Regular paper

Journal of the KIECS. pp. 527-534, vol. 18, no. 3, Jun. 30. 2023, t. 119, pISSN 1975-8170 | eISSN 2288-2189
http://dx.doi.org/10.13067/JKIECS.2023.18.3.527

삼중구조 시스템의 실시간 태스크 최적 체크포인터 및 분산 고장 탐지 구간 선정

곽성우* · 양정민**

Determination of the Optimal Checkpoint and Distributed Fault Detection Interval for Real-Time Tasks on Triple Modular Redundancy Systems

Seong Woo Kwak* · Jung-Min Yang**

요 약

삼중구조 시스템에서는 하나의 프로세서에서 고장이 발생해도 여유도 때문에 주어진 임무를 계속 수행할 수 있다. 본 연구에서는 삼중구조 시스템에 체크포인터 기법을 도입한 후 고장 탐지와 체크포인터를 분리하는 새로운 고장 극복 방법을 제안한다. 먼저 한 개 프로세서에서 고장이 발생하면 고장 탐지와 동시에 모든 프로세서의 상태를 동기화함으로써 고장을 복구한다. 또한 두 개 이상의 프로세서에서 동시에 고장이 발생하면 직전의 체크포인터로 회귀하여 태스크를 재실행함으로써 고장을 복구한다. 본 논문에서는 태스크가 데드라인 이내에서 성공적으로 수행될 확률을 최대화하는 고장 탐지 구간과 체크포인터 구간의 선정 방법을 제안한다. 제안된 방식을 탑재한 삼중구조 시스템을 마코프 체인으로 모델링하고 실시간 태스크의 성공적 수행 확률을 도출하는 모의실험을 수행하여 최적의 해를 구하는 과정을 제시한다.

ABSTRACT

Triple modular redundancy (TMR) systems can continue their mission by virtue of their structural redundancy even if one processor is attacked by faults. In this paper, we propose a new fault tolerance strategy by introducing checkpoints into the TMR system in which data saving and fault detection processes are separated while they corporate together in the conventional checkpoints. Faults in one processor are tolerated by synchronizing the state of three processors upon detecting faults. Simultaneous faults occurring to more than one processor are tolerated by re-executing the task from the latest checkpoint. We propose the checkpoint placement and fault detection strategy to maximize the probability of successful execution of a task within the given deadline. We develop the Markov chain model for the TMR system having the proposed checkpoint strategy, and derive the optimal fault detection and checkpoint interval.

키워드

Triple Modular Redundancy, Checkpoints, Fault Detection, Rollback And Re-Execution, Real-Time Task
삼중 구조, 체크포인터, 고장 탐지, 회귀 및 재실행, 실시간 태스크

* 부경대학교 제어계측공학과(ksw@pknu.ac.kr)
** 교신저자 : 경북대학교 전자공학부
· 접 수 일 : 2023. 05. 02
· 수정완료일 : 2023. 05. 22
· 게재확정일 : 2023. 06. 17
· Received : May. 02, 2023, Revised : May. 22, 2023, Accepted : Jun. 17, 2023
· Corresponding Author : Jung-Min Yang
School of Electronics Engineering, Kyungpook National University
Email : jmyang@knu.ac.kr

Ⅰ. 서 론

산업 현장에서 발생하는 고장 분석 결과에 따르면 영구적인 부품의 결함과 같은 하드웨어적인 원인보다는 외부의 전기적 환경 변화 등에 의한 과도 고장이 대부분이다[1]. 실시간 태스크는 대부분 데드라인(deadline)이라는 제한된 시간 이내에 실행을 마쳐야하는 작업을 수행한다. 실시간 태스크의 개념은 다양한 분야에 적용되어 왔다[2-5]. 체크포인팅(checkpointing) 기법은 데드라인 내에서 태스크가 가지는 여유시간을 사용하여 과도 고장을 복구하는 방법이다[6]. 이 기법의 핵심은 고장이 발생한 체크포인터 구간을 재수행하여 과도 고장을 극복한다는 것이다[7].

체크포인팅 기법과 관련된 기존 연구들에는 태스크의 수행 시간을 최소화하는 체크포인터 삽입 방법[8], 데드라인 이내에 수행이 끝날 확률을 최대화시키는 체크포인터 삽입 방법[9] 등이 있다. 이들 연구는 실시간으로 시스템을 감시하여 고장이 발생하자마자 즉시 탐지하거나 체크포인터를 삽입하는 과정에서 고장을 탐지할 수 있다고 가정하였다. 하지만 실시간으로 고장을 탐지하는 과정은 매우 복잡하며 많은 부가적인 회로가 필요하다. 또한 체크포인터를 삽입하는 과정에서 고장을 탐지하는 방법은 단순하고 구현하기 쉽지만 고장 탐지 시 재실행 구간이 길어질 수 있는 단점이 있다.

본 논문에서는 삼중구조(TMR: Triple Modular Redundancy) 시스템에 체크포인터 기법을 도입한다. 삼중구조의 특징을 이용하여 고장을 탐지하고, 삼중구조와 체크포인터를 이용하여 고장을 복구하는 새로운 방법을 제안한다. 제안된 방법에서는 체크포인터에서 수행되는 고장 탐지와 데이터 저장 과정을 분리하여[10] 하나의 체크포인터 구간 사이에 여러 번의 고장 탐지 과정이 수행되도록 한다.

본 연구에서는 고장 탐지와 데이터 저장 과정이 분리되어 운영되는 체크포인터 기법에 대한 모델을 개발하고 이를 바탕으로 최적의 체크포인터 구간과 고장 탐지 구간을 찾는다. 먼저 삼중구조에서 고장 탐지와 데이터 저장 과정이 분리된 체크포인터 방식에 대한 마코프(Markov) 체인 모델링을 실시한다. 또한 실시간 태스크가 데드라인 이내에 수행이 끝날 확률을 구할 수 있도록 상태(state)를 정의하고 상태 천이 확률 식을 유도한다. 다음으로 고장 탐지에 소요되는 오버헤드, 프로세서 사이의 중간 상태 값 동기화에 소요되는 시간, 상태 데이터 저장에 걸리는 시간이 고려된 태스크의 실제 수행 시간을 계산한다. 마지막으로 제안된 삼중구조 시스템에서 실행되는 태스크가 데드라인 이내에서 수행이 끝날 확률을 최대화하도록 고장 탐지 간격과 데이터 저장 간격에 따른 최적 체크포인터 삽입 방식을 선정한다.

Ⅱ. 삼중구조에서 고장 탐지와 체크포인터 삽입 방식

2.1 체크포인터 삽입과 고장 탐지 방식

체크포인터를 이용한 고장 극복 방식은 태스크의 실행 시간 내에 다수의 체크포인터를 삽입하고 고장이 발생하면 고장 발생 직전의 체크포인터로 되돌아가서 태스크를 재실행하는 것이다. 고장이 발생한 구간을 재실행함으로써 태스크의 실행 중 발생한 과도 고장에 의한 오류를 복구할 수 있다.

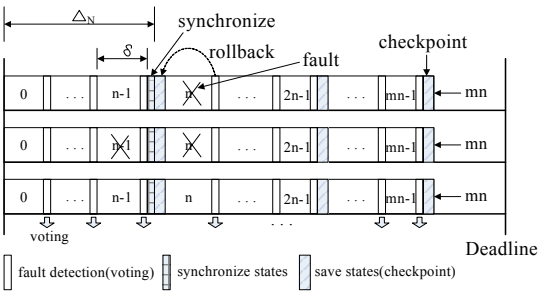

그림 1. 삼중구조에서 고장 탐지와 데이터 저장이 분리된 체크포인터 방식

Fig. 1 Checkpoints with separated fault detection and data saving on the TMR structure.

그림 1은 본 연구에서 제안하는 삼중구조 시스템에서의 체크포인터 삽입 방식이다. 체크포인터 삽입과 고장 탐지가 분리되어 있으며 하나의 체크포인터 구간 내에 다수의 고장 탐지 과정이 실행된다. 각 체크포인터에서는 태스크의 중간 상태 값을 저장하기 전에 항상 고장 탐지 과정을 실행하여 고장

이 없는 온전한 중간 상태 값이 저장되도록 한다. 본 연구에서의 고장 탐지 방식은 삼중구조의 특징을 이용하여 세 프로세서의 태스크 중간 상태 값을 다수결 회로를 이용하여 표결(voting)함으로써 고장을 탐지한다. 삼중구조의 특성상 하나의 프로세서에서의 고장은 극복 가능하다. 하지만 고장이 발생한 프로세서의 상태를 온전한 프로세서와 동기화하지 않으면 고장의 영향이 지속적으로 남는다. 만약 이와중에 또 다른 프로세서에서 고장이 발생한다면 두 개의 프로세서가 고장에 노출되어 출력단에서는 오류를 발생시킨다.

본 연구가 제안하는 방식에서 한 개 프로세서에 고장이 탐지되면 세 개의 프로세서 상태를 동기화 한다(그림 1의 (n-1) 고장 탐지 구간 예제 참조). 만약 두개 이상의 프로세서에서 동시에 고장이 탐지되면 가장 가까운 체크포인터로 회귀하여 태스크를 재실행하여 고장을 복구한다(그림 1의 n 고장 탐지 구간 예제 참조). 제안된 방식에서는 하나의 체크포인터 구간 내에 n번의 고장 탐지가 실행된다.

2.2 태스크 실행 시간 및 재실행 횟수

체크포인터가 삽입되기 전 원래 태스크의 수행 시간을 e라고 하고, 다수결 회로를 이용한 표결을 통해 고장 탐지 과정을 수행하는 데 걸리는 시간을 t_d, 세 개 프로세서의 태스크 중간 상태 값을 동기화 시키는 데 걸리는 시간을 t_s, 그리고 중간 상태 값을 저장하는 데 걸리는 시간을 t_c라고 하자. 또한 그림 1에서 보인 바와 같이 각 고장 탐지 과정이 실행되는 간격을 δ라고 하자. 고장 탐지 간격 δ, 고장이 없는 경우의 태스크의 실제 실행 시간 E_N, 그리고 체크포인터 사이의 구간 Δ_N는 식 (1), (2), (3)과 같이 구할 수 있다.

$$\delta = e/(n \cdot m) + t_d \quad \cdots (1)$$

$$E_N = e + n \cdot m \cdot t_d + m \cdot t_c \quad \cdots (2)$$

$$\Delta_N = e/m + n \cdot t_d \quad \cdots (3)$$

여기서 m은 체크포인터 개수이고, n은 하나의 체크포인터 구간 내에 실행되는 고장 탐지 과정의 개수이다. 체크포인터 삽입 후 첫 번째 고장 탐지 구간에서 두 개 프로세서 이상에서 고장이 발생하면 가장 가까운 체크포인터까지 δ구간 만큼 회귀하여 태스크를 재실행한다. 만약 두 번째 고장 탐지 구간에서 고장이 발생하였다면 상태 값이 저장된 가장 가까운 체크포인터까지 2δ만큼 재실행이 일어나고, k번째 구간에서 고장이 발생하였다면 kδ만큼 재실행된다.

만약 세 프로세서 중 한 개 프로세서에 고장이 발생하여 동기화 과정이 수행되어야 하는 곳이 u개 존재하고, 두 개 이상의 프로세서에서 고장이 발생하여 재수행이 필요한 곳이 r개 존재한다고 하자. 이때 동기화 과정과 재수행 때문에 늘어난 태스크의 전체 실행 시간 E(u,r)은 식 (4)와 같다.

$$E(u,r) = e + n \cdot m \cdot t_d + m \cdot t_c + u \cdot t_s + r \cdot \delta \quad \cdots (4)$$

태스크의 데드라인을 D라고 하면 동기화 과정이 실행될 수 있는 최대값 u_{mn}과 재수행 횟수의 최대값 r_{mn}은 식 (5)와 같이 구할 수 있다.

$$u_{mn} = \left\lfloor \frac{D - m t_c}{\delta + t_s} \right\rfloor , \; r_{mn} = \left\lfloor \frac{D - E(0,0)}{\delta} \right\rfloor \quad \cdots (5)$$

여기서 ⌊x⌋는 x를 넘지 않는 최대 정수이다.

u개의 동기화 과정이 수행되었다면 재수행 가능 횟수는 식 (6)과 같이 구할 수 있다.

$$r_{mn}(u) = \left\lfloor \frac{D - E(u,0)}{\delta} \right\rfloor \quad \cdots (6)$$

III. 모델링 및 태스크 실행 성공 확률

각 체크포인터에서 태스크의 상태 값을 저장할 때에는 voter를 통과시킨 후 그 결과 값을 저장한다. 이 과정에서 한 개 프로세서에서의 고장은 voter에 의해 복구된다. 각 프로세서에서 발생하는 고장은 발생율 λ를 가진 Poisson 분포를 따른다고 하자. 각 프로세서의 고장 탐지 구간 δ 내에서 고장이 발생하지 않을 확률(α)와 고장이 한 개 이상 발생할 확률(β)은 식 (7)과 같이 구할 수 있다[10].

$$\alpha = e^{-\lambda\delta}, \qquad \beta = 1 - e^{-\lambda\delta} \quad \cdots (7)$$

고장 탐지 구간 δ 내에서 세 개 프로세서 모두에 고장이 없을 확률을 p라고 하고, 한 개 프로세서에서 고장이 발생할 확률을 q, 그리고 두 개 이상의 프로세서에서 고장이 발생할 확률을 w라고 정의하면, 각각의 확률은 식 (8), (9)와 같이 구할 수 있다.

$$p = e^{-3\lambda\delta}, \quad q = 3e^{-2\lambda\delta}(1-e^{-\lambda\delta}) \quad \cdots (8)$$

$$w = 1-[e^{-3\lambda\delta}+3e^{-2\lambda\delta}(1-e^{-\lambda\delta})] \quad \cdots (9)$$

본 연구에서 다루는 그림 1의 삼중구조 체크포인터 시스템에서는 하나의 체크포인터와 다음 체크포인터 사이에 n개의 고장 탐지 과정이 등 간격으로 실행된다. 태스크에는 m개의 체크포인터가 삽입된다고 했으므로 태스크의 수행은 m·n개의 고장 탐지 구간을 수행하는 것과 같다. m·n개 구간이 고장 없이 수행되면 태스크의 한 주기가 성공적으로 수행되는 것으로 간주된다. 그림 1의 0~(mn-1)로 표시된 구간 중에서 구간 i를 수행하고 있는 상태를 $x_{u,i}$라고 정의하자. 여기서 인덱스 $u(0\le u\le u_{mn})$은 구간 i를 수행하기 전에 u번의 동기화 과정이 실행되었다는 것을 표시한다. 앞에서 설명하였듯이 하나의 프로세서에서 고장이 발생하였다면 동기화 과정만 실행된다. 또한 $x_{u,mn}$은 m·n개의 구간이 모두 고장 없이 실행되어 태스크의 수행이 완료된 상태라고 정의한다.

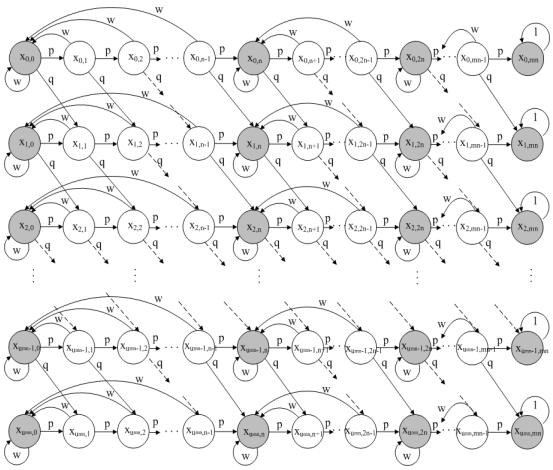

그림 2. 삼중구조에서 고장 탐지 과정이 분리된 체크포인터의 마코프 모델

Fig. 2 Markov model for the TMR checkpoint system with a separated fault detection procedure.

그림 2는 본 연구에서 제안하고 있는 고장 탐지 과정이 분리된 삼중구조 체크포인터 시스템을 마코프 체인(Markov chain)으로 모델링한 것이다. 상태 $x_{u,0}$는 그림 1의 i=0인 구간을 수행하는 상태를 나타낸다. 또한 상태 $x_{u,vn}$는 v번째 체크포인터의 바로 다음 구간을 실행하는 상태이다. $x_{u,vn+j}(0\le j\le n-1)$에서 두 개 이상의 프로세서에서 고장이 탐지되면 가장 가까운 체크포인터 $x_{u,vn}$로 회귀하여 그 이후 구간을 재실행한다. 만약 구간 i를 수행하는 상태 $x_{u,i}$에서 세 프로세서 모두에 고장이 발생하지 않았다면 다음 구간(i+1)을 실행하는 상태 $x_{u,i+1}$로 천이한다. 상태 $x_{u,i}$에서 한 개 프로세서에서 고장이 탐지되었다면 체크포인터로 회귀하지 않고 동기화 과정만 실행되므로 동기화 과정의 인덱스 u가 1 증가된 (u+1)로 된다. 이때 태스크의 실행도 다음 구간으로 진행하므로 인덱스 i도 1 증가된 상태 $x_{u+1,i+1}$로 옮겨간다.

$k\delta\le t<(k+1)\delta$ 시간에서 시스템이 $x_{u,i}$에 있을 확률을 $\Upsilon_{u,i}(k)$라고 하자. $\Upsilon_{u,0}(k+1)$은 다음과 같이 구할 수 있다. 만약 $k\delta\le t<(k+1)\delta$ 시간에서 시스템이 상태 $x_{u,0}$에 있을 때 두 개 이상의 프로세서에서 고장이 발생하면(이때의 고장 발생 확률은 w) 시스템은 다시 상태 $x_{u,0}$로 회귀하여 i=0의 구간을 재실행해야 한다. 이 사건이 일어날 확률은 $w\Upsilon_{u,0}(k)$이다. 그림 2의 마코프 모델에서 $x_{u,0}\rightarrow x_{u,0}$로 천이하는 확률 w가 이 경우를 표현한다. 또한 $k\delta\le t<(k+1)\delta$ 시간에 시스템이 상태 $x_{u,1}$에 있을 때 역시 두 개 이상의 프로세서에 고장이 발생하여도 시스템은 가장 가까운 체크포인터가 있는 $x_{u,0}$로 회귀하여 i=0 구간부터 재실행해야 하므로 $x_{u,0}$에 머무른다. 이 사건이 일어날 확률은 $w\Upsilon_{u,1}(k)$로 표현된다. 그림 2의 마코프 모델에서 $x_{u,1}\rightarrow x_{u,0}$로 천이하는 확률 w가 이 확률에 대응된다. 이러한 해석을 일반화하면 시스템이 상태 $x_{u,n-1}$에서 $x_{u,0}$로 천이하는 확률은 $w\Upsilon_{u,n-1}(k)$이다. 상태 $x_{u,0}$로 회귀하는 모든 경우의 확률을 합하면 $\Upsilon_{u,0}(k+1)(0\le u\le u_{mn})$을 구할 수 있다.

$$\gamma_{u,0}(k+1) = w\gamma_{u,0}(k)+\cdots+w\gamma_{u,n-1}(k) \quad \cdots (10)$$

다음으로 시스템이 $(k+1)\delta\le t<(k+2)\delta$ 시간에 $x_{u,vn}$에 있을 확률 $\Upsilon_{u,vn}(k+1)$은 식 (10)의 유도 과정과

비슷하게 구할 수 있다. $k\delta \le t < (k+1)\delta$ 시간에 상태 $x_{u,vn+j}$에 있을 때 두 개 이상의 프로세서에서 고장이 발생하면 가장 가까운 체크포인터인 i=vn 구간(상태 $x_{u,vn}$)으로 회귀하여 태스크를 재실행한다. 이 확률은 $w\gamma_{u,vn+j}(k)$로 표현된다. 그림 2에서 $x_{u,vn+j} \rightarrow x_{u,vn}$로 천이하는 확률 w가 여기에 대응된다. 또한 만약 $k\delta \le t < (k+1)\delta$ 시간에서 i=(vn-1) 구간을 실행하였을 때 고장이 발생하지 않으면(확률 p) i=vn 구간을 수행하는 상태 $x_{u,vn}$로 천이한다. 이 확률은 $p\gamma_{u,vn-1}(k)$이다. 그림 2에서 $x_{u,vn-1} \rightarrow x_{u,vn}$로 천이하는 확률 p가 이것을 나타낸다. 상태 $x_{u-1,vn-1}$에서 한 개 프로세서에서 고장이 탐지되었다면(확률 q) 체크포인터로 회귀하지 않고 동기화 과정만 실행되므로 동기화 과정의 인덱스가 (u-1)에서 1이 증가된 u로 된다. 또한 태스크의 수행은 다음 구간(vn)으로 진행하므로 상태 $x_{u,vn}$로 천이한다. 이와 같이 천이할 확률은 $q\gamma_{u-1,vn-1}(k)$이다. 그림 2의 $x_{u-1,vn-1} \rightarrow x_{u,vn}$로 천이하는 사선의 확률 q가 이를 표현한다. 따라서 상태 $x_{u,vn}$로 천이하는 확률은 위의 모든 경우의 확률을 더하면 된다. 이 과정을 일반화하면 $\gamma_{u,vn}(k+1)$은 식 (11)과 같이 구할 수 있다.

$$\gamma_{u,vn}(k+1) = w\gamma_{u,vn}(k) + w\gamma_{u,vn+1}(k) + \cdots + w\gamma_{u,vn+n-1}(k) + p\gamma_{u,vn-1}(k) + q\gamma_{u-1,vn-1}(k) \quad \cdots (11)$$

여기서 $0 \le u \le u_{mn}$이며 $0 \le v \le (m-1)$이다. $u<0$ 또는 $i<0$일 때는 $\gamma_{u,i}(k)=0$으로 정의한다.

그림 1의 시스템은 상태 $x_{u,vn+j}$에서 고장이 발생하지 않으면 $x_{u,vn+j+1}$로 천이한다. $k\delta \le t < (k+1)\delta$ 시간에 상태 $x_{u,vn+j}$에 있으면서 고장이 발생하지 않을 확률은 $p\gamma_{u,vn+j}(k)$이다. 그림 2에서 $x_{u,vn+j} \rightarrow x_{u,vn+j+1}$로 천이하는 확률 p가 이것을 나타낸다. 또한 한 개의 프로세서에서 고장이 발생한 경우에는 프로세서 간의 동기화를 통해 다음 상태로 옮겨갈 수 있다. 프로세서 간의 동기화가 수행될 확률은 q이다. 그림 2에서 $x_{u-1,vn+j} \rightarrow x_{u,vn+j+1}$로 천이하는 사선의 확률 q가 이것을 나타낸다. 따라서 $(k+1)\delta \le t < (k+2)\delta$ 시간에 시스템이 상태 $x_{u,vn+j+1}$에 있을 확률 $\gamma_{u,vn+j+1}(k+1)$은 식 (12)와 같다.

$$\gamma_{u,vn+j+1}(k+1) = p\gamma_{u,vn+j}(k) + q\gamma_{u-1,vn+j}(k) \quad \cdots (12)$$

상태 $x_{u,mn}$는 태스크의 수행이 끝난 상태를 나타내므로 시스템이 이 상태에 도달하면 이후에는 계속 여기에 머무른다. 따라서 $\gamma_{u,mn}(k+1)$은 식 (13)과 같이 구할 수 있다.

$$\gamma_{u,mn}(k+1) = p\gamma_{u,mn-1}(k) + q\gamma_{u-1,mn-1}(k) + \gamma_{u,mn}(k) \quad \cdots (13)$$

다음으로 태스크의 수행이 끝나는 상태 $x_{u,mn}$에 도달할 때까지 필요한 시간 스텝 k의 최대값을 구한다. 상태 $x_{u,mn}$에 도달할 때까지 u번의 동기화가 실행된다고 했을 때 재수행 가능 횟수는 식 (6)에서 구한 $r_{mn}(u)$이다. 그림 1에서 볼 수 있듯이 태스크의 수행을 끝내기 위해서는 mn개의 δ 구간에서 태스크의 수행이 필요하다. 데드라인 D 이내에서 태스크의 수행을 끝내기 위해서는 mn개의 δ 구간이 필수적으로 필요하며, $r_{mn}(u)$개의 여유 δ 구간은 고장 극복을 위해 사용할 수 있다. 만약 $r_{mn}(u)<0$이면 mn개의 필수 구간을 수행할 수 없으므로 태스크의 수행을 완료할 수 없다는 뜻이다. 따라서 u번의 동기화가 실행된다고 했을 때 상태 $x_{u,mn}$에 도달할 때까지 소요되는 시간 스텝의 최대값 k_u은 식 (14)와 같이 구할 수 있다.

$$k_u = \begin{cases} mn + r_{mn}(u) & if\ r_{mn}(u) \ge 0 \\ 0 & if\ r_{mn}(u) < 0 \end{cases} \quad \cdots (14)$$

식 (14)를 이용하면 데드라인 이내에 태스크의 수행이 끝나는 상태 $x_{u,mn}$에 도달할 확률은 $\gamma_{u,mn}(k_u)$로 표현된다. 동기화 과정의 최대값은 식(5)에서 제시된 u_{mn}이므로 그림 1의 시스템이 데드라인 이내에서 태스크의 수행을 완료하는 확률(P_D)은 식 (15)와 같이 나타낼 수 있다.

$$P_D = \sum_{u=0}^{u_{mn}} \gamma_{u,mn}(k_u) \quad \cdots (15)$$

여기서 초기값은 $\gamma_{0,0}(0)=1$이다.

Ⅳ. 시뮬레이션 결과

모의실험에 사용된 태스크의 데드라인 D, 태스크의 원래 실행 시간 e, 고장 탐지에 소요되는 시간 t_d, 체크포인터에서 중간 상태값 저장에 소요되는 시간 t_s, 그리고 평균 고장 발생율 λ는 다음과 같다.

D=10, e=6, t_d=0.05, t_c=0.1, t_s=0.05, λ=0.5

그림 3은 태스크가 데드라인 이내에서 성공적으로 수행이 끝날 확률 P_D를 나타낸 것이다. x축은 체크포인터 수 m이고 y축은 체크포인터 사이에 삽입된 고장 탐지 과정의 수 n이다. 그림 3에서 볼 수 있듯이 태스크 수행을 완료하는 확률을 최대로 하는 지점은 m=10, n=2일 때이다. 즉 태스크에 10개의 체크포인터를 삽입하고 체크포인터 사이에는 두 개의 고장 탐지 과정을 실행할 때가 최적이라는 것을 보여준다.

그림 4는 태스크 실행시간, 고장 발생률 등의 변수 값을 다음과 같이 설정했을 때 나오는 시뮬레이션 결과이다.

D=10, e=4, t_d=0.05, t_c=0.5, t_s=0.05, λ=0.7

그림 4는 그림 3의 실험 설정에서 태스크의 실행 시간을 줄이고 데이터 저장에 소요되는 오버헤드는 증가시킨 후 구한 실험 결과이다. 태스크의 수행을 완료하는 확률을 최대로 하는 지점은 m=5, n=3일 때이다. 즉 태스크에 5개의 체크포인터를 삽입하고 체크포인터 사이에 세 번의 고장 탐지 과정을 넣는 것이 최적이라는 사실을 알 수 있다.

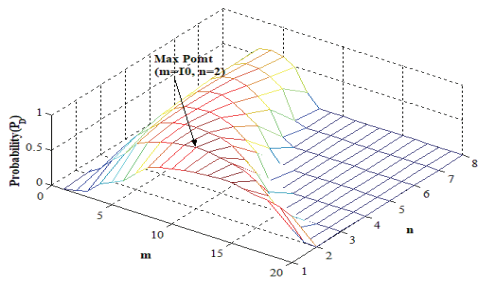

그림 3. 데드라인 내에서 태스크의 실행 성공 확률
(D=10, e=6, td=0.05, tc=0.1, ts=0.05, λ=0.5)
Fig. 3 Probability of a task completion within a deadline.

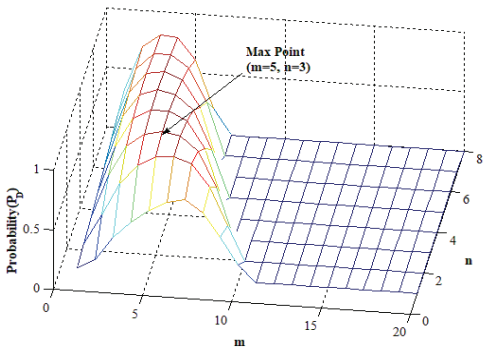

그림 4. 데드라인 내에서 태스크의 실행 성공 확률
(D=10, e=4, td=0.05, tc=0.5, ts=0.05, λ=0.7)
Fig. 4 Probability of a task completion within a deadline.

Ⅴ. 결 론

본 논문에서는 삼중구조 시스템을 위한 새로운 체크포인터 삽입 방법을 제안하였다. 삼중구조 시스템의 각 체크포인터에서 실행되는 고장 탐지와 데이터 저장 과정을 분리하여 체크포인터 구간 내에서 다수의 고장 탐지를 실행하는 방법을 제시하였다. 제안된 체크포인터 방법을 가진 삼중구조 시스템에 대한 마코프 모델을 개발하고 데드라인 이내에서 태스크의 수행이 끝날 확률을 계산하였다. 이 확률을 기반으로 최적의 체크포인터 갯수와 체크포인터 사이에서 실행되는 최적의 고장 탐지 횟수를 구하였다. 모의실험을 통하여 태스크와 체크포인터에 대한 적절한 매개변수 값이 주어졌을 때 최적의 해를 구할 수 있음을 입증하였다.

감사의 글

이 논문은 부경대학교 자율창의학술연구비(2022년)에 의하여 연구되었음.

References

[1] S. Punnekkat, A. Burns, and R. Davis, "Analysis of checkpointing for real-time systems," *International Journal of Time-Critical Computing Systems*, vol. 20, no. 1, 2001, pp. 83-102.

[2] H. Suh, "An improved algorithm of distributed QoS in real-time networks," *J. of the Korea Institute of Electronic Communication Sciences*, vol. 7, no. 1, Feb. 2012, pp. 53-59.

[3] Y. Park, J. Li, and Y. Lee, "AUV platform design for unmanned remotely construction and harbor infrastructure," *J. of the Korea Institute of Electronic Communication Sciences*, vol. 16, no. 6, Dec. 2021, pp. 1089-1094.

[4] S. Ko and T. Kwon, "Intermediate node mobility management technique by real-time monitoring in CCN environment," *J. of the Korea Institute of Electronic Communication Sciences*, vol. 17, no. 5, Oct. 2022, pp. 783-790.

[5] C. Kim and J. Seo, "Design and implementation of realtime things control system using MQTT and websocket in IoT environment," *J. of the Korea Institute of Electronic Communication Sciences*, vol. 13, no. 3, June 2018, pp. 517-524.

[6] T. Ozaki, T. Dohi, H. Okamura, and N. Kaio, "Distribution-free checkpoint placement algorithms based on min-max principle," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 2, Apr.-Jun. 2006, pp. 130-140.

[7] J. Young, "A first order approximation to the optimal checkpoint intervals," *Communications of the ACM*, vol. 17, no. 9, Sept. 1974, pp. 530-531.

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

[9] S. Kwak and Y. Jung, "Determination of optimal checkpoint interval for RM scheduled real-time tasks," *The Transactions of the Korean Institute of Electrical Engineers*, vol. 56, no. 6, June 2007, pp. 1122-1129.

[10] S. Kwak and J.-M. Yang, "Determination of optimal checkpoint intervals for real-time tasks using distributed fault detection," *Journal of Korean Institute of Intelligent Systems*, vol. 26, no. 3, July 2016, pp. 202 - 207.

저자 소개

곽성우(Seong-Woo Kwak)

1993년 한국과학기술원 전기및전자공학과 졸업(공학사)
1995년 한국과학기술원 전기및전자공학과 졸업(공학석사)
2000년 한국과학기술원 전기및전자공학과 졸업(공학박사)
2003년~2020년 계명대학교 전자공학과 교수
2020년~현재 부경대학교 제어계측공학과 교수
※ 관심분야 : 실시간 시스템, 비동기 시스템 제어, 내고장성 시스템, 자율주행 자동차

양정민(Jung-Min Yang)

1993년 한국과학기술원 전기및전자공학과 졸업(공학사)
1995년 한국과학기술원 전기및전자공학과 졸업(공학석사)
1999년 한국과학기술원 전기및전자공학과 졸업(공학박사)
2013년~현재 경북대학교 전자공학부 교수
※ 관심분야 : 비동기 순차 머신 교정 제어, 실시간 시스템 고장 진단 및 극복, 불리언 제어 네트워크