Journal of Institute of Control, Robotics and Systems (2011) 17(1):72-77
DOI:10.5302/J.ICROS.2011.17.1.72 ISSN:1976-5622 eISSN:2233-4335

동시 결함 검출 기능이 있는 실시간 제어 시스템의 결함 허용성을 위한 적응형 체크포인팅 기법

An Adaptive Checkpointing Scheme for Fault Tolerance of Real-Time Control Systems with Concurrent Fault Detection

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

Abstract: The checkpointing scheme is a well-known technique to cope with transient faults in digital systems. This paper proposes an adaptive checkpointing scheme for the reliability improvement of real-time control systems with concurrent fault detection capability. With concurrent fault detection capability the effect of transient faults are assumed to be detected with no latency. The proposed adaptive checkpointing scheme is based on the reliability analysis of an equidistant checkpointing scheme. Numerical data show the proposed adaptive scheme outperforms the equidistant scheme from a reliability point of view.

Keywords: real-time control system, fault tolerance, concurrent fault detection, adaptive checkpointing

I. 서론

실시간 제어 시스템은 주어진 대상에 대하여 제어 작업을 수행하기 위한 특정 목적의 컴퓨터 시스템으로서, 제어 작업 수행에 있어 요구되는 시간 제약 조건을 충실히 만족해야 한다. 실시간 제어 시스템을 포함한 모든 컴퓨터 시스템은 일시적 결함(transient faults)이 유발하는 일시적 오류(transient errors)로 인하여 오동작할 수 있다[1-3]. 그러나 열악하고 급변하는 환경 속에서도 중단 없이 오랜 시간 지속적으로 동작하도록 요구되는 경우에는 이러한 일시적 오류를 극복해야 하며, 그 대표적인 방안으로 체크포인팅(checkpointing) 기법[4]을 꼽을 수 있다.

체크포인팅(checkpointing) 기법은 컴퓨터 시스템에서 실행되는 태스크(task)의 상태를 안정성이 높은 저장 장치에 주기적으로 저장하는 것을 의미한다. 이렇게 저장된 태스크의 상태를 체크포인트(checkpoint)라고 하며, 일시적 오류가 감지되었을 때 최근에 생성된 체크포인트를 이용하여 태스크를 오류 발생 이전 상태로 되돌리는 것을 롤백 복구(rollback recovery)라 한다.

체크포인팅 작업이 일정 시간 간격으로 수행되는 경우를 등간격 체크포인팅(equidistant checkpointing) 기법이라 하고, 체크포인팅 작업의 수행 시간 간격이 변화하는 경우를 적응형 체크포인팅(adaptive checkpointing) 기법이라 한다.

대부분의 등간격 체크포인팅 기법에 관한 이전 연구는 데이터베이스 시스템이나 범용 컴퓨터 시스템에서의 시스템의 가용성 최대화나 성능 평가, 프로그램의 평균 실행시간 최소화, 체크포인팅 부담 최소화 등이 주요 주제였으며[5,6], 실시간 시스템에 대해서는 태스크 평균 실행시간이나 응답 시간 분석 등에 관한 연구[7,8]가 주요 주제였었다[9,10]. [11]에서는 동시 결함 검출(concurrent fault detection)이 적용된 경우 즉, 일시적 결함의 발생으로 인한 오류가 즉시 검출되는 경우에 등간격 체크포인팅에 따른 시스템의 신뢰도에 대한 문제가 다루어졌다.

적응형 체크포인팅 기법에 관한 이전 연구[12-15]는 실시간 제어 시스템과는 성격이 다른 시스템들을 대상으로 이루어졌다. 실시간 제어 시스템에 대해 적용할 만한 적응형 체크포인팅 기법에 관한 이전 연구 중 [16-18]은 실시간 태스크의 실행에 소요되는 평균 시간을 최소화하여 요구되는 시간 제약을 만족할 수 있는 확률을 최대로 하는 기법을 제안하였다. 그리고 이전 연구 [9]와 [10]은 일시적 오류의 영향이 발생 즉시 검출되지 않고 일정 간격으로 수행되는 오류 검출 방식에 의해 검출되는 구조를 갖는 실시간 제어 시스템의 결함 허용성에 대한 문제를 다루었다. [9]에서는 등간격 체크포인팅 기법이 적용된 실시간 제어 태스크가 일시적 오류를 극복하고 성공적으로 실행될 확률을 최대화하는 기법이 제안되었고, [10]에서는 [9]의 결과를 바탕으로 하여 실시간 제어 태스크가 성공적으로 실행될 확률을 최대화하는 적응형 체크포인팅 기법이 제안되었다.

본 논문에서는 동시 결함 검출 기능이 구비된 즉, 결함의 발생을 즉시에 검출할 수 있는 기능이 구비된 실시간 제어 시스템의 신뢰도 향상을 목표로 하는 적응형 체크포인팅 기법을 제안하고 그 효과를 보인다. 실시간 제어 시스템이 일시적 오류를 극복하고 주어진 시간 조건을 만족하며 센서로부터의 입력에 따른 구동장치를 위한 출력 계산을 완료할 확률, 즉 주기적 실시간 제어 태스크(이하 태스크)의 1회 실행 성공 확률을 성능 지표로 정하고 이를 통하여 제안하는 적응형 체크포인팅 기법의 효과를 검토한다.

II 장에서는 대상 시스템의 모델 및 가정에 대해 기술하고, III 장에서는 적응형 체크포인팅 기법 유도를 위해 필요한 등간격 체크포인팅 기법에 관한 사항을 소개한다. 그리고 IV

* 책임저자(Corresponding Author)
논문접수: 2010. 6. 3., 수정: 2010. 10. 18., 채택확정: 2010. 11. 8.
류상문: 군산대학교 제어로봇공학과(smryu@kunsan.ac.kr)

장에서 적응형 기법을 제안하고 V 장에서 그 효과를 보인 후 VI 장에서 결론을 맺는다.

II. 가정 및 시스템 모델

이전 연구 [5-13,16-18]에서와 마찬가지로 일시적 오류는 평균값 λ 를 갖는 포아송(poisson) 분포에 따라 발생한다고 가정한다. 그리고 이러한 일시적 오류가 이전 연구 [19-25]에서 제안된 다양한 동시 검출 기법들이 시스템의 특성과 환경에 적합하게 적용되어 모두 검출된다고 가정한다. 실제로 모든 일시적 오류를 검출할 수 있는 완벽한 검출기법은 존재할 수 없으므로 적용된 오류 검출 기법의 성능은 관련 성능 지표인 오류 검출 범위(fault detection coverage) [1]를 고려하면 된다. 체크포인팅과 롤백 작업 구간의 길이는 태스크 실행 구간의 길이에 비해 매우 짧고, 체크포인팅과 롤백 작업에서 동작하는 하드웨어는 상대적으로 안정성이 높게 설계되므로 이들 구간에서의 일시적 오류의 발생 가능성은 상대적으로 매우 작아서 고려하지 않는다.

주기적 태스크가 센서의 값을 바탕으로 제어 출력을 계산하는데 소요되는 시간을 실행시간(execution time)이라 하고, 1회의 제어 출력 계산에 허용된 시간을 상대적 마감시간(relative deadline)이라고 한다[26]. 상대적 마감시간과 실행시간의 차이를 여유시간(slack time)이라 하며, 이 여유시간을 활용해서 주기적인 체크포인팅 작업과 필요한 롤백 복구를 수행하여 일시적 결함의 극복할 수 있는 것이다.

본 논문에서는 태스크의 실행시간, 여유시간, 체크포인팅 작업 소요 시간 그리고 롤백 복구 작업 소요 시간 각각 T_e, T_s, T_c, T_r 로 표현하기로 한다.

그림 1은 체크포인팅 기법이 적용된 경우에 태스크의 수행 중 나타날 수 있는 실행 패턴을 보여준다. 그림에서 E, C, R은 각각 태스크의 실행 구간, 체크포인팅 작업 그리고 롤백 복구 작업을 의미하며, 회색으로 표현된 것은 해당 구간에서 'x' 기호로 표현되는 일시적 오류가 발생하였음을 나타낸다. 그림 1(a)는 태스크의 1주기 실행 동안 일시적 오류의 발생에 대비해서 체크포인팅 작업을 수행했지만 일시적 오류가 발생하지 않아 롤백 복구 작업의 수행 없이 태스크의 1주기 수행이 완료된 경우를 보여준다. 그림 1(b)는 길이가 $T_{v,1}$인 첫 번째 태스크 실행 구간 완료 후 체크포인팅 작업이 수행되었고, 길이가 각각 T_{v2}와 T_{v3}인 두 번째, 세 번째 태스크 실행 구간에서 각각 $t_{f,1}$과 $t_{f,2}$ 시점에 일시적 오류가 발생하여 롤백 복구 작업이 수행된 경우를 보여준다.

체크포인팅 작업이 일정한 간격으로 수행되는 등간격 체크포인팅 기법의 경우에는 $T_{v,l}, l=1,2,\cdots$ 의 값이 모두 동일하게 되며, 본 논문이 제안하는 적응형 체크포인팅 기법에서는 태스크가 활용할 수 있는 잔여 여유시간과 태스크의 한 주기 수행을 완료하기 위해 요구되는 시간에 따라서 체크포인팅 작업 수행 시점이 동적으로 정해지므로 $T_{v,l}$ 의 값이 오류 발생 상황에 따라 변하게 된다.

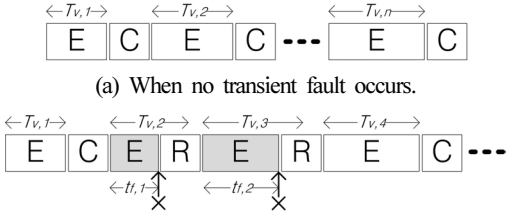

그림 1. 체크포인팅 기법이 적용된 태스크의 수행 패턴 예.
Fig. 1. Example execution patterns of a task with checkpointing.

III. 등간격 체크포인팅 기법에 대한 고찰

본 장에서는 등간격 체크포인팅 기법이 적용될 때 태스크의 1 주기가 성공적으로 실행될 즉, 시간 제약을 만족하고 1 주기 수행을 완료할 확률을 구한다. 이 결과는 다음 장의 적응형 체크포인팅 기법에서 이용된다.

태스크의 1 주기 동안에 T_v $(T_{v,l}=T_v, l=1,2,\cdots)$ 마다 체크포인팅 작업이 수행되면 태스크는 길이가 T_v 인 T_e/T_v 개의 작은 실행 구간들로 분리되고 T_e/T_v 회의 체크포인팅 작업이 수행된다. 그리고 등간격 조건이 만족되려면 T_e/T_v는 정수 값을 가져야 한다.

일시적 오류가 평균값 λ 를 갖는 포아송 분포로 발생한다고 가정했으므로 실행시간이 T_e 동안 일시적 오류가 발생하지 않을 확률은 $e^{-\lambda T_e}$ 이다. 태스크의 1 주기 동안 발생 가능한 일시적 오류의 수를 N_f 로 표기하면 N_f 는 이론적으로 1부터 ∞의 값을 가질 수 있다. 따라서 T_v 마다 체크포인팅 작업이 수행되는 경우 태스크의 1 주기가 성공적으로 실행될 확률을 $P_S(T_v)$ 라고 하면 $P_S(T_v)$ 는 다음과 같이 표현될 수 있다.

$$P_S(T_v)=e^{-\lambda T_e}\left[1+\sum_{N_f=1}^{\infty}\Pr\{N_f\text{개의 허용 가능한 오류 발생}\}\right] \quad (1)$$

태스크의 1 주기가 정상적으로 수행되기 위해서는 T_e/T_v개의 실행 구간들 중 마지막 실행 구간이 일시적 오류를 겪지 않고 수행되어야 하므로, T_e/T_v 개의 실행 구간들로 분리된 태스크에 N_f 개의 일시적 오류가 발생할 경우의 수는 (T_e/T_v+N_f-1) 개의 장소 중에서 임의로 N_f 개를 고르는 경우와 같다. 그리고 길이가 T_v인 실행 구간에서 일시적 오류가 발생할 확률은 $(1-e^{-\lambda T_v})$ 이다. 따라서 길이가 T_v 인 T_e/T_v 개의 작은 실행 구간들로 나뉘어진 태스크의 1 주기 수행 중 N_f개의 일시적 오류가 발생할 확률은 다음과 같다.

$$\binom{\frac{T_e}{T_v}+N_f-1}{N_f}(1-e^{-\lambda T_v})^{N_f} \quad (2)$$

발생 가능한 N_f 개의 오류는 각각 그림 1(b)에서 표현된 $t_{f,i}, i=1,2,\cdots,N_f$ 의 총합에 해당하는 시간을 태스크로 하여금 소비하게 한다. 태스크의 여유시간 T_s 가 일시적 오류의 발생에 의해 허비되는 시간 $t_{f,i}$ 의 합과 T_e/T_v 회의 체크포인팅 작업을 위해 소모되는 시간 그리고 N_f 개의 오류를 극복하기 위한 N_f 회의 롤백 작업에 소모되는 시간을 포용할 수 있어야 태스크가 상대적 마감시간 이내에 수행을 마칠 수

있다. 따라서 발생한 N_f 개의 오류가 허용 가능할 확률 즉, 태스크에 의해 포용 가능할 확률은 다음과 같이 표현될 수 있다.

$$\Pr\left\{\sum_{i=1}^{N_f} t_{f,i} \le (T_s - T_c\frac{T_e}{T_v} - T_r N_f)\right\} \quad (3)$$

(2)와 (3)를 (1)에 대입하면 태스크의 1 주기가 성공적으로 실행될 확률 $P_S(T_v)$ 는 다음과 같이 표현된다.

$$P_S(T_v) = e^{-\lambda T_e}\left[1+\sum_{N_f=1}^{\infty}\binom{\frac{T_e}{T_v}+N_f-1}{N_f}(1-e^{-\lambda T_v})^{N_f}\Pr\left\{\sum_{i=1}^{N_f} t_{f,i} \le (T_s - T_c\frac{T_e}{T_v} - T_r N_f)\right\}\right] \quad (4)$$

T_f 를 $T_f = \sum_{i=1}^{N_f} t_{f,i}$ 라 정의하면, (3)의 확률을 구하기 위해서 랜덤 변수 T_f 의 확률 밀도 함수를 구하여야 하는데 다음과 같은 과정을 통하여 구할 수 있다.

동일한 지수 함수 분포를 갖는 독립적인 랜덤 변수들의 합은 어랑(Erlang) 분포를 갖는 것으로 알려져 있다[27]. 만일 (5)의 지수 함수 분포를 확률 밀도 함수로 하며, (6)의 특성 함수(Characteristic function)를 갖고 서로 독립적인 n개의 랜덤 변수를 $t_i, i=1,2,\cdots,n$ 라고 하면, 이들의 합 $\sum_{i=1}^{n} t_i$ 로 정의된 랜덤 변수 S_n의 확률 밀도 함수와 특성 함수는 각각 (7)과 (8)로 표현된다[27].

$$f_{t_i}(x) = \lambda e^{-\lambda x},\ x \ge 0 \quad (5)$$

$$\Phi_{t_i}(\omega) = \frac{\lambda}{\lambda - j\omega} \quad (6)$$

$$f_{S_n}(x) = \frac{(\lambda x)^{n-1}}{(n-1)!}\lambda e^{-\lambda x},\ x \ge 0 \quad (7)$$

$$\Phi_{S_n}(\omega) = \left(\frac{\lambda}{\lambda - j\omega}\right)^n \quad (8)$$

랜덤 변수 $t_{f,i}$ 는 $[0, T_v]$ 구간에서 지수 함수 분포를 가지며 그 확률 밀도 함수와 특성 함수는 각각 (9)과 (10)과 같다.

$$f_{t_{f,i}}(x) = \begin{cases}\dfrac{\lambda e^{-\lambda x}}{1-e^{-\lambda T_v}}, & 0 < x < T_v \\ 0, & otherwise\end{cases} \quad (9)$$

$$\Phi_{t_{f,i}}(\omega) = \frac{\lambda}{\lambda - j\omega}\frac{1-e^{-\lambda T_v}e^{jT_v\omega}}{1-e^{-\lambda T_v}} \quad (10)$$

(10)로부터 랜덤 변수 T_f 의 특성 함수를 (11)와 같이 구할 수 있다.

$$\Phi_{T_f}(\omega) = (\Phi_{t_{f,i}}(\omega))^{N_f} = \left(\frac{\lambda}{\lambda - j\omega}\right)^{N_f}\frac{\sum_{m=0}^{N_f}\binom{N_f}{m}(-e^{-\lambda T_v})^m e^{jT_v m\omega}}{(1-e^{-\lambda T_v})^{N_f}} \quad (11)$$

(11)의 특성 함수로부터 랜덤 변수 T_f 의 확률 밀도 함수를 (12)와 같이 구할 수 있다.

$$f_{T_f}(x) = \frac{1}{\left(1-e^{-\lambda T_v}\right)^{N_f}}\sum_{m=0}^{N_f}\binom{N_f}{m}\left(-e^{-\lambda T_v}\right)^m f\left(x - mT_v\right) \quad (12)$$

$$f(x) = \begin{cases}\dfrac{(\lambda x)^{N_f-1}}{(N_f-1)!}\lambda e^{-\lambda x}, & x \ge 0 \\ 0, & x < 0\end{cases} \quad (13)$$

따라서 태스크의 1 주기가 성공적으로 실행될 확률 $P_S(T_v)$ 는 (4), (12) 그리고 (13)을 이용하여 (14)와 같이 구해진다.

$$\begin{aligned} P_S(T_v) &= e^{-\lambda T_e}\left[1+\sum_{N_f=1}^{\infty}\binom{\frac{T_e}{T_v}+N_f-1}{N_f}\left(1-e^{-\lambda T_v}\right)^{N_f}\int_0^{T_s - T_c\frac{T_e}{T_v} - T_r N_f} f_{T_f}(x)dx\right] \\ &= e^{-\lambda T_e}\left[1+\sum_{N_f=1}^{\infty}\binom{\frac{T_e}{T_v}+N_f-1}{N_f}\sum_{m=0}^{N_f}\binom{N_f}{m}\left(-e^{-\lambda T_v}\right)^m\int_0^{T_s - T_c\frac{T_e}{T_v} - T_r N_f} f\left(x - mT_v\right)dx\right] \end{aligned} \quad (14)$$

표 1은 (14)의 검증을 위하여 $T_e = 100,\ T_s = 20,\ T_c = 1,\ T_r = 2$ 인 태스크가 $\lambda = 0.001$ 인 환경에서 수행되는 상황을 가정하여 (14)의 결과와 시뮬레이션 결과를 비교한 것이다. 시뮬레이션은 T_e/T_v 값에 따라 태스크를 10^7회 반복 수행하여 얻은 결과이다.

표 1. (14)의 결과와 시뮬레이션 결과 비교.

Table 1. Comparison bet. the values from formula (14) and simulation data.

$\frac{T_e}{T_v}$	$P_S(T_v)$	시뮬레이션	
		평균	95% 신뢰 구간
5	0. 964090	0.964097	0.964009 - 0.964185
6	0. 970544	0.970453	0.970314 - 0.970592
7	0. 975152	0.975219	0.975135 - 0.975303
8	0. 977904	0.977960	0.977871 - 0.978048
9	0. 978796	0.978860	0.978761 - 0.978959
10	0. 977830	0.977794	0.977709 - 0.977878
11	0. 975011	0.975085	0.975003 - 0.975167
12	0. 970354	0.970305	0.970289 - 0.970420
13	0. 963874	0.963734	0.963606 - 0.963882
14	0. 955596	0.955650	0.955566 - 0.955735
15	0. 945548	0.945490	0.945424 - 0.945556

IV. 적응형 체크포인팅 기법

1. 적응형 체크포인팅 기법

적응형 체크포인팅 기법은 현재 태스크의 수행 상황에 따라 태스크의 1 주기가 시작하는 시점과 체크포인팅 또는 롤백 복구 작업이 수행 완료되는 시점에서 $l(=1,2,3,\cdots)$ 번째 체크포인팅 작업 수행 시점 $T_{v,l}$ 을 결정하여 이를 적용하는 것이다. $T_{v,l}$ 을 계산하는 시점에서 태스크가 활용할 수 있는 '잔여 여유시간'을 $T_{s,l}$, 태스크의 한 주기 수행을 완료하기

위한 '잔여 실행시간'을 $T_{e,l}$ 이라고 하면, 이들은 다음과 같이 계산된다.

- 첫 번째 ($l=1$) 실행 구간 수행 전:

$$\begin{aligned} T_{e,l} &= T_e \\ T_{s,l} &= T_s \end{aligned} \tag{15}$$

태스크의 1 주기가 실행되기 시작하는 시점이므로 잔여 여유시간과 잔여 실행시간은 각각 여유시간과 실행시간과 같다.

- $l(=1,2,3,\cdots)$ 번째 실행 구간이 일시적 오류를 겪지 않고 해당 체크포인팅 작업이 수행된 경우:

$$\begin{aligned} T_{e,l+1} &= T_{e,l} - T_{v,l} \\ T_{s,l+1} &= T_{s,l} - T_c \end{aligned} \tag{16}$$

l 번째 실행 구간이 수행되었으므로 잔여 실행시간은 $T_{v,l}$ 만큼 줄어들고 잔여 여유시간은 T_c 만큼 줄어든다.

- $l(=1,2,3,\cdots)$ 번째 실행 구간에서 오류가 발생하여 롤백 복구 작업이 수행된 경우:

$$\begin{aligned} T_{e,l+1} &= T_{e,l} \\ T_{s,l+1} &= T_{s,l} - T_{v,l} - T_r \end{aligned} \tag{17}$$

l 번째 실행 구간이 수행되지 못했으므로 잔여 실행시간은 줄어들지 않고 잔여 여유시간만 $T_{v,l}$ 과 T_r의 합만큼 줄어든다.

2. $T_{v,l}$ 의 계산

적응형 체크포인팅 기법에서는 l 번째 실행 구간이 수행되기 직전에 l 번째 체크포인팅 작업을 수행할 시기 즉, l 번째 실행 구간의 길이 $T_{v,l}$ 을 정하는 것이 매우 중요한 사항이다.

l 번째 실행 구간을 시작하려는 태스크는 여유시간과 실행시간이 각각 $T_{s,l}$ 과 $T_{e,l}$ 인 태스크로 여겨질 수 있다. 따라서 (14)에서 T_v, T_s, T_e 을 각각 $T_{v,l}$, $T_{s,l}$, $T_{e,l}$ 로 대체한 후 $P_S(T_{v,l})$ 을 최대로 하여주는 $T_{v,l}$ 을 구하여 이를 적용하면 태스크의 1 주기 실행 성공 확률을 최대화할 수 있다.

실제의 경우 일시적 오류 발생률 λ 는 매우 작아서 $\lambda \ll 1$ 을 만족하므로 (14)는 $N_f=1$ 인 우세항만을 이용하여 근사화하고 T_v, T_s, T_e 을 각각 $T_{v,l}$, $T_{s,l}$, $T_{e,l}$ 로 대체하여 (18)과 같이 표현할 수 있다.

$$P_S(T_{v,l}) \approx e^{-\lambda T_{e,l}} \left[1 + \frac{T_{e,l}}{T_{v,l}} \left(1 - e^{-\lambda \left(T_{s,l} - T_c \frac{T_{e,l}}{T_{v,l}} - T_r \right)} \right) \right] \tag{18}$$

(18)은 $e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots$ 을 이용하여 한 번 더 근사화하고 정리하면 (19)과 같이 된다.

$$P_S(T_{v,l}) \approx e^{-\lambda T_{e,l}} \left[1 + \lambda (T_{s,l} - T_r) \frac{T_{e,l}}{T_{v,l}} - \lambda T_c \left(\frac{T_{e,l}}{T_{v,l}} \right)^2 \right] \tag{19}$$

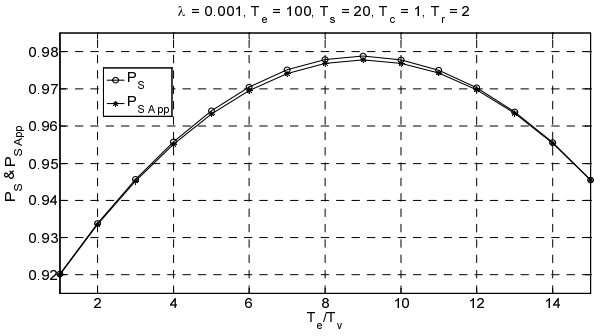

그림 2. 식 (14)와 근사식 (19)의 비교.

Fig. 2. Comparison bet. formula (14) and (19).

그림 2는 $\lambda = 0.001$, $T_e(=T_{e,l}) = 100$, $T_s(=T_{s,l}) = 20$, $T_c = 1$, $T_r = 2$ 에 대해 $T_e/T_v(=T_{e,l}/T_{v,l})$ 에 따른 원래의 (14)와 근사식 (19) 를 비교하여 보여준다. 그래프에서 P_S는 원래의 (14)를 P_{SApp}는 근사식 (19)를 나타내며 상당히 유사함을 알 수 있다.

근사식 (19)를 최대로 하여주는 $T_{v,l}$ 을 구하여 $T_{v,l}^*$ 로 나타내면 (20)과 같게 된다.

$$T_{v,l}^* = \frac{2T_cT_{e,l}}{T_{s,l} - T_r} \tag{20}$$

그리고 (20)의 $T_{v,l}^*$ 은 잔여 실행 구간의 길이 $T_{e,l}$ 보다 클 수 없으므로 $T_{v,l}^* \le T_{e,l}$ 을 만족해야 한다. 따라서 (21)의 조건이 만족되어야 한다.

$$T_{s,l} \ge 2T_c + T_r \tag{21}$$

이상으로부터 적응형 체크포인팅 기법에서 l 번째 실행 구간이 수행되기 직전에 l 번째 체크포인팅 작업을 수행할 시기를 (21)의 조건이 만족되는 경우에는 (20)의 $T_{v,l}^*$ 로 정하여 태스크의 실행 성공 확률을 개선할 수 있을 것으로 기대할 수 있다. (21)의 조건이 만족되지 않는 경우에는 잔여 여유시간이 부족함을 의미하므로 (22)과 같이 $T_{v,l}^*$ 의 값을 잔여 실행시간과 같게 하여 주어 태스크의 1주기 수행 완료를 도모한다.

$$T_{v,l}^* = T_{e,l} \tag{22}$$

따라서 본 논문에서 제안되는 적응형 체크포인팅 기법은 $l(=1,2,3,\cdots)$ 번째 실행 구간이 수행되기 직전에 l 번째 실행 구간의 길이 $T_{v,l}$ 을 다음의 규칙을 이용하여 계산한다.

```
if (T_{s,l} ≥ 2T_c + T_r) then
    T_{v,l} = 2T_cT_{e,l} / (T_{s,l} - T_r)
else
    T_{v,l} = T_{e,l}
end if
```

그림 3. 제안된 적응형 체크포인팅 기법에서의 $T_{v,l}$ 계산.

Fig. 3. Calculation of $T_{v,l}$ in the proposed adaptive checkpointing scheme.

V. 성능 고찰

그림 4와 표 2는 논문에서 제안되는 적응형 체크포인팅 기법의 성능을 등간격 체크포인팅 기법의 성능과 비교한 것을 보여준다. $T_e = 100$, $T_c = 1$, $T_r = 2$ 인 태스크가 $\lambda = 0.001$ 인 환경에서 수행되는 상황을 가정하였으며, 주어진 여유시간 T_s를 활용하여 등간격 체크포인팅 기법을 적용하여 얻을 수 있는 최대의 태스크 성공적 실행 확률 P_E과 앞서 제안된 적응형 체크포인팅 기법을 적용하여 얻을 수 있는 태스크 성공적 실행 확률 P_A를 함께 나타낸 것이다. P_E는 주어진 여유시간 T_s에 대해서 얻을 수 있는 (14)의 최대값을 구하였고, P_A는 제안되는 적응형 체크포인팅 기법을 적용하여 태스크를 10^7 회 반복 수행한 시뮬레이션을 통하여 얻었다.

그림 4와 표 2로부터 적응형 기법이 등간격 기법에 비해 신뢰도 성능면에서 뛰어남을 알 수 있다. 상대적으로 적은 여유시간으로도 태스크의 실행 성공 확률이 커지므로 다수의 제어 태스크가 실행되는 시스템에서는 그 효과가 더욱 클 것이다.

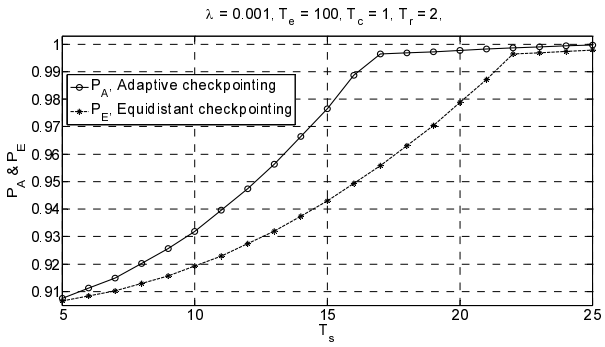

그림 4. 등간격 기법과 적응형 기법의 성능 비교.

Fig. 4. Performance comparison bet. the equidistant scheme and the adaptive scheme.

표 2. 등간격 기법과 적응형 기법의 성능 비교.

Table 2. Performance comparison bet. the equidistant scheme and the adaptive scheme.

T_s	P_E	P_A	
		평균	95% 신뢰 구간
5	0.906645	0.907550	0.907385 - 0.907715
6	0.908453	0.911223	0.911064 - 0.911383
7	0.910259	0.914871	0.914713 - 0.915029
8	0.912971	0.920262	0.920068 - 0.920456
9	0.915684	0.925688	0.925517 - 0.925860
10	0.919304	0.931801	0.931639 - 0.931964
11	0.922929	0.939577	0.939487 - 0.939666
12	0.927462	0.947368	0.947264 - 0.947472
13	0.932009	0.956271	0.956138 - 0.956403
14	0.937465	0.966394	0.966294 - 0.966495
15	0.942944	0.976567	0.976469 - 0.976665
16	0.949335	0.988670	0.988598 - 0.988742
17	0.955761	0.996447	0.996428 - 0.996467
18	0.963101	0.996896	0.996862 - 0.996929
19	0.970490	0.997302	0.997270 - 0.997335
20	0.978796	0.997780	0.997764 - 0.997796
21	0.987168	0.998239	0.998222 - 0.998257
22	0.996461	0.998663	0.998645 - 0.998680
23	0.996885	0.999053	0.999039 - 0.999066
24	0.997330	0.999427	0.999419 - 0.999435
25	0.997850	0.999671	0.999656 - 0.999685

VI. 결론

본 논문에서는 일시적 결함에 의한 일시적 오류가 발생과 동시에 검출될 수 있는 결함 허용 실시간 제어 시스템에 대해 적용할 수 있는 적응형 체크포인팅 기법을 제안하였다. 제안된 적응형 체크포인팅 기법이 적용된 실시간 제어 태스크가 상대적 마감시간을 만족할 수 있는 확률을 수학적으로 표현할 수 없기 때문에 시뮬레이션을 통하여 그 효과를 보였다.

제안된 기법은 실시간 태스크의 1 주기 실행 동안에 잔여 여유시간과 잔여 실행시간을 고려하여 체크포인팅 작업 수행 시기를 동적으로 설정하여 실시간 태스크가 결함을 극복하고 상대적 마감시간을 만족할 확률을 최대화하도록 하는 것이다. 과거의 오류 발생 상황을 고려하기 때문에 태스크에게 주어진 여유시간을 더욱 효율적으로 사용할 수 있는 것이다. 따라서 기존의 등간격 기법보다 신뢰도 성능을 더욱 향상할 수 있다. 그림 3에서 보인 것과 같이 체크포인팅 작업 수행 시기 결정에 필요한 계산량이 많지 않기 때문에 실시간 제어 시스템의 성능에는 큰 영향이 없다고 판단된다.

참고문헌

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

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

[3] 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.

[4] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Engineering*, vol. 1, no. 2, pp. 220-232, June 1975.

[5] A. Ziv and J. Bruck, "An on-line algorithm for checkpoint placement," *IEEE Trans. Computers*, vol. 46, no. 9, pp. 976-985, Sep. 1997.

[6] 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.

[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] 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.

[9] S.-M. Ryu, "Performance analysis of checkpointing and dual modular redundancy for fault tolerance of real-time control system," *Journal of Institute of Control, Robotics and Systems*, vol. 14, no. 4, pp. 376-380, Apr. 2008.

[10] S.-M. Ryu, "An adaptive checkpointing scheme for fault tolerance of real-time control systems," *Journal of Institute of Control, Robotics and Systems,* vol. 15, no. 6, pp. 598-602, Jun. 2009.

[11] S.-M. Ryu and D.-J. Park, "Checkpointing for the reliability of real-time systems with on-line fault detection," *Lecture Notes in*

Computer Science, no. 3824, pp. 194-202, Aug. 2005.

[12] C.-M. Lin and C.-R. Dow, "Efficient techniques for adaptive independent checkpointing in distributed systems," *IEICE Trans. on Information & Systems*, vol. E83-D, no. 8, pp. 1642-1653, Aug. 2000.

[13] N. Chen and S. Ren, "Architecture support for behavior-based adaptive checkpointing," *Journal of Software*, vol. 3, no. 2, pp. 61-68, Feb. 2008.

[14] Y. Gao, C. Deng, and Y. Che, "An adaptive index-based algorithm using time-coordination in mobile computing," *Proc.* of *2008 International Symposiums on Information Processing*, pp. 578-585, May 2008.

[15] M. Chtepen, F. Claeys, B. Dhoedt, F. Turck, P. Vanrolleghem, and P. Demeester, "Providing fault-tolerance in unreliable grid systems through adaptive checkpointing and replication," *Lecture Notes in Computer Science*, vol. 4487, pp. 454-461, 2007.

[16] Z. Li, H. Chen, and S. Yu, "Performance optimization for energy-aware adaptive checkpointing in embedded real-time systems," *Proc. the conference on Design, Automation and Test in Europe*, pp. 678-683, Mar. 2006.

[17] Y. Xiang, Z. Li, and H. Chen, "Optimizing adaptive checkpointing schemes for grid workflow systems," *Proc. the Fifth International Conference on Grid and Cooperative Computing Workshops*, pp. 181-188, Oct. 2006.

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

[19] S. M. A. H. Jafri, et al., "Design of a fault-tolerant coarse-grained reconfigurable architecture: a case study," *Proc. of 2010 11th International Symposium on Quality Electronic Design*, pp. 845-852, 2010.

[20] M. Maniatakos, et al., "Instruction-level impact analysis of low-level faults in a modern microprocessor controller," *IEEE Transactions on Computers*, vol. 59, 2010.

[21] L. Costas-Perez and J. J. Rodriguez-Andina, "Algorithmic concurrent error detection in complex digital-processing systems," *IEEE Design & Test of Computers*, vol. 26 , no. 1, pp. 60-67, Jan.-Feb. 2009.

[22] M. Richter and M. Goessel, "Concurrent checking with split-parity codes," *Proc. of 15th IEEE International On-Line Testing Symposium*, pp. 159-163, Jun. 2009.

[23] V. S. Veeravalli, "Fault tolerance for arithmetic and logic unit," *Proc. IEEE Southeastcon 2009*, pp. 329-334, Mar. 2009.

[24] R. Vemu, et al., "A low-cost concurrent error detection technique for processor control logic," *Proc. Design, Automation and Test in Europe 2008*, pp. 897-902, 2008.

[25] C. Yen and B. Wu, "Simple error detection methods for hardware implementation of advanced encryption standard," *IEEE Trans. on Computers*, vol. 55, no. 6, pp. 720-731, Jun. 2006.

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

[27] A. Leon-Garcia, *Probability and Random Processes for Electrical Engineering*, Addison Wesley, 1994.

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.