurrences detected
The FE approach	5.41	30
The PP approach	3.22	28

6. CONCLUSION

This paper considers a workload scaling mechanism for uniprocessor real-time embedded systems. The design objectives are to hide complex precedence constraints among tasks from outside, and to expose a set of simple but effective primitives for workload scaling. A new approach that combines firm-real-time scheduling and an event-driven paradigm is proposed: The feeding of external hardware events are controlled by a

deterministic algorithm, and dependencies among task jobs are formulated as events. Because task jobs are triggered by events, the whole system automatically reacts to when and how external hardware events are fed in. We also show that, with our approach, the system still have deterministic timing behaviors, thus timing-sensitive algorithms such as Kalman filters are benefited a lot. An on-line admission control policy is proposed so that system timing correctness can be verified before any changes to workloads can be made. A series of experiments were conducted on a real-time surveillance system based on the proposed approach. Experimental results showed that, under heavily stressing workloads, the proposed approach still provided better accuracy and higher responsiveness for visual object tracking than a system using proportional period adjustment for purely periodic tasks.

REFERENCES

1. K. Jeffay, "The real-time producer/consumer paradigm: a paradigm for the construction of efficient, predictable real-time systems," in *Proceedings of ACM Symposium on Applied Computing*, 1993, pp. 796-804.
2. A. K. Mok and W. Wang, "Window-constrained real-time periodic task scheduling," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, 2001, pp. 15-24.
3. R. West and C. Poellabauer, "Analysis of a window-constrained scheduler for real-time and best-effort packet streams," in *Proceedings of the 21st IEEE Real-Time Systems Symposium*, 2000, pp. 239-248.
4. G. Koren and D. ShaSha, "Skip-over: algorithms and complexity for overloaded systems that allow skips," in *Proceedings of the 16th IEEE Real-Time Systems Symposium*, 1995, pp. 110-117.
5. P. Ramanthan, "Overload management in real-time control applications using (*m*, *k*)-firm guarantee," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, 1999, pp. 549-559.
6. M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (*m*, *k*)-firm deadlines," *IEEE Transactions on Computers*, Vol. 44, 1995, pp. 1443-1451.
7. C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, Vol. 20, 1973, pp. 46-61.
8. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System architecture directions for networked sensors," in *Proceedings of the 9th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 93-104.
9. W. K. Shih and J. W. S. Liu, "On-line scheduling of imprecise computations to minimize error," in *Proceedings of the 13th IEEE Real-Time Systems Symposium*, 1992, pp. 280-289.
10. "The open computer vision library," http://sourceforge.net /projects /opencvlibrary/.
11. B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, 1981, pp. 121-130.
12. G. Welch and G. Bishop, "An introduction to the Kalman filter," Technical Report No. TR 95-041, Dept. of Computer Science, University of North Carolina, 1995.

13. K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming*, Vol. 40, 1994, pp. 117-134.
14. P. Richard, F. Cottet, and M. Richard, "On-line scheduling of real-time distributed computers with complex communication constraints," in *Proceedings of the 7th IEEE International Conference on Engineering of Complex Computer Systems*, 2001, pp. 26-34.
15. K. Ramamritham, "Allocation and scheduling of precedence-related periodic tasks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, 1995, pp. 412-420.
16. T. Y. Yen and W. Wolf, "Performance estimation for real-time distributed embedded systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, 1998, pp. 1125-1136.
17. Y. Shin and K. Choi, "Rate assignment for embedded reactive real-time systems," in *Proceedings of the 24th EUROMICRO Conference*, Vol. 1, 1998, pp. 237-242.
18. T. W. Kuo and A. K. Mok, "Incremental reconfiguration and load adjustment in adaptive realtime systems," *IEEE Transactions on Computers*, Vol. 46, 1997, pp. 1313-1324.
19. T. Gustafsson and J. Hansson, "Data freshness and overload handling in embedded systems," Technical Report, http://www.ida.liu.se/~thogu/gustafsson06admission.pdf, 2006.
20. K. D. Kang, S. H. Son, and J. A. Stankovic, "Managing deadline miss ratio and sensor data freshness in real-time databases," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, 2004, pp. 1200-1216.
21. L. P. Chang, "Event-driven scheduling for dynamic workload scaling in uniprocessor embedded systems," in *Proceedings of the 21st ACM Symposium on Applied Computing*, 2006, pp. 1462-1466.

Li-Pin Chang (張立平) received his Ph.D. and M.S. degrees from Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, in 2003 and 1997, respectively. After two years of military service in Judicial Yuan, Taiwan, he serves as Assistant Professor at Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, since 2005. His current research interests are of two main themes: real-time computing, and embedded storage systems.

Ya-Shu Chen (陳雅淑) received her B.S. degree from National Chiao Tung University, Taiwan, R.O.C., in 2001. She received her M.S. and Ph.D degree from National Taiwan University, Taiwan, R.O.C., in 2003 and 2007, respectively. She serves as Assistant Professor at Department of Electrical Engineering, National Taiwan University of Science and Technology, since 2007. Her current research interests include real-time systems, embedded systems, and system-on-a-chip.