Journal of Institute of Control, Robotics and Systems (2011) 17(11):1101-1105
http://dx.doi.org/10.5302/J.ICROS.2011.17.11.1101 ISSN:1976-5622 eISSN:2233-4335

자가 복구 오류 검출 및 정정 회로 적용을 고려한 최적 스크러빙 방안

An Optimal Scrubbing Scheme for Auto Error Detection & Correction Logic

류 상 문*
(Sang-Moon Ryu[1])
[1]Kunsan National University

Abstract: Radiation particles can introduce temporary errors in memory systems. To protect against these errors, so-called soft errors, error detection and correcting codes are used. In addition, scrubbing is applied which is a fundamental technique to avoid the accumulation of soft errors. This paper introduces an optimal scrubbing scheme, which is suitable for a system with auto error detection and correction logic. An auto error detection and correction logic can correct soft errors without CPU's writing operation. The proposed scrubbing scheme leads to maximum reliability by considering both allowable scrubbing load and the periodic accesses to memory by the tasks running in the system.

Keywords: fault tolerance, soft error, reliability improvement, memory scrubbing

I. 서론

우주 공간에서 동작하는 컴퓨터 시스템에서 발견된 현상으로, 고에너지 입자가 반도체 부품을 통과할 때 생성되는 전자-정공 쌍에 의해서 저장되어 있던 정보의 값이 반전되는 현상을 SEU (Single Event Upset) [1,2]라고 한다. 항공기와 인공위성과 같은 항공 우주 시스템 및 원자력 발전소와 같은 시스템에서는 SEU에 대한 대응이 필수적이다.

SEU의 영향에 가장 많이 노출되는 전자 부품은 메모리 소자이다. 요구되는 집적도가 높아지고 저전력 설계로 인해 동작 전압이 낮아지면서 그 영향이 지속적으로 증가하고 있고, 항공기 운항 고도 및 지상에서도 SEU 현상이 발견되고 있다 [1,3,4].

SEU 현상에 의해 발생한 오류는 물리적인 손상에 기인한 것이 아니기 때문에 올바른 정보를 다시 기입하여 주면 정정된다. 이러한 오류를 소프트 에러라고 한다. 메모리의 소프트 에러 극복을 위한 방안으로 사용되는 것이 오류 검출 및 정정(EDAC: Error Detection & Correction) 회로[5]를 이용하는 것이다. 그림 1처럼 CPU는 오류 검출 및 정정 회로를 통하여 메모리에 접근한다. CPU가 메모리에 정보를 저장할 때는 추가의 정보(패리티)가 함께 저장되고, 정보를 읽을 때에는 이 추가 정보를 함께 읽어 들여 소프트 에러 발생 여부를 검출하고 일정 범위 이내에서 소프트 에러를 정정할 수 있다.

그리고 소프트 에러의 발생 정도가 적용된 오류 검출 및 정정 회로의 정정 범위 이내에서 유지되도록 메모리의 모든 워드에 대해 스크러빙(scrubbing) [1,6-8]을 수행해 준다. 스크러빙은 CPU가 주기적으로 메모리의 모든 워드를 읽고 다시 기입하는 동작으로, 메모리의 정보를 처리할 목적으로 메모리에 접근하는 것이 아니고 메모리에 소프트 에러가 발생했는지 확인하고 발생했으면 정정하기 위해 수행되는 것이다.

소프트 에러의 검출 및 정정은 CPU가 메모리를 읽는 과정에서 오류 검출 및 정정 회로에 의해 이루어지며 소프트 에러가 정정된 정보는 CPU가 읽은 정보를 다시 메모리에 기입할 때 패리티 정보와 함께 저장된다. 스크러빙을 목적으로 메모리에 접근하는 것은 시스템에 요구되는 작업을 수행하는 것이 아니기 때문에 시스템의 성능을 저하시키는 요인이 된다.

메모리에 저장되어 있는 정보는 메모리 소자의 물리적인 특성을 배제하면 오류 검출 및 정정 회로의 정정 능력과 스크러빙 주기로 결정된다. 오류 검출 및 정정 회로의 정정 능력을 키우려면 관련 회로의 복잡도가 증가하고 정정 소요 시간도 증가한다. 그리고 본 정보 이외에 추가로 저장되어야 하는 패리티 정보의 양도 증가되어 메모리 요구량이 증가된다.

스크러빙의 주기가 짧을수록, 즉 빈번하게 스크러빙을 수행할 수록 정정 범위를 벗어나는 소프트 에러의 누적을 방지할 수 있다. 하지만 빈번한 스크러빙은 CPU 및 버스 자원을 많이 사용하여 시스템의 성능을 떨어뜨리게 된다. 따라서 주어진 오류 검출 및 정정 회로에 대해 시스템의 성능을 고려하면서 신뢰도를 최대로 할 수 있는 스크러빙 주기를 적용하여야 하며 이와 관련된 연구가 진행되었다[9-11].

그림 1의 경우는 CPU가 메모리의 정보를 읽고 다시 쓸 때 오류가 정정된다. 그림 2처럼 CPU가 메모리의 정보를 읽을 때 소프트 오류가 검출되면 CPU의 개입 없이 오류 검출 및 정정 회로 스스로 오류가 제거된 정보를 메모리에 기입하도록 구성할 수 있다. 본 논문에서는 이런 기능이 있는 회로를 '자가 복구 오류 검출 및 정정 회로(Auto Error Detection & Correction logic)'라고 부르기로 한다.

* 책임저자(Corresponding Author)
논문접수: 2011. 8. 20., 수정: 2011. 9. 5., 채택확정: 2011. 9. 25.
류상문: 군산대학교 제어로봇공학과(smryu@kunsan.ac.kr)

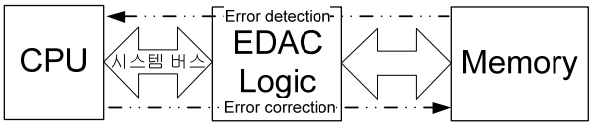

그림 1. 오류 검출 및 정정 회로.
Fig. 1. Error detection & correction logic.

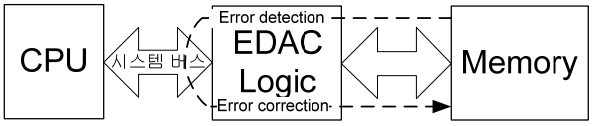

그림 2. 자가 복구 오류 검출 및 정정 회로.
Fig. 2. Auto error detection & correction logic.

자가 복구 오류 검출 및 정정 회로가 적용되면 오류가 제거된 정보를 메모리에 기입하기 위해 CPU가 메모리를 대상으로 쓰기 동작을 수행할 필요가 없기 때문에 CPU의 스크러빙 로드(scrubbing load)를 경감할 수 있다. 특히 메모리 영역 중에서 항상 읽기 동작만 일어나는 프로그램 코드 영역이나 상수 데이터 영역 등에서는 그 경감 효과가 클 것이다.

그리고 주기적으로 실행되는 태스크에 할당된 메모리 영역은 CPU가 해당 태스크를 수행할 때 자동으로 스크러빙이 이루어지게 된다. 따라서 시스템 내에서 가용한 스크러빙 로드를 비주기적인 태스크의 메모리 영역이나 평소에는 접근되지 않는 메모리 영역을 스크러빙하는 데 사용하면 시스템의 메모리 신뢰도를 높일 수 있을 것이다. 본 논문은 자가 복구 오류 검출 및 정정 회로로 보호되는 메모리를 갖는 컴퓨터 시스템에 적용할 수 있는 메모리 신뢰성을 고려한 최적 스크러빙 기법을 제안한다. 주기적 그리고 비주기적 태스크가 함께 실행되는 상황에서 메모리 시스템의 신뢰도를 최대화하도록 허용된 스크러빙 로드를 활용할 수 있는 방안을 소개한다.

II 장에서는 제안된 방안을 소개하기 위한 시스템 모델과 가정 그리고 이를 바탕으로 한 신뢰도 해석에 대해 설명한다. III 장에서는 신뢰도를 최대화하는 최적 스크러빙 방안을 제안하고, IV 장에서는 제안된 방안의 적용 효과를 보인다.

II. 시스템 모델과 신뢰도 해석

다음은 본 논문에서 사용한 가정과 시스템 모델이다.

1. 시스템에는 n 개의 태스크 $\tau_1, \tau_2, \cdots, \tau_n$ 가 실행 중이다.
2. 태스크 τ_i 가 점유하는 메모리 워드 양은 N_i 이다. 따라서 시스템의 총 메모리 워드는 $\sum_{i=1}^{n} N_i$ 이다.
3. 태스크 τ_i 의 실행 주기는 t_i 이다. 태스크 τ_i 가 주기적으로 실행되지 않고 특정 이벤트 발생 시에만 실행된다면 실행 주기를 ∞ 로 간주한다.
4. 태스크 τ_i 는 실행될 때 마다 점유하는 모든 메모리 워드에 1회 이상 접근한다.
5. CPU는 자가 복구 오류 검출 및 정정 회로를 통하여 메모리에 접근하며, 이때 검출된 소프트 에러는 CPU의 개입 없이 자가 복구 오류 검출 및 정정 회로에 의해 정정된다.
6. 메모리 워드의 크기는 $(w+c)$ 이다. w 는 정보의 저장에 할당된 비트의 수를, c 는 패리티 비트의 수를 의미한다.
7. 적용된 오류 검출 및 정정 회로는 $(w+c)$ 비트 워드에서 발생할 수 있는 1 비트 소프트 에러를 정정할 수 있다
8. 소프트 에러는 평균값 λ 를 갖는 포아송(Poisson) 분포에 따라 발생한다[9-12].

가정 4가 만족되지 않는 어플리케이션이 존재할 수 있다. 각 태스크에서 주기적으로 접근되지 않는 메모리 영역은 실행 주기가 ∞ 인 태스크 영역으로 간주하면 본 논문에서 제안하는 방안을 적용할 수 있다.

이러한 메모리 시스템에 대한 신뢰도를 이전 연구 [10]를 참고해 해석해보면 다음과 같다. 임의의 시간 구간 $[0, t]$ 에서 비트당 일시적 소프트 에러가 발생하지 않을 확률은 $e^{-\lambda t}$ 이며, 소프트 에러가 발생할 확률은 $(1-e^{-\lambda t})$ 이다. 하나의 워드에서 발생한 소프트 에러가 오류 정정 범위 안에 있기 위해서는 $(w+c)$ 비트들 중 한 개 이하의 비트에서만 소프트 에러가 발생하여야 한다. 따라서 한 개의 워드에 대한 신뢰도 함수 $r_o(t)$ 는 식 (1)과 같다.

$$\begin{aligned} r_o(t) &= e^{-\lambda(w+c)t} + (w+c)(1-e^{-\lambda t})e^{-\lambda(w+c-1)t} \\ &\approx 1 - \tfrac{1}{2}(w+c)(w+c-1)\lambda^2 t^2, \quad t \geq 0 \end{aligned} \tag{1}$$

여기에서 $\lambda \ll 1$ 를 이용하여 우세항만을 이용한 근사화를 적용하였다.

편의를 위하여 α 를

$$\alpha = \tfrac{1}{2}(w+c)(w+c-1)\lambda^2 \tag{2}$$

같이 정의하면 $\alpha \ll 1$ 을 만족하며, 식 (1)은 다음과 같이 된다.

$$r_o(t) \approx 1 - \alpha t^2, \quad t \geq 0 \tag{3}$$

태스크 τ_i 의 메모리 점유량은 N_i 이므로 태스크 τ_i 가 사용하는 메모리에 대한 신뢰도 함수 $R_{o,i}(t)$ 는 다음과 같다

$$\begin{aligned} R_{o,i}(t) &= \left(r_o(t)\right)^{N_i} \\ &\approx 1 - \alpha N_i t^2, \quad t \geq 0 \end{aligned} \tag{4}$$

여기에서 $\alpha \ll 1$ 를 이용하여 우세항만을 이용한 근사화를 적용하였다.

식 (4)는 스크러빙이 적용되지 않는 경우에 태스크 τ_i 가 점유하는 메모리 영역에 대한 신뢰도 함수이다. 태스크 τ_i 가 매 t_i 마다 주기적으로 실행되면 매 t_i 마다 스크러빙이 수행되는 것으로 간주할 수 있다. 스크러빙이 적용되면 t_i 마다 정정 가능한 범위 내의 소프트 오류들이 정정되므로 신뢰도 함수는 매 스크러빙 작업이 완료된 시점에서 모든 오류가 정정되고, 최근 스크러빙 작업 이후에 정정 불가능한 오류가 발생하지 않을 확률과 같다. 따라서 스크러빙을 효과를 고려한 태스크 τ_i 의 메모리에 대한 신뢰도 함수 $R_{l,i}(t)$ 는 식 (5)과 같다.

$$\begin{aligned} R_{l,i}(t) &= \left[R_{o,i}(t_i)\right]^l R_{o,i}(\tau) \\ &\approx (1-\alpha N_i t_i^2)^l (1-\alpha N_i \tau^2), \quad t \geq 0 \end{aligned} \tag{5}$$

여기서 $t = t_i l + \tau$ 이고 τ 는 $0 \le \tau < t_i$ 를 만족하는 실수이며 l 은 $l \ge 0$ 를 만족하는 정수이다.

각 태스크 영역의 신뢰도 함수로부터 메모리 시스템의 신뢰도 함수 $R_t(t)$ 를 식 (6)과 같이 구할 수 있다.

$$R_t(t) = \prod_{i=1}^{n} R_{t,i}(t) \tag{6}$$

신뢰도 함수와 함께 신뢰도의 지표로 사용되는 MTTF (Mean-Time-to-Failure)를 구하기 위해 식 (5)를 좀 더 근사화하면 식 (7)과 같다.

$$R_{t,i}(t) \approx (1 - \alpha N_i t_i^2)^{\frac{t}{t_i}}, \quad t \ge 0 \tag{7}$$

식 (5)과 (7)는 t 가 t_i 의 정수배에 해당할 때 마다 동일한 결과를 갖는다.

전체 메모리에 대한 $MTTF_t$ 는 그 정의에 따라 식 (6)과 (7)을 이용하여 구하면 다음과 같다.

$$\begin{aligned} MTTF_t &= \int_0^{\infty} R_t(t)dt \\ &\approx -\frac{1}{\sum_{i=1}^{n} \frac{1}{t_i} \ln\left(1 - \alpha N_i t_i^2\right)} \end{aligned} \tag{8}$$

III. 최적 스크러빙 방안

이 장에서는 II 장의 결과를 바탕으로 시스템에서 허용된 스크러빙 로드를 유지하면서 전체 메모리의 신뢰도를 최대화할 수 있는 스크러빙 방안을 소개한다.

메모리 스크러빙에 따른 부담은 스크러빙 대상 메모리의 총 워드 수에 비례하고 스크러빙 주기에 반비례한다. 따라서 시스템에 요구되는 모든 작업을 수행하면서 전체 메모리를 T_s 의 주기로 스크러빙할 수 있다면 가용한 스크러빙 로드를 다음과 같이 표현할 수 있다.

$$\frac{1}{T_s} \sum_{i=1}^{n} N_i \tag{9}$$

식 (9)로 표현된 시스템의 가용한 스크러빙 로드를 이용해 각 태스크에 할당된 메모리 영역을 추가로 스크러빙하면 전체 메모리의 신뢰도를 개선할 수 있을 것이다. 그리고 가용한 스크러빙 로드를 유지하면서 전체 메모리의 신뢰도를 최대화하는 각 태스크 영역별 최적 스크러빙 주기가 존재할 것이다. 추가의 스크러빙이 적용된 각 태스크별 스크러빙 주기를 $T_i (\le t_i, i = 1,2,\cdots,n)$ 라고 하면 이를 위해 다음과 같은 최적화 문제를 구성할 수 있다.

$$\text{maximize} \quad -\frac{1}{\sum_{i=1}^{n} \frac{1}{T_i} \ln\left(1 - \alpha N_i T_i^2\right)} \tag{10}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \frac{N_i}{T_i} = \sum_{i=1}^{n} \frac{N_i}{t_i} + \frac{1}{T_s} \sum_{i=1}^{n} N_i \tag{11}$$

식 (10)는 식 (8)의 $MTTF_t$ 에서 t_i 를 T_i 로 대체한 것으로 각 태스크의 스크러빙 주기 T_i 가 전체 메모리의 신뢰도를 최대화하는 것을 최적화 목적 함수로 설정한 것이다. 식 (11)의 좌변은 각 태스크를 스크러빙 주기 T_i 로 스크러빙하는 총 스크러빙 로드를 의미하여, 우변은 각 태스크가 각자의 실행 주기 마다 실행되어 자동으로 수행되는 스크러빙 양과 식 (9)의 가용한 스크러빙 로드의 합을 의미한다.

$0 < x \ll 1$ 을 만족하는 x 에 대해 $\ln(1-x) \approx -x$ 이므로 식 (10)의 목적 함수는 그 분모에 $\ln(1-x) \approx -x$ 를 적용하여 다음과 같이 바꿀 수 있다.

$$\text{minimize} \quad \sum_{i=1}^{n} N_i T_i \tag{12}$$

목적 함수 식 (10)로부터 식 (12)를 구하는 과정에서 α 는 상수이므로 제거하였다.

Lagrange 정리[13]에 따르면 다음의 최적화 문제

$$\text{minimize} \quad f(x) \tag{13}$$

$$\text{s.t.} \quad h(x) = 0 \tag{14}$$

에 대해, $h(x) = 0$ 을 만족하고 $f(x)$ 를 최소화하는 국부 최소점(Local minimizer) x^*가 존재한다면,

$$\nabla f(x^*) + \kappa \nabla h(x^*) = 0 \tag{15}$$

를 만족하는 κ 가 존재하게 된다. 여기서 ∇ 는 Gradient이며, κ 를 Lagrange multiplier라고 한다.

식 (12)와 (11)를 각각 식 (13)과 (14)에 대응시킨 후 Lagrange 정리를 적용하면 다음의 연립 방정식이 얻어진다.

$$\begin{aligned} &N_j + \kappa\left(\sum_{i=1, i\ne j}^{n} \frac{N_i}{T_i} - \sum_{i=1}^{n} \frac{N_i}{t_i} - \frac{1}{T_s}\sum_{i=1}^{n} N_i\right) = 0, \quad j = 1,2,\cdots,n \\ &\sum_{i=1}^{n} \frac{N_i}{T_i} - \sum_{i=1}^{n} \frac{N_i}{t_i} - \frac{1}{T_s}\sum_{i=1}^{n} N_i = 0 \end{aligned} \tag{16}$$

식 (16)를 만족하는 $T_i > 0, i = 1,2,\cdots,n$ 를 구하는 것이 가용한 스크러빙 로드를 유지하면서 전체 메모리의 신뢰도를 최대화하는 각 태스크 영역별 최적 스크러빙 주기를 구하는 것이다. 그런데 구해진 태스크 영역별 스크러빙 주기 T_i 중에 $T_i \ge t_i$ 를 만족하는 태스크 τ_i 가 있을 수 있다. 구해진 최적 스크러빙 주기가 태스크의 실행 주기보다 크거나 같다는 것은 이 태스크에 대해 추가적인 스크러빙이 필요 없음을 의미한다. 따라서 만일 식 (16)의 해 T_i 중 $T_i \ge t_i$ 를 만족하는 것이 존재하면 T_i 를 $T_i = t_i$ 로 대입하여 식 (16)를 다시 구성하고, 해 T_i 중 $T_i \ge t_i$ 를 만족하는 것이 없을 때까지 이 과정을 반복한다.

$T_i < t_i$ 를 만족하는 태스크 τ_i 에 대해서는 메모리 시스템의 신뢰도 향상을 위해서 추가의 스크러빙이 필요하다는 것을 의미한다. 태스크 τ_i 를 위한 추가 스크러빙 수행 주기를 e_i 라고 하면, T_i 와 t_i 그리고 e_i 의 관계는 식 (17)로 표현되며, 이로부터 e_i 는 식 (18)과 같이 구할 수 있다.

$$\frac{N_i}{T_i} = \frac{N_i}{t_i} + \frac{N_i}{e_i} \tag{17}$$

$$e_i = \frac{T_i t_i}{t_i - T_i} \tag{18}$$

메모리 시스템의 신뢰도를 최대화하는 스크러빙 주기 $T_i, i = 1,2,\cdots,n$ 를 적용했을 때 얻어지는 각 태스크의 메모리 영역별 신뢰도 함수 $R_{T,i}(t)$ 는 식 (7)에서 t_i 를 T_i 로 대체하면 식 (19)와 같다.

$$R_{T,i}(t) \approx (1-\alpha N_i T_i^2)^{\frac{t}{T_i}}, \quad t \geq 0 \tag{19}$$

그리고 메모리 시스템의 신뢰도 함수 $R_T(t)$를 식 (20)과 같이 구할 수 있다.

$$R_T(t) = \prod_{i=1}^{n} R_{T,i}(t) \tag{20}$$

그리고 메모리 시스템의 $MTTF_T$ 는 식 (21)와 같다.

$$\begin{aligned} MTTF_T &= \int_0^\infty R_T(t)dt \\ &\approx -\frac{1}{\sum_{i=1}^{n} \frac{1}{T_i} \ln\left(1-\alpha N_i T_i^2\right)} \end{aligned} \tag{21}$$

IV. 적용 효과

표 1과 같은 실행 주기가 1, 5, 10인 3개의 주기적 태스크와 1 개의 비주기적 태스크가 실행되는 상황을 가정한다. 비주기적인 태스크 4의 주기는 ∞로 표현한다. 실행 주기와 스크러빙 주기 설정에 따른 효과를 비교하기 위해서 태스크에 할당된 메모리 워드의 양는 동일하게 10^5로 가정한다. 각 워드의 크기는 7 비트 ($w=4$, $c=3$)이며, 워드에서 발생하는 1 비트 오류를 정정할 수 있는 자가 오류 검출 및 정정 회로를 가정한다. 소프트 에러 발생률 $\lambda = 2\times10^{-8}$ [bit/sec]를 가정하고, 시스템에서는 모든 워드를 10초(T_s=10)의 주기로 스크러빙할 수 있는 성능 여유가 있다고 가정한다.

스크러빙 주기에 따른 효과 비교를 위하여 3가지의 경우를 고려한다.

- Case 1: 가용한 스크러빙 로드의 여유, 즉 10초 주기로 모든 워드를 스크러빙할 수 있는 여력을 모두 비주기적인 태스크 4 영역을 스크러빙하는데 사용한다. 4×10^5워드를 10초 주기로 스크러빙하는 것은 10^5 워드를 2.5초 주기로 스크러빙하는 것과 동일하기 때문에 태스크 4의 스크러빙 주기는 2.5가 된다. 나머지 태스크들은 추가의 스크러빙을 수행하지 않으므로 태스크의 실행 주기가 스크러빙 주기가 된다. 각 태스크의 스크러빙 주기와 추가 스크러빙 주기가 표 2에 있다.

표 1. 가정된 시스템.
Table 1. System parameter assumption.

Task #	Period [sec]	# of Words
1	1	10^5
2	5	10^5
3	10	10^5
4	∞	10^5

표 2. Case 1: 가용한 스크러빙 로드가 태스크 4의 스크러빙에 할당된 경우.
Table 2. Case 1 in which the allowable scrubbing load is allocated to the scrubbing for task 4.

Task #	Scrubbing period [sec]	Extra scrubbing period [sec]
1	1	0
2	5	0
3	10	0
4	2.5	2.5

표 3. Case 2: 가용한 스크러빙 로드가 태스크 2, 3, 4의 스크러빙에 균등하게 할당된 경우.
Table 3. Case 2 in which the allowable scrubbing load is evenly distributed to the scrubbing for task 2, 3 and 4.

Task #	Scrubbing period [sec]	Extra scrubbing period [sec]
1	1	0
2	3	7.5
3	4.3	7.5
4	7.5	7.5

표 4. Case 3: 제안된 최적 스크러빙 방안 적용.
Table 4. Case 3 in which the proposed optimal scrubbing scheme is applied.

Task #	Scrubbing period [sec]	Extra scrubbing period [sec]
1	1	0
2	4.3	30.7
3	4.3	7.5
4	4.3	4.3

- Case 2: 가용한 스크러빙 로드의 여유를 태스크 2, 3, 4의 영역을 추가 스크러빙하는데 동일하게 분배한다. 4×10^5 워드를 10초 주기로 스크러빙하는 것은 10^5 워드를 차지하는 3개의 태스크를 각각 7.5초 주기로 스크러빙하는 것과 동일하기 때문에 태스크 2, 3, 4의 추가 스크러빙 주기는 7.5가 된다. 그러면 각 태스크는 자체 실행 주기를 고려하여 실제 스크러빙 주기는 각각 3, 4.3, 7.5초가 된다. 각 태스크의 스크러빙 주기와 추가 스크러빙 주기가 표 3에 있다.
- Case 3: 본 논문에서 제안한 최적화 방안을 적용한 경우로 태스크 1, 2, 3, 4의 영역을 각각 1, 4.3, 4.3, 4.3초로 스크러빙한다. 태스크 1은 추가 스크러빙이 필요 없으며, 4×10^5 워드를 10초 주기로 스크러빙하는 로드를 태스크 2, 3, 4을 각각 30.7, 7.5, 4.3초 추가 스크러빙에 할당한다. 각 태스크의 스크러빙 주기와 추가 스크러빙 주기가 표 4에 있다.

그림 3과 표 5는 각 경우에 대한 메모리 시스템의 신뢰도 함수와 MTTF를 보여 준다. Case 1이 신뢰도 측면에서 가장 좋지 않다. 메모리 시스템 전체의 신뢰도를 고려하지 않고 가용한 스크러빙 로드를 한 개의 비주기적 태스크 영역 스크러빙에 사용했기 때문이다. Case 2는 가용한 스크러빙 로드를 실행 주기가 가장 짧은 태스크를 제외한 나머지 태스크 영역의 추가 스크러빙에 균등 배분하여 Case 1 보다는 메모리 시스템 전체의 신뢰도를 고려하였다. MTTF가 17% 정도 개선되었다. 가용한 스크러빙 로드 여유를 제안된 방안에 따라 최적화해서 각 태스크 영역 스크러빙에 할당한 Case 3은 Case 1 대비 33%, Case 2 대비 14% 개선된 MTTF를 얻을 수 있다.

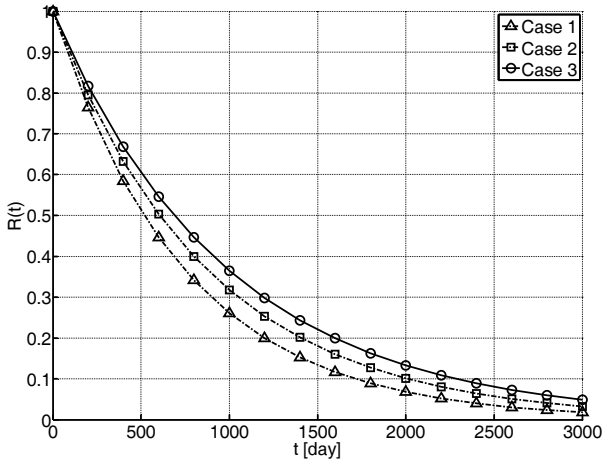

그림 3. 신뢰도 함수의 그래프.
Fig. 3. Graphs of reliability function.

표 5. MTTF.
Table 5. MTTF.

	Case 1	Case 2	Case 3
MTTF [day]	745	873	991

V. 결론

소프트 에러가 발생할 수 있는 환경에서 동작하는 컴퓨터 시스템은 요구되는 신뢰도 수준을 만족하기 위해 설계, 제작, 시험 단계에서 다양한 사항이 고려되어야 한다. 특히 소프트 에러에 매우 취약한 CPU의 레지스터나 메모리 소자에는 소프트 에러 정정 기능이 필수적으로 요구된다. 그리고 소프트 에러의 누적을 방지하기 위하여 메모리 소자에는 전 영역에 걸쳐서 주기적인 스크러빙을 수행하여 준다.

본 논문은 CPU가 메모리의 정보를 읽을 때 소프트 오류가 검출되면 CPU의 개입 없이 오류를 정정할 수 있는 자가 복구 오류 검출 및 정정 회로에 의해 보호되는 메모리 시스템에 적용할 수 있는 최적 스크러빙 방안을 제안하였다. 제안된 방안은 메모리를 점유하는 태스크의 실행 주기를 고려하여 메모리 시스템의 신뢰도가 최대가 되도록 가용한 스크러빙 로드를 태스크 영역별로 할당하는 것이다. 비주기적인 태스크에 의해 점유되는 메모리 영역이나 주기적인 태스크에 의해 점유되지만 주기적으로 접근되지 않는 메모리 영역은 그 주기를 ∞라고 간주하고 제안된 방안을 적용할 수 있다.

참고문헌

[1] S. Karp and B. K. Gilbert, “Digital system design in the presence of single event upsets,” *IEEE Trans. Aerospace and Electronic Systems*, vol. 29, no. 2, pp. 310-316, Apr. 1993.

[2] R. Harboe-Sorensen, E. Daly, F. Teston, H. Schweitzer, R. Nartallo, P. Perol, F. Vandenbussche, H. Dzitko, and J. Cretolle, “Observation and analysis of single event effects on-board the SOHO satellite,” *IEEE Trans. Nuclear Science*, vol. 49, no. 3, pp. 1345-1350, Jun. 2002.

[3] A. Taber and E. Normand, “Single event upset in avionics,” *IEEE Trans. Nuclear Science*, vol. 40, no. 2, pp. 120-126, Apr. 1993.

[4] E. Normand, “Single event upset at ground level,” *IEEE Trans. Nuclear Science*, vol. 43, no. 6, pp. 2742-2750, Dec. 1996.

[5] R. Morelos-Zaragoza, *The Art of Error Correcting Coding*, Wiley, 2002.

[6] A. M. Saleh, J. J. Serrano, and J. H. Patel, “Reliability of scrubbing recovery-techniques for RAMs,” *IEEE Trans. Reliability*, vol. 39, no. 1, pp. 114-122, Apr. 1990.

[7] G. C. Yang, “Reliability of semiconductor RAMs with softerror scrubbing techniques,” *IEE Proc. in Computers and Digital Techniques*, vol. 142, pp. 337-344, Sep. 1995.

[8] R. M. Goodman and M. Sayano, “The reliability of semiconductor RAM memories with on-chip error-correction coding,” *IEEE Trans. Information Theory*, vol. 37, no. 3, pp. 884-896, May. 1991.

[9] P. Reviriego, J. A. Maestro, and S. H. Baeg, “Optimizing scrubbing sequences for advanced computer memories,” *IEEE Trans. Device and Materials Reliability*, vol. 10, no. 2, pp. 192-200, Jun. 2010.

[10] S.-M. Ryu and D.-J. Park, “Transient bit error recovery scheme for ROM-based embedded systems,” *IEICE Trans. Information and System*, vol. EE88-D, no. 9, pp. 2209-2212, Sep. 2005.

[11] S. Baeg, S. Wen, and R. Wong, “Minimizing soft errors in TCAM devices: a probabilistic approach to determining scrubbing intervals,” *IEEE Trans. Circuits and Systems*, vol. 57, no. 4, pp. 814-822, Apr. 2010.

[12] Z. Ming, X. L. Yi, L. Chang, and Z. J. Wei, “Reliability of memories protected by multibit error correction codes against MBUs,” *IEEE Trans. Nuclear Science*, vol. 58, no. 1, pp. 289-295, Feb. 2011.

[13] E. K. P. Chong and S. H. Zak, *An Introduction to Optimization*, John Wiley& Sons, 2001.

류 상 문

1992년 금오공과대학교 전자공학과 졸업. 1995년 한국과학기술원 전기및전자공학과 석사. 2006년 동 대학원 전자전산학과 박사. 1995년~2000년 LG전자(주). 2000년~2004년 한국과학기술원. 2006년~현재 군산대학교 제어로봇공학과 조교수. 관심분야는 임베디드 제어 시스템, 실시간 제어 시스템, 결함허용 임베디드 시스템, 스페이스와이어 네트워크.