실시간 제어 시스템의 결함 극복을 위한 이중화 구조와 체크포인팅 기법의 성능 분석

Performance Analysis of Checkpointing and Dual Modular Redundancy for Fault Tolerance of Real-Time Control System

유 상 문*
(Sang-Moon Ryu)

Abstract : This paper deals with a performance analysis of real-time control systems, which engages DMR(dual modular redundancy) to detect transient errors and checkpointing technique to tolerate transient errors. Transient errors are caused by transient faults and the most significant type of errors in reliable computer systems. Transient faults are assumed to occur according to a Poisson process and to be detected by a dual modular redundant structure. In addition, an equidistant checkpointing strategy is considered. The probability of the successful task completion in a real-time control system where periodic checkpointing operations are performed during the execution of a real-time control task is derived. Numerical examples show how checkpoiniting scheme influences the probability of task completion. In addition, the result of the analysis is compared with the simulation result.

Keywords : real-time control system, DMR(Dual Modular Redundancy), checkpointing

I. 서론

실시간 제어 시스템은 물리적인 외부 환경으로부터 입력을 받아 제어 작업을 수행하여 출력을 내는 과정에서 요구되는 시간 제약 조건을 항상 만족하는 제어 시스템을 말하며, 일상 생활 및 산업 부문에 있어서 컴퓨터의 활용 범위 및 영향이 급격히 확대되어 감에 따라 실시간 제어 시스템의 성능에 대한 요구는 갈수록 늘어갈 것이다. 실시간 제어 시스템은 종종 열악하고 급변하는 환경 속에서 중단 없이 오랜 시간 지속적으로 동작하도록 요구되며 높은 안정성과 시간 제약 조건을 만족하여야 한다. 이러한 실시간 제어 시스템은 그 핵심이 소형 컴퓨터 시스템의 형태로 구현되는 것이 일반적이며 하드웨어나 소프트웨어 오류에 의해서 오동작할 수 있다.

Single Event Upsets(SEU)[1-3]나 전자파간섭(EMI)등의 환경적 요인으로 발생할 수 있는 일시적 결함(transient faults)[4]은 컴퓨터 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.

체크포인팅(checkpointing) 기법[6]은 일시적 오류를 극복하는 방법으로, 컴퓨터 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.

일시적 오류에 가장 취약한 부품은 메모리이며 이에 대한 대비책이 비용 대비 가장 큰 효과를 보는 것으로 알려져 있다. 메모리에서 발생하는 오류 다음으로 고려 대상이 되는 것이 프로세서에서 발생하는 오류이다. 메모리에서 발생하는 오류는 오류 검출 코드와 추가의 메모리를 결합하면 비교적 쉽게 대응할 수 있지만, 프로세서 내부에서 발생하는 오류 검출을 위해서는 오류 검출 기능을 내장한 프로세서를 사용해야 하며, 가격이 매우 비싸고 선택의 폭이 매우 좁다. 이중화 구조(dual modular redundancy)를 적용하면 오류 검출 기능이 없는 저가의 상용 프로세서와 메모리를 사용하여 오류를 쉽게 검출할 수 있다. 이중화 구조는 프로세서 레벨에서 구현할 수도 있고 시스템 레벨에서 구현할 수도 있다.

대부분의 체크포인팅에 관한 이전 연구는 데이터베이스 시스템이나 범용 컴퓨터 시스템에서의 시스템의 가용성 최대화나 성능 평가, 프로그램의 평균 실행 시간 최소화, 체크포인팅 부담 최소화 등이 주요 주제였으며 [7]과 [8]에 잘 정리되어 있다. 실시간 시스템에 대해서는 태스크 평균 실행 시간이나 응답 시간 분석 등에 관한 연구[9,10]가 주요 주제였었다.

본 논문에서는 이전 연구들과 달리 신뢰도 관점에서 체크포인팅 기법을 다룬다. 오류 검출을 위해 시스템 레벨의 이중화 구조를 채택한 실시간 제어 시스템에 체크포인팅 기법을 적용했을 때의 시스템의 일시적 오류 극복 성능을 분석하고 최적의 방안이 있음을 보인다. 2장에서는 고려 대상 시스템의 모델과 가정에 대해 기술하고, 3장에서는 실시간 제어 시스템이 체크포인팅 기법을 이용하여 일시적 오류를 극복하고 시간 제약을 만족할 수 있는 확률을 해석적으로 구하며,

* 책임저자(Corresponding Author)
논문접수 : 2007. 7. 19., 채택확정 : 2007. 11. 14.
유상문 : 군산대학교 전자정보공학부(smryu@kunsan.ac.kr)
※ 본 연구는 과학기술부의 과학기술위성 3호 개발 사업의 지원을 받아 수행되었음.

4장에서는 3장의 해석의 결과를 고찰하고 이를 모의 실험 결과와 비교하고, 끝으로 5장에서 결론을 맺는다.

II. 가정 및 시스템 모델

일시적 결함에 따른 오류는 평균값 λ 를 갖는 포아송(poisson) 분포에 따라 발생한다고 가정한다. 이것은 일시적 결함 및 오류를 다루는 다수의 이전 연구[7-11]에서 사용된 가정이다. 발생된 결함은 매우 짧은 시간 동안 만 존재하다 사라지지만, 시스템 내부에 표현되는 정보에 오류를 만들어 낸다. 이 오류가 정정되어야 시스템은 일시적 결함을 극복하고 요구되는 기능을 발휘할 수 있는 것이다.

일시적 결함에 의해 발생한 오류는 그림 1과 같은 이중화 구조와 비교기를 통해서 검출될 수 있다. 제어 대상의 상태를 파악하는 센서로부터 인코딩(encoding)된 값은 제어기 A와 B에 동일하게 입력되고, 각 제어기의 프로세서가 제어 알고리즘을 수행하는 제어 태스크를 실행한다. 일시적 결함에 의한 오류가 각 제어기의 프로세서나 메모리에서 발생하면 이것은 중간 계산 결과나 주요 변수의 값에 영향을 미칠 것이다. 따라서 주기적으로 두 제어기의 중간 계산 결과나 메모리에 저장되어 있는 중요 정보를 비교기 블록에서 비교하고, 두 결과가 일치하면 오류가 발생하지 않았다고 판단하여 비교기 블록에 있는 고신뢰(high-reliable) 메모리에 체크포인트를 생성한다. 만일 두 제어기의 중간 계산 결과가 일치하지 않으면 오류가 발생한 것으로 판단하여 최근에 저장되어 있던 체크포인트를 이용하여 두 제어기의 상태를 최근의 체크포인트가 저장되던 상태로 되돌리기 위한 롤백 복구를 수행한다. 실시간 제어 태스크의 한 주기가 종료되고 체크포인트가 정상적으로 생성되면, 제어 태스크의 실행 결과에 따라 비교기 블록에서 제어에 필요한 구동 장치를 구동한다.

안정적 체크포인팅을 위해 두 개의 체크포인트 저장공간을 구비하고 이들을 교대로 사용한다. 만일 체크포인팅 과정에서 일시적 오류가 검출되면 최근의 무결점 상태가 저장되어 있는 다른 체크포인트를 이용하여 롤백 복구를 수행한다. 그리고 체크포인트 저장을 위한 고신뢰 메모리는 각 제어기에 장착된 메모리 보다 매우 작은 저장 용량이 요구되며, 오류 검출 및 정정 코드와 추가의 메모리를 결합하여 원하는 정도까지 안정하게 만들 수 있다. 그리고 비교기에서 발생할 수 있는 오류는 WDT(Watchdog timer)나 고신뢰 메모리가 갖는 오류 검출 기능을 통하여 감지할 수 있다.

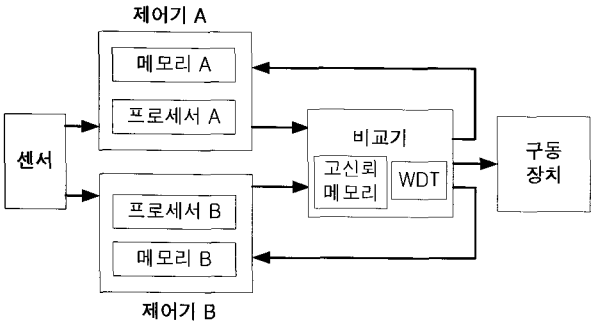


그림 1. 이중화 구조 제어 시스템.
Fig. 1. DMR structured control system.

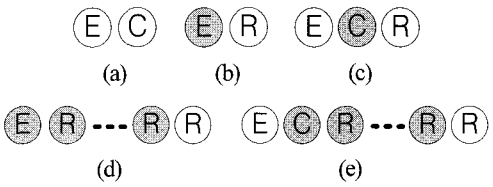


그림 2. 발생 가능한 모든 수행 패턴.
Fig. 2. All possible execution patterns.

그림 3. 태스크의 수행 패턴 예.
Fig. 3. Example pattern of a task .

체크포인팅과 롤백 복구가 적용되는 태스크의 실행 중 발생할 수 있는 수행 패턴을 그림 2에 나타내었다. 그림에서 E, C와 R은 각각 태스크의 실행 구간, 체크포인팅 작업 그리고 롤백 복구 작업을 의미하며, 회색으로 표현된 것은 해당 구간에서 일시적 결함에 따른 오류가 발생하였음을 나타낸다.

그림 2(a)는 태스크의 일정 부분이 실행된 후 체크포인팅 작업이 정상적으로 수행된 경우를 의미하고, (b)는 태스크 실행 구간에서 오류가 검출되어 롤백 복구를 실행한 경우를 의미한다. 그리고 (c)는 체크포인팅 작업 도중 오류가 발생되어 롤백 복구를 수행한 경우이다. (b)와 (c)의 경우에 실행되던 롤백 복구 과정에서 1회 또는 그 이상의 롤백 복구 중 오류가 발생될 수 있는데 이를 각각 (d)와 (e)에 나타내었다.

일시적 결함에 의한 오류를 체크포인팅과 롤백 복구 기법에 의해 극복해가는 실시간 태스크의 한 주기 동안 실행 양상은 그림 2(a)-(e)에 나열된 수행 패턴들의 일련의 조합으로 표현될 수 있으며 그림 3에 그 한 예가 있다. 그림 3은 태스크가 세 번째 체크포인팅 작업과 4번째 실행 구간 수행 중 발생한 오류를 극복하는 경우를 보여 준다.

각 제어기에서 동일하게 실행되는 주기적 실시간 제어 태스크가 센서의 값을 바탕으로 제어 출력을 계산하는데 소요되는 시간을 실행 시간이라 하고, 1회의 제어 출력 계산에 허용된 시간을 상대적 마감 시간(relative deadline)[12]이라고 한다. 상대적 마감 시간과 실행 시간의 차이를 여유 시간(slack time)이라 하며, 이 여유 시간을 활용해서 주기적인 체크포인팅 작업과 필요한 롤백 복구를 수행하여 일시적 결함의 극복할 수 있는 것이다.

본 논문에서는 제어 태스크의 한 주기 안에 일정 횟수의 체크포인팅 작업이 동일 간격으로 수행되는 정적 등간격 체크포인팅 기법이 적용되는 상황을 고려한다.

III. 성능 분석

제어 시스템이 일시적 결함에 의한 오류를 극복하면서 주어진 시간 조건을 만족하며 센서로부터의 입력에 따른 구동 장치를 위한 출력 계산을 완료할 확률, 즉 실시간 제어 태스크의 성공적 실행 확률을 성능 지표로 정하고 이를 해석적으로 구해 체크포인팅 기법의 효과를 검토한다.

성능 해석을 위해 실시간 제어 태스크의 실행 시간을 T_e, 여유 시간을 T_s, 체크포인팅 작업 소요 시간을 T_c, 롤백 복

구 소요 시간을 T_r 그리고 제어 태스크의 1회 실행에 수행되는 체크포인팅 작업 횟수를 N_c 로 표기한다.

여유 시간이 T_s 인 태스크의 한 주기에 T_c 가 소요되는 N_c 회의 체크포인팅 작업을 실행하면 태스크가 일시적 오류를 극복하는데 사용할 수 있는 실제 여유 시간 T_{ns} 는 (1)과 같게 된다.

$$T_{ns} = T_s - N_c T_c \tag{1}$$

그림 2에 그려진 실행 구간 E의 길이는 실행 시간이 T_e 인 태스크에 N_c 개의 체크포인트가 동일 간격으로 삽입되었으므로 T_e/N_c 이다. 그림 2의 실행 패턴 (b)처럼 태스크의 실행 구간에서 일시적 오류가 발생하여 실시간 제어 태스크가 손실하게 되는 시간과 실행 패턴 (c)처럼 체크포인팅 작업 구간에서 일시적 오류가 발생하여 실시간 제어 태스크가 손실하게 되는 시간을 각각 T_l 과 T_m 이라 하면 이들은 다음과 같게 된다.

$$T_l = \frac{T_e}{N_c} + T_r \tag{2}$$

$$T_m = \frac{T_e}{N_c} + T_c + T_r \tag{3}$$

실시간 제어 태스크의 1주기 동안 일시적 오류에 의해 손상될 수 있는 태스크의 실행 구간의 수를 i 라 하면 i 는 태스크의 성공적 실행을 위해 다음의 N_i 보다 크지 않아야 한다.

$$N_i = \left\lfloor \frac{T_{ns}}{T_l} \right\rfloor \tag{4}$$

그리고 실시간 제어 태스크의 1주기 동안 일시적 오류에 의해 손상될 수 있는 체크로인팅 작업 구간의 수를 j 라 하면 j 는 i 개의 실행 구간이 일시적 오류에 의해 손상된 경우에도 태스크가 성공적으로 실행되기 위해서 다음의 N_j 보다 크지 않아야 한다.

$$N_j = \left\lfloor \frac{T_{ns} - iT_l}{T_m} \right\rfloor \tag{5}$$

실시간 제어 태스크의 1주기 동안 일시적 오류에 의해 j 개의 손상된 실행 구간과 j 개의 손상된 체크포인팅 작업 구간이 발생하는 사건을 $\langle i, j \rangle$ 로 나타내고 사건 $\langle i, j \rangle$ 가 발생할 확률을 $\Pr\{\langle i, j \rangle\}$ 로 표기하면, 태스크가 일시적 오류를 극복하고 시간 제약을 만족시킬 확률, 즉 성공적으로 실행될 확률 P_s 는 (6)과 같이 표현될 수 있다.

$$P_s = \sum_{i=0}^{N_i} \sum_{j=0}^{N_j} \Pr\{\langle i, j \rangle\} \tag{6}$$

이제 확률 $\Pr\{\langle i, j \rangle\}$ 을 구하면 실시간 제어 태스크의 성공적 실행 확률을 구할 수 있다.

태스크가 요구되는 계산을 완료하고 N_c 회의 체크포인팅 작업이 완료되기 위해서는 그림 2(a) 패턴이 N_c 회 나타나야 한다. 그림 2(a) 패턴이 N_c 회 나타날 확률은 일시적 오류가 평균값 λ 를 갖는 포아송 분포에 따라 발생한다고 가정했으므로 다음과 같다[4].

$$e^{-\lambda\left(\frac{T_e}{N_c}+T_c\right)N_c} \tag{7}$$

일시적 오류에 의해 태스크의 실행 구간이 손상되어 그림 2(b)와 같은 실행 패턴이 i 회 나타날 확률은 다음과 같다.

$$\left(1 - e^{-\lambda\frac{T_e}{N_c}}\right)^i e^{-i\lambda T_r} \tag{8}$$

일시적 오류에 의해 체크포인팅 작업 구간이 손상되어 그림 2(c)와 같은 실행 패턴이 j 회 나타날 확률은 다음과 같다.

$$e^{-j\lambda\frac{T_e}{N_c}}\left(1 - e^{-\lambda T_c}\right)^j e^{-i\lambda T_r} \tag{9}$$

태스크 1회 실행 동안 그림 2(a), (b)와 (c) 패턴이 각각 N_c 회, i 회 그리고 j 회 출현할 가능성과 함께 이들이 구성할 수 있는 가능한 조합의 경우의 수도 고려되어야 한다. 태스크가 요구되는 작업을 정상적으로 수행하기 위해서는 마지막 실행 구간과 마지막 체크포인트가 오류 없이 완료되어야 한다. 따라서 $(N_c - 1)$ 개, i 개 그리고 j 개의 그림 2(a), (b)와 (c) 패턴이 구성할 수 있는 조합의 경우의 수를 고려하면 되며, 이때 가능한 조합의 수는 다음과 같다.

$$\binom{N_c - 1 + i + j}{i}\binom{N_c - 1 + j}{j} \tag{10}$$

태스크의 실행 구간에서 일시적 오류가 발생하고 뒤이은 롤백 복구 작업에서도 일시적 오류가 발생하여 롤백 복구 구간이 손상되면 그림 2(d)와 같은 패턴이 발생할 수 있다. 마찬가지로 체크포인팅 작업 도중 일시적 오류가 발생하고 뒤이은 롤백 복구 구간이 손상된다면 그림 2(e)와 같은 패턴이 발생할 수 있다. 오류에 의해 손상되는 롤백 복구 작업의 수를 k 라고 하면, k 는 실시간 제어 태스크가 사건 $\langle i, j \rangle$ 와 k 개의 손상된 롤백 작업 구간을 포용하면서 실시간 제약을 만족시켜야 하기 때문에 다음의 N_k 보다 클 수 없다.

$$N_k = \begin{cases} 0, & \text{when } i = j = 0 \\ \left\lfloor \dfrac{T_{ns} - i\left(\frac{T_e}{N_c} + T_r\right) - j\left(\frac{T_e}{N_c} + T_c + T_r\right)}{T_r} \right\rfloor, & \text{otherwise} \end{cases} \tag{11}$$

k 개의 손상된 롤백 작업 구간은 사건 $\langle i, j \rangle$ 가 발생했을 때 한번도 발생하지 않거나 1회 이상 그리고 (11)의 N_k 이하로 발생할 수 있다. 그리고 사건 $\langle i, j \rangle$ 에 대해 k 개의 손상된 롤백 작업 구간이 발생할 확률과 그 경우의 수는 각각

$$\left(1 - e^{-\lambda T_r}\right)^k \tag{12}$$

와

$$\frac{(i+j)^k}{k!} \tag{13}$$

이다.

따라서 (11), (12), (13)으로부터 사건 $\langle i,j \rangle$ 가 발생했을 때 롤백 복구 작업 중 발생한 일시적 오류를 포용하고 시간 제약 조건을 만족할 확률은 다음처럼 구해진다.

$$\sum_{k=0}^{N_k} \frac{(i+j)^k}{k!} \left(1-e^{-\lambda T_r}\right)^k \tag{14}$$

(7), (8), (9), (10), (14)로부터 실시간 제어 태스크의 1주기 동안 일시적 오류에 의해 i 개의 손상된 실행 구간과 j 개의 손상된 체크포인팅 작업 구간이 출현하는 사건 $\langle i,j \rangle$ 가 발생할 확률 $\Pr\{\langle i,j \rangle\}$ 를 (15)와 같이 구할 수 있다.

(15)를 (6)에 대입하여 실시간 제어 태스크가 체크포인팅과 롤백 복구를 통하여 일시적 오류를 극복하고 요구되는 시간 제약을 만족할 성공적 실행 확률을 구할 수 있다.

$$\begin{aligned} P\{\langle i,j \rangle\} = & \binom{N_c - 1 + i + j}{i} \binom{N_c - 1 + j}{j} \\ & \cdot \left(1 - e^{-\lambda \frac{T_e}{N_c}}\right)^i e^{-i\lambda T_r} \\ & \cdot e^{-j\lambda \frac{T_e}{N_c}} \left(1-e^{-\lambda T_c}\right)^j e^{-j\lambda T_r} \\ & \cdot e^{-\lambda\left(\frac{T_e}{N_c}+T_c\right)N_c} \cdot \sum_{k=0}^{N_k} \frac{(i+j)^k}{k!} \left(1-e^{-\lambda T_r}\right)^k \end{aligned} \tag{15}$$

IV. 고찰 및 모의 실험

그림 4의 그래프는 $\lambda = 0.001$ 인 환경에서 $T_e = 100$, $T_c = 1$, $T_r = 2$ 인 실시간 제어 태스크에 대해 태스크의 1회 실행 당 수행되는 체크포인팅 작업의 횟수 N_c 에 따른 (6)의 태스크의 성공적 실행 확률 P_s 의 변화를 여유 시간 T_s 가 22, 25, 33인 경우에 대하여 보여준다.

T_s 가 22인 경우에는 $N_c = 10$ 인 경우에만 성공적 실행 확률이 체크포인팅 기법을 적용하지 않았을 때($N_c = 0$) 보다 커짐을 알 수 있다. 이것은 10회 이외의 횟수로 체크포인팅 작업을 실행하게 되면 (4)와 (5)의 N_i 와 N_j 가 0보다 큰 값을 가질 만큼 (1)의 실제 여유 시간 T_{ms} 가 확보되지 않기 때문이다. 따라서, 체크포인팅 기법의 적용에 따른 효과를 보기 위해서는 실시간 제어 태스크의 1회 실행 당 수행되는 체크포인팅 작업 횟수가 상당히 중요함을 알 수 있다.

T_s 가 25와 33인 경우에는 성능을 개선시킬 수 있는 체크포인팅 작업 횟수가 두 개 이상 존재함을 알 수 있으며, 성능을 최대로 하여주는 체크포인팅 작업 횟수가 존재함을 알 수 있다. 그리고 T_s 가 33인 경우가 25인 경우보다 성공적 실행 확률을 1에 근접하게 할 수 있다. 이론적으로는 여유 시간 T_s가 클수록 성공적 실행 확률이 점점 1에 근접하게 되나, T_s가 증가할수록 그 증가된 양에 비해 성능이 개선되는 폭은 작아진다.

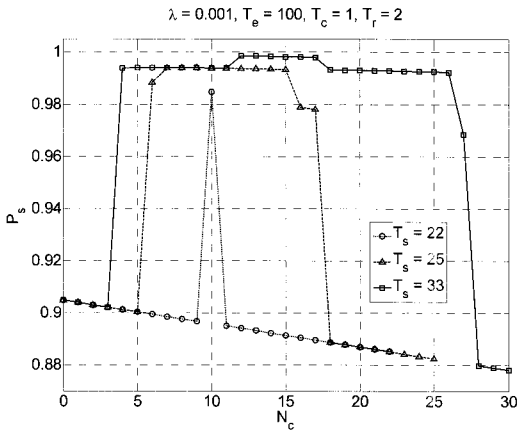


그림 4. 체크포인팅 작업 횟수에 따른 성능($\lambda = 0.001$).

Fig. 4. Performance according to the number of checkpointing operation ($\lambda = 0.001$).

표 1. N_c 에 따른 N_i 와 N_j (그림 4. $T_s = 33$).

Table 1. N_i and N_j according to N_c (Fig. 4. $T_s = 33$).

N_c	N_i	N_j
0 - 3	0	0
4 - 11	1	1
12 - 17	2	1
18 - 26	1	1
27	1	0
28 - 30	0	0

표 1은 그림 4에서 $T_s = 33$ 경우에 대해 N_c 에 따른 N_i 와 N_j 의 값을 보여준다. 그림 4와 비교해보면 N_c 의 변화에 따라 N_i 나 N_j 값이 변화할 때 P_s 가 상대적으로 큰 폭으로 변화하는 것을 볼 수 있다.

표 2는 그림 4에서 표시된 내용 중의 일부에 대해 (6)으로부터 얻은 P_s와 모의 실험을 통해 얻은 P_s의 값을 비교한 것이다. 세 번째 열은 (6)으로부터 얻은 값이고 네 번째 열과 다섯 번째 열은 각각 모의 실험을 통하여 실시간 제어 태스크를 10^7회 반복 수행하여 구한 평균 성공적 실행률과 95% 신뢰구간이다. 해석적으로 구한 결과가 모의 실험 결과와 일치함을 알 수 있다.

그림 5는 그림 4의 경우에서 일시적 오류의 평균 발생률만 다른 환경($\lambda = 0.002$) 에서의 체크포인팅 작업의 횟수 N_c와 여유 시간 T_s에 따른 태스크의 성공적 실행 확률 P_s의 변화를 보여 준다. 오류의 발생 빈도가 증가했으므로 태스크의 성공적 실행 확률은 감소했으나, 체크포인팅 작업의 횟수와 여유 시간 변화에 따른 태스크의 성공적 실행 확률 변화 양상은 동일함을 알 수 있다. 그리고 표 1의 내용은 그림 5에도 동일하게 적용된다. 따라서, 성공적 실행 확률을 최대로 만들어주는 체크포인팅 작업 수는 일시적 오류의 평균 발생률과 무관함을 알 수 있다.

표 2. 모의 실험 결과.
Table 2. Simulation result.

T_s	N_c	P_s		
		해석	평균	95% 신뢰 구간
22	10	0.98479	0.98474	0.98447 - 0.98503
25	6	0.98844	0.98846	0.98842 - 0.98850
	7	0.99375	0.99374	0.99370 - 0.99378
	8	0.99390	0.99390	0.99387 - 0.99393
	9	0.99387	0.99386	0.99384 - 0.99388
	10	0.99383	0.99382	0.99380 - 0.99385
	11	0.99376	0.99377	0.99375 - 0.99380
33	4	0. 99374	0.99373	0.99369 - 0.99376
	5	0. 99387	0.99386	0.99384 - 0.99388
	6	0.99392	0.99393	0.99390 - 0.99396
	7	0.99393	0.99394	0.99392 - 0.99397
	8	0.99392	0.99395	0.99391 - 0.99400
	9	0.99388	0.99388	0.99384 - 0.99393
	10	0.99384	0.99385	0.99383 - 0.99387
	11	0.99378	0.99382	0.99376 - 0.99388
	12	0.99848	0.99846	0.99844 - 0.99848
	13	0.99839	0.99841	0.99838 - 0.99844
	14	0.99829	0.99829	0.99825 - 0.99832

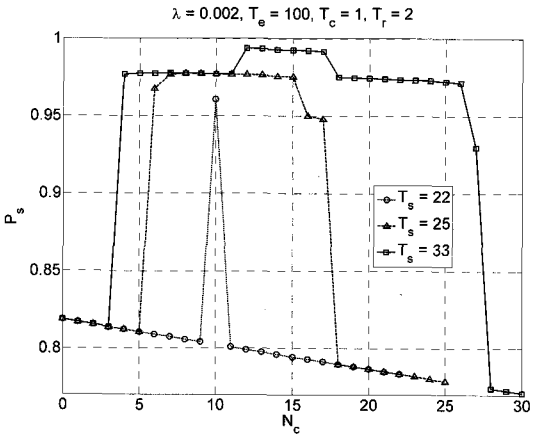


그림 5. 체크포인팅 작업 횟수에 따른 성능($\lambda = 0.002$).
Fig. 5. Performance according to the number of checkpointing operation ($\lambda = 0.002$).

V. 결론

실시간 제어 시스템에서 발생할 수 있는 일시적 오류가 이중화 구조에 의해 검출되고, 검출된 오류가 일정 주기로 실행되는 체크포인트 생성 작업과 롤백 복구 작업에 의해 극복되는 상황에서 단일 제어 태스크가 시간 제약을 만족할 확률을 해석하여 체크포인팅 기법의 효과를 보여 주었다. 그리고 해석 결과는 모의 실험을 통하여 검증되었다.

체크포인팅 기법이 실시간 시스템에 효과를 발휘하기 위해서는 실행되는 소프트웨어 태스크에 여유 시간이 확보되어야 한다. 적절한 여유 시간이 확보되지 않은 상황에서 체크포인팅 기법을 적용하면 일시적 오류 발생시 시간 제약을 만족하면서 이를 극복할 롤백 복구를 수행할 수 없고, 체크포인팅 작업 수행에 따라 제어 태스크가 일시적 오류에 노출되는 시간이 증가하게 되어 역효과를 가져온다. 그리고 여유 시간이 많을 수록 일시적 오류를 극복할 가능성이 커지게 되며, 주어진 여유 시간에 대해 성능을 최대로 만들어 주는 체크포인팅 작업의 횟수가 존재한다.

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] A. Ziv and J. Bruck, "An On-Line Algorithm for Checkpoint Placement," *IEEE Trans. Computers*, vol. 46, no. 9, pp. 976-985, Sep. 1997.

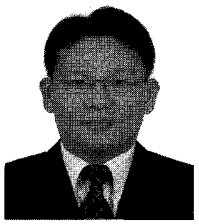
[8] Y. Ling, J. Mi, and X. Lin, "A variational calculus approach to optimal checkpoint placement," *IEEE Trans. Computers*, vol. 50, no. 7, pp. 699-708, Jul. 2001.

[9] 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.

[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] 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.

[12] J. W. S. Liu, *Real-Time Systems*, Prentice-Hall, 2000.



유 상 문

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