Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 307869, 8 pages
http://dx.doi.org/10.1155/2014/307869

Research Article

An Improved Task Scheduling Algorithm for Intelligent Control in Tiny Mechanical System

Jialiang Wang, Hai Zhao, Yuanguo Bi, Xingchi Chen, Ruofan Zeng, and Yu Wang

College of Information Science & Engineering, Northeastern University, Shenyang 110819, China

Correspondence should be addressed to Yuanguo Bi; biyuanguo@ise.neu.edu.cn

Received 28 February 2014; Accepted 1 April 2014; Published 22 April 2014

Academic Editor: Weichao Sun

Wireless sensor network (WSN) has been already widely used in many fields in terms of industry, agriculture, and military, and so forth. The basic composition is WSN nodes that are capable of performing processing, gathering information, and communicating with other connected nodes in the network. The main components of a WSN node are microcontroller, transceiver, and some sensors. Undoubtedly, it also can be added with some actuators to form a tiny mechanical system. Under this case, the existence of task preemption while executing operating system will not only cost more energy for WSN nodes themselves, but also bring unacceptable system states caused by vibrations. However for these nodes, task I/O delays are inevitable due to the existence of task preemption, which will bring extra overhead for the whole system, and even bring unacceptable system states caused by vibrations. This paper mainly considers the earliest deadline first (EDF) task preemption algorithm executed in WSN OS and proposes an improved task preemption algorithm so as to lower the preemption overhead and I/O delay and then improve the system performance. The experimental results show that the improved task preemption algorithm can reduce the I/O delay effectively, so the real-time processing ability of the system is enhanced.

1. Introduction

WSN consists of spatially distributed autonomous sensors to monitor physical or environmental conditions, such as temperature, sound, and pressure. These WSN nodes cooperatively pass their data through the network to a main location. Energy issue and system performance of WSN have always been considered because WSN nodes always work in some harsh environment. Energy is the scarcest resource for WSN nodes, and it finally determines the lifetime of WSNs. Energy consumption of the sensor nodes should be minimized since their limited energy resource determines their lifetime [1]. For the WSN OS used in some practical application of connecting some actuators, the task preemption will inevitably bring I/O delay and even unacceptable system states caused by the vibrations [2]. Nowadays, while WSN nodes formed a tiny mechanical system by adding actuators, how to decrease the system extra overhead and improve the system performance caused by the vibrations becomes very important.

Currently, there are many researches about improving system performance for WSN nodes; for example, [3] used error and kernel density approach of sleeping nodes in cluster-based WSNs; [4] used low energy online self-test of embedded processor in dependable WSN nodes. Reference [5] studied low-energy symmetric key distribution in WSN; [6] used a dynamic programming approach for QoS-aware power management in wireless video sensor. Reference [7] used virtual backbone scheduling to maximize the lifetime of WSNs. Reference [8] used hybrid-storage units to extend runtime of low-power WSN nodes. Reference [9] studied distributed tracking with energy management in wireless sensor networks. Reference [10] used joint routing and sleep scheduling to maximize lifetime of WSN. Thus it can be seen that most solutions are implemented by the way of utilizing extra resources. But the innovation for this paper is to improve the lifetime of WSN nodes in terms of real-time operating system itself and the task scheduling algorithm to be exact. The decrease of system overhead and the improvement of system performance are implemented by reducing

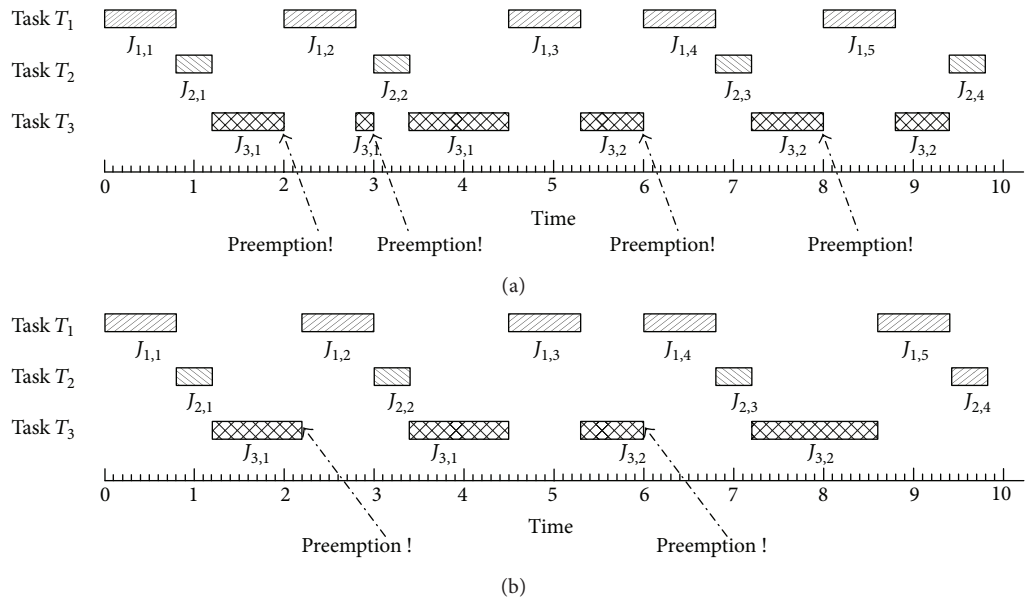

Figure 1: Execution case of EDF task scheduling before and after optimization.

the overhead of task preemptions. The most important is that this method can be combined with all of the others to achieve better system performance in tiny mechanical system composed of WSN nodes, as this improvement is focused on the real-time operating system itself.

2. EDF Task Scheduling Analysis

Compared with fixed priority task scheduling algorithm, dynamic priority algorithm allocates different priorities to every period task, so the task priority is constantly changing. EDF task scheduling algorithm has already been widely used as a classic dynamic priority scheduling algorithm in WSN [11].

EDF task scheduling algorithm assigns tasks with different priority according to execution deadline of task, the earlier task execution deadline, and the higher task priority. In a word, system allocates the highest priority to task having the earliest task execution.

2.1. Judging Whether the System Is Schedulable. Supposing that system has n independent and preemptive period tasks τ_i $(1 \leq i \leq n)$. E_i, D_i, and T_i represent the execution time, relative deadline, and period of task τ_i, respectively. The sufficient condition for EDF task scheduling algorithm in single processor is given by

$$\sum_{i=1}^{n} \frac{E_i}{\min(D_i, T_i)} \leq 1. \quad (1)$$

While task τ_i $(1 \leq i \leq n)$ has max block time b_i, the above sufficient condition will be expressed as

$$\sum_{i=1}^{n} \frac{E_i + b_i}{\min(D_i, T_i)} \leq 1. \quad (2)$$

If this formula can be met for all tasks, the system is regarded as schedulable. The all below discussions are all based on formula (2).

2.2. Task Preemption Optimization about EDF Task Scheduling Algorithm. Use the attribute set (T_i, E_i) to describe a period task, where T_i is the period of task and E_i is the execution time of the task, while using $J_{m,n}$ to describe one specific job in a task, which means that the task m has n jobs totally. For example, supposing that there are three jobs in the task T_1, the first one is $J_{1,1}$, the second one is $J_{1,2}$, and the last one is $J_{1,3}$.

Suppose that there are three period tasks $T_1(2, 0.8)$, $T_2(3, 0.4)$, and $T_3(5, 2.1)$ in the single processor system. For these three tasks, their super period is 30.

As Figure 1(a) shows, task T_1, T_2, and T_3 are all released at time 0, but their execution deadline is 2, 3, and 5, so task T_1 has the highest priority and it begins to run firstly. At time 0.8, task T_2 begins to run as it has higher priority than task T_3. Till time 1.2, task T_3 begins to run. While at time 2, task T_1 is released again, task T_3 is preempted by task T_1 then. Figure 1(a) shows the whole execution case before optimization for these three tasks during the time period $[0, 10]$.

Figure 1(a) shows that the beginning time for jobs on task T_1, T_2, and T_3 is $\{0, 2, 4.5, 6, 8\}$, $\{0.8, 3, 6.8, 9.4\}$, and $\{1.2, 2.8, 3.4, 5.3, 7.2, 8.8\}$, respectively. We can see from the results that the jobs of task T_3 are preempted 4 times totally, and each time point of preemption is shown as in Figure 1(a).

Figure 1(b) shows the execution case after optimization during the time period $[0, 10]$; the beginning time for jobs of tasks T_1, T_2, and T_3 becomes $\{0, 2.2, 4.5, 6, 8.6\}$, $\{0.8, 3, 6.8, 9.4\}$, and $\{1.2, 3.4, 5.3, 7.2\}$, respectively. Under this case, task T_3 is preempted 2 times totally, and in the meanwhile, task T_1 and T_2 do not exceed their deadline. This

is implemented by postponing the execution of some jobs for task T_1 and T_2, which means that nonpreemptive scopes are added for some jobs in task T_3, and the below discussion will describe the process of implementation in detail.

2.3. Extra Overhead Caused by Task Preemption. $\text{Pree}_{i,u}$ is a task preemptive set, which describes that task τ_i is executed u times totally. The number of elements for $\text{Pree}_{i,u}$ is signed as $w_{i,u}$, which means the times that task τ_i is preempted totally, so the total amount for all these tasks are given by

$$N_{\text{pree}}(I,t) = \sum_{i=1}^{n-1} \sum_{u=0}^{\lceil t/T_i \rceil - 1} w_{i,u}. \tag{3}$$

If a low priority task is preempted by another one, then the extra cost for the preemption is C_{pree}. While there is no preemption exists and then the extra cost is C_{nonpree}, for any task, no matter whether it is preempted, the total extra cost for the system is $C_{\text{pree}} - C_{\text{nonpree}}$ [11].

If the phase position is I for the real-time task set $\tau_0, \ldots, \tau_{n-1}$, the extra overhead caused by task preemption during the time period $[0, t]$ is described as

$$O_{\text{preempt}}(I,t) = \sum_{i=1}^{n-1} \sum_{u=0}^{\lceil t/T_i \rceil - 1} w_{i,u} \cdot \left(C_{\text{pree}} - C_{\text{nonpree}}\right). \tag{4}$$

So we can know that the task preemption will bring extra overhead for the whole system. As saving energy is very important for WSN nodes that formed a tiny mechanism system, how to reduce the task preemption becomes very necessary.

3. Analysis of I/O Delay for EDF Task Scheduling

For the release time of all jobs in set S, there exists a max I/O delay for certain job [11, 12].

3.1. Analysis of I/O Delay Caused by Task Preemption. Supposing that allocated preemptive threshold PT_i equals E_i and ignoring the restrain of preemptive threshold time, then according to the analysis of time requirement, the max I/O delay for preemptive task scheduling of τ_i is given by

$$IOL_i(a) = \max\left\{E_i, L_i(a) - a\right\}, \tag{5}$$

where a is the release time of job; then, $L_i(a)$ can be calculated by

$$\begin{aligned} L_i^{(0)}(a) &= 0, \\ L_i^{(m+1)}(a) &= W_i\left(a, L_i^{(m)}(a)\right) + \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor\right) C_i. \end{aligned} \tag{6}$$

While $L_i^{(m+1)}(a) = L_i^{(m)}(a)$, the above iterative calculation will end.

For task τ_i, while the release time of its one job is a, the job load of higher priority can be represented as

$$W_i(a,t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i}} \min\left\{\left\lceil \frac{t}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor\right\} \cdot E_j. \tag{7}$$

A preemptive threshold PT_i ($0 \leq \text{PT}_i \leq E_i$) is assigned to task τ_i, and the I/O delay should be calculated by the preemptive part and nonpreemptive part, respectively, as below [11].

The max I/O delay of preemptive part for task τ_i is given by

$$IOL_i^{\text{pree}}(a) = \max\left\{\text{PT}_i, L_i^{\text{pree}}(a) - a\right\} \quad i = 1, 2, \ldots, n; \tag{8}$$

$$\begin{aligned} L_i^{\text{pree}}(a) &= W_i\left(a, L_i^{\text{pree}}(a)\right) \\ &\quad + \left\lfloor \frac{a}{T_i} \right\rfloor \cdot E_i + \text{PT}_i + \max_{D_j > D_i}\left(E_j - \text{PT}_j\right), \end{aligned} \tag{9}$$

where $L_i^{\text{pree}}(a)$ is the max value of formula (9), and it can be calculated by the initialization value $L_i^{\text{pree}(0)} = 0$.

The max I/O delay of nonpreemptive part for task τ_i is represented as

$$IOL_i^{\text{nonp}} = \left(E_i - \text{PT}_i\right) \quad i = 1, 2, \ldots, n; \tag{10}$$

so the max I/O delay for task τ_i based on EDF scheduling algorithm is described as

$$\begin{aligned} IOL_i^{\max} &= IOL_i^{\text{pree}}(a) + IOL_i^{\text{nonp}} \\ &= \max\left\{\text{PT}_i, L_i^{\text{pree}}(a) - a\right\} + \left(E_i - \text{PT}_i\right) \\ &\qquad i = 1, 2, \ldots, n. \end{aligned} \tag{11}$$

While $\text{PT}_i \geq L_i^{\text{pree}}(a) - a$, then

$$IOL_i^{\max} = \text{PT}_i + \left(E_i - \text{PT}_i\right) = E_i. \tag{12}$$

Under this case, the I/O delay is a constant value, and it is irrelevant to task attribute.

While $\text{PT}_i < L_i^{\text{pree}}(a) - a$,

$$\begin{aligned} IOL_i^{\max} &= IOL_i^{\text{pree}}(a) + IOL_i^{\text{nonp}} \\ &= L_i^{\text{pree}}(a) - a + \left(E_i - \text{PT}_i\right) \\ &= W_i\left(a, L_i^{\text{pree}}(a)\right) + \left\lfloor \frac{a}{T_i} \right\rfloor \cdot E_i + \text{PT}_i \\ &\quad + \max_{D_j > D_i}\left(E_j - \text{PT}_j\right) - a + \left(E_i - \text{PT}_i\right) \\ &= W_i\left(a, L_i^{\text{pree}}(a)\right) + \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor\right) \cdot E_i \\ &\quad + \max_{D_j > D_i}\left(E_j - \text{PT}_j\right) - a. \end{aligned} \tag{13}$$

From the above formulas, we can know that PT_i is the main factor for $IOL_i^{\max}$, and the next section will describe how to assign PT_i for jobs.

(1) Input: $n, T_i, E_i, D_i\ (i = 1, 2, \ldots, n)$;
(2) $\mathrm{PT}_1^{\min *} = E_1$, and allocating preemptive threshold $\mathrm{PT}_1 = E_1$ to task τ_1.
(3) Initializing $i = 1$;
(4) Calculating b_i^* by formula (19);
(5) Assigning $b_{i+1}^*, b_{i+2}^*, \ldots, b_n^*$ to b_i^* separately;
(6) $i = i + 1$, returning to Step (4) till all b_i^* are calculated.
(7) **for** $i = 1, 2, \ldots, n$,
(8) Obtaining all preemptive time threshold by formula (20) and (21).
(9) **endfor**
(10) **for** tasks $\tau_i\ (i = 1, 2, \ldots, n)$
(11) Calculating max I/O delay $IOL_i^{\max}$, and min I/O delay min I/O delay.
(12) **endfor**
(13) Output: $\mathrm{PT}_1, \mathrm{PT}_2, \ldots, \mathrm{PT}_n$, $IOL_i^{\max}$ and $IOL_i^{\min}$.

Algorithm 1

3.2. Improved Task Preemption Scheduling Algorithm. In the EDF task scheduling algorithm, task is indexed by the relative time slice. For task τ_i, the bigger the index, the bigger relative time slice. So only the task with big relative time slice can block the task with smaller one and the time requirement of preemptive part for task τ_i having below restriction of preemptive time threshold [11]. Consider

$$\begin{aligned} &w_i^{\mathrm{pree}}\left(a, \mathrm{PT}_1, \mathrm{PT}_2, \ldots, \mathrm{PT}_n\right) \\ &\quad = W_i\left(a, L_i^{\mathrm{pree}}(a)\right) + \left\lfloor \frac{a}{T_i} \right\rfloor E_i + \mathrm{PT}_i + b_i, \\ &\quad b_i = \max_{D_j > D_i}\left(\mathrm{PT}_j\right). \end{aligned} \tag{14}$$

The max execution time of nonpreemptive part is $E_j - \mathrm{PT}_j$ for task τ_i, so the necessary condition that task can be schedulable is there is existing a $(a \in S)$, which meets the below formula:

$$w_i^{\mathrm{pree}}\left[a, \mathrm{PT}_1, \mathrm{PT}_2, \ldots, \mathrm{PT}_n\right] + E_i - \mathrm{PT}_i \le a + D_i \tag{15}$$

as S is the detecting set, and it is given by

$$\begin{aligned} &S = \bigcup_{p=1}^{n} S_p, \\ &S_p = \left\{ D_p + kT_p - D_i,\ k \in N, \right. \\ &\qquad \left. \left\lceil \frac{D_i - D_p}{T_p} \right\rceil \le k < \left\lceil \frac{L + D_i - D_p}{T_p} \right\rceil \right\}, \end{aligned} \tag{16}$$

where $a \in S$, and it meets the following formula:

$$\begin{aligned} &\max_{j=i+1}^{n}\left(E_j - \mathrm{PT}_j\right) \\ &\quad \le a + D_i - W_i\left(a, L_i^{\mathrm{pree}}(a)\right) + \left\lfloor \frac{a}{T_i} \right\rfloor E_i + \mathrm{PT}_i + b_i, \\ &\quad b_i = \max_{D_j > D_i}\left(E_j - \mathrm{PT}_j\right). \end{aligned} \tag{17}$$

The necessary and sufficient condition for EDF task scheduling is described as

$$\begin{aligned} &\sum_{h=1}^{n} \frac{E_h + b_i}{\min\left(D_h, T_h\right)} \le 1, \\ &b_i = \max_{D_j > D_i}\left(E_j - \mathrm{PT}_j\right). \end{aligned} \tag{18}$$

Then b_i can be calculated by

$$b_i \le \min\left(D_i, T_i\right)\left(1 - \sum_{h=1}^{n} \frac{E_h}{\min\left(D_h, T_h\right)}\right). \tag{19}$$

To make the task τ_i schedulable, the min value of PT_{i+1} is given by

$$\mathrm{PT}_{i+1}^{\min} = E_{i+1} - \min\left(D_i, T_i\right) \cdot \left(1 - \sum_{h=1}^{n} \frac{E_h}{\min\left(D_h, T_h\right)}\right). \tag{20}$$

So to make all tasks schedulable, the min value of PT_{i+1} is represented as

$$\mathrm{PT}_{i+1}^{\min^*} = \max_{j=1}^{i}\left(\mathrm{PT}_{j+1}^{\min}\right). \tag{21}$$

From the above analyses, system allocates the $\mathrm{PT}_1 = \mathrm{PT}_1^{\min^*} = E_1$ to the task τ_1 that has the smallest relative deadline. The PT value of other jobs will also be allocated by formula (21).

The system obtains block time b_i for every task and calculates threshold of tasks whose relative time slices are larger than PT_j according to b_i and then allocates threshold to task $i = 2, 3, \ldots, n$ by Algorithm 1.

Algorithm 1 implements an optimization allocation of preemptive time threshold for each job. From the above formula (18) and (19), we can know that the time complexity of this algorithm is $O(n)$.

$\mathrm{TID} = \{\mathrm{tid}_1, \mathrm{tid}_2, \ldots, \mathrm{tid}_N\}$ which corresponds to task release time. If several tasks are released at the same time, then the system uses the task TID having the highest priority.

(1) Input: $n, \beta, T_i, E_i, \varphi_i$ $(i = 1, 2, \ldots, n)$;
(2) Initializing $\xi_i = 0$; $pree_i(\beta) = 0, (i = 1, 2, \ldots, n)$; $pree\ (\beta) = 0$;
(3) Constructing TID = $\{tid_1, tid_2, \ldots, tid_N\}$ and Set $S(r) = \{r_1, r_2, \ldots, r_N\}$; // N $\leq \sum_{i=1}^{n} \left\lfloor \frac{\tau - \varphi_i}{T_i} \right\rfloor + n$
(4) **for** $k = 1$ to N;
(5) Detecting release time of all tasks;
(6) $t = r_k; m = tid_k$;
(7) **if** $s_m(t) - s_m(\xi_m) \leq 0$ **then**
(8) Ending this loop, and detecting next release time;
(9) **endif**
(10) **if** $s_n(t) - s_n(\xi_n) > 0$ **then**
(11) Refreshing $\xi_i = t\ (i = m, m + 1, \ldots, n)$;
(12) Ending this loop, and detecting next release time;
(13) **endif**
(14) System finds the index $\lambda = i$ in $m + 1, m + 2, \ldots, n$, and makes $s_{i-1}(t) - s_{i-1}(\xi_{i-1}) > 0$ and $s_i(t) - s_i(\xi_i) \leq 0$;
(15) **if** $s_\lambda(t) - s_\lambda(\xi_\lambda) \neq 0$ **then**
(16) $pree_\lambda(\beta) = pree_\lambda(\beta) + 1$;
(17) **endif**
(18) Refreshing $\xi_i = t\ (i = m, m + 1, \ldots, \lambda - 1)$;
(19) **endfor**
(20) Output: $pree_i(\beta), (i = 1, 2, \ldots, n)$; the amount of task i is preempted
$pree\ (\beta) = \sum_{i=1}^{n} pree_i(\beta)$; the amount of all preemptions

Algorithm 2

At any given time t, $s_i(t) = t - \sum_{k=1}^{i} \lceil (t - \varphi_k)/T_k \rceil \cdot E_k$ is used to express the time difference between system supplying and finishing the all jobs whose priority is no less than job i during time period $[0, t)$.

For any one task set based on EDF task scheduling algorithm, system can calculate the amount of task preemption by task attribute during the time period $[0, \gamma]$ according to Algorithm 2.

The above algorithms describe the implementation of postponing the execution of some jobs for some tasks; at the meanwhile, nonpreemptive scope is added for some jobs in others tasks, and the below discussion will describe the experiments about the performance test.

4. Experimental Platform

In order to test the performance of the improved task scheduling algorithm, we used the hardware platform of Webit&NEU [13] and its operating system Webit&NEU OS which were implemented by our China Liaoning Province Embedded Technique Key Laboratory.

4.1. Hardware Platform. Webit&NEU (Atmega128L Processor) uses the basic structure of Webit2.0. It can collect and handle the data collected from the CC2520 and store the core data to data memory.

Webit2.0 (AT90S8515 Processor) is an embedded Internet device that has implemented the function about making the standard industry device that can access the Internet. Webit2.0 has successfully been developed by our China Liaoning Province Embedded Technique Key Laboratory and has been manufactured by our China Shenyang NeuEra Information Technology Stock CO., LTD. The most important is that Webit2.0 has also achieved the China's product patent (numbers ZL 00 2 52482.1 and ZL 01 2 29725.9) [14].

4.2. Software Platform. A real-time operating system must have the ability to respond to external events fast. In order to coordinate and execute tasks effectively, Webit&NEU OS system provides the following services: interrupt handlers, intertask communication facilities, task synchronization, internal memory management, timing services, task priority assignment, and so on as Table 1 shows [15, 16]. Webit&NEU OS supports up to 16 tasks, which is committed to achieve low cost application in the IoT. These tasks have 8 different priorities of 0–7. The bigger the value, and higher the priority.

All of these functions implemented by the unique design of system call can easily execute real-time applications. System call is a program supported to user by operating system, while users need to implement certain system function; then, what they need to do is to call the accordingly system call in the application program; thus, it is very convenient for users to use the system kernel.

All the system calls are written in AVR assembly language. One of the important advantage for AVR assembly language is that it can flexibly and easily manage a certain storage location in memory, even its certain bit of the storage location, so system can effectively handle task, interruption, message, timing, and so on.

All of these functions can easily implement real-time applications, so the Webit&NEU OS system can well be

Table 1: The structure of Webit&NEU OS kernel.

Function module	Implementation of system calls
(a) Task management services	(i) CREATE TASK: Creating and scheduling a system task dynamically. (ii) DELETE TASK: Deleting a specific system task. (iii) GET FUNCTION ID: Obtaining task's function ID (00H to 0FFH). (iv) SUSPEND TASK: Suspending a task of being executed.
(b) Intertask communication services	(i) ALLOCATE: Allocating buffer space where a task creates a message for sending. (ii) SEND MESSAGE: Sending a message to the specific destination. (iii) WAIT MESSAGE: Causing a task to wait for receiving a message. (iv) DEALLOCATE: Returning a buffer allocated to the system buffer.
(c) Internal memory management services	(i) GET MEMORY: Getting the address of a block of internal memory with specific length is currently available in the system. (ii) RELESE MEMORY: Returning a block of memory of specific length to the system memory pool.
(d) Interrupt-handling services	(i) DISABLE INTERRUPT: Disabling specific interrupts. (ii) ENABLE INTERRUPT: Reenabling disabled interrupts. (iii) SYNCHRONIZE: Synchronizing tasks with interrupts.
(e) Timing services	(i) SET INTERVAL: Setting a time interval when the interval event has occurred. (ii) WAIT TIME: Waiting for interval event or timeout to occur.

Table 2: Case of task preemption comparison before and after algorithm optimization.

System load U	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9
Before optimization	4	6	9	11	15	18	24	29	34	37	45	49	56	61	72	80	91	99
After optimization	2	3	5	6	8	10	13	16	19	21	26	28	33	36	44	50	59	66
Preemption optimization rate	50.0%	50.0%	44.4%	45.5%	46.7%	44.4%	45.8%	44.8%	44.1%	43.2%	42.2%	42.9%	41.1%	41.0%	38.9%	37.5%	35.2%	33.3%

used to effectively manage data collection, wireless communication in the field of WSN, and it has successfully been developed by our China Liaoning Province Embedded Technique Key Laboratory.

5. Performance Analysis of Improved EDF Scheduling Algorithm

For system load $U = 0.05, 0.1, 0.15, \ldots, 0.9$, the system generates 18 period task sets totally, and each set includes 6 period tasks generated at random. Table 2 shows the case of task preemption comparison before and after algorithm optimization.

From the experimental results, we can know that the optimization scope ranges from 50.0% to 33.3%; with the increase of system load, the preemption optimization rate decreases correspondingly, which means that system can get better optimization while system load is lower [17–19].

We used the tools of logic analyzer (TLA603), arbitrary waveform generator (AWG2021), and digital storage oscilloscope (TDS1012). In the test programs, we added some control codes by transforming the value of high and low levels of pulses through I/O pin while task is preempted.

The I/O pin of Webit&NEU is connected to the digital storage oscilloscope (TDS1012). While there are the task preemptions happening, the system will alter the current value of pulses when certain task is about to be preempted; accordingly, the system will also change the current value of pulses again while the preempted task begins to execute again. So from the waveform of the experimental results in the digital storage oscilloscope, we can obtain the whole time T_w for the whole process. Besides, we can also get the whole execution time of the preemption task T_e. So I/O delay time $T_{\text{I/O delay}}$ can be calculated by $T_{\text{I/O delay}} = T_w - T_e$.

Under the case of different system load, we obtained the I/O delay according to the above method, and the experimental results are shown in Figure 2.

As Figure 2 shows, while the system load is 0.05, system can achieve the best optimization rate of 50.8%; however, even the system load is 0.9 and the optimization rate can still reach 31.6%. Obviously, the proposed improved task scheduling algorithm can well decrease the I/O delay for the real-time operating system. The system I/O delay is obviously reduced because of the decrease of task preemption, which also means that the system overhead is also reduced. Consequently system real-time ability is enhanced, and at the meanwhile, energy consumption is also obviously decreased [20].

Table 3: Comparison of system kernel size (KB).

Operating system	System kernel code size
Webit&NEU OS	3.856
Webit&NEU OS with improved algorithm	4.031
μCOS-II	About 12

Table 3 shows the system kernel code size of Webit5.0 OS (the fifth version operation system for Webit2.0), Webit&NEU OS, and μC/OS-II separately. Because Webit&NEU OS supports only up to 16 tasks, it is committed

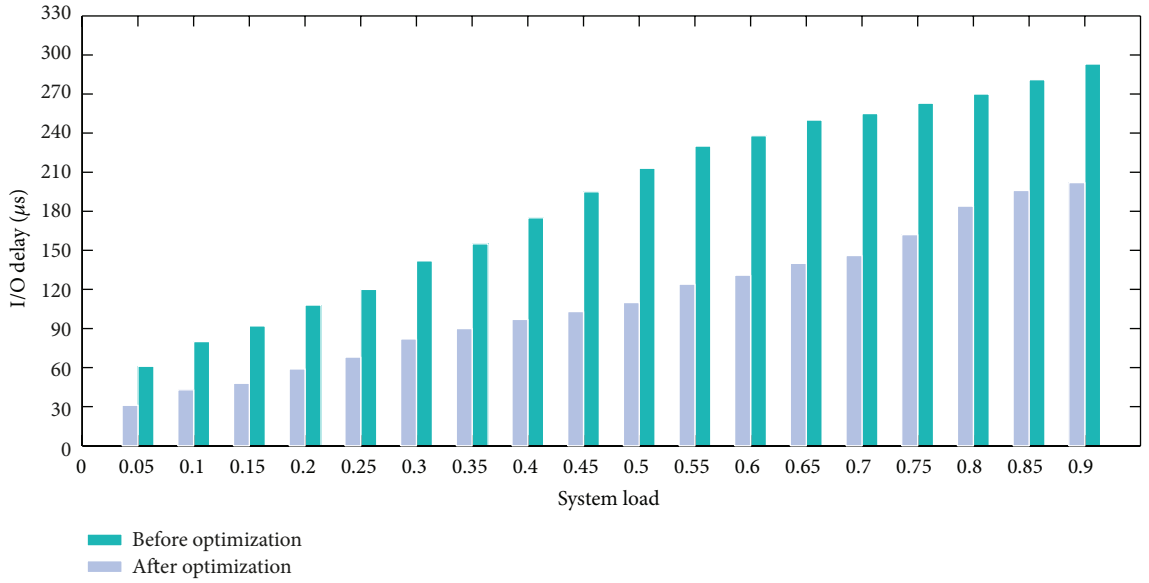

Figure 2: I/O delay comparison before and after algorithm optimization.

to low cost application in the IoT, so its kernel code size is smaller than μC/OS-II who manages up to 64 tasks. This is just the design goal of light-weight operating systems for Webit&NEU OS.

As the code memory of Atmega128L is 128 KB, the code size of Webit&NEU OS with improved algorithm (4.031 KB) is 4.34% larger than Webit&NEU OS's (3.856 KB); its size only occupies 3.15% of the whole code memory, so it still leaves enough space for user applications.

6. Conclusions

Compared with other current common solutions to increase the lifetime of WSN nodes, this paper aims to improve the lifetime and system performance by optimizing task scheduling algorithm when WSN nodes are applied to actuator applications. On the premise that these jobs can be executed within their execution deadline, system postpones executing some jobs to ensure that more jobs can better be executed, which can reduce task preemption greatly. By executing the improved task scheduling algorithm in the platform of Webit&NEU and Webit&NEU OS both designed and implemented by our laboratory, the experimental results show that the system I/O delay and overhead are decreased effectively and the occurrences of unacceptable system states caused by vibrations are also reduced, due to the reduction of task preemption in tiny mechanical system composed of WSN nodes.

Conflict of Interests

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

Acknowledgments

This work is partly supported by National Natural Science Foundation of China (no. 61101121), National High Technology Research and Development Program (863 Program) (no. 2013AA102505), Key Laboratory Project Funds of Shenyang Ligong University under Grant (no. 4771004kfs03), and Educational Committee of Liaoning Province Science and Technology Research Projects under Grant (no. L2013096). The authors thank Jan Vitek, Ales Plsek, and Lei Zhao for their help during their studying at Purdue University in USA from September 2010 to September 2012. They also thank the anonymous reviewers for their valuable comments.

References

[1] http://en.wikipedia.org/wiki/Wireless_sensor_network.

[2] W. Sun, H. Gao, and B. Yao, "Adaptive robust vibration control of full-car active suspensions with electro-hydraulic actuators," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 6, pp. 2417–2422, 2013.

[3] M. Peng, Y. Xiao, and P. P. Wang, "Error analysis and kernel density approach of scheduling sleeping nodes in cluster-based wireless sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 9, pp. 5105–5114, 2009.

[4] A. Merentitis, N. Kranitis, A. Paschalis, and D. Gizopoulos, "Low energy online self-test of embedded processors in dependable WSN nodes," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 86–100, 2012.

[5] K. McCusker and N. E. O'Connor, "Low-energy symmetric key distribution in wireless sensor networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 3, pp. 363–376, 2011.

[6] A. Fallahi and E. Hossain, "A dynamic programming approach for QoS-aware power management in wireless video sensor

networks," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 2, pp. 843–854, 2009.

[7] Y. Zhao, J. Wu, F. Li, and S. Lu, "On maximizing the lifetime of wireless sensor networks using virtual backbone scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1528–1535, 2012.

[8] M. T. Penella and M. Gasulla, "Runtime extension of low-power wireless sensor nodes using hybrid-storage units," *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 4, pp. 857–865, 2010.

[9] N. Roseveare and B. Natarajan, "Distributed tracking with energy management in wireless sensor networks," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 4, pp. 3494–3511, 2012.

[10] F. Liu, C.-Y. Tsui, and Y. J. Zhang, "Joint routing and sleep scheduling for lifetime maximization of wireless sensor networks," *IEEE Transactions on Wireless Communications*, vol. 9, no. 7, pp. 2258–2267, 2010.

[11] J. W. S. Liu, *Real-Time Systems*, Higher Education Press, Beijing, China, 2003.

[12] W. Sun, Y. Zhao, J. Li, L. Zhang, and H. Gao, "Active suspension control with frequency band constraints and actuator input delay," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 1, pp. 530–537, 2012.

[13] J. Wang, H. Zhao, J. Xu, and Y. Bi, "Webit&NEU: an embedded device for the Internet of things," *International Journal of Distributed Sensor Networks*, vol. 2014, Article ID 839540, 10 pages, 2014.

[14] Z. Hai and C. Yan, *Pervasive Computing*, Northeastern University Press, Shenyang, China, 2005.

[15] Intel Corporation, "iRMX 51 DISTRIBUTED CONTROL EXECUTIVE USER'S GUIDE for Release 2.0," Intel Corporation, 1987.

[16] J. Wang, H. Zhao, P. Li, H. Li, and B. Li, "Analysis and comparison of five kinds of typical device-level embedded operating systems," *Journal of Software Engineering and Applications*, vol. 3, no. 1, pp. 81–90, 2010.

[17] W. Sun, Z. Zhao, and H. Gao, "Saturated adaptive robust control for active suspension systems," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 9, pp. 3889–3896, 2013.

[18] W. Sun, H. Gao, and O. Kaynak, "Adaptive backstepping control for active suspension systems with hard constraints," *IEEE Transactions on Mechatronics*, vol. 18, no. 3, pp. 1072–1079, 2013.

[19] W. Sun, H. Gao Sr., and O. Kaynak, "Finite frequency H_{∞} control for vehicle active suspension systems," *IEEE Transactions on Control Systems Technology*, vol. 19, no. 2, pp. 416–422, 2011.

[20] R. Majumder, G. Bag, and K.-H. Kim, "Power sharing and control in distributed generation with wireless sensor networks," *IEEE Transactions on Smart Grid*, vol. 3, no. 2, pp. 618–634, 2012.