Journal of Korean Institute of Intelligent Systems
Vol. 26, No. 3, June 2016, pp. 202-207
http://dx.doi.org/10.5391/JKIIS.2016.26.3.202

ISSN(Print) 1976-9172
ISSN(Online) 2288-2324

JKIIS

# 분산 고장 탐지 방식을 이용한 실시간 태스크에서의 최적 체크포인터 구간 선정

# Determination of Optimal Checkpoint Intervals for Real-Time Tasks Using Distributed Fault Detection

**곽성우* · 양정민****†
Seong Woo Kwak and Jung-Min Yang†

***계명대학교 전자공학과, **경북대학교 전자공학부**
*Department of Electronic Engineering, Keimyung University
**School of Electronics Engineering, Kyungpook National University

## 요 약

체크포인터를 삽입한 실시간 시스템에서는 고장이 발생하면 고장 직전의 체크포인터로 회귀하여 태스크를 재실행함으로써 과도 고장을 효과적으로 극복할 수 있다. 이번 논문에서는 체크포인터에서 실행되는 데이터 저장과 고장 탐지 과정을 분리한 새로운 체크포인터 방식을 제안한다. 하나의 체크포인터 구간 내에 여러 개의 고장 탐지 과정을 추가하면 고장 발생에서 탐지까지의 지연 시간을 줄일 수 있다. 본 논문에서는 태스크가 데드라인 이내에서 성공적으로 수행될 확률을 최대화하는 고장 탐지 과정의 삽입 방법을 제안한다. 고장 탐지 과정이 분리된 체크포인터 방식을 마코프 체인으로 모델링하고 실시간 태스크의 성공적 수행 확률을 계산하는 모의실험을 수행하여 최적의 해를 구하는 과정을 제시한다.

키워드 : 체크포인터, 고장 탐지, 재수행, 실시간 태스크

## Abstract

Checkpoint placement is an effective fault tolerance technique against transient faults in which the task is re-executed from the latest checkpoint when a fault is detected. In this paper, we propose a new checkpoint placement strategy separating data saving and fault detection processes that are performed together in conventional checkpoints. Several fault detection processes are performed in one checkpoint interval in order to decrease the latency between the occurrence and detection of faults. We address the placement method of fault detection processes to maximize the probability of successful execution of a task within the given deadline. We develop the Markov chain model for a real-time task having the proposed checkpoints, and derive the optimal fault detection and checkpoint interval.

Key Words : Checkpoints, Fault detection, Rollback, Real-time Ttask.

Received: Apr. 18, 2016
Revised : May. 12, 2016
Accepted: May. 23, 2016
†Corresponding authors
jmyang@ee.knu.ac.kr

## 1. 서 론

산업 현장에서 발생하는 고장들을 분석한 결과에 따르면 소자의 영구적 결함과 같은 하드웨어적인 원인보다는 외부의 전기적, 기계적 환경 변화 등과 같은 과도 고장에 의한 고장 발생이 다수를 이루고 있다[1]. 과도 고장은 다중구조를 이용한 하드웨어적 방법보다는 체크포인터(checkpoints) 삽입 등과 같은 소프트웨어적인 방법으로 더 잘 대처할 수 있다[2]. 실시간 태스크들은 제한된 시간, 즉 데드라인(deadline) 이내에 출력을 생성해야 하는 작업을 수행한다. 체크포인팅 기법은 태스크의 데드라인 이내의 여유시간을 이용하여 과도 고장을 극복하는 방식이다. 고장이 발생한 경우 해당 체크포인터 구간을 재수행함으로써 과도 고장을 극복한다[3].

체크포인팅 기법에 대한 기존 연구들은 태스크의 수행 시간을 최소화하는 체크포인터 삽입 방법[4], 데드라인 이내에 수행이 끝날 확률을 최대화시키는 체크포인터 삽입 방법[5, 6] 등이 있다. 이들 기존 연구들은 고장이 발생하자마자 실시간으로 고장을 탐지하거나 체크포인터를 삽입하는 과정에서 고장이 탐지된다고 가정하였다. 하지만 실시간으로 고장을 바로 탐지하는 것은 매우 어려우며 많은 부가적인 회로를 요구한다. 또한 체크포인터를 삽입하는 과정에서 고장을 탐지하는 방법은 단순하고

한국연구재단 바이오 · 의료기술개발사업의 지원을 받아 수행된 연구임(No. 2015M3A9A7067220). 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2015R1A2A1A15054026). 이 논문은 2015년도 정부(교육부)의 재원으로 한국 연구재단의 지원을 받아 수행된 기초연구사업임(No. 2015R1D1A1A01056764).

구현하기 쉽지만 고장 탐지 시 재수행 구간이 길어질 수 있다.

본 논문에서는 매 체크포인터에서 수행되는 고장 탐지와 데이터 저장 과정을 분리하여 하나의 체크포인터 구간 이내에서 여러 번 고장 탐지 과정을 수행하는 기법을 제안한다. 고장 탐지 과정이 많아질수록 재수행 구간을 줄일 수 있지만 고장 탐지를 위한 오버헤드(overhead)가 증가한다. 본 연구의 목표는 고장 탐지와 데이터 저장 과정이 분리된 경우에 데드라인 이내에서 태스크의 성공적 수행 확률을 최대화하는 최적 체크포인터 구간과 고장 탐지 구간을 찾는 것이다.

먼저 고장 탐지와 데이터 저장 과정이 분리된 체크포인터 방식에 대하여 마코프(Markov) 체인 모델링을 실시한다. 실시간 태스크가 데드라인 이내에 수행이 끝날 확률을 구할 수 있도록 각 상태(state)를 정의하고 상태 천이 확률식을 유도한다. 다음으로 고장 탐지에 소요되는 부가적인 오버헤드와 데이터 저장에 걸리는 시간을 고려한 태스크의 실제 수행 시간을 계산한다. 마지막으로 고장 탐지 간격과 데이터 저장 간격에 따른 최적 체크포인터 삽입 방식을 선정한다.

## 2. 고장 탐지 과정이 분리된 체크포인터 방식

체크포인터를 이용한 고장 극복은 실시간 태스크의 수행 시간 내에 다수의 체크포인터를 삽입하고 고장이 발생하면 고장 발생 직전에 수행된 체크포인터로 되돌아가는 식이다. 되돌아간 체크포인터에 저장된 이전의 태스크 수행 값을 기반으로 고장이 발생한 구간을 재수행하여 고장을 극복한다. 일반적으로 고장은 태스크의 수행 중 임의의 순간에서 발생할 수 있다. 만약 고장이 발생하자마자 실시간으로 고장을 탐지 할 수 있다면 고장 탐지와 동시에 직전 체크포인터로 회귀하여 그 이후부터 태스크를 재실행한다. 하지만 실시간으로 고장을 탐지하기 위해서는 오류 탐지 코드(error detection code) 등 고장을 탐지하기 위한 부가적인 로직(logic)이 다수 필요하다.

또 다른 고장 탐지 방법은 그림 1에서와 같이 체크포인터에 데이터를 저장하기 전 데이터의 유효성 검사를 실시하여 고장 발생 유무를 체크하는 것이다. 이 방식은 실시간 탐지 방법에 비해 부가적인 로직을 적게 사용하는 장점이 있다. 고장 탐지 시점이 고장 발생 시점과 다르고 재수행 구간이 데이터를 저장하는 구간으로 고정된다는 단점이 있으나 간단하게 고장 탐지가 가능하기 때문에 널리 사용되는 방법이다. 데이터를 저장하기 전에 고장 발생 유무가 탐지하므로 체크포인터에 저장된 데이터는 항상 고장이 없는 온전한 값을 유지한다.

그림 1과 같은 체크포인터 시스템에서 체크포인터가 삽입되기 전 원래 태스크의 수행 시간을 e라고 하고, 유효성 검사 등 고장 탐지 과정을 수행하는 데 걸리는 시간을 $t_d$, 그리고 데이터를 저장하는 데 걸리는 시간을 $t_s$라고 하자. 태스크 수행 중 m개의 체크포인터를 삽입한다고 하면 고장이 없을 때 태스크의 실행시간 E와 체크포인터 사이의 구간 Δ는 다음과 같이 구할 수 있다[7].

$$E = e + m \cdot (t_d + t_s) \quad (1)$$

$$\Delta = e/m + t_d \quad (2)$$

매 체크포인터에서는 고장 탐지와 데이터 저장 과정이 수행된다. 고장 탐지는 매 체크포인터의 끝에서 수행되므로 고장이 탐지되었을 때 재수행되는 구간은 항상 Δ이다. 태스크 수행 중 고장에 의해 r개의 체크포인터에서 재수행이 일어났다고 했을 때 재수행에 의해 늘어난 태스크의 수행 시간을 E(r)로 정의하자. E(r)은 다음 식과 같이 구할 수 있다.

$$E(r) = E + r \cdot \Delta = e + m \cdot (t_d + t_s) + r \cdot \Delta \quad (3)$$

태스크의 데드라인을 D라고 하면 고장 극복을 위해 사용가능한 재수행 횟수의 최대값 $r_{max}$은 다음과 같이 구할 수 있다.

$$r_{max} = \left\lfloor \frac{D - (e + m(t_d + t_s))}{\Delta} \right\rfloor \quad (4)$$

여기서 $\lfloor x \rfloor$는 x를 넘지 않는 최대 정수이다. 따라서 그림 1에서 제시된 기존 체크포인터 삽입 방법의 경우 최대 $r_{max}$ 구간에서 발생한 고장을 극복할 수 있다. $r_{max} < 0$일 때는 태스크의 수행 시간이 데드라인 D를 넘는 경우이다.

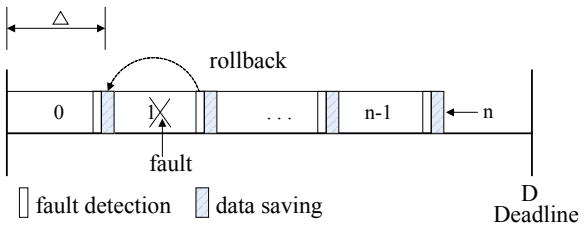


그림 1. 기존의 체크포인터 삽입 방법
Fig. 1. Conventional checkpoints

그림 2는 본 논문에서 제시하는 체크포인터 삽입 과정을 나타낸 것이다. 제안된 체크포인터 방식은 데이터 저장 과정과 고장 탐지 과정을 분리한 것이다. 매 체크포인터에서는 기존 방식과 동일하게 고장 탐지와 데이터 저장 과정이 수행된다. 하지만 하나의 체크포인터 구간 Δ내에 n개의($n \geq 1$) 고장 탐지 과정을 실행하여 고장 발생 후 탐지까지의 시간을 줄일 수 있다. n=1인 경우는 체크포인터 구간 내에 하나의 고장 탐지 과정만 삽입되므로 그림 1의 기존 체크포인터 방식과 동일하게 된다. n>1인 경우는 기존 체크포인터 구간에 별도로 n-1개의 고장 탐지 과정이 추가됨으로써

기존 방식에 비해 오버헤드는 증가하지만 고장 탐지 후 재실행되는 구간을 줄일 수 있다.

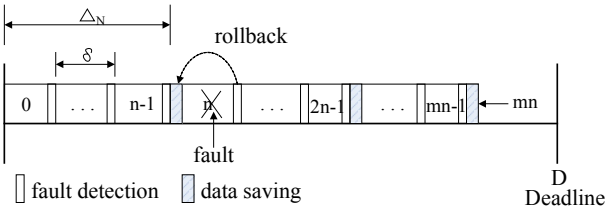


그림 2. 고장 탐지와 데이터 저장이 분리된 체크포인터 방식
Fig. 2. Checkpoints with separated fault detection and data saving

그림 2에서 보인 바와 같이 하나의 고장 탐지 과정이 수행된 후 다음 고장 탐지까지의 구간을 $\delta$라고 하자. 이때 고장 탐지 간격 $\delta$와 고장이 없는 경우의 태스크의 실제 실행 시간 $E_N$, 그리고 체크포인터 구간 $\Delta_N$는 다음 식과 같이 구할 수 있다.

$$\delta = e/(n \cdot m) + t_d \quad (5)$$

$$E_N = e + nm \cdot t_d + m \cdot t_s \quad (6)$$

$$\Delta_N = e/m + n \cdot t_d \quad (7)$$

여기서 m은 기존 방식에서처럼 고장 탐지와 데이터 저장을 함께 수행하는 체크포인터 개수이고, n은 하나의 체크포인터 구간 내에 실행되는 고장 탐지 과정의 개수이다. 그림 2에서와 같이 n번째 고장 탐지 구간에서 고장이 발생하면 데이터가 저장된 가장 가까운 체크포인터까지 $\delta$구간만큼 회귀하여 태스크를 재실행한다. 만약 (n+1)번째 고장 탐지 구간에서 고장이 발생하였다면 데이터가 저장된 가장 가까운 체크포인터까지 $2\delta$만큼 재실행이 일어난다. (n+2)번째 구간에서 고장이 발생하면 $3\delta$만큼 재실행되며, 이 관계를 일반화해서 (2n-1)번째 구간에서 고장이 발생하였다면 $n\delta$ ($=\Delta_N$)만큼 재실행된다. 한편 그림 1의 기존 방식에서는 항상 데이터 저장 시 고장을 탐지하므로 일률적으로 $\Delta$만큼 재수행되기 때문에 제안된 방법과 차이를 보임을 알 수 있다. n의 최대값 $n_{max}$는 다음과 같이 구할 수 있다.

$$n_{max} = \left\lfloor \frac{D-(e+mt_s)}{mt_d} \right\rfloor \quad (8)$$

## 3. 실시간 태스크 모델링 및 성공 확률 계산

본 논문의 목표는 고장 탐지 과정이 분리된 경우에 태스크가 데드라인 이내에서 성공적으로 수행될 확률을 최대로 하는 체크포인터 개수와 고장 탐지 과정의 개수를 구하는 것이다. 고장 발생이 발생율 $\lambda$를 가진 Poisson 분포를 따른다고 하자. 데이터를 저장하는 시간 $t_s$ 이내에서는 고장이 발생하지 않는다고 가정한다. 하나의 고장 탐지 구간 $\delta$에서 고장이 발생하지 않을 확률 p와 고장이 1개 이상 발생할 확률 q는 다음과 같이 구할 수 있다.

$$p = e^{-\lambda\delta}, \quad q = 1 - e^{-\lambda\delta} \quad (9)$$

본 논문에서 제안된 그림 2의 체크포인터 시스템에서 태스크의 수행은 m·n개의 고장 탐지 구간으로 구성된다. m은 데이터가 저장되는 체크포인터 수를 의미하며 n은 하나의 체크포인터 내에 수행되는 고장 탐지 과정의 수이다. 그림 2에서 나타낸 바와 같이 i번째 고장 탐지 구간을 수행하고 있는 상태를 $x_i$(0≤i≤mn-1)라고 정의하자. $x_0$는 "0"번째 구간을 수행하는 초기 상태를 나타낸다. 또한 $x_{mn}$을 태스크의 수행이 완료된 상태라고 정의한다. 상태 $x_{vn}$(0≤v≤m-1)는 데이터 저장과정이 실행되는 v번째 체크포인터의 바로 다음 구간을 나타낸다. 즉 vn번째 고장 탐지 구간을 수행하는 상태를 말한다. 고장이 탐지되면 데이터가 저장되어 있는 가장 가까운 체크포인터, 즉 $x_{vn}$으로 회귀한다. 또한 i번째 고장 탐지 구간을 수행하는 상태 $x_i$에서 고장이 발생하지 않으면 $x_{i+1}$로 진행한다.

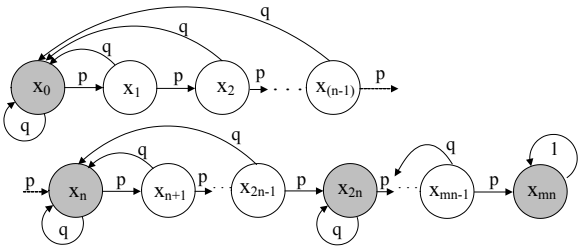


그림 3. 고장 탐지 과정이 분리된 체크포인터의 마코프 모델
Fig. 3. Markov model for the checkpoint system with a separated fault detection

그림 3은 고장 탐지 과정이 분리된 체크포인터 시스템에 대한 마코프 모델을 보여준다. $k\delta \le t < (k+1)\delta$ 시간에서 시스템이 $x_i$에 있을 확률을 $\gamma_i(k)$(0≤i≤mn)이라고 하자. 이 정의에 따라서 $\gamma_i(k+1)$은 $(k+1)\delta \le t < (k+2)\delta$ 시간에 $x_i$에 있을 확률이다. $k\delta \le t < (k+1)\delta$ 구간에서 발생한 고장은 $t=(k+1)\delta$ 시점에 고장 탐지 과정이 실행될 때 탐지된다. $\gamma_0(k+1)$은 $(k+1)\delta \le t < (k+2)\delta$ 시간에 상태 $x_0$에 있을 확률, 즉 "0"번째 고장 탐지 구간이 실행되는 확률이다. 확률 $\gamma_0(k+1)$은 귀납적 방법에 의해서 다음과 같이 구할 수 있다. 만약 $k\delta \le t < (k+1)\delta$ 구간에서 시스템이 상태 $x_0$에 있을 때 고장이 발생하면(이때의 고장 발생 확률은 q), $t=(k+1)\delta$ 시점에 고장이 탐지되므로 시스템은 다시 초기 상태 $x_0$로 회귀하여 "0"번째 구간을 재실행한다. 따라서 $(k+1)\delta \le t < (k+2)\delta$에서도 계속 $x_0$에 머무른다. 이 사건이 일어날 확률은 $q\gamma_0(k)$이다. 그림 3의 마코프

모델에서 $x_0$에서 다시 $x_0$로 가는 확률 q가 이 경우를 표현한다. 또한 $k\delta \le t < (k+1)\delta$ 구간에서 시스템이 상태 $x_1$에 있을 때 고장이 발생하여도 시스템은 다시 $x_0$로 회귀하므로 $(k+1)\delta \le t < (k+2)\delta$ 구간에서 $x_0$에 머무른다. 이 사건이 일어날 확률은 $q\gamma_1(k)$로 표현된다. 그림 3의 마코프 모델에서 $x_1$에서 $x_0$로 천이하는 확률 q가 이 경우에 대응된다. 이러한 해석을 일반화하면 시스템이 상태 $x_{n-1}$에서 $x_0$로 천이하는 확률은 $q\gamma_{n-1}(k)$이 된다. 따라서 $(k+1)\delta \le t < (k+2)\delta$에 $x_0$에 있을 확률 $\gamma_0(k+1)$은 위의 모든 경우의 확률을 합하면 된다. $\gamma_0(k+1)$는 다음과 같다.

$$\gamma_0(k+1) = q\gamma_0(k) + q\gamma_1(k) + \cdots + q\gamma_{n-1}(k) \quad (10)$$

그림 3에서 회색으로 표시된 $x_{vn}$는 앞에서 설명하였듯이 데이터 저장이 수행되는 v번째 체크포인터의 바로 다음 구간, 즉 vn번째 구간을 실행하는 상태이다. 따라서 $x_{vn+j}(0 \le j \le n-1, 0 \le v \le m-1)$ 상태에서 고장이 탐지되면 가장 가까운 체크포인터인 $x_{vn}$로 되돌아가서 태스크를 재실행한다. $(k+1)\delta \le t < (k+2)\delta$ 구간에서 시스템이 $x_{vn}$에 있을 확률 $\gamma_{vn}(k+1)(1 \le v \le m-1)$은 식 (10)의 유도 과정과 비슷하게 구할 수 있다. $k\delta \le t < (k+1)\delta$ 구간에서 상태 $x_{vn+j}$에 있을 때 고장이 발생하면 $x_{vn}$으로 회귀한다. 이 확률은 $q\gamma_{vn+j}(k)$로 표현된다. 또한 $k\delta \le t < (k+1)\delta$ 시간에서 (vn-1)번째 구간을 실행하였을 때 고장이 없으면(이 확률은 p) vn번째 구간을 수행하는 상태 $x_{vn}$로 진행한다. 이때의 확률은 $p\gamma_{vn-1}(k)$이다. 따라서 상태 $x_{vn}$로 천이하는 확률은 위의 모든 경우의 확률을 더하면 된다.

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

그림 2의 체크포인터 시스템은 상태 $x_{vn+j}$에서 고장이 없으면 $x_{vn+j+1}$로 천이한다. 상태 $x_{vn+j}$에 있으면서 $k\delta \le t < (k+1)\delta$ 구간에 고장이 없을 확률은 $p\gamma_{vn+j}(k)$이다. 따라서 $(k+1)\delta \le t < (k+2)\delta$ 구간에 시스템이 상태 $x_{vn+j+1}$에 있을 확률 $\gamma_{vn+j+1}(k+1)(0 \le v \le m-1, 0 \le j \le n-2)$은 다음과 같다. 여기서 v의 인덱스는 0부터 시작됨에 유의하자.

$$\gamma_{vn+j+1}(k+1) = p\gamma_{vn+j}(k) \quad (12)$$

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

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

식 (4)와 유사하게 데드라인 이내에서 재수행이 가능한 $\delta$ 구간의 개수 $r_{max}(\ge 0)$는 다음과 같이 구할 수 있다.

$$r_{max} = \left\lfloor \frac{D - (e + mnt_d + mt_s)}{\delta} \right\rfloor \quad (14)$$

데드라인 이내에 태스크의 수행이 끝나도록 하는 k의 최대값 $k_{max}$는 태스크에 삽입된 mn개의 $\delta$ 구간의 수와 재수행이 가능한 최대 $\delta$ 구간의 개수 $r_{max}$를 합한 것이다.

$$k_{max} = mn + r_{max} \quad (15)$$

따라서 데드라인 D 이내에 태스크의 모든 수행이 끝날 확률은 아래 식과 같이 그림 3의 마코프 체인 모델이 $k_{max}$ 스텝 후에 $x_{mn}$ 상태에 있을 확률이다.

$$\text{Prob(task completion within D)} = \gamma_{mn}(k_{max}). \quad (16)$$

## 4. 시뮬레이션 결과

모의실험에 사용된 태스크의 데드라인 D, 태스크의 원래 실행시간 e, 고장 탐지에 소요되는 시간 $t_d$, 데이터 저장에 소요되는 시간 $t_s$, 그리고 평균 고장 발생율 $\lambda$는 다음과 같다.

$$D=10,\ e=8,\ t_d=0.05, t_s=0.1,\ \lambda=0.1$$

그림 4는 태스크가 데드라인 이내에서 성공적으로 수행이 끝날 확률 $\gamma_{mn}(k_{max})$을 나타낸 것이다. x축은 데이터 저장 과정이 포함된 체크포인터 수 m이고 y축은 한 개의 체크포인터 구간에 삽입된 고장 탐지 과정의 수 n이다. m과 n에 따라 식 (5), (14), (15)의 $\delta$, $r_{max}$, 그리고 $k_{max}$ 값이 결정된다. 그림에서 알 수 있듯이 태스크 수행 확률을 최대로 하는 지점은 m=3, n=2일 때이다. 즉 태스크에 데이터를 저장하는 3개의 체크포인터를 삽입하고 각 체크포인터 구간 내에는 2개의 고장 탐지 과정을 실행할 때가 가장 좋다는 것을 보여준다.

앞의 모의실험 결과로 유도되는 최적의 고장 탐지구간 $\delta$, 최적 체크포인터 구간 $\Delta_N$, 이때의 태스크 수행시간 $E_N$, 재수행이 가능한 $\delta$ 구간의 개수 $r_{max}$는 다음과 같다.

$$\delta = e/(n \cdot m) + t_d = 8/(2 \cdot 3) + 0.05 = 1.383$$

$$\Delta_N = e/m + n \cdot t_d = 8/3 + 2 \cdot 0.05 = 2.767$$

$$E_N = r_{max} = \left\lfloor \frac{D - E_N}{\delta} \right\rfloor = \left\lfloor \frac{10 - 8.6}{1.383} \right\rfloor = 1$$

그림 5는 태스크 실행시간, 고장 발생률 등의 매개변수 값을 다음과 같이 설정했을 때 나오는 시뮬레이션 결과이다.

$$D=10, e=6, t_d=0.05, t_s=0.05, \lambda=0.5$$

그림 5는 그림 4의 실험 설정에서 데이터 저장에 소요되는 오버헤드와 태스크의 실행시간을 증가시킨 후 구한 실험 결과이다. 태스크의 수행 확률을 최대로 하는 지점은 m=17, n=1일 때이다. 즉 태스크에 17개의 체크포인터를 삽입하고 각 체크포인터 구간에 한 번의 고장 탐지 과정을 넣는 것이 최적이라는 사실을 알 수 있다.

$$\delta = 6/17+0.05 = 0.403$$
$$\Delta_N = 6/17+0.05 = 0.403$$
$$E_N = 6+17 \cdot 0.05+17 \cdot 0.05 = 7.7$$
$$r_{\max} = \left\lfloor \frac{10-7.7}{0.403} \right\rfloor = 5$$

## 5. 결 론

본 논문에서는 고장 탐지 과정을 분리한 체크포인터 삽입 방법을 제안하였다. 매 체크포인터에서 수행되는 고장 탐지와 데이터 저장 과정 중에서 고장 탐지 부분을 분리하여 체크포인터 구간 내에 따로 삽입하는 방법을 제시하였다. 제안된 체크포인터를 가진 태스크를 마코프 체인으로 모델링하고 데드라인 이내에서 성공적으로 수행이 끝날 확률을 계산하였다. 이 확률을 기반으로 태스크의 수행시간과 데드라인, 고장 발생율, 체크포인터 오버헤드 등이 주어졌을 때 최적의 체크포인터 구간과 고장 탐지 구간을 구하였다. 마지막으로 모의실험을 통하여 본 논문에서 제시된 방법을 검증하였다.

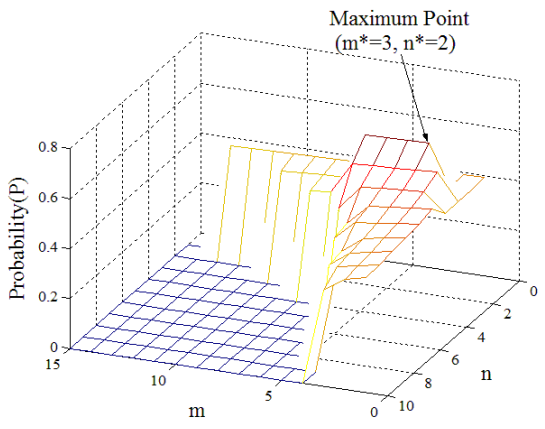


그림 4. 데드라인 이내에서 태스크의 성공적 수행 확률
(D=10, e=6, $t_d$=0.05, $t_s$=0.05, $\lambda$ =0.5)
Fig. 4. Probability of a task completion within the deadline
(D=10, e=6, $t_d$=0.05, $t_s$=0.05, $\lambda$ =0.5)

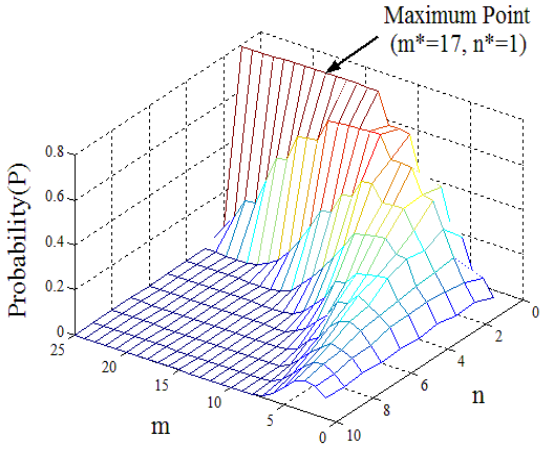


그림 5. 데드라인 이내에서 태스크의 성공적 수행 확률
(D=10, e=8, $t_d$=0.05, $t_s$=0.1, $\lambda$ =0.1)
Fig. 5. Probability of a task completion within the deadline
(D=10, e=8, $t_d$=0.05, $t_s$=0.1, $\lambda$ =0.1)

## 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, pp. 83-102, 2001.

[2] 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, pp. 130-140, 2006.

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

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

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

[6] S. W. Kwak and J.-M. Yang, "Determining checkpoint intervals of non-preemptive rate monotonic scheduling using probabilistic optimization," *Journal of Korean Institute of Intelligent Systems*, vol. 21, no. 1, pp. 120-127, 2011.

[7] S. W. Kwak and J.-M. Yang, "Optimal checkpoint placement for real-time systems with multi-tasks having deadlines longer than periods," *Transactions of the Korean Institute of Electrical Engineers*, vol. 61, no. 1, pp. 148-154, 2012.

## 저 자 소 개

**곽성우(Seong Woo Kwak)**

1993년 : 한국과학기술원 전기및전자공학과 (공학사)
1995년 : 한국과학기술원 전기및전자공학과 (공학석사)
2000년 : 한국과학기술원 전기및전자공학과 (공학박사)
2003년~현재 : 계명대학교 전자공학과 교수

관심분야 : 실시간 시스템, 교정제어, 우주용 시스템 설계, 무인 자율주행 제어
E-mail : ksw@kmu.ac.kr

**양정민(Jung-Min Yang)**

1993년 : 한국과학기술원 전기및전자공학과 (공학사)
1995년 : 한국과학기술원 전기및전자공학과 (공학석사)
1999년 : 한국과학기술원 전기및전자공학과 (공학박사)
2013년~현재 : 경북대학교 전자공학부 교수

관심분야 : 비동기 머신 교정 제어, 실시간 시스템 고장 진단 및 극복, 불리언 제어 네트워크
E-mail : jmyang@ee.knu.ac.kr