실시간 임베디드 시스템의 결함 허용성 개선을 위한 정적 체크포인팅 방안

Fault-Tolerance Improvement of Real-Time Embedded System using Static Checkpointing

유 상 문*
(Sang-Moon Ryu)

Abstract : This paper deals with a scheme for fault-tolerance improvement of real-time embedded systems, which engages an equidistant checkpointing technique to tolerate transient errors. Transient errors are caused by transient faults which are the most significant type of fault in reliable computer systems. Transient faults are assumed to occur according to a Poisson process and to be detected in a non-concurrent manner (e.g., checked periodically). The probability of the successful real-time task completion in the presence of transient errors is derived with the consideration of the possible effects of the transient errors. Based on this, a condition under which inserting checkpoints improves the fault-tolerance of the system is introduced and an optimal equidistant checkpointing strategy that achieves the highest fault tolerance is presented.

Keywords : real-time embedded system, fault-tolerance, checkpointing

I. 서론

실시간 임베디드 시스템은 종종 열악하고 급변하는 환경 속에서 중단 없이 오랜 시간 지속적으로 동작하도록 요구되며 높은 안정성과 시간 제약 조건을 만족하여야 한다. 특히 안전중요(safety critical) 시스템의 경우 시스템 오동작 시 그 피해가 막대하므로 신뢰성이 매우 강조된다.

Single Event Upsets(SEU)[1-3]나 전자파간섭(EMI)등의 환경적 요인으로 발생할 수 있는 일시적 결함(transient faults)[4]은 임베디드 시스템 하드웨어에서 가장 발생 빈도가 높은 유형의 결함이며[5], 하드웨어가 물리적으로 손상되는 것이 아니므로 원천적으로 회피하거나 수리할 수 없다. 일시적 결함은 일시적 오류(transient errors)를 유발하며 반도체 소자의 집적도의 증가에 따른 정보의 표현에 사용되는 전하의 양의 감소, 사용 전압 감소, 동작 속도 증대 등에 따라 일시적 결함에 의한 일시적 오류의 발생 빈도도 지속적으로 증가할 것으로 예상된다.

일시적 결함의 영향을 극복하기 위한 결함허용(fault-tolerance) 기법으로 체크포인팅(checkpointing) 기법[6]이 종종 적용되는데, 이것은 임베디드 시스템에서 실행되는 태스크(task)의 상태를 안정성이 높은 저장 장치에 주기적으로 저장하는 것을 의미한다. 태스크의 상태는 주요 변수의 값, 주요 레지스터의 값, 스택 영역에 저장된 정보 등으로 정의될 수 있다. 저장된 태스크의 상태를 체크포인트(checkpoint)라고 하며 일시적 오류가 감지 되었을 때 최근에 생성된 체크포인트를 이용하여 태스크를 오류 발생 이전, 체크포인트를 저장하기 직전의 상태로 되돌리는 것을 롤백 복구(rollback recovery)라 한다.

* 책임저자(Corresponding Author)
논문접수 : 2007. 9. 29., 채택확정 : 2007. 10. 26.
유상문 : 군산대학교 전자정보공학부(smryu@kunsan.ac.kr)
※ 본 연구는 과학기술부의 과학기술위성 3호 개발 사업의 지원을 받아 수행되었음.

체크포인팅에 관한 이전 연구는 데이터베이스 시스템이나 범용 컴퓨터 시스템에서의 가용성(availability) 최대화, 시스템 성능 예측, 체크포인팅 부담 최소화 등이 주요 주제였으며, 실시간 시스템에 대해서는 태스크 평균 실행 시간 최소화[7,8], 복구 동작의 수행 횟수가 제한된 상황에서 태스크의 실행 성공 확률 최대화[9], 체크포인팅 수행에 따른 태스크의 응답 시간 분석[10], 결함 허용 시스템의 에너지 소모 최소화[11,12] 등에 관한 연구가 수행되었다.

본 논문에서는 정적 체크포인팅 기법을 이용한 실시간 임베디드 시스템의 결함 허용성 최대화 방안에 대해 다룬다. 실시간 태스크에게 주어진 여유 시간을 최대한 활용하여 결함이 발생하는 상황에서도 실시간 태스크의 성공적 실행 확률을 최대화 할 수 있는 체크포인팅 기법을 소개한다. 2장에서는 고려 대상 시스템의 모델과 가정에 대해 기술하고, 3장에서는 실시간 임베디드 시스템이 체크포인팅 기법을 이용하여 일시적 오류를 극복하고 시간 제약을 만족할 수 있는 확률을 해석적으로 구하며, 4장에서는 3장의 해석의 결과를 고찰하고 이를 모의 실험 결과와 비교한다. 5장에서는 결함 허용 성능을 개선할 수 있는 체크포인팅 수행 조건과 최적 방안에 대해 소개하고 6장에서 결론을 맺는다.

II. 시스템 모델과 가정

실시간 임베디드 시스템에서 실행되는 주기적 실시간 태스크의 매 실행마다 일정 횟수의 체크포인팅 작업이 동일 간격으로 실행되는 정적 체크포인팅 기법이 적용되는 상황을 고려한다. 일시적 결함은 시스템 내부에 표현되는 정보에 일시적 오류를 만들어 내고 이것은 복구 작업을 통하여 그 영향이 사라지게 된다.

그림 1은 실시간 태스크의 1회 실행에 4회의 체크포인팅 작업이 수행되는 예를 보여준다. 그림에서 두 번째 체크포인팅 작업 완료 후 오류가 발생했고 세 번째 체크포인팅 작업 수행 시점에 오류가 검출되어 체크포인팅 작업 대신에 두 번

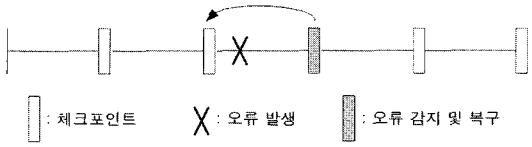


그림 1. 체크포인팅 기법이 적용되는 실시간 태스크.
Fig. 1. A real-time task with checkpointing and rollback recovery.

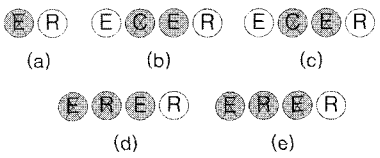


그림 2. 실행 구간, 체크포인팅 구간, 복구 구간의 조합.
Fig. 2. Possible combination of execution, checkpointing and rollback recovery.

째 체크포인팅이 완료된 시점으로 시스템의 상태가 복구되고, 세 번째와 마지막 네 번째 체크포인팅이 수행되어 태스크의 실행이 완료되는 상황을 보여준다.

실시간 태스크의 실행 중에 일정 간격으로 체크포인팅 작업이 수행되면 태스크는 수행되는 체크포인팅 작업의 수와 동일한 실행 구간으로 나뉘게 되고, 실시간 태스크가 일시적 오류를 극복하는 형태는 그림 2에 그려진 실행 구간, 체크포인팅 구간 그리고 복구 구간의 결합으로 나타낼 수 있다.

그림 2에서 E, C, R은 각각 실행 구간, 체크포인팅 구간, 복구 구간을 나타내며, 흰색으로 그려진 구간은 일시적 오류의 영향 없이 정상적으로 수행되었음을 의미하고, 회색으로 그려진 구간은 직접 또는 간접적으로 일시적 오류의 영향을 받아 정상적으로 수행되지 못했음을 의미한다. 회색으로 표시된 구간의 ×는 해당 구간에서 일시적 오류가 발생해 오류의 직접적인 영향을 받았음을 나타내며, (b)와 (d)의 × 없이 회색으로 표시된 실행 구간은 이전의 체크포인팅 작업이나 복구 작업이 정상적으로 수행되지 않아 간접적으로 오류에 영향을 받았음을 의미한다.

일시적 결함에 따른 일시적 오류는 평균값 λ를 갖는 포아송(poisson) 분포에 따라 발생한다고 가정한다. 이것은 일시적 결함 및 오류를 다루는 연구에서 대부분 사용된 가정이다[4,7-10,12,19]. 그리고 λ의 실제 값은 매우 작으므로 실시간 태스크의 1회 수행 당 발생하는 일시적 오류의 평균 발생 횟수는 1보다 매우 작다고 가정한다[1-4].

오류의 극복을 위해서는 그 검출이 우선되어야 한다. 일시적 오류는 체크포인팅 생성 전에 수행되는 오류 검출 기법에 의해 검출되는 상황을 가정한다. 알고리즘 기반 결함 허용(algorithm-based fault tolerance) 기법[13], 소프트웨어로 구현된 오류 검출 및 정정 코드(error detection and correction code) [14], 수락 시험(acceptance test)[15-17], 외부 검사(external check)[18], WDT(Watchdog timer), 이중화 구조 시스템의 주기적 출력 비교 등이 검출 기법의 예가 될 수 있다.

오류 검출 기법은 시스템의 특성이나 요구 사항 등에 따라 다양한 방법으로 다양한 레벨로 구현될 수 있으며, 완벽한 오류 검출을 보장하는 기법은 없다. 본 논문에서는 성능 분석을 위해 완벽한 오류 검출 상황을 가정한다. 이러한 가정을 바탕으로 구한 결과에 시스템에 적용된 오류 검출 기법의 검출 범위(detection coverage) [19]를 적용할 수 있다.

체크포인팅 작업 중에도 발생할 수 있는 일시적 오류에 대응하기 위해서는 두 개의 체크포인트 저장공간을 구비하고 이들을 교대로 사용해야 한다. 체크포인팅 작업이 일시적 오류에 의해 정상적으로 수행되지 못했다면 최근의 무결점 상태가 저장되어 있는 다른 체크포인트를 이용하여 롤백 복구를 수행해야 한다. 그리고 체크포인트 저장을 위한 고신뢰 메모리는 오류 검출 및 정정 코드와 추가의 메모리를 결합하여 원하는 정도까지 안정하게 만들 수 있다.

체크포인팅에 따른 성능 해석과 최적 방안 유도를 위해 하나의 주기적 태스크가 실행되는 실시간 임베디드 시스템을 가정했으나, 그 결과는 두 개 이상의 주기적 태스크가 실행되는 시스템에 쉽게 확장할 수 있다.

III. 체크포인팅에 따른 결함 허용 성능

실시간 임베디드 시스템에서 실행되는 태스크가 한 주기에 수행되는 체크포인팅 작업의 횟수에 따라 성공적으로 실행할 확률, 즉 일시적 오류를 극복하고 주어진 시간 제약 조건을 만족하여 완료할 확률, P_s를 구한다. 이를 위해 태스크의 실행 시간(execution time), 여유 시간(slack time), 체크포인팅 작업과 롤백 복구 작업의 소요 시간을 각각 T_e, T_s, T_c라고 한다. 그리고 태스크의 1회 실행에 수행되는 체크포인팅 작업 횟수를 N_c로 한다.

여유 시간은 실시간 태스크에게 주어진 상대적 마감 시간(relative deadline)에서 실행 시간을 뺀 값으로 이 시간적 여유는 일시적 오류의 영향을 극복하기 위해 필수적이다. 그리고 체크포인팅 작업과 롤백 작업은 오류의 검출 메커니즘이 동작된 후 그 결과에 따라 선택되어 수행되며, 태스크의 상태를 저장 또는 복구하는 과정이므로 임베디드 시스템에서는 각각의 소요시간이 동일하다고 본다.

논의의 편리를 위하여 결함 섹션(faulty section)과 무결함 섹션(fault-free section)을 정의한다. 섹션은 태스크의 하나의 실행 구간과 이 실행 구간 다음에 오는 하나의 체크포인팅 구간 또는 하나의 실행 구간과 이 실행 구간 다음에 오는 하나의 롤백 복구 구간으로 정의한다. 결함 섹션은 오류의 영향을 받은 구간이 포함되는 섹션을 의미하며 그림 2의 (a)는 하나의 결함 섹션을 (b)-(e)는 두 개의 결함 섹션의 예를 보여준다. 무결함 섹션은 오류에 영향 받지 않고 정상적으로 수행된 하나의 실행 구간과 그에 이은 체크포인팅 구간으로 정의한다. 일시적 오류는 태스크의 실행 중 결함 섹션을 만들어 내고 발생된 결함 섹션의 길이에 해당하는 시간이 해당 오류에 의해 손실된다.

N_c 회의 체크포인팅 작업이 태스크의 수행 중 동일 간격으로 삽입되면 태스크는 N_c 개의 섹션으로 나뉘어지게 되며, 섹션의 길이는 $\frac{T_e}{N_c}+T_c$ 가 된다. 그리고 N_c 회의 체크포인팅 작업에 소요되는 시간이 N_cT_c 이므로 이를 제외한 알짜 여유 시간은 $T_s - N_cT_c$ 가 되고, 일시적 오류의 극복에 사용될 수 있는 여유 섹션의 수 N_t는 (1)과 같다.

$$N_t = \left\lfloor \frac{T_s - N_c T_c}{\frac{T_e}{N_c} + T_c} \right\rfloor \quad (1)$$

여기에서 $\lfloor \cdot \rfloor$ 는 주어진 값보다 크지 않은 최대의 정수를 의미한다.

2장에서 일시적 오류는 평균값 λ 를 갖는 포아송 분포에 따라 발생한다고 가정했으므로, 태스크의 실행 시간 동안 그리고 N_c 회의 체크포인팅 작업 실행 동안 일시적 오류가 발생하지 않을 확률은 다음과 같다[4,19].

$$e^{-\lambda(T_e + N_c T_c)} \quad (2)$$

태스크가 성공적으로 실행되기 위해서는 일시적 오류가 발생하지 않거나, 일시적 오류가 발생하더라도 결함 섹션의 수가 (1)의 여유 섹션 수 보다 적게 발생해야한다. 태스크의 실행중 k 개의 결함 섹션이 발생할 확률을 $P_f(k)$ 라고 하면, 태스크의 실행 성공 확률 P_s 는 (2)와 (1)의 N_t 를 이용하여 다음과 같이 표현될 수 있다.

$$P_s = e^{-\lambda(T_e + N_c T_c)} \left[1 + \sum_{k=1}^{N_t} P_f(k) \right] \quad (3)$$

태스크의 실행 중 k 개의 결함 섹션들이 발생했다고 가정하고 이들을 추려낸 예가 그림 3에 나타나있다. 이 결함 섹션들은 그림 2에 그려진 가능한 결함 섹션 중 하나의 형태를 가지며, k 번째 결함 섹션이 마지막 결함 섹션이 되기위해서는 해당 롤백 복구 구간이 정상적으로 수행되어야한다.

$P_f(k)$ 를 구하기 위하여 k 개의 결함 섹션들을 표현하기 위한 $(2k-1)$-tuple 벡터 $\mathbf{x}^{(k)}$ 를 다음과 같이 정의한다.

$$\mathbf{x}^{(k)} = (x_0, x_1, \cdots, x_{2k-2}),\ x_i \in \{0,1\} \quad (4)$$

x_i 는 그림 3에 표현된 것처럼 k 개의 결함 섹션들을 구성하는 실행 구간, 체크포인팅 구간, 롤백 복구 구간에 대응한다. 아래 첨자 i 가 짝수이면 실행 구간에, 홀수이면 체크포인팅 구간이나 롤백 구간에 x_i 가 대응된다. 그리고 x_i 가 대응되는 구간에서 일시적 오류가 발생했으면 x_i 는 1, 그렇지 않으면 0을 갖는다. 끝으로 마지막 k 번째 결함 섹션의 롤백 복구 구간은 정상적으로 수행되어야 하므로 x_i 가 할당되지 않았다. 결국 벡터 $\mathbf{x}^{(k)}$ 는 태스크의 실행 중에 나타나는 일시적 오류의 발생 양상을 표현한다고 볼 수 있다.

벡터 $\mathbf{x}^{(k)}$ 가 일련의 k 개의 결함 섹션을 표현하기 위해서는 다음 조건을 만족하여야 한다.

A. $k=1$ 이면 $x_0 = 1$

B. $k \geq 2$ 이면 x_0 와 x_1 는 동시에 0이 아니고, x_{2k-3} 와 x_{2k-2} 는 동시에 0이 아님

C. $k \geq 3$ 이면 $i(=2,3,\cdots,k-1)$ 에 대해 $x_{2(i-1)-1}$, $x_{2(i-1)}$ 그리고 $x_{2(i-1)+1}$ 는 동시에 0이 아님

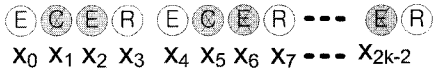


그림 3. 추출된 k 개의 결함 섹션.

Fig. 3. Extracted k faulty sections.

조건 A는 결함 섹션이 그림 2(a)의 형태를 가져야한다는 것이며, 조건 B는 첫번째로 발생한 결함 섹션이 그림 2의 (b-d)에 표현된 첫번째 결함 섹션의 형태를 가져야함을 의미한다. 마찬가지로 조건 C와 D는 첫번째를 제외한 결함 섹션들이 그림 2의 (b-e)에 표현된 결함 섹션의 형태를 가져야함을 의미한다. 주어진 k 에 대해 상기의 네가지 조건들을 만족하는 벡터 $\mathbf{x}^{(k)}$ 를 만들어내는 것은 그리 어렵지 않다.

$P_f(k)$ 를 구하기 위하여 벡터 $\mathbf{x}^{(k)}$ 가 표현하는 형태로 k 개의 결함 섹션들이 발생할 확률을 구해야한다. 이를 위해 $\mathbf{x}^{(k)}$ 의 함수 $f_0(\mathbf{x}^{(k)})$, $f_1(\mathbf{x}^{(k)})$, $g_0(\mathbf{x}^{(k)})$ 그리고 $g_1(\mathbf{x}^{(k)})$ 를 정의한다. $g_1(\mathbf{x}^{(k)})$ 과 $g_0(\mathbf{x}^{(k)})$ 는 첨자 i 가 짝수이며 값이 각각 1과 0인 x_i 의 개수를 반환하고, $f_1(\mathbf{x}^{(k)})$ 과 $f_0(\mathbf{x}^{(k)})$ 는 첨자 i 가 홀수이며 값이 각각 1과 0인 x_i 의 개수를 반환하는 함수로 정의하면, $g_1(\mathbf{x}^{(k)})$ 과 $g_0(\mathbf{x}^{(k)})$ 는 각각 '오류가 발생한' 그리고 '발생하지 않은' 실행 구간의 개수를 의미하고, $f_1(\mathbf{x}^{(k)})$ 는 '오류가 발생한' 체크포인팅 구간 또는 롤백 복구 구간의 개수를 의미한다. 그리고 $f_0(\mathbf{x}^{(k)})$ 는 '오류가 발생하지 않은' 롤백 복구 구간의 개수를 의미하며, $\mathbf{x}^{(k)}$ 가 표현하는 k 개의 결함 섹션들 중 마지막 결함 섹션의 복구 구간은 $\mathbf{x}^{(k)}$ 에 포함되지 않았으므로, '오류가 발생하지 않은' 롤백 복구 구간의 실제 수는 $f_0(\mathbf{x}^{(k)})+1$ 이다.

$g_1(\mathbf{x}^{(k)})$ 개의 실행 구간에서 오류가 발생할 확률은

$$\left(1 - e^{-\lambda \frac{T_e}{N_c}}\right)^{g_1(\mathbf{x}^{(k)})} \quad (5)$$

이고, $g_0(\mathbf{x}^{(k)})$ 개의 실행 구간에서 오류가 발생하지 않을 확률은

$$e^{-\lambda \frac{T_e}{N_c} \cdot g_0\left(\mathbf{x}^{(k)}\right)} \quad (6)$$

이다.

마찬가지 방법으로 $f_1(\mathbf{x}^{(k)})$ 개의 체크포인팅 또는 롤백복구 구간에서 오류가 발생할 확률은

$$\left(1 - e^{-\lambda T_c}\right)^{f_1(\mathbf{x}^{(k)})} \quad (7)$$

이고, $f_0(\mathbf{x}^{(k)})+1$ 개의 롤백 복구 구간에서 오류가 발생하지 않을 확률은

$$e^{-\lambda T_c \cdot (f_0(\mathbf{x}^{(k)})+1)} \quad (8)$$

이다.

그림 3의 k 개의 결함 섹션은 일시적 오류를 극복하며 실행된 실시간 태스크로부터 추출되었던 것이므로 추출되기

이전의 상황을 고려하여야 한다. 이것은 $\mathbf{x}^{(k)}$ 로 표현되는 k 개의 결함 섹션이 N_c 개의 무결함 섹션들과 구성할 수 있는 가능한 조합의 수를 고려하면 된다. $\mathbf{x}^{(k)}$ 가 표현하는 오류 발생 및 극복 양상이 유지되기 위해서는 일련의 k 개의 결함 섹션이 무결함 롤백 복구 구간에서만 분리되어 N_c 개의 무결함 섹션들과 구성할 수 있는 가능한 조합의 수만이 고려되어야 한다. 그리고 이들 조합은 실시간 태스크의 1회 실행 완료를 위하여 마지막 섹션이 무결함 섹션이어야 한다. 이상의 두조건을 만족하면서 $\mathbf{x}^{(k)}$ 로 표현되는 k 개의 결함 섹션이 N_c 개의 무결함 섹션들과 구성할 수 있는 가능한 조합의 수는 다음과 같다.

$$\begin{pmatrix} N_c + f_0\left(\mathbf{x}^{(k)}\right) \\ f_0\left(\mathbf{x}^{(k)}\right)+1 \end{pmatrix} \tag{9}$$

k 개의 결함 섹션을 만들 수 있는 가능한 모든 벡터 $\mathbf{x}^{(k)}$ 의 집합을 $\mathbf{X}^{(k)}$ 라고 정의하면, $\mathbf{X}^{(k)}$ 는 k 개의 결함 섹션을 만들어 내는 일시적 오류의 발생 양상을 모두 포함하게 되며 이것과 (5-9)를 이용하여 태스크의 실행 중 k 개의 결함 섹션이 발생할 확률을 $P_f(k)$ 를 다음과 같이 표현할 수 있다.

$$\begin{aligned} P_f(k) = & \sum_{\mathbf{x}^{(k)} \in \mathbf{X}^{(k)}} \begin{pmatrix} N_c + f_0\left(\mathbf{x}^{(k)}\right) \\ f_0\left(\mathbf{x}^{(k)}\right)+1 \end{pmatrix} \\ & \cdot \left(1 - e^{-\lambda \frac{T_e}{N_c}}\right)^{g_1\left(\mathbf{x}^{(k)}\right)} \cdot e^{-\lambda \frac{T_e}{N_c} \cdot g_0\left(\mathbf{x}^{(k)}\right)} \\ & \cdot \left(1 - e^{-\lambda T_c}\right)^{f_1\left(\mathbf{x}^{(k)}\right)} \cdot e^{-\lambda T_c \cdot \left(f_0\left(\mathbf{x}^{(k)}\right)+1\right)} \end{aligned} \tag{10}$$

(10)의 $P_f(k)$ 를 (3)에 대입하면 체크포인팅 수행 횟수에 따른 태스크의 성공적 실행 확률 P_s 를 구할 수 있다.

IV. 모의 실험

그림 4와 표 1은 λ =0.001이고 여유 시간 T_s 가 25, 33, 40 인 경우에 T_e =100, T_c =1인 실시간 태스크에 대해 태스크의 1회 실행 당 수행되는 체크포인팅 작업의 횟수 N_c 에 따른 (3)의 태스크의 성공적 실행 확률 P_s 와 (1)의 여유 섹션의 수 N_t 를 보여준다.

체크포인팅의 실행 횟수에 따라 여유 섹션의 수가 1 이상이 되면 체크포인팅을 실행하지 않을 때보다 결함 허용성능이 증가함을 알 수 있다. 그리고 과도한 체크포인팅의 실행이 역효과를 가져오는 것도 확인할 수 있다. 여유 시간이 클수록 일시적 오류를 극복하는데 사용할 수 있는 여유 섹션의 수가 증가하여 태스크의 성공적 실행 확률이 1로 접근하는 것을 볼 수 있다. 하지만 T_s 가 증가할수록 그 증가된 양에 비해 결함허용 성능이 개선되는 폭은 작아진다.

표 2는 그림 4에서 표시된 내용 중의 일부에 대해 (3)으로부터 얻은 P_s 와 모의 실험을 통해 얻은 태스크의 실행 성공률을 비교한 것이다. 세 번째 열은 (3)으로부터 얻은 값이고 네 번째 열과 다섯 번째 열은 각각 모의 실험을 통하여 실시

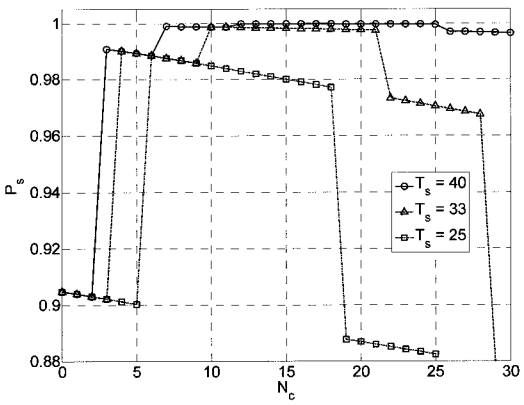


그림 4. 체크포인팅 작업 횟수에 따른 결함허용 성능.

Fig. 4. Fault-tolerance according to the number of checkpointing operations.

표 1. T_s 와 N_c 에 따른 N_t.

Table 1. N_t according to N_c and T_s.

T_s =25		T_s =33		T_s =40	
N_c	N_t	N_c	N_t	N_c	N_t
1 - 5	0	1 - 3	0	1 - 2	0
6 - 18	1	4 - 9	1	3 - 6	1
19 - 25	0	10 - 21	2	7 - 11	2
		22 - 28	1	12 - 25	3
		29 - 30	0	26 - 30	2

표 2. 모의 실험 결과.

Table 2. Simulation result.

T_s	N_c	P_s		
		해석	평균	95% 신뢰 구간
25	6	0. 98853	0. 98854	0.98850 - 0.98857
	7	0. 98765	0. 98763	0.98758 - 0.98769
	8	0. 98674	0. 98676	0.98672 - 0.98680
33	4	0. 99014	0. 99015	0.99013 - 0.99018
	5	0. 98937	0. 98936	0.98929 - 0.98943
	6	0. 98853	0. 98853	0.98847 - 0.98858
	7	0. 98765	0. 98764	0.98756 - 0.98771
	8	0. 98674	0. 98677	0.98667 - 0.98687
	9	0. 98581	0. 98581	0.98572 - 0.98589
	10	0. 99869	0. 99869	0.99868 - 0.99871
40	3	0. 99076	0. 99081	0.99075 - 0.99087
	4	0. 99014	0. 99015	0.99008 - 0.99021
	5	0. 98937	0. 98935	0.98931 - 0.98938
	6	0.98853	0. 98856	0.98851 - 0.98860
	7	0.99898	0. 99900	0.99897 - 0.99902
	8	0.99889	0. 99888	0.99886 - 0.99890
	9	0.99879	0. 99879	0.99876 - 0.99882
	10	0.99869	0. 99867	0.99865 - 0.99870
	11	0.99859	0. 99861	0.99857 - 0.99864
	12	0.99984	0. 99985	0.99985 - 0.99986
	13	0.99982	0. 99984	0.99983 - 0.99984

간 제어 태스크를 10^7회 반복 수행하여 구한 평균 성공적 실행률과 95% 신뢰구간이다. 해석적으로 구한 결과가 모의 실험 결과와 일치함을 알 수 있다.

V. 수행 조건 및 최적 방안

그림 4와 표 1로부터 여유 섹션이 확보되지 않는 체크포인팅 적용은 성능을 악화시킨다는 것을 알 수 있다. 이장에서는 체크포인팅 적용이 효과를 발휘할 수 있는 조건에 대해 정리한다. 여유 섹션의 수가 많을수록 결함 허용 성능이 개선되는 것을 확인하였으므로, 여유 섹션이 오직 한 개만 확보되는 경우, 즉 N_t=1인 경우만을 고려하면 된다.

N_t=1이면 k=1이므로 (3)과 (10)으로부터 (11)을 얻게 된다.

$$P_s = e^{-\lambda(T_e+N_cT_c)}\left[1+N_c(1-e^{-\lambda\frac{T_e}{N_c}})e^{-\lambda T_c}\right] \tag{11}$$

결함허용성능이 개선되기 위해서는 부등식 (12)가 만족되어야 하며 이를 정리하면 (13)을 얻게 된다.

$$e^{-\lambda T_e} < e^{-\lambda(T_e+N_cT_c)}\left[1+N_c(1-e^{-\lambda\frac{T_e}{N_c}})e^{-\lambda T_c}\right] \tag{12}$$

$$e^{\lambda N_cT_c}-1 < N_c(1-e^{-\lambda\frac{T_e}{N_c}})e^{-\lambda T_c} \tag{13}$$

가정에 의해 $\lambda T_e \ll 1$ 이고, 체크포인팅 작업의 총 소요시간이 태스크의 실행 시간 보다는 작아야할 것이므로 $N_cT_c < T_e$ 이다. 따라서 $\lambda N_cT_c \ll 1$ 이다. 그리고 $0<a<1$ 을 만족하는 실수 a 에 대해 $e^a-1<2a$, $1-e^{-a}<a$ 그리고 $e^{-a}<1-a+\frac{1}{2}a^2$ 이므로, 다음의 세부등식을 얻을 수 있다.

$$e^{\lambda N_cT_c}-1 < 2\lambda N_cT_c \tag{14}$$

$$1-e^{-\lambda\frac{T_e}{N_c}} < \lambda\frac{T_e}{N_c} \tag{15}$$

$$e^{-\lambda T_c} < 1-\lambda T_c+\frac{1}{2}\lambda^2T_c^2 \tag{16}$$

(14-16)의 부등식을 (13)에 대입하면 (17)을 얻는다.

$$2N_cT_c < T_e(1-\lambda T_c+\frac{1}{2}\lambda^2T_c^2) \tag{17}$$

따라서 여유 섹션이 한 개 이상만 확보되면 (17)을 만족하는 체크포인팅 작업 횟수는 실시간 임베디드 시스템의 결함허용 성능을 개선할 수 있다. 그리고 (17)에서 λ 를 포함하는 항을 무시한다면 (18)과 같은 간략화된 조건을 얻을 수 있다.

$$N_c < \frac{T_e}{2T_c} \tag{18}$$

표 1과 그림 4로부터 결함허용 성능을 최대로 만들어주는 최적의 체크포인팅 횟수가 존재함을 알 수 있다. 그것은 오류의 극복에 필요한 여유 섹션의 수를 최대로 만들어주는 가장 적은 체크포인팅 횟수이다.

(1)의 N_t 를 최대로 만드는 N_c 값의 후보는 (1)의 $\lfloor\cdot\rfloor$ 의 내부항을 최대로 만들어주는 값을 구하여 다음과 같이 얻을 수 있다.

$$\left\lfloor\frac{-T_e+\sqrt{T_e(T_e+T_s)}}{T_c}\right\rfloor \text{와} \left\lceil\frac{-T_e+\sqrt{T_e(T_e+T_s)}}{T_c}\right\rceil$$

여기에서 $\lceil\cdot\rceil$ 는 주어진 값보다 작지않은 최소의 정수를 의미한다.

위의 두 N_c 값 중 N_t 를 최대로 만들어주는 값을 N_c^* 라고 하고, 이때의 N_t 값을 N_t^* 라고 하면 N_t^* 는 (19)와 같이 표현된다.

$$N_t^* = \left\lfloor\frac{N_c^*(T_s-N_c^*T_c)}{T_e+N_c^*T_c}\right\rfloor \tag{19}$$

그리고 최적의 N_c 값 N_c^O 는 (20)과 같이 표현된다.

$$N_c^O = \min\left\{N_c : N_t^* = \left\lfloor\frac{N_c(T_s-N_cT_c)}{T_e+N_cT_c}\right\rfloor\right\} \tag{20}$$

VI. 결론

일시적 오류가 포아송 분포에 따라 발생하고 체크포인팅 생성 직전에 수행되는 오류 검출 기법에 의해 검출되어, 오류에 의한 영향이 롤백 복구 작업에 의해 복구되는 상황에서 실시간 태스크가 시간 제약을 만족할 확률을 해석하여 체크포인팅 기법의 적용에 따른 내고장성 개선 효과를 보여 주었다. 그리고 해석 결과는 모의 실험을 통하여 검증되었다.

해석 결과를 바탕으로 여유 시간이 많을 수록 일시적 오류를 극복할 가능성이 커지게 되며, 주어진 여유 시간에 대해 성능을 최대로 만들어 주는 체크포인팅 작업의 횟수가 존재함을 보여주었다. 그리고 결함 허용성을 개선할 수 있는 체크포인팅 수행 횟수에 대한 조건을 제시하였다.

참고문헌

[1] R. Harboe-Sørensen, 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.

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

[3] E. Normand, "Signle event upset at ground level," *IEEE Trans. Nuclear Science*, vol. 43, no. 6, pp. 2742-2750, Dec., 1996.

[4] D. P. Siewiorek, *Reliable Computer Systems: Design and Evaluation*, A K Peters, 1998.

[5] E. Dupont, M. Nicolaidis, and P. Rohr, "Embedded robustness IPs for transient-error-free ICs," *IEEE Design & Test of Computers*, vol. 19, pp. 56-70, May-Jun., 2002.

[6] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Engineering*, vol. 1, no. 2, pp. 220-232,

June, 1975.

[7] K. G. Shin, T.-H. Lin, and Y.-H. Lee, "Optimal checkpointing of real-time tasks," *IEEE Trans. Computers*, vol. C-36, no. 11, pp. 1328-1341, Nov, 1987.

[8] Z. Li, H. Chen and S. Yu, "Performance optimization for energy-aware adaptive checkpointing in embedded real-time systems," *Proc. Design, Automation and Test in Europe 2006*, vol. 1, pp.6-11, Mar, 2006.

[9] R. Geist, R. Reynolds, and J. Westall, "Selection of a checkpoint interval in a critical-task environment," *IEEE Trans. Reliability*, vol. 37, no. 4, pp. 395-400, Nov., 1988.

[10] S. Punnekkat, A. Burns, and R. Davis, "Analysis of checkpointing for real-time systems," *The Int'l Journal of Time-Critical Computing Systems (Real-Time Systems)*, vol. 20, no. 1, pp. 83-102, Jan, 2001.

[11] R. Melhem, D. Mosse, and E. Elnozahy, "The interplay of power management and fault recovery in real-time systems," *IEEE Trans. Computers*, vol. 53, no. 2, pp.217-231, Feb., 2004.

[12] Y. Zhang and K. Chakrabarty, "Dynamic adaptation for fault tolerance and power management in embedded real-time systems," *ACM Trans. Embedded Computing Systems*, vol. 3, no. 2, pp. 336-360, May 2004.

[13] V. K. Stefanidis and K. G. Margaritis, "Algorithm based fault tolerance: Review and experimental study," *Int'l Conference of Numerical Analysis and Applied Mathematics 2004 (ICNAAM 2004)*, 2004.

[14] P. P. Shirvani, N. R. Saxena, and E. J. McCluskey, "Software-implemented EDAC protection against SEUs," *IEEE Trans. Reliability*, vol. 49, no. 3, pp. 273-284, Sep, 2000.

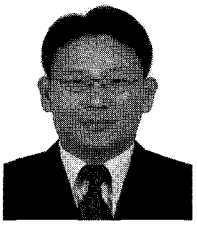
[15] R. Stroph and T. Clarke, "Dynamic acceptance tests for complex controllers," *Proc. 24th Euromicro Conference*, vol. 1, pp. 411-417, Aug, 1998.

[16] J. Sosnowski, "Transient fault tolerance in digital systems," *IEEE Micro*, vol. 14, no. 1, pp. 24-35, Feb, 1994.

[17] C. M. Krishna and K. G. Shin, *Real-Time Systems*, McGraw-Hill, 1997.

[18] C. N. Hadjicostis, "Finite-state machine embeddings for nonconcurrent error detection and identification," *IEEE Trans. Automatic Control*, vol. 50, no. 2, pp. 142-153, Feb., 2005.

[19] B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley, 1989.



유 상 문

1992년 금오공과대학교 전자공학과(공학사). 1995년 한국과학기술원 전기및전자공학과(공학석사). 2006년 동 대학원 전자전산학과(공학박사). 1995년~2000년 LG전자(주). 2000년~2004년 한국과학기술원 인공위성연구센터. 2006년~현재 군산대학교 전자정보공학부 전임강사. 관심분야는 임베디드 제어 시스템, 실시간 제어 시스템, 결함허용 시스템.