Security-Aware Periodic-Write Scheduling for Mission-Critical Embedded Storage System

Wei Jiang, Guang-Ze Xiong, Zheng-Wei Chang, Xu-Yang Ding, and Nan Sang

Abstract—High quality of security and guaranteed real-time requirements are two key goals of mission-critical embedded storage systems. But most existing real-time disk scheduling algorithms do not consider improving security performance of disk requests. A security-aware periodic-write (SAPW) scheduling algorithm is proposed to judiciously select appropriate security level for each disk request to maximize security value of \( N \) periodic disk users, while without sacrificing timing constraint of each user. Simulation results show the significant effectiveness of SAPW algorithm, and the average security improvement is up to 223.6% over other three algorithms.

Index Terms—Disk scheduling, embedded storage system, real-time, security-aware.

1. Introduction

Next-generation cyber-physical systems will pose great challenges in future embedded and real-time system design\cite{1}. Since many mission-critical embedded systems need to access, store, and manipulate security-sensitive data\cite{2}, how to improve the quality of security service in embedded and real-time systems has become an important problem. Embedded storage systems have been a hot research topic because of an increasing number of emerging data-intensive applications like video surveillance\cite{3}, cooperative storage in wireless sensor networks\cite{4,5}, and digital libraries\cite{6}. Data intensive storage managements are required to provide strong security mechanisms against talented damage or disclosure while guaranteeing response times for storage requests\cite{7}.

Recently, researchers are making much effort to time-sensitive disk scheduling\cite{8,9}. The Scan-EDF can be employed to provide temporal requirements\cite{10}. In [11], the authors developed an energy-aware scheduling for soft real-time disk I/O requests. Reference [12] proposed a novel real-time disk-scheduling algorithm called WRR-SCAN to provide quality guarantees for all in-service streams encoded at variable bit rates and bounded response times for aperiodic requests in multimedia servers. However, conventional disk scheduling algorithms nearly do not take into account security factors completely. In [13], a common security-evaluating framework was developed. Cryptographic file systems were developed to protect confidential data in untrusted storage systems\cite{14,15}. Key distribution schemes were proposed to enhance security of distributed storage systems\cite{16,17}. Reference [18] designed a secure and reliable local storage system for wireless sensor nodes. These mentioned security techniques did not function in disk scheduling mechanisms. Therefore they are not suitable to manage disk requests with time-critical constraints. Some dynamic security-aware scheduling schemes were proposed for clusters\cite{19} and embedded system\cite{20}, but these can not be directly applied to storage system. EDF-based adaptive security-aware write strategy was proposed to improve the security quality of local and distributed storage system\cite{7,21}. Differently, focusing on rate monotonic scheduling (RMS) policy, we study the static disk scheduling schemes for periodically multi-user disk requests, which have both security and real-time constraints.

The rest of this paper is organized as follows. Related work is presented in section 2. In section 3, system model and problem statement are illustrated. In section 4, a security-aware disk write scheduling algorithm is proposed. Simulation results are discussed in section 5 and section 6 concludes the paper.

2. Model Description and Problem Statement

2.1 Modeling for Security-Critical Storage System

In this study, we pay attention to periodic real-time storage applications with security requirements. Our system is a security-aware embedded storage architecture, as shown in Fig. 1. Disk requests are periodically issued by independent multiple users. Security library provides a number of security services, such as RC4, to protect mission-critical data against information damage. The responsibility of security-aware scheduler is to select the most proper security level for each periodic disk request.
We consider a group of time-critical disk write requests \( R = \{ R_1, R_2, \ldots, R_N \} \). Requests of each user are independent and periodic. For every user, each disk request is represented by a four-tuple, \( R_i = (l_i, w_i, s_i, p_i) \), where \( l_i \) is the size of confidential data to be wrote, \( w_i \) is the security weight of class \( R_i \), \( s_i \) is the finally assigned security level, and \( p_i \) is the period of class \( R_i \). In this paper, we assume \( p_i \) coinciding with \( R_i \)'s relative deadline. The \( j \)-th request of \( R_i \) which arrives at time \((j-1)\times p_i\) needs to finish its processing before time \( jp_i \). Note that we only consider write requests in this study.

### 2.2 Security Overhead

Since high security is achieved at the cost of performance degradation, we need to take into account the security overhead generated by security services.

In security overhead model of embedded storage system, we consider confidentiality security services, which are widely applied in storage system. Encryption is a good approach to guarantee the confidentiality of data stored in local storage system. Nine encryption algorithms are implemented for security mechanism, and the corresponding overhead of them are presented in Fig. 2. Each encryption algorithm is assigned a security indicator, meaning the level of importance. For example, security level of SEAL algorithm is 1, which is the weakest but fastest encryption algorithm. We do not consider the overhead of storing the cipher separately, because that overhead can be envisioned as a part of total security overhead.

### 2.3 Problem Statement

The main goal of this paper is to maximize security profit of periodic multi-user disk requests while without violating temporal constraints. The Security Profit (SP) of each user's disk request is related to data size, security weight and assigned security level, that is,

\[
SP_i = \sum_{j=1}^{P_i} w_i s_{ij}
\]

where \( P \) is the least common multiple of users’ disk request periods and \( S_{ij} \) is the security level of the \( j \)-th request of \( R_i \).

The total security profit (denoted by symbol \( \phi \)) is defined as the sum of the security profit of all users’ disk requests. So, we convert this problem as a linear optimizing problem. Our object is to maximize function (2), subjecting to some timing and security constraints:

Maximize

\[
\phi = \sum_{i=1}^{N} \sum_{j=1}^{P_i} w_i s_{ij}
\]

Subject to

\[
s_{ij} \in [s_{\text{min}}(i), s_{\text{max}}(i)]
\]

\[
\sum_{i=1}^{N} \sum_{j=1}^{P_i} (T_{\text{seek}} + T_{\text{rotate}} + l_i / B + l_if(s_{ij})) \leq P
\]

\[
R_{ij} < j p_i
\]

\[
\sum_{i=1}^{N} w_i = 1
\]

where \( N \) is the number of users; \( s_{\text{min}}(i) \) and \( s_{\text{max}}(i) \) are minimal and maximal security requirements of disk request \( R_i \) respectively; \( T_{\text{seek}} \) and \( T_{\text{rotate}} \) are the seek time and rotation time of disk head respectively; \( B \) is the bandwidth of disk I/O; \( f(\cdot) \) is a discrete function which is used to map a security level to the overhead of corresponding encryption service; \( R_{ij} \) is the end time of \( j \)-th request for user \( R_i \). The first constraint means the security requirements have to be met. The second constraint indicates that the total processing time demand should not exceed the processing capacity. The third condition shows that the disk request response must be guaranteed.

### 3. RMS-Based Security-Aware Scheduling Policy

In this paper, we consider RMS based static priority schemes for periodic disk requests. RMS is known as the optimal fixed-priority periodic scheduling algorithm. Furthermore, with the direct support in common real-time operating systems and well-established timing analysis methodologies, RMS remains as the most well-known and common real-time scheduling policy in practice. Each class of disk request is allocated a priority, which is sorted by the reverse order of periods. Without loss of generality, we assume that \( p_i < p_{i+1} \) \((i=1, 2, \ldots, N-1)\), which means that \( R_i \) has higher priority than \( R_{i+1} \).
To maximize security benefit of all disk requests, we strive to select the most appropriate security levels for disk requests and make sure that the deadlines of disk requests are not violated. Some approaches are now proposed to verify that whether it is possible to complete all disk requests within temporal constraints under chosen security quality. Given a group of real-time disk users with strict security requirements, the following two theorems are necessary and sufficient condition for feasibility check.

According to the exact time demand analysis (TDA) in [22], we define the time demand function \( w_q(t) \) of disk user \( R_i \) as follows:

\[
w_q(t) = C^i_{\text{base}} + C^i_{\text{sec}} + \sum_{k=1}^{t} \left( \frac{t}{p_i} \right) (C^i_{\text{base}} + C^i_{\text{sec}}), \quad t \in (0, p_i]
\]

where \( C^i_{\text{base}} = T_{\text{seek}} + T_{\text{rotate}} + l_i / B \) and \( C^i_{\text{sec}} = 1, f(s_i) \). \( C^i_{\text{base}} \) is the basic execution time of \( R_i \) and \( C^i_{\text{sec}} \) is the security overhead of \( R_i \).

**Theorem 1.** Given a group of periodic disk requests \( R = \{R_1, R_2, \ldots, R_N\} \) and a set of security levels \( S = \{s_1, s_2, \ldots, s_N\} \), all disk requests with fixed priorities can be feasibly scheduled for single disk if and only if each user’s request satisfies following conditions:

\[
\forall i, \exists t, w_q(t) \leq t \leq p_i
\]

**Proof.** This is the direct application of Lehoczky’s research efforts on time demand analysis [22]. Hence, we omit the detail of the proof.

**Theorem 2.** For a set of periodic disk requests \( R = \{R_1, R_2, \ldots, R_N\} \) with corresponding security levels \( S = \{s_1, s_2, \ldots, s_N\} \), if the following condition is satisfied, they are schedulable by RMS policy.

\[
\forall i: 1 \leq i \leq N, \min_{m \in M_i} \sum_{j=1}^{m} \frac{C^i_{\text{base}} + C^i_{\text{sec}}}{m} \leq 1
\]

where

\[
M_i = \left\{ q \mid 1 \leq k \leq q = 1, \ldots, \frac{p_i}{p_j} \right\}
\]

**Proof.** It is also the direct results of Lehoczky’s research efforts on fixed priority scheduling [22].

Theorem 1 and 2 are two sufficient and necessary RMS feasibility test methods, but both of them are very complex and time consuming to test the feasibility of a group of disk I/O requests. Therefore, it is necessary to introduce a simple and easy-to-implement method for feasibility test.

**Theorem 3.** Given a group of periodic disk requests \( R = \{R_1, R_2, \ldots, R_N\} \) and a set of security levels \( S = \{s_1, s_2, \ldots, s_N\} \), the security-aware schedulable condition is

\[
\mu^\text{base} + \mu^\text{sec} \leq N(2^{1/N} - 1)
\]

where

\[
\mu^\text{base} = \sum_{i=1}^{N} \frac{C^i_{\text{base}}}{p_i}, \quad \mu^\text{sec} = \sum_{i=1}^{N} \frac{T_{\text{seek}} + T_{\text{rotate}} + l_i / B}{p_i}
\]

**Proof.** It has been proved that the sufficient feasibility check for a task set under RMS scheduling requires the total utilization of the worst-case utilizations being equal to or less than the Liu-Layland bound (LLB) [23], i.e.,

\[
U_{\text{sys}} \leq \text{LLB}(N) = N(2^{1/N} - 1)
\]

In this paper the total utilization consists of base utilization \( \mu^\text{base} \) and security utilization \( \mu^\text{sec} \). After substituting \( U_{\text{sys}} \) by \( \mu^\text{base} + \mu^\text{sec} \), we obtain the sufficient feasibility condition for security-sensitive disk requests.

Theorem 3 shows a low-cost but efficient approach to check the feasibility of a set of periodic disk requests. Based on Theorem 3, we are in a position to present a scheduling algorithm for periodic disk requests with security constraints. To maximize security profit of a group of disk requests, SAPW judiciously allocates the slack time to the most suitable security service for these security-critical disk requests. Since distributing slack time to the disk requests with high weight, large data size and low security overhead leads to high security profit, we define a slack-security factor \( \delta_i \) to measure the decrease of slack time by unit security improvement for disk request \( R_i \),

\[
\delta_i = \frac{l_i(f(s_i + \Delta s_i) - f(s_i))}{w_i \Delta s_i / l_i} = \frac{f(s_i + \Delta s_i) - f(s_i)}{w_i / \Delta s_i}
\]

where \( \Delta s_i \) is the increase of security level for request \( R_i \).

Security-aware periodic-write (SAPW) aims at maximizing system security quality while not violating temporal constraints. The main idea of SAPW is to iteratively raise the security level of the disk request with minimal slack-security factor, provided that the schedulable condition is satisfied. SAPW algorithm is illustrated in Fig. 3. In the first step, we calculate the spare utilization, and check whether disk requests set \( R \) can be schedulable without considering security factor. In step 2, SAPW initiates security level of each request at the minimal security requirements. In step 3, the core of SAPW strives to iteratively find out the request with minimal slack-security factor and raises its security level. If security level of any request is equal to its maximal security requirement, SAPW ignores that request in next round. When the security utilization is not less than spare utilization, it comes to the terminal condition. After selecting the most suitable security levels for all disk requests, step 4 (line 18) generates schedulable table for each instance of \( R \) based on RMS policy.
Step 1:
Calculate Spare Utilization (SU) for security improving

\[ SU = N(2^{|S|} - 1) - \mu_{\text{fired}} \]
\[ = N(2^{|S|} - 1) - \sum_{i=1}^{|S|} T_{\text{seek}} + T_{\text{rotate}} + T_{\text{read}}/p_i \]

If \( SU < 0 \)
then disk requests of \( R \) are not schedulable
EndIf.
Step 2:
For \( i=1 \) to \( N \) do
\[ s_i = s_{\text{max}}(i) \]
EndFor.
Step 3:
While \( \mu_{\text{fired}} = \sum_{i=1}^{|S|} l_{i}/p_i \leq SU \) do
Find \( s_i \) from \( S = \{s_1, s_2, ..., s_N\} \), which subject to
\[ \Delta \delta_i = \min \left\{ f(s_i + \Delta x) - f(s_i), \frac{j}{w_i + \Delta s} \right\}, \text{for } 1 \leq i \leq N \]
\[ s_i = s_i + \Delta x \]
If \( s_i > s_{\text{max}}(i) \)
then \( S = S - \{s_i\} \)
EndIf.
\[ \mu_{\text{fired}} = \sum_{i=1}^{|S|} l_{i}/p_i \]
//Renew security utilization
EndWhile.

Fig. 3. Description of SAPW.

Theorem 4. Time complexity of SAPW algorithm is \( O(bN) \), where \( b \) is the number of “While” iterations.

Proof. The “For” operation takes \( O(N) \) time to initialize the security levels of all disk requests. In “While” operation, it takes \( O(N) \) time to execute “Min” operation. Therefore, the total time complexity of SAPW is \( O(N) + O(b*N) = O(bN) \).

4. Simulations and Analysis

To quantitatively evaluate our algorithm through extensive simulations with synthetic real-time disk requests, we used MATLAB7.0 to simulate a local disk system. For comparison, in addition to SAPW proposed in this work, we implemented three variant versions of RMS scheme, which are referred to as RMS-ORG, RMS-MAX and RMS-AVG. RMS-ORG means the original strategy which allocates minimal security quality. RMS-MAX and RMS-AVG choose the max security level and average security level within their security range, respectively. Important simulated disk parameters are presented in Table 1, which are similar to IBM Ultrastar 36Z15 [7].

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk capacity</td>
<td>18.4 GB</td>
</tr>
<tr>
<td>Revolutions per minute</td>
<td>15000</td>
</tr>
<tr>
<td>Seek time</td>
<td>6.0 ms</td>
</tr>
<tr>
<td>Rotational time</td>
<td>4.02 ms</td>
</tr>
<tr>
<td>Disk bandwidth</td>
<td>20 MB/s</td>
</tr>
</tbody>
</table>

Table 2: Disk requests with various data sizes

<table>
<thead>
<tr>
<th>Requests</th>
<th>( l ) (KB)</th>
<th>( p_i ) (ms)</th>
<th>( s_{\text{read}}(l) )</th>
<th>( s_{\text{read}}(l) )</th>
<th>( w_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_1 )</td>
<td>[200, 500]</td>
<td>80</td>
<td>1</td>
<td>9</td>
<td>0.1</td>
</tr>
<tr>
<td>( R_2 )</td>
<td>[300, 600]</td>
<td>90</td>
<td>1</td>
<td>9</td>
<td>0.4</td>
</tr>
<tr>
<td>( R_3 )</td>
<td>[400, 700]</td>
<td>120</td>
<td>2</td>
<td>7</td>
<td>0.1</td>
</tr>
<tr>
<td>( R_4 )</td>
<td>[500, 800]</td>
<td>100</td>
<td>3</td>
<td>9</td>
<td>0.2</td>
</tr>
<tr>
<td>( R_5 )</td>
<td>[600, 900]</td>
<td>110</td>
<td>1</td>
<td>5</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Table 3: Disk requests with various deadlines

<table>
<thead>
<tr>
<th>Requests</th>
<th>( l ) (KB)</th>
<th>( p_i ) (ms)</th>
<th>( s_{\text{read}}(l) )</th>
<th>( s_{\text{read}}(l) )</th>
<th>( w_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_1 )</td>
<td>500</td>
<td>[50, 150]</td>
<td>1</td>
<td>9</td>
<td>0.2</td>
</tr>
<tr>
<td>( R_2 )</td>
<td>600</td>
<td>[55, 155]</td>
<td>1</td>
<td>9</td>
<td>0.5</td>
</tr>
<tr>
<td>( R_3 )</td>
<td>700</td>
<td>[70, 170]</td>
<td>2</td>
<td>7</td>
<td>0.1</td>
</tr>
<tr>
<td>( R_4 )</td>
<td>800</td>
<td>[60, 160]</td>
<td>3</td>
<td>9</td>
<td>0.3</td>
</tr>
<tr>
<td>( R_5 )</td>
<td>400</td>
<td>[65, 165]</td>
<td>1</td>
<td>5</td>
<td>0.1</td>
</tr>
</tbody>
</table>

We conducted two groups of simulations to evaluate the performance improvement of SAPW over the others. In the first group simulations, we studied the impact of the data size on security improvement. There are five periodic users demanding to periodically store security data in the local disk. Each user’s disk request has different period, security requirement, and security weight. In particular, the data sizes of all users are adjusted in a way that the mean data size is set to 500 KB, 600 KB, 700 KB and 800 KB. Detailed information is presented in Table 2. In the second group simulations, we investigated the security enhancement under various deadlines. We also chose five periodic disk requests. The period of each disk request is properly selected so that the average period varies from 60 ms to 160 ms, while other parameters are kept unchanged. Table 3 shows the details.

Results of the first group simulations are illustrated in Fig. 4 and Fig. 5. Several important observations are discovered from Fig. 4. First, we can see that SAPW outperforms over other three algorithms greatly, because it always obtains the highest security value. Specifically, SAPW achieves security profit improvement over RMS-ORG and RMS-AVG by averages of 223.6% and 50.1%, respectively. Second, SAPW and RMS-MAX both outperforms over other three algorithms greatly, because it discovered from Fig. 4. First, we can see that SAPW achieves security profit improvement over RMS-ORG scheme are fixed. Larger data size brings larger security value. Security levels of SAPW are changeable under different data sizes. In Fig. 5, security overhead of SAPW is decreasing with the increasing average data size, while overheads of other algorithms are gradually increased. This is because larger data size results less spare time for SAPW.
In Fig. 6, we can see security improvements with the impact of various mean periods of disk requests. Clearly, SAPW obtains the largest security performance improvement among the four algorithms. For instance, the security enhancement of SAPW over RMS-ORG and RMS-AVG is by average of 208.5% and 49.6%, respectively. When average period is small, like 300 ms, SAPW and RMS-ORG can only satisfy the least security demand while RMS-MAX and RMS-AVG can not be schedulable. With the increasing of the average period, RMS-AVG becomes feasible at 400 ms case. When the periods of all requests are larger enough, RMS-MAX becomes schedulable and gets the same security improvement as SAPW. This is because larger period, more spare time is left for improving security value of every user’s disk request. Fig. 7 shows the security overhead of each algorithm under various periods. The security overhead of SAPW is increased with larger period. The other algorithms’ overheads are fixed if they are schedulable, like RMS-ORG.

5. Conclusion and Future Work

In this paper, the disk scheduling issues of embedded storage system with both time-critical and security-critical constraints are considered. For the situation of multi-user and periodic disk requests, SAPW is proposed to schedule security-sensitive periodic write requests in a way to maximize the security profit. SAPW is a static scheduling policy based on RMS policy. Compared with other three algorithms in extensive simulations, SAPW shows its effectiveness of improving overall security profit of real-time disk write requests while not sacrificing the timing requirement of each request.

In the future work, we will consider security-aware schemes for multi-disk storage system. Considering the seek time of disk is not fixed, a dynamic policy will also be studied to efficiently utilize the slack time.

References


