1999년도 대한전기학회 하계학술대회 논문집 1999. 7.19-21

고성능컴퓨터의 고신뢰도보장을 위한 이중(Duplex) 시스템의 작업 할당/시퀀싱 기법 연구

임한승,김학배
연세대학교 전기 · 컴퓨터공학과

A Task Scheduling to Minimize the Effect of Coincident Faults in a Duplex Controller Computer with Time Constraints

Hanseung Lim and Hagbae Kim
Dep. of Electrical & Computer Engineering, Yonsei University

Abstract - 본 연구는 시스템의 신뢰도(reliability)를 향상시키기 위해 사용되는 이중(Duplex) 시스템에서, EMI(전자기파 간섭현상) 같은 원인에 의한 동시 발생적(coincident) 고장의 영향을 최소화하는 기법을 제안하고 신뢰성 있는 고성능 컴퓨터를 위한 운영체계 및 H/W 구조의 설계와 최적 평가에 기여하는데 그 목적이 있다. 이중 시스템에 동시 발생적 고장이 일어나면 두 개의 모듈이 고장의 영향을 받게 되므로 고장 포용능력을 상실하게 된다. 이 같은 영향을 최소화하기 위해서 같은 작업들을 가능한 한 다른 시간대로 중복 수행하도록 시퀀싱(sequencing) 및 스케쥴링(scheduling) 함으로써 동시발생적 고장으로 야기되는 전체 작업의 고장 결과를 피할 수 있다. 또한 실시간 시스템에서 작업들은 기본적으로 수행이 완료되어야 할 시간적 제약(hard deadline)을 지니고 있으므로, 이러한 엄격한 마감시한 내에서 모든 작업을 완수하고 기본조건을 만족시키고자 한다.

1. 서 론

고장포용(fault-tolerance)시스템은 추가적인 설비, 즉 여분(redundancy)을 이용하여 고장을 검출하고 고장의 영향을 없애주어 고장의 존재하에서도 작업이 올바르게 수행되도록 해주는 시스템이다. 여기서 추가적인 설비란 소프트웨어나 하드웨어, 시간, 또는 이들의 조합 등이 될 수 있다. 이중 시스템을 이용하면 한 모듈에서의 고장을 검출할 수 있으며, 별도의 고장진단 장비와 함께 고장을 극복시킬 수도 있다. 각각의 작업(task)은 동시에 두 개의 프로세싱 모듈(PM)에서 병렬로 수행되며, 이중 시스템을 형성한다. 고장의 검출은 별도의 비교기(comparator)를 이용하여 가능해 진다. 두 모듈에서 나오는 출력을 연속적으로 비교하여 고장이 발생했는지의 여부를 알아낸다. 이 결과에 불일치가 검출되면 진단(diagnostic) 프로그램이 고장의 위치를 찾아내고(fault location), 고장난 모듈을 떼어낸 뒤 단일(simplex) 시스템으로 재시작(reinitialization)시킬 수 있다.

그러나 동일하지 않은 모듈에서 다중(multiple) 고장이 거의 동시에 발생한다면, 고장의 검출이 불가능하게 된다. 이중 시스템은 다중 모듈에서 공통된 근원에 의한 동시발생적(coincident) 고장에 대해서는 고장 검출 능력을 상실하게 된다. 환경적인 외란(disruption)이나 공통된 구성요소의 기능불량(malfunction) 등에 의한 동시발생적 고장을 고려해야 한다. 예를들어, EMI에 의해 발생하는 고장은 이중시스템에 큰 확률로 동시발생적 고장을 발생시킬 수 있다.

이러한 동시발생적 고장의 영향을 줄이기 위하여 동일하지 않은 모듈에서의 작업들을 시퀀싱(sequencing) 및 스케쥴링(scheduling) 하는 방법을 제시한다. 작업들의 수행 시간과 순서를 두 모듈에서 달리함으로써 양 모듈에서 고장이 동시에 발생하더라도 고장 검출이 가능하도록 할 수 있다.

이와같은 착상에는 (i)다중 동시발생적 모듈 고장의 근원이 오래 지속되지 않는다는 가정과 (ii)작업들은 서로간에 독립적이라는 가정이 들어가 있다.

기존에 TMR(Triple Modular Redundancy) 시스템에서, 동일한 길이의 비주기적 작업에 대해 스케쥴링 기법을 적용하는 연구가 있었다[1]. 이 스케쥴링 기법에 의해 동시발생적 고장이 발생할 가능성을 줄여줄 수 있음을 보였다. 여기에서 나아가 본 논문에서는 비록 이보다 간단한 시스템이지만, 마감시한(deadline)을 고려하여 다양한 길이가 주어진 작업에 대해 새로운 스케쥴링 기법을 적용해보는 새로운 시도를 하고자 한다.

동일한 작업들을 수행할 때, 두 모듈에서 고장이 동시에 발생할 확률을 확률밀도함수(pmf)를 이용하여 구할 수 있다. 이러한 수식 유도 결과를 수치예제를 통해 입증해 보이고자 한다.

2. 본 론

2.1 기본적인 가정들

수행해야 할 모든 작업들은 비주기적으로 들어오며, 작업들은 수행시간(C_i)과 마감시한(D_i)에 대한 정보가 모두 주어져 있다고 가정한다. 이 정보를 이용하여 수행시작 시간(S_i)과 작업 거리(TD_i)를 결정하여 모든 작업의 수행 순서를 정하게 된다. 단, 이때 마감시한을 넘어서지 않도록 모든 작업이 수행되야 한다. 모든 작업들은 비선점형(nonpreemptable)이라 가정한다.

고장의 발생과 지속은 지수적(exponential) 분포를 갖으며, 각각의 발생 비율은 서로 독립적이라고 가정한다. 또한, 고장 지속은 너무 길지 않다고 가정한다. 고장발생과 지속에 관한 확률변수를 각각 X와 Y라고 정의한다.

2.2 작업들이 두 모듈에서 동시에 진행될 경우

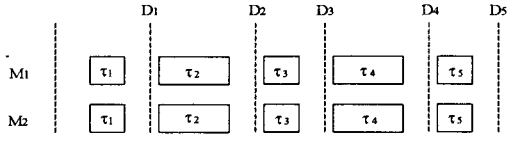


그림 1. 일반적인 이중 시스템의 작업 수행

작업수행이 우선적으로 이루어져야 하는 작업부터 수행된다. 마감시한에 해당하는 D_i가 작은 작업부터 수행되는데, 전체 시스템의 고장(dynamic failure)을 막기 위해서는 모든 작업들의 시작 시간은 반드시 D_i를 넘지 말아야 한다.

동시발생적 고장이 발생할 확률을 유도하기 위해 작업의 영역을 다음과 같이 두 부분으로 나눈다.

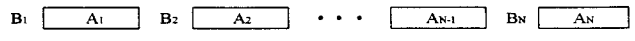


그림 2. 작업의 수행영역(영역A)과 비수행영역(영역B)

A영역은 작업이 수행되는 도중 고장이 발생하는 영역이며, B영역은 작업 수행이 일어나지 않을 때 고장이 발생하는 영역이다.

X와 Y는 각각 고장의 발생시간과 고장의 지속을 의미하는 확률변수이며 S_i는 모듈1의 i번째 작업의 수행 시작시간을 의미한다. B영역에서 고장이 발생할 경우는 $X+Y > S_i$이 될 때 동시발생적 고장을 발생시키게 한다. S_i'은 모듈2의 작업 수행 시간을 의미한다.

따라서 작업 전체가 수행되는 동안 동시발생적 고장이 한번이라도 일어날 확률을 다음과 같이 구할 수 있다.

$$P_c = \sum_{i=1}^{N} P(X \in A_i) + \sum_{i=1}^{N} P(X \in B_i)\, P(X+Y > S_i)$$

$$= \sum_{i=1}^{N} e^{-\lambda_X S_i}(1 - e^{-\lambda_X C_i})$$

$$+ (1 - e^{-\lambda_X S_1})\left(\frac{\lambda_X}{\lambda_X - \lambda_Y} e^{-\lambda_Y S_1} + \frac{\lambda_Y}{\lambda_Y - \lambda_X} e^{-\lambda_X S_1}\right) \times$$

$$\left(\frac{\lambda_X}{\lambda_X - \lambda_Y} e^{-\lambda_Y S_i} + \frac{\lambda_Y}{\lambda_Y - \lambda_X} e^{-\lambda_X S_i}\right)$$

여기서 λ_X는 동시발생적 고장의 발생 비율을 의미하며, λ_Y는 동시발생적 고장의 지속 비율을 의미하는데, 이러한 파라미터는 field data와 추정 기법을 통해 구해질 수 있다.

2.2 작업들이 두 모듈에서 시간차를 두고 진행될 경우

그림3 에서와 같이 모든 작업들은 두 모듈에서 될 수 있는 한 시간적 간격을 많이 두고 수행된다. 모듈1에서는 가능한 한 우선적으로 작업수행을 시작하며, 모듈2에서는 마감시한에 임박하여 작업이 수행된다. 마감시한을 넘지 않는 범위내에서 시간 여분을 최대한 활용한다. 작업 수행의 마지막 단계에선 별도의 비교회로를 통해 두 모듈의 결과값을 비교하는 시간이 포함되어 있다.

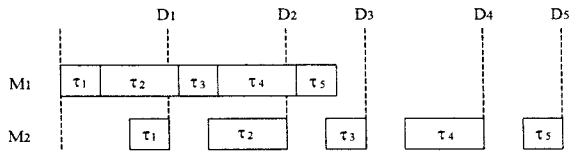


그림 3. 시퀀싱 기법이 적용된
이중 시스템의 작업 수행

영역 A에서 고장이 발생할 경우는 다음 그림과 같으며, $Y > TD_i$가 될 때 동시발생적 고장의 영향을 받는다. 영역 A에서 동시발생적 고장이 일어나더라도 양 모듈의 수행시간에 차이가 있으므로 고장의 영향을 피할 수가 있다.

영역 B에서 동시발생적 고장이 일어날 경우에는 $X+Y > S_i' = S_i + TD_i$가 될 때 작업들이 동시발생적 고장의 영향을 받는다. 영역 A에서와 마찬가지로 고장의 지속이 충분이 길 때에만 동시발생적 고장의 영향을 받게 되므로 시퀀싱기법을 적용하기 전보다 고장의 영향을 덜 받게 된다.

따라서 작업 전체가 수행되는 동안 동시발생적 고장이 일어날 확률이 다음과 같이 구해진다.

$$P_c = \sum_{i=1}^{N} P(X \in A_i)\, P(Y > TD_i)$$

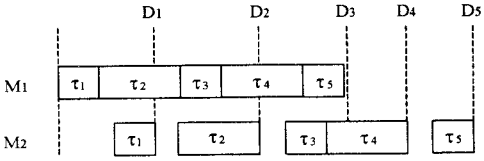
$$+ \sum_{i=1}^{N} P(X \in B_i \text{ and } X+Y > S_i')$$

$$= \sum_{i=1}^{N} e^{-\lambda_X S_i}(1 - e^{-\lambda_X C_i}) e^{-\lambda_Y TD_i}$$

$$+ \frac{\lambda_X}{\lambda_Y - \lambda_X} e^{-\lambda_Y S_1'}(e^{S_1(\lambda_Y - \lambda_X)} - 1)$$

$$+ \sum_{i=2}^{N} \frac{\lambda_X}{\lambda_Y - \lambda_X} e^{-\lambda_Y S_i'}(e^{S_i(\lambda_Y - \lambda_X)} - e^{(S_{i-1} + C_{i-1})(\lambda_Y - \lambda_X)})$$

모듈2에서 두 개의 작업이 중복되게(overlapping) 될 경우는 마감시한이 늦은 작업에 우선순위를 주어 마감시한 이내에 작업 수행을 마칠수 있도록 한다.



priority $\tau_4 > \tau_3$

그림 4. 작업3과 작업 4의 수행 시간이 겹치는 경우

따라서 모듈1과 모듈2에서의 작업수행 시작시간은 다음과 같이 결정된다.

$$\begin{cases} S_{i+1} = S_i + C_i \\ S_{N-i}' = \text{Min}\,(S_{N-i+1}' - C_{N-i},\ D_{N-i} - C_{N-i}) \end{cases} \quad (1 \le i \le N-1)$$

마감시한이 동일한 작업이 있을 경우에는 이와는 다른 방식을 적용한다. 만일 동일한 마감시한을 갖는 작업들을 수행함에있어서 충분한 시간여유가 있다면, 임의로 정해진 모듈1의 작업수행 순서에대해 모듈2의 작업수행은 최대한 늦추어 수행하며 모듈1과 같은 순서로 이루어 진다.

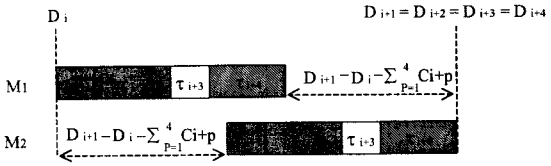


그림9. 동일한 마감시한을 갖는 4개의 작업들
(충분한 시간여유가 있을 경우)

마감시한이 동일한 작업들이 M개 있을 경우 작업수행 시작시간은 다음과 같이 정해진다.

$$D_{i+1} - D_i \ge \frac{1}{2} \sum_{p=1}^{M} C_{i+p} + C_{\max} \quad \text{일때}$$

$$S_{i+q} = \begin{cases} D_i & \text{if } q = 1 \\ D_i + \sum_{p=1}^{q-1} C_{i+p} & \text{if } 2 \le q \le M \end{cases}$$

$$S_{i+q}' = \begin{cases} D_{i+1} - \frac{1}{2} \sum_{p=1}^{M} C_{i+p} & \text{if } q = 1 \\ D_i + \sum_{p=1}^{q-1} C_{i+p} & \text{if } 2 \le q \le M \end{cases}$$

그러나, 동일한 마감시한을 갖는 작업들의 수행에 있어서 시간적 여유가 충분하지 못하다면 두 모듈의 작업들의 수행순서를 뒤바꾼다.

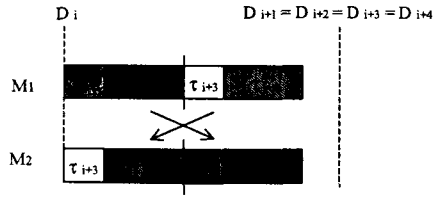


그림10. 동일한 마감시한을 갖는 4개의 작업들
(충분한 시간여유가 없을 경우)

이 때의 작업수행 시작시간은 다음과 같이 정해진다.

$$D_{i+1} - D_i < \frac{1}{2}\sum_{p=1}^{M} C_{i+p} + C_{\max} \quad \text{일때}$$

$$S_{i+q} = \begin{cases} D_i & \text{if } q=1 \\ D_i + \sum_{p=1}^{q-1} C_{i+p} & \text{if } 2 \le q \le M \end{cases}$$

$$S_{i+q}' = \begin{cases} D_i - \sum_{p=k+1}^{M} C_{i+p} & \text{if } q=1 \\ D_i + \sum_{p=k+1}^{M} C_{i+p} + \sum_{p=1}^{q-1} C_{i+p} & \text{if } 2 \le q \le M \\ D_i & \text{if } q=k+1 \\ D_i + \sum_{p=k+1}^{q-1} C_{i+p} & \text{if } k+2 \le q \le k+M \end{cases}$$

여기서 $C_{\max}$는 동일한 마감시한을 갖는 작업들 중 최대 수행시간을 의미한다. 또한 k는 1부터 M까지의 정수 중에서 $\left| \sum_{p=1}^{k} C_{i+p} - \frac{1}{2}\sum_{p=1}^{M} C_{i+p} \right|$을 최소화시키는 정수로 정한다.

2.3 수치예제

작업들의 마감시한과 수행 시간이 각각 {3,6,9,13,18, 20,24,26,29,32}와 {2,3,1,4,3,3,2,5,2,2}일 경우일 때, 동시발생적 고장의 발생비율(λ_X)과 지속비율(λ_Y)을 변화시켜가며 고장의 영향을 받는 작업의 수를 세어본다.

(1) 두 모듈의 작업 수행이 동시에 진행될 경우

두 모듈에서 작업들의 수행 시작시간이 {1,3,7,9, 13,16,19,21,27,30}로 동일하며 동시발생적 고장의 발생과 지속비율이 각각 10^{-4}와 0.5인 경우를 살펴본다. 이 경우 10개의 작업들 중 동시발생적 고장의 영향을 받게되는 작업의 개수는 평균 2.6×10^{-3}개가 나온다. 또한, 이와 동일한 작업들에 대해 동시발생적 고장의 발생과 지속비율이 각각 5×10^{-4}와 0.5인 경우 고장의 영향을 받는 작업의 개수는 1.41×10^{-3}개가 나왔다.

(2) 두 모듈의 작업 수행이 거리를 두고 진행될 경우

두 모듈에서의 작업 수행이 주어진 스케쥴링 방식에 따라 변형되어 진행된다. 모듈1에서의 작업 수행 시작 시간은 {0,2,5,6,10,13,16,18,23,25}이 되며, 모듈2에서의 작업 수행 시작 시간은 {1,3,8,9,13,16,19,21,27,30} 이 되며 따라서 모듈1과 모듈2에서 수행되는 작업들간 거리는 {1,1,3,3,3,3,3,3,4,5}이 된다. 동시발생적 고장의 발생과 지속비율이 각각 10^{-4}와 0.5인 경우를 살펴보면, 10개의 작업들 중 동시발생적 고장의 영향을 받게 되는 작업의 개수는 평균 7.4×10^{-4}개가 나온다. 또한, 이와 동일한 작업들에 대해 동시발생적 고장의 발생과 지속비율이 각각 5×10^{-4}와 0.5인 경우 고장의 영향을 받는 작업의 개수는 3.8×10^{-3}개가 나왔다.

이같은 결과를 통해 본 논문에서 제안한 시퀀싱기법 적용 이후에 동시발생적 고장의 영향을 받는 작업의 개수가 현격히 줄어드는 것을 알 수 있다.

고장발생비율 / 고장난 작업개수	10^{-5}	10^{-4}	5×10^{-4}
시퀀싱 이전	0.000323	0.002633	0.014112
시퀀싱 이후	0.00006	0.000737	0.003814

3. 결 론

이중시스템은 동시발생적 고장에 대해서 고장 검출 능력을 상실하게 되므로 이에 대한 영향을 염두해 두어야 한다. 공통된 구성요소의 기능불량 또는 EMI와 같은 외란 등은 공간여분을 이용한 이중 시스템에 고장을 발생시킬 수 있다. 이같은 고장에 의한 피해를 줄이기 위해 두 모듈의 태스크 수행에 시간차를 두는 시퀀싱 및 스케쥴링 방법을 제시하였다. 또한 수치예제를 통해 본 결과 제시된 시퀀싱 및 스케쥴링 기법이 효과적임을 보였다. 이처럼 시간여분을 추가시킨 이중시스템은 고장의 발생에 영향받지 않고 보다 높은 신뢰도(Reliability)를 제공할 수 있을 것이다. 또한 실시간 고장포용 시스템의 운용체계(O/S) 설계에 이 방식을 적용시키면 병렬로 수행되는 컴퓨터의 모든 작업들이 동시 발생적 고장에 대해 보다 높은 안정성을 보장받을 수 있게 될 것이다. 따라서 이 시퀀싱 방식은 높은 신뢰도를 지닌 고장포용 O/S를 설계하는데 중요한 방향을 제시해 준다.

[참 고 문 헌]

[1] H. Kim and K. Shin, "Task Sequencing to Minimize the Effect of Near-Coincident Faults in TMR Controller Computers", *IEEE Trans. on Computer*, vol. 5, no. 11, pp. 1331-1337, November, 1996

[2] K. Ramamrithan, "Scheduling Algorithms and Operation Systems Support for Real-Time Systems", *IEEE*, vol. 82, no. 1, pp. 55-67, Jan. 1994

[3] B. Kao and G. Molina, "Scheduling Soft Real-Time Jobs Over Dual Non-Real-Time Servers", *IEEE Trans. Parallel and Distributed System*, vol. 7, no. 1, pp. 56-68, Jan 1996

[4] H. Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm", *IEEE Trans. Software*, vol. 15, no. 10, pp. 1261-1269, Oct. 1989

[5] K. Shin, and Y. Chang, "A Reservation-Based Algorithm for Scheduling Both Periodic and Aperiodic Real-Time Tasks", *IEEE Trans. Computers*, vol. 44 no. 12 pp. 1405-1419 Dec. 1995

[6] A. Burns, K Tindell and A. Wellings, "Effective Analysis for Engineering Real-time Fixed Priority Schedulers", *IEEE Trans. Software*, vol. 21 no. 5 pp. 475-480 May. 1995

[7] J. Blazewicz, M. Drozdowski, D. de Werra and J. Weglarz, "Deadline Scheduling of Multiprocessor Tasks", *Discrete Applied Mathematics* 65 pp. 81-95 1996

[8] J. Hong, X. Tan and D. Towsley, "A Performance Analysis of Minimum Laxity and Earliest Deadline Scheduling in a Real-Time System", *IEEE Trans. Computer*, vol. 38, no. 12, pp. 1736-1744, Dec. 1989

[9] J. Dey and J. Kurose, "On-Line Scheduling Policies for a Class of IRIS Real-Time Tasks", *IEEE Tran.s Computers*, vol. 45, no. 7pp 802-813, July 1996