성능 함수를 고려한 실시간 제어 태스크에서의 최적 체크 포인터 구간 선정

論 文
57-5-22

Determination of Optimal Checkpoint Interval for Real-time Control Tasks Considering Performance Index Function

郭 成 祐† · 鄭 容 朱*
(Seong-Woo Kwak · Young-Joo Jung)

Abstract - In this paper, a novel method to determine the optimal checkpoint interval for real-time control task is proposed considering its performance degradation according to tasks's execution time. The control task in this paper has a specific sampling period shorter than its deadline. Control performance is degraded as the control task execution time is prolonged across the sampling period and eventually zero when reached to the deadline. A new performance index is defined to represent the performance variation due to the extension of task execution time accompanying rollback fault recovery. The procedure to find the optimal checkpoint interval is addressed and several simulation examples are presented.

Key Words : Checkpoint, Performance index, Real-time control task, Transient fault, Deadline, Sampling period

1. 서 론

최근 산업 현장에서 발생하는 시스템 고장들을 분석한 결과에 따르면 소자의 영구적 결함과 같은 하드웨어적인 원인보다는 외부의 전기적, 기계적 환경변화등과 같은 과도 고장에 의한 것이 다수를 이루고 있는 것으로 보고되고 있다[1]. 이와 같은 과도 고장은 중복구조를 이용한 하드웨어적 방법보다는 체크 포인터 삽입등과 같은 소프트웨어적인 방법으로 더 잘 대처할 수 있다. 최근에는 컴퓨터의 처리 성능 향상과 OS(Operating System) 기술의 발전으로 체크 포인팅 기법을 실제 실시간 시스템에 적용하는 것이 가능하여졌다.

실시간 제어 태스크들은 매 샘플링 시간마다 일정한 제어 알고리즘을 반복적으로 수행함으로서 시스템을 제어한다. 샘플링 주기는 시스템의 제어 안정성을 고려하여 선정되며, 제어 태스크의 데드라인에 비해 짧게 결정된다. 체크포인팅 기법은 태스크의 데드라인 이내의 여유시간을 이용하여 과도 고장을 극복하는 방식이다. 과도 고장이 발생한 경우 해당 체크포인터 구간을 재 수행함으로서 고장을 극복한다. 따라서 이러한 고장 극복 과정은 태스크의 수행 시간 연장을 가져온다. 연장된 태스크 수행시간이 샘플링 주기이내에 종료되면 시스템의 제어 성능에 큰 영향을 미치지 않지만 샘플링 주기를 넘어서면 제어 성능의 감소로 이어진다. 이때 수행시간이 데드라인을 초과하는 경우 시스템은 Fail이 된다. 따라서 고장 극복 시간이 늘어남으로 인해 발생하는 시스템의 성능 감소를 고려하여 체크포인팅 구간을 설정하여야 한다. 태스크 수행시간 변화에 따른 성능 변화를 고려하여 체크 포인팅 기법을 설계한 연구는 현재까지 없었다.

체크 포인팅 기법이 탑재된 실시간 시스템을 해석하여 최적의 체크 포인팅 방법을 구하려는 연구는 기존의 다수 연구자들에 의해 연구 되어왔다[2,3,4,5,6]. Geist et. al[4]는 태스크의 수행 시간을 최소화하는 체크 포인터 삽입 방법을 연구 하였고, Shin et. al[5]은 평균 수행 시간을 최소화 시키는 체크 포인터 삽입 방법을 연구하였다. 논문 저자도 이러한 연구를 수행 한바 있다[7,8,10,11]. 하지만 기존 연구의 대부분은 샘플링 주기와 데드라인이 동일하다는 가정 하에 샘플링 주기를 벗어나는 고장극복은 시스템 Fail로 간주하였다. 이들 연구 결과들은 샘플링 주기와 데드라인이 서로 같다는 현실적이지 않은 측면이 있었다. 일반적으로 실시간 제어 태스크의 설계에 있어서 샘플링 주기는 데드라인보다 짧게 선정된다.

본 논문에서는 샘플링 주기와 데드라인이 다른 경우에 대하여 체크포인터 구간을 결정하는 문제를 해결하였다. 태스크의 실행 시간이 샘플링 주기를 벗어난 경우 태스크의 성능이 감소하기 시작하여 데드라인을 초과하면 성능이 0으로 된다는 것을 기반으로 태스크의 수행 시간 변화가 고려된 새로운 성능 지수를 정의한다. 이 성능 지수는 과도 고장 발생율에 따른 태스크의 실행 시간 확률 분포로부터 유도하였다. 이 성능 지수를 최대화 하는 구간이 최적 체크포인터 구간이 된다.

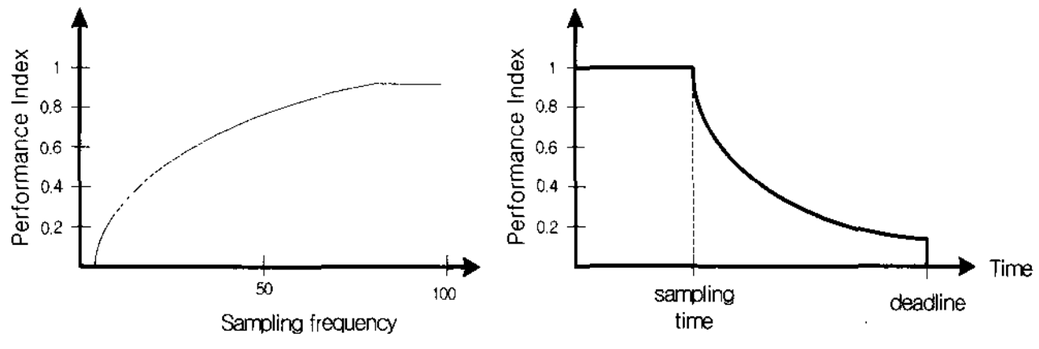
본 논문의 구성은 다음과 같다. 2장에서는 실시간 태스크의 일반적인 성능함수에 대하여 알아보고, 3장에서는 과도 고장 발생을 Poisson 확율 분포를 이용하여 모델링한다. 4장에서는 태스크의 수행 시간 확률 분포를 바탕으로 새로운 성능지수를 정의하고 최적 체크포인터 구간을 선정한다. 5장에서 몇 가지 예제에 대한 시뮬레이션을 통하여 최적 체크포인터 구간을 구하고 6장에서 결론을 맺는다.

† 교신저자, 正會員 : 啓明大 工大 電子工學科 助敎授 · 工博
E-mail : ksw@kmu.ac.kr
* 正 會 員 : 啓明大 工大 電子工學科 副敎授 · 工博
接受日字 : 2008年 3月 6日
最終完了 : 2008年 3月 24日

2. 실시간 제어 태스크의 성능 함수

디지털 제어 컴퓨터는 제어 알고리즘이 구현된 태스크를 일정한 샘플링 주기로 실행함으로서 제어 임무를 수행한다. 샘플링 주기는 제어 대상 플랜트의 주파수 특성을 고려하여 결정된다. 일반적으로 샘플링 주파수가 증가함에 따라 시스템의 성능 함수는 증가하다 포화(saturation)되고, 샘플링 주파수가 감소하면 성능 함수는 감소하며 일정 주파수 이하에서는 시스템이 불안정(unstable) 해진다[12]. 시스템이 불안정해지는 샘플링 주기가 제어 시스템 데드라인(deadline)이 된다.

그림 1(a)는 샘플링 주기에 따른 시스템의 성능 함수의 한 예를 나타낸 것으로 최대값을 1로 표준화하 하였다[12]. 그림 1(a)와 같은 성능 함수를 바탕으로 제어 태스크의 샘플링 주기가 결정된다. 일단 샘플링 주기가 결정되면 이 주기마다 디지털 컴퓨터(제어기)에서는 태스크를 실행하여 제어기 출력을 제어 대상 플랜트로 내보낸다. 하지만 고장 극복 등의 이유로 제어 태스크의 수행 시간이 늘어나 샘플링 주기를 넘어서면 샘플링 주파수가 줄어드는 효과가 나타나 시스템의 성능이 감소하고 데드라인을 넘어서면 시스템이 불안정해진다. 따라서 제어 태스크의 수행시간에 따른 시스템의 성능 함수는 그림 1(b)와 같이 나타낼 수 있다. 태스크가 샘플링 주기 이내에서 수행이 끝나는 경우 최대 성능을 나타내며 수행 시간이 늘어남에 따라 감소한다. 이때 수행 시간이 데드라인을 넘어서면 성능 함수 값이 0이 된다.



(a) Performance index vs. sampling frequency

(b) Performance index function

그림 1 디지털 제어 시스템의 성능 함수

Fig. 1 Performance Index Function for Digital Control System

그림 2는 디지털 컴퓨터(제어기)에서 태스크가 수행되는 형태를 나타낸 것이다. 매 샘플링 주기 T 마다 제어 태스크가 수행되고 태스크의 수행이 끝나면 제어기 출력을 제어 대상 플랜트로 내보낸다. 태스크의 수행 중 고장이 발생하면 이를 극복하기 위한 고장 극복 알고리즘이 수행된다. 본 논문에서는 고장 극복을 위해 최근의 체크포인트로 회귀하는 동작을 수행한다. 이러한 고장 극복 과정은 태스크 수행 시간의 연장을 가져와 경우에 따라서는 샘플링 시간을 초과할 수 있고 최악의 경우 데드라인을 초과할 수도 있다. 이때 태스크의 수행 시간이 샘플링 시간을 초과하게 되면 필연적으로 시스템의 성능 감소를 가져온다.

그림 2의 예 에서는 kT ~ $(k+1)T$ 구간의 태스크 수행 중 고장이 발생하고, 이를 복구하기위한 고장 극복 과정으로 인해 태스크의 수행 시간이 샘플링 주기를 넘어 $(k+1)T$ 시간 이후까지 연장된다. 태스크 수행이 완료된 시점에 제어기 출력이 발생하며 다음 샘플링 주기 $(k+1)T$을 지났으므로 다음 주기에서의 태스크 수행이 바로 시작된다.

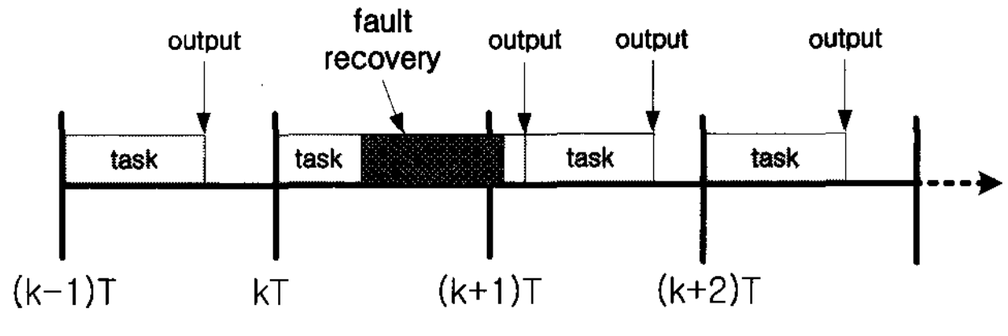


그림 2 제어 태스크의 실행 형태

Fig. 2 Execution pattern of Control Task

본 논문에서 다루는 제어 시스템의 성능 함수는 다음과 같은 특성을 가진 것으로 가정한다.

A.1 제어 태스크의 수행 시간 연장에 따라 그림 2(b)와 같이 샘플링 주기를 넘어서면 시스템의 성능이 지수적으로 감소하고 데드라인에서 "0"이 되는 값을 가진다.

그림 2(b)의 성능 함수는 다음과 같은 수식으로 나타낼 수 있다.

$$w(t)=\begin{cases}1 & \text{if } t<T\\ e^{-a(t-T)} & \text{if } T\le t<D\\ 0 & \text{if } t\ge D\end{cases} \tag{1}$$

3. 과도 고장 발생 및 극복

디지털 제어 시스템이 직면하는 고장들 중 많은 부분이 과도 고장에 기인하는 것으로 보고되고 있다[1]. 본 논문에서는 과도 고장과 관련하여 다음과 같이 가정한다.

A2. 고장은 고장 발생율 λ를 가진 Poisson 분포에 따라 발생한다.

A3. 고장에 대한 탐지는 체크 포인터 삽입 시 고장 탐지 알고리즘에 의해 수행되며 발생한 모든 고장은 탐지된다.

과도 고장이 λ의 고장 발생율 가진 Poisson 분포에 따라 발생한다고 하면, t 시간 내에 n개의 고장이 발생할 확률은 다음과 같다.

$$\alpha_{n(\lambda,t)}=\frac{(\lambda t)^n}{n!}e^{-\lambda t} \tag{2}$$

따라서 Δ 시간 구간에 고장이 없을 확률은

$$p=\alpha_0(\lambda,\Delta)=e^{-\lambda\Delta} \tag{3}$$

이며, Δ 시간 구간에 1개 이상의 고장이 발생할 확률은 다음과 같다[11].

$$q=\sum_{n=1}^{\infty}\frac{(\lambda\Delta)^n}{n!}e^{-\lambda\Delta}=1-e^{-\lambda\Delta} \tag{4}$$

과도 고장은 실시간 태스크에 체크포인터를 삽입하여 극복할 수 있다. 체크 포인터에서는 태스크의 모든 상태를 저

장하고, 이들 상태에 오류가 있는지 검증(고장 탐지)하는 2가지 과정을 수행한다. 그림 3은 실시간 태스크에 과도 고장이 발생한 경우, 체크포인터에서 발생한 고장을 탐지하고, 고장이 발생한 구간을 재수행 함으로써 고장을 극복하는 과정을 나타낸 것이다. 재수행 과정은 가장 최근의 체크 포인터에서 저장된 태스크 상태를 불러와 그 시점부터 태스크의 수행을 재개하는 것이다. 하지만 그림 3에서 보이듯이 고장이 발생한 구간을 재수행하는 것은 태스크의 수행시간 연장을 가져온다. 태스크의 연장된 시간은 데드라인 이내이어야 하며, 경우에 따라서는 샘플링 주기를 초과할 수 있다. 하지만 제어 태스크의 실행시간이 샘플링 주기를 초과하면 제어 성능이 감소하고 데드라인을 초과하면 시스템이 Fail 한다. 따라서 제어 성능을 최대화하기 위해서는 가능한 매 샘플링 주기 이내에 제어기의 출력이 제어 대상 플랜트로 전달되도록 하는 것이 가장 효율적이다.

체크 포인터가 삽입되는 구간을 어떻게 결정하느냐에 따라 재수행 구간의 길이가 결정되고, 고장 극복 확율 및 태스크의 수행시간이 달라진다. 체크포인터 구간을 줄이면 재수행 구간이 줄어들지만 체크 포인팅을 위한 부담(overhead)은 늘어난다[10,11]. 따라서 체크포인터에 수반된 오버헤드, 고장 극복에 따른 태스크 수행시간의 연장 및 이에 따른 성능 감소 등을 고려하여 체크포인터 구간을 설정하여야 한다.

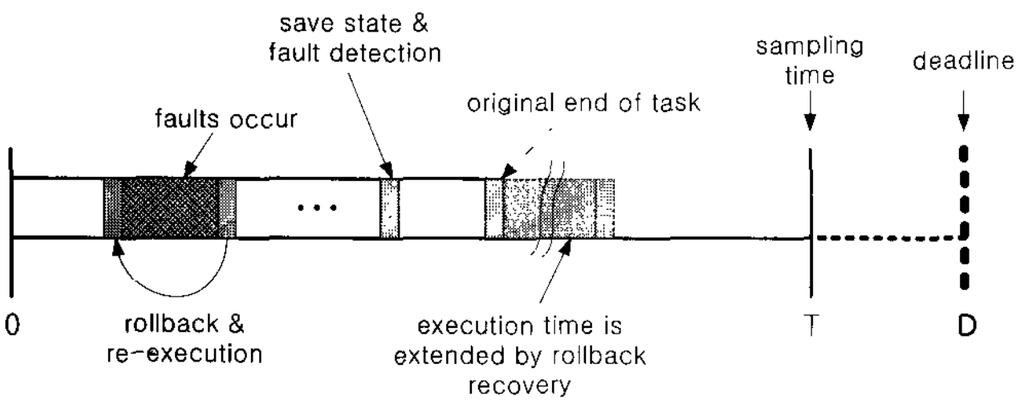


그림 3 체크포인터 삽입을 통한 고장 극복

Fig. 3 Fault Recovery using Checkpoint Placement

4. 성능 함수를 고려한 최적 체크포인터 구간

실시간 제어 태스크의 샘플링 주기를 T, 데드라인을 D라고 하고, 고장극복을 위해 제어 태스크에 삽입된 체크 포인터 수를 n 이라고 하자. 또한 체크 포인터 오버헤드(overhead)는 t_{cp} , 체크포인터가 삽입되기 전 제어 태스크의 원래 실행 시간은 E, 태스크의 체크포인터 삽입 구간을 Δ로 표시하자. 체크 포인터 오버헤드(overhead) t_{cp}는 태스크의 상태를 저장하는데 걸리는 시간과 고장 탐지알고리즘을 수행하는데 걸리는 시간이다. 제어 태스크에 삽입된 체크 포인터 수가 n이므로 체크포인터 구간 Δ는 다음과 같이 구할 수 있다.

$$\Delta = \frac{E}{n} + t_{cp} \tag{5}$$

따라서 체크포인터가 n개 삽입된 후 제어 태스크의 실행시간 e 는 다음과 같다[11].

$$e = n\Delta = n \cdot \left(\frac{E}{n} + t_{cp}\right) = E + nt_{cp} \tag{6}$$

즉 태스크의 원래 실행 시간에 체크 포인터를 삽입 하는데 걸리는 시간이 더해져 실행 시간이 nt_{cp} 만큼 늘어난다.

체크 포인터를 이용한 고장 극복은 태스크가 자신의 샘플링 주기 및 데드라인 이내에 사용할 수 있는 여유 시간을 이용하여 고장이 발생한 부분을 재수행하는 것이다. 따라서 몇 개의 체크 포인터를 삽입해야 하는가는 고장 발생율 뿐만 아니라 태스크의 샘플링 시간 이내의 여유시간, 데드라인 내에서의 여유시간, 태스크의 성능 지수 함수와 밀접한 관계가 있다.

고장이 발생하지 않은 상황에서 각 태스크가 가질 수 있는 샘플링 주기 내에서의 여유 시간(S_T)과 데드라인 이내에서의 여유시간(S_D)은 다음과 같이 각각 구할 수 있다.

$$S_T = T - e \tag{7}$$

$$S_D = D - e \tag{8}$$

따라서 각 여유시간에서 사용할 수 있는 재수행 구간의 수는 다음과 같다.

$$n_T = \left\lfloor \frac{S_T}{\Delta} \right\rfloor = \left\lfloor \frac{T-e}{\Delta} \right\rfloor \tag{9}$$

$$n_D = \left\lfloor \frac{S_D}{\Delta} \right\rfloor = \left\lfloor \frac{D-e}{\Delta} \right\rfloor \tag{10}$$

여기서 $\lfloor x \rfloor$ 는 x를 초과하지 않는 최대 정수 이다.

제어 태스크가 샘플링 주기 T 이내에서 고장을 극복하기 위해서는 n_T 개의 재수행을 통하여 고장을 복구할 수 있어야 하고, 데드라인 D 이내에서 복구하기 위해서는 n_D 개의 재수행을 통하여 고장을 복구할 수 있어야 한다. A2.~A3 의 가정에 따라 과도 고장이 발생하고 발생된 고장이 체크포인터에서 탐지 된다고 하면 샘플링 주기 이내에서 고장 복구를 완료하고 태스크 수행이 끝날 확률(P_T)은 다음과 같다.

$$P_T = \sum_{k=0}^{n_T} {}_{n-1-k}C_k \, p^n q^k = e^{-n\lambda\Delta} \sum_{k=0}^{n_T} {}_{n-1+k}C_k \, (1-e^{-\lambda\Delta})^k \tag{11}$$

또한 데드라인 이내에 태스크의 수행이 끝날 확률(P_D)은 다음과 같이 구할 수 있다.

$$P_D = \sum_{k=0}^{n_D} {}_{n-1+k}C_k \, p^n q^k \qquad (12)$$
$$= e^{-n\lambda\Delta} \sum_{k=0}^{n_D} {}_{n-1+k}C_k \, (1-e^{-\lambda\Delta})^k$$

여기서 인덱스 k는 고장 극복 과정에서 재수행한 체크 포인터 구간(Δ)의 수를 나타낸다. 태스크의 수행이 끝나기 위해서는 n개의 구간에 고장이 없어야 한다. 따라서 태스크 수행 중 k개 구간에 고장이 발생한 경우 $(n+k)$ 구간 후에 태스크의 수행이 끝난다. 태스크의 수행이 끝나는 마지막 $(n+k)$ 번째 구간은 항상 고장 없이 수행되어야 하므로 k개의 고장은 마지막 구간을 제외한 $(n+k-1)$ 개의 구간에서 존재할 수 있다. $(n+k-1)$구간에 k개의 고장 구간이 있을 경우의 수는 ${}_{(n+k-1)}C_k$ 이다.

체크 포인터 구간에 고장이 없을 확률은 p이고 고장이 존재할 확률이 q 이므로 n개 구간에 고장이 없고 k개 구간에는 고장이 있으면서 $(n+k)$번째 구간에서 태스크의 수행이 끝날 확률은 ${}_{(n+k-1)}C_k p^n q^k$ 이다. 샘플링 시간 내에는 n_T 개의 재수행을 위한 여유 구간이 존재하고, 데드라인 내에는 n_D 개의 여유 구간이 존재하므로 P_T 와 P_D는 위의 식(11)과 식(12)로 구할 수 있다.

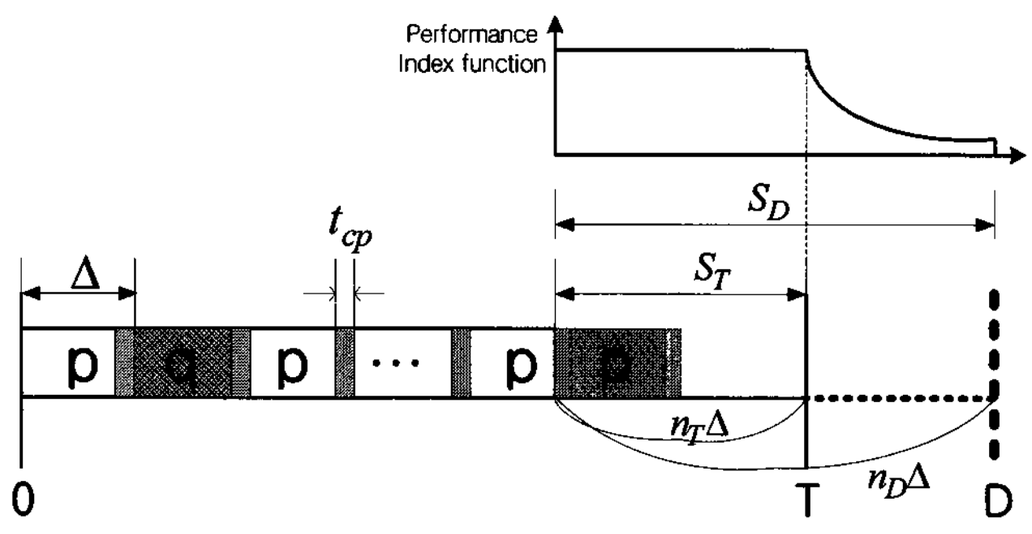


그림 4 태스크의 수행시간과 성능 지수 함수
Fig. 4 Task Execution Time and Performance Index Function

제어 태스크의 성능 함수는 그림 1(b) (식(1))에서와 같이 태스크의 수행이 샘플링 시간을 초과함에 따라 감소하여 데드라인이 초과되면 0이 된다. 그림 4는 태스크의 연장된 수행시간에 따른 성능 함수를 동시에 표시한 것 이다. 그림에서 "p" 로 표시된 구간은 고장이 발생하지 않은 곳이며, "q" 로 표시된 구간은 고장이 발생하여 재수행하여야 하는 구간이다. 재수행 구간이 존재하면 태스크의 수행 시간이 연장되며 마지막 구간이 고장 없이 수행되었을 때 태스크의 수행이 끝난다. 태스크의 수행시간에 따른 성능함수의 변화를 그림 4의 상단부분에 겹쳐 나타내었다.

태스크의 수행 시간은 고장 발생율, 체크포인터 구간, 체크포인터 오버헤드 등에 따라 달라지고 일정한 값으로 고정된 것이 아니라 확률 분포를 따른다. 따라서 태스크의 수행시간 확률을 고려한 제어 태스크의 성능 지수(J)를 다음과 같이 새로이 정의한다.

$$J = \int_0^\infty P(t) \bullet w(t) dt \qquad (13)$$

여기서 $P(t)$는 태스크의 수행시간이 t일 확률이고, $w(t)$는 식(1)로 표현되는 성능 함수이다.

$w(t)$는 $t \le T$ 일 때 1이고, $t \ge D$ 인 경우는 0이므로 위식은 다음과 같이 나타낼 수 있다.

$$J = \int_0^\infty P(t) \bullet w(t) dt \qquad (14)$$
$$= P(t \le T) + \int_T^D P(t) \bullet w(t) dt$$
$$= P_T + \int_T^D P(t) \bullet w(t) dt$$

고장 탐지는 체크포인터 삽입 시 수행되므로 태스크의 수행 시간은 항상 체크포인터 구간(Δ)의 정수배만 가능하다. 따라서 식(14)는 다음과 같이 나타낼 수 있다.

$$J = \sum_{k=0}^{n_D} {}_{n-1+k}C_k \bullet w((n+k)\Delta) \bullet p^n q^k$$
$$= P_T + \sum_{k=n_T+1}^{n_D} {}_{n-1+k}C_k \bullet w((n+k)\Delta) \bullet p^n q^k \qquad (15)$$

위 식(15)에 식(11)의 P_T 와 식(1)의 $w(t)$를 대입하면 다음과 같다.

$$J = e^{-n\lambda\Delta} \sum_{k=0}^{n_T} {}_{n-1+k}C_k \bullet (1-e^{-\lambda\Delta})^k$$
$$+ e^{-n\lambda\Delta} \sum_{k=n_T+1}^{n_D} {}_{n-1+k}C_k \bullet e^{-a[(n+k)\Delta - T]} (1-e^{-\lambda\Delta})^k \qquad (16)$$

위의 성능 지수 J 를 최대로 하도록 체크포인터 구간을 선정하는 것이 최적이 된다. 고장이 없는 경우 샘플링 주기 이내에서 태스크의 수행이 끝나야 하므로 태스크에 삽입할 수 있는 최대 체크 포인터 수는 다음과 같다.

$$\overline{n} = \left\lfloor \frac{T-E}{t_{cp}} \right\rfloor \qquad (17)$$

따라서 J의 값이 최대가 되는 최적 체크 포인터 수(n^*)는 다음 식(18)과 같이 구할 수 있다.

$$n^* = \underset{n}{Max} \{J\}, \quad 1 \le n \le \overline{n} \qquad (18)$$

따라서 제어 태스크의 최적 체크포인터 구간은 다음과 같다.

$$\Delta^* = \frac{E}{n^*} + t_{cp} \qquad (19)$$

5. 시뮬레이션 결과

몇 가지 간단한 예제에 대하여 본 논문에서 제시한 성능 지수를 바탕으로 최적 체크 포인터 수를 구하였다. 태스크의 실행 시간 $E=0.4$, 샘플링 주기 $T=1$, 데드라인 $D=2$ 인 경우를 생각한다. 고장 발생율 $\lambda=5$, 체크포인터 오버헤드는 $t_{cp}=0.02$, 태스크의 성능 함수는 식(1)과 같고 $a=1$로 가정한다. 최적의 체크포인터 수를 구하기 위하여 식(16)의 성능 지수 J 를 최대로 하는 n^* 값을 구하였다. 그림 5는 체크 포인터 수의 변화에 따른 성능 지수 J 의 변화를 나타낸 것이다. $n^*=11$ 일 때 J 가 최대가 되므로 이것이 최적 체크포인터 수이다.

그림 6은 $E=0.6$ 인 경우의 체크포인터 수에 따른 성능 지수 J 의 변화를 나타낸 것이다(나머지 파라메타는 그림 5의 경우와 동일). 이 경우 $n^*=12$ 이 최적 체크포인터 수가 된다. 그림 7은 $E=0.7$, $\lambda=1$ (나머지 파라메타는 동일)인 경우로 최적 체크포인터 수는 $n^*=9$이다.

그림 5~그림 7에서 보는 바와 같이 최적 체크포인터 수는 고장 발생율(λ), 태스크의 원래 수행시간(E), 데드라인(D), 체크포인터 오버헤드(t_{cp}), 성능함수($w(t)$)와 같은 제어 태스크의 수행 환경 조건에 따라 조금씩 달라짐을 알 수 있다. 따라서 위와 같은 태스크의 수행 환경을 고려하여 체크 포인팅 기법을 설계하는 것이 최적의 고장 극복 효과를 얻을 수 있다.

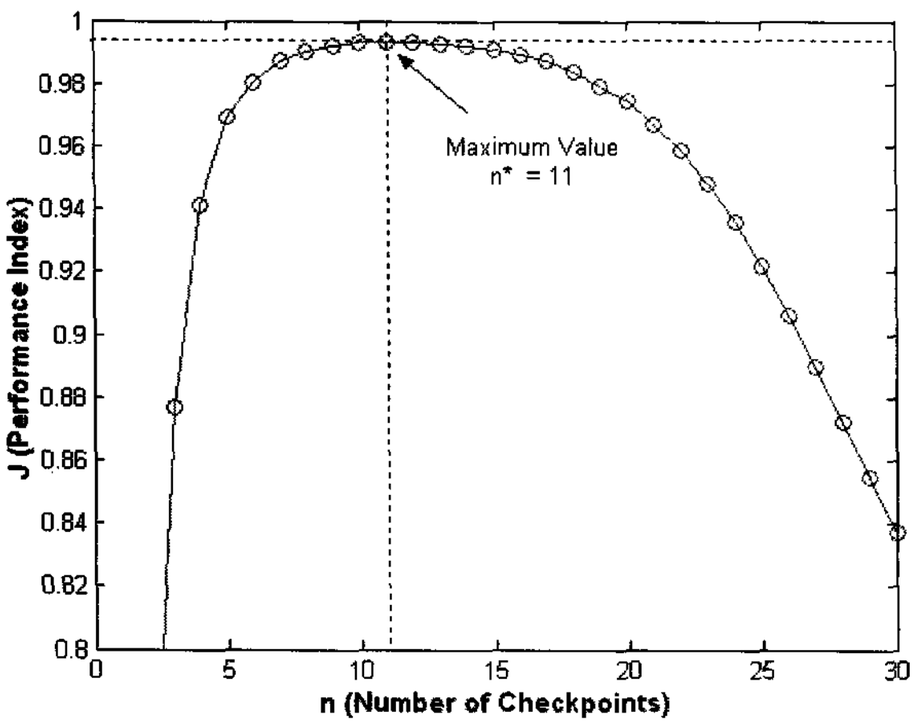


그림 5 성능 지수(J) vs. 체크포인터 수

Fig. 5 Performance Index vs. number of checkpoints
($E=0.4$, $\lambda=5$, $T=1$, $D=2$, $t_{cp}=0.02$, $a=1$)

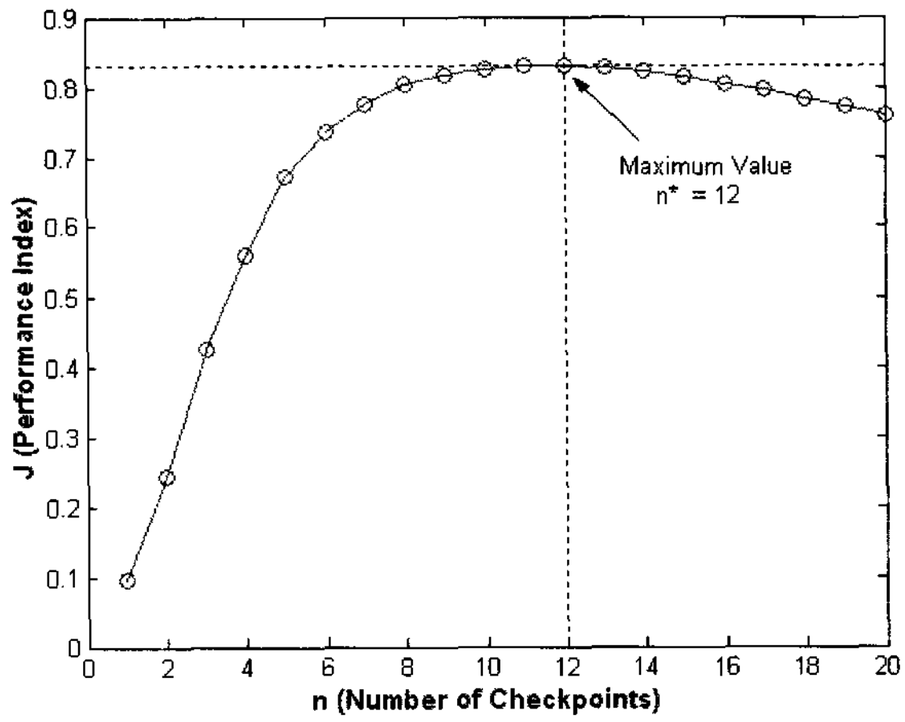


그림 6 성능 지수(J) vs. 체크포인터 수

Fig. 6 Performance Index vs. number of checkpoints
($E=0.6$, $\lambda=5$, $T=1$, $D=2$, $t_{cp}=0.02$, $a=1$)

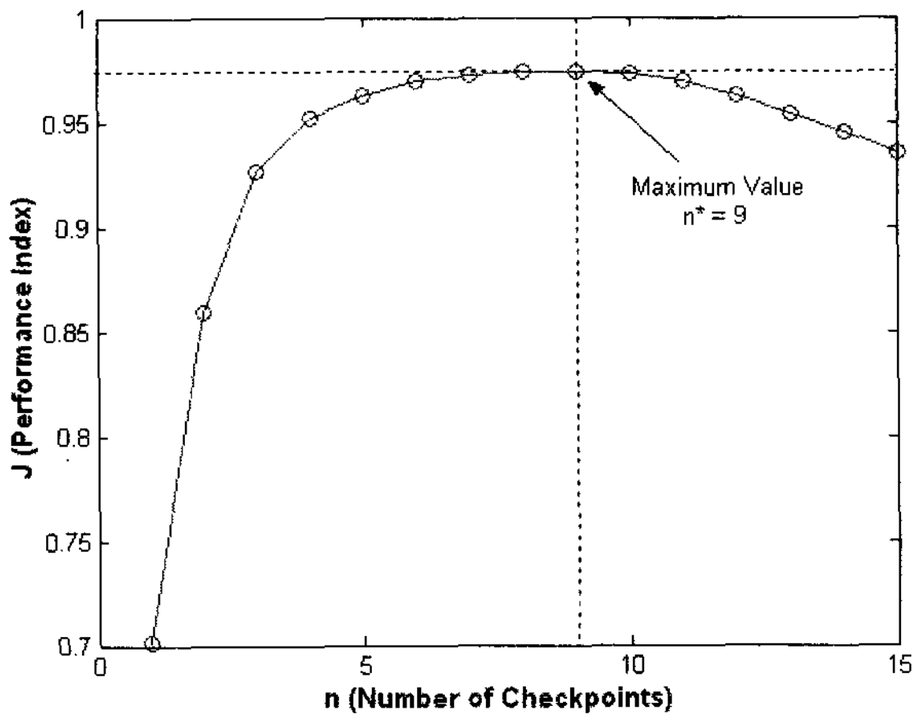


그림 7 성능 지수(J) vs. 체크포인터 수

Fig. 7 Performance Index vs. number of checkpoints
($E=0.7$, $\lambda=1$, $T=1$, $D=2$, $t_{cp}=0.02$, $a=1$)

6. 결 론

본 논문에서는 샘플링 주기와 데드라인이 다른 경우에 대하여 체크포인터 구간을 결정하는 문제를 해결하였다. 체크포인터 삽입을 통한 고장 극복은 태스크의 실행 시간 변화를 가져와 샘플링 시간 이내에 수행이 종료되지 않을 수 있다. 이와 같은 상황을 고려하기 위하여 고장 극복과정에서 발생하는 태스크의 수행 시간 변화에 따른 새로운 성능 지수를 정의하였고 이를 기반으로 최적의 체크포인터 구간을 유도 하였다. 기존 연구의 대부분은 샘플링 주기와 데드라인이 동일하다는 가정 하에 샘플링 주기를 벗어나는 고장극복은 시스템 Fail로 간주하였지만, 본 연구에서는 샘플링 주기와 데드라인이 서로 다른 매우 현실적인 상황을 다루었다.

참 고 문 헌

[1] H. Kim and K. G. Shin, "Design and Analysis of an Optimal Instruction Retry Policy for TMR Controller Computers", IEEE Trans. on Computers, vol 45, pp. 1217-1225, Nov. 1996.

[2] C. M. Krishna and A. D. Singh, "Optimal configuration of redundant real-time systems in the face of correlated failure," IEEE Trans. on Reliability, vol. 44, pp. 587-594. Dec.1995.

[3] Avi Ziv and Jehoshua Bruck, "An on-line algorithm for checkpoint placement," IEEE Trans. on Computers, vol. 46, pp. 976-984, Sep. 1997.

[4] R. Geist, R. Reynolds, and J. Westall, "Selection of a checkpoint interval in a critical-task environment," IEEE Trans. on Reliability, vol. 37, pp. 395-400, Oct. 1988.

[5] Kang G. Shin, Tein-Hsiang Lin, and Yann-Hang Lee, "Optimal checkpointing of real-time tasks," IEEE Trans. on Computers, vol. C-36, pp. 1328-1341, Nov. 1987.

[6] C. M. Krishna and A. D. Singh, "Reliability of checkpointed real-time systems using time redundancy," IEEE Trans. on Reliability, vol. 42, pp. 427-435, Sep. 1993.

[7] Seong Woo Kwak, Byung Jae Choi and Byung Kook Kim, "Optimal Checkpointing Strategy for Real-Time Control Systems under Faults with Exponential Duration", IEEE Trans. on Reliability, vol.50, no.3, pp. 293-301, Sep. 2001.

[8] Seong Woo, Kwak, "Reliability Analysis and Design of Real-time Fault Tolerant Control Systems under Transient Faults", Ph.D thesis, KAIST, 2000.

[9] John W. Young, "A first order approximation to the optimal checkpoint intervals," Comm. of the ACM, vol. 17, pp.530-531, Nov. 1974.

[10] 곽성우, 하드데드라인을 가지는 다중 실시간 주기적 태스크에서의 체크포인팅 기법, 전기학회논문지-D, 제53권 제8호, pp. 594-601, 2004.

[11] 곽성우, 정용주, RM 스케줄링된 실시간 태스크에서의 최적 체크포인터 구간 선정, 전기학회논문지, 제56권 제6호, pp. 1122-1129, 2007.

[12] D. Seto, J. P. Lehoczky, L. Sha, K. G. Shin, "On task schedulability in real-time control systems", Real-Time Systems Symposium(RTSS96), pp.134-21, 1996.

저 자 소 개

곽 성 우 (郭 成 祐)

1970년 3월 104일생. 1993년 한국과학기술원 전기및전자공학과 졸업(학사) 1995년 동 대학원 전기및전자공학과 졸업(석사). 2000년 동 대학원 전기및전자공학과 졸업(공박). 2000년~2002년 인공위성연구센터 선임연구원, 연구교수. 2003년~현재 계명대 전자공학과 전임강사, 조교수

Tel : 053 - 580 - 5926

Fax : 053 - 580-5165

E-mail : ksw@kmu.ac.kr

정 용 주 (鄭 容 朱)

1965년 5월 24일생. 1988년 서울대학교 전자공학과 졸업(학사) 1990년 한국과학기술원 전기및전자공학과 졸업(석사). 1995년 동 대학원 전기및전자공학과 졸업(공박). 1995년~1998년 LG전자 선임연구원. 1999년~현재 계명대 전자공학과 전임강사, 조교수, 부교수