고성능 컴퓨터의 고신뢰도 보장을 위한 이중(Duplex) 시스템의 작업 시퀀싱/스케쥴링 기법 연구

임 한 승† · 김 학 배††

요 약

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 동시 발생적 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.

Task Scheduling to Minimize the Effect of Coincident Faults in a Duplex Controller Computer

Han-Seung Lim† · Hag-Bae Kim††

ABSTRACT

A duplex system enhances reliability by tolerating faults through spatial redundancy. Faults can be detected by duplicating identical tasks in pairs of modules. However, this kind of systems cannot even detect the fault if it occurs coincidently due to either malfunctions of common component such as power supply and clock or due to such environmental disruption as EMI. In the paper, we propose a method to reduce those effects of coincident faults in the duplex controller computer. Specifically, a duplex system tolerates coincident faults by using a sophistication sequencing of scheduling technique with certain timing redundancy. In particular when all tasks should be completed in the sense of real-time, the suggested scheduling method works properly to minimize the probability of faulty tasks due to coincident fault without missing the timing constraints.

1. 서 론

이중(duplex) 시스템은 한 모듈에서의 고장을 검출할 수 있으며, 별도의 고장진단 장비와 함께 고장을 극복시킬 수 있는 고장 포용(fault-tolerant) 시스템이다. 이러한 시스템에서는 각각의 작업이 동시에 두 개의 프로세싱 모듈(PM)에서 병렬로 수행 되며 고장의 검출은 별도의 비교기를 이용하여 가능해 진다. 비교된 결과에 불일치가 검출되면 진단 프로그램이 고장의 위치를 찾아내고(fault location), 고장난 모듈을 떼어낸

※ 본 논문은 정보통신연구진흥원의 1999년 대학기초지원사업의 지원 결과임.
† 준 회 원 : 연세대학교 대학원 전기·컴퓨터공학과
†† 정 회 원 : 연세대학교 전기·컴퓨터공학과 교수
논문접수 : 1999년 7월 14일, 심사완료 : 1999년 10월 12일

뒤 단일(simplex) 시스템으로 재시작 시킬 수 있다[1].

그러나 동일하지 않은 모듈에서 다중 고장이 동시에 발생한다면 동일한 한 쌍의 작업들이 고장의 영향을 받게 되므로 고장의 검출 및 포용이 불가능하게 된다. 따라서 환경적인 외란(disruption)이나 공통된 구성요소의 기능불량 등에 의해 발생할 가능성이 있는 동시 발생적 고장을 고려함은 주목해야 할 문제가 된다[2, 3]. 본 논문에서는 이러한 동시 발생적 고장의 영향을 줄이기 위하여 동일하지 않은 모듈에서의 작업들을 시퀀싱/스케쥴링하는 방법을 제시한다. 작업들의 수행 순서를 두 개의 모듈에서 달리함으로써 양 모듈에서 고장이 동시에 발생하더라도 고장 검출이 가능하도록 할 수 있다.

수행할 작업을 이중화시키고 고장을 감지하여 고장이 발생할 경우 고장난 작업을 여분의 모듈에서 재수행시키는 방법(roll-forward)을 제시한 연구가 있었다[4]. 그러나 이 방법으로는 동시 발생적 고장에 대해서는 고장난 모듈을 찾아낼 수가 없으며, 고장의 검출도 불가능하게 된다. 따라서 동시 발생적 고장의 검출 및 포용을 위한 다중 프로세서 시스템에서의 작업 스케쥴링 기법이 필요하다. 기존에 TMR(Triple Modular Redundancy) 시스템에서, 마감시한을 고려하지 않은 동일한 길이의 비주기적 작업에 대해 동시 발생적 고장을 극복하기 위한 스케쥴링기법을 적용하는 연구가 있었으며[2], 스케쥴링 기법에 의해 다중 모듈 시스템에서 동시 발생적 고장의 영향을 줄여줄 수 있음을 보였다. 여기에서 나아가 본 논문에서는 이중 시스템에서 마감시한(deadline)을 고려하여 다양한 길이의 비주기적 작업 및 주기적 작업에 대해 동시 발생적 고장을 포용하기 위한 새로운 시퀀싱/스케쥴링 기법을 제시한다.

동일한 작업을 수행할 때, 두 모듈에서 고장이 동시에 발생할 확률을 확률 밀도 함수(pdf)를 이용하여 구할 수 있다. 마감시한을 지닌 작업에 대해 제안하려는 스케쥴링 기법을 적용하고 두 모듈의 모든 작업들을 동일한 시간대로 중복 수행하는 일반적인 이중 시스템에서의 방법과 비교하여 제안된 기법의 유효성을 입증해 본다.

2. 고장 및 작업 모델

기존의 고장에 대한 연구들은 대부분 구성요소들의 독립적인 고장에 대해서 다루고 있다. 그러나 시스템의 구성요소들은 종종 동시 발생적 고장에 영향을 받게 된다. 하드웨어 모듈은 동일한 환경에 놓이게 되며, 종종 동일한 클럭(clock)과 전원 공급장치를 공유한다. 따라서 동일하지 않은 모듈에 대한 고장이 독립적으로 발생한다고 단정지을 수가 없다. 특히 번개, 높은 강도의 라디오 주파수 영역, 핵 전자기 펄스 등에서 발생하는 EMI 등의 환경적 외란은 이중 시스템의 모듈에 동시 발생적 고장을 발생시키는 근원이 될 수 있다[2, 3]. 본 연구에서의 모든 고장의 발생과 지속은 지수함수적(exponential) 분포를 갖는다고 가정하며, 고장의 발생과 지속은 서로 독립적이라고 가정한다. 고장발생과 지속에 관한 확률변수를 각각 X와 Y라고 정의하고, 고장의 발생과 지속의 비율을 각각 λ_X와 λ_Y라고 정의한다. 고장의 발생은 자주 일어나지 않는다고 가정한다.

하나의 작업 미션(mission)은 여러 개의 연속적인 단(phase)으로 나누어지며, 이러한 단은 몇 개의 세부작업을 수행함으로써 완료될 수 있다. 이 중 시스템의 작업들은 동시에 두 개의 모듈에서 수행이 된다. 작업의 수행이 완료된 후, 모듈의 출력 값을 비교하여 오류를 검출하게 된다. 스케쥴링 방식을 따를 경우, 수행결과를 언제 비교할 것인가를 결정하기가 어려워지는데, 이러한 어려움을 피해가기 위해서, 모든 작업의 수행 결과는 저장되었다가 나중에 비교된다고 가정한다. 또한 모든 작업들은 서로간에 독립적이라고 가정한다.

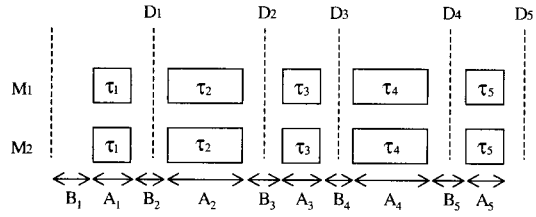
3. 비주기적 작업의 스케쥴링

비주기적인 작업에 대해, 작업 수행시간(C_i)과 마감시한(D_i)에 대한 정보가 모두 주어져 있다고 가정한다. 이 정보를 이용하여 작업의 수행 시작 시간(S_i)과 작업 거리(TD_i)를 결정하여 모든 작업의 수행 순서를 결정하게 된다. 모든 작업들은 비선점형(nonpreemtable)이라고 가정한다.

3.1 두 모듈의 작업 수행이 동시에 진행

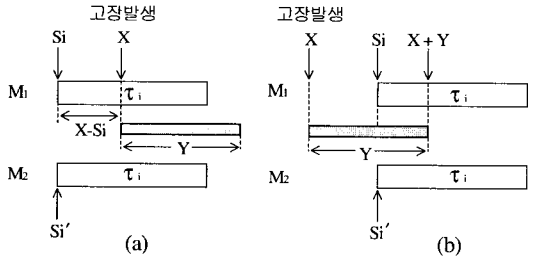
작업 수행이 우선적으로 이루어져야 하는 작업부터 실행된다. 마감시한에 해당하는 D_i가 작은 작업부터 수행되는데, 실시간 작업의 특성상 모든 작업들의 수행완료 시간은 반드시 D_i를 경과하지 말아야 한다. 동시 발생적 고장이 발생할 확률을 유도하기 위해 작업의 영역을 (그림 1)과 같이 두 부분으로 나눈다. A영

역은 작업이 수행되는 시간 영역이며, B영역은 작업 수행이 일어나지 않는 시간 영역이다.



(그림 1) 일반적인 이중 시스템에서의 작업수행

A영역에서 고장이 발생하는 경우는 (그림 2 (a))와 같으며, 이 영역에서 발생하는 동시 발생적 고장은 양 모듈의 작업에 직접 영향을 미치게 된다. S_i와 S_i'는 각각 모듈 1과 모듈 2의 i번째 작업의 수행 시작시간을 의미한다. B영역에서 고장이 발생할 경우에는 (그림 2 (b))와 같이 $X+Y > S_i$이 될 때 작업들이 동시 발생적 고장의 영향을 받게 된다.



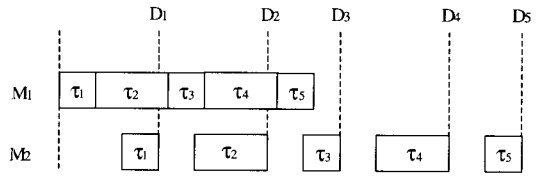
(그림 2) (a) 영역A (b) 영역B 에서의 고장발생

따라서 작업 전체가 수행되는 동안 최소한 한 개 이상의 작업이 동시 발생적 고장의 영향을 받게 될 확률을 다음과 같이 유도할 수 있다.

$$
\begin{aligned}
P_c = & \sum_{i=1}^{N} P(X \in A_i) + \sum_{i=1}^{N} P(X \in B_i)\, P(X+Y > S_i) \\
= & \sum_{i=1}^{N} e^{-\lambda_X S_i}(1-e^{-\lambda_X C_i}) \\
& + (1-e^{-\lambda_X S_1})\left(\frac{\lambda_X}{\lambda_X-\lambda_Y} e^{-\lambda_Y S_1} + \frac{\lambda_Y}{\lambda_Y-\lambda_X} e^{-\lambda_X S_1}\right) \\
& + \sum_{i=2}^{N} (e^{-\lambda_X (S_{i-1}+C_{i-1})} - e^{-\lambda_X S_i}) \times \\
& \left(\frac{\lambda_X}{\lambda_X-\lambda_Y} e^{-\lambda_Y S_i} + \frac{\lambda_Y}{\lambda_Y-\lambda_X} e^{-\lambda_X S_i}\right)
\end{aligned}
$$

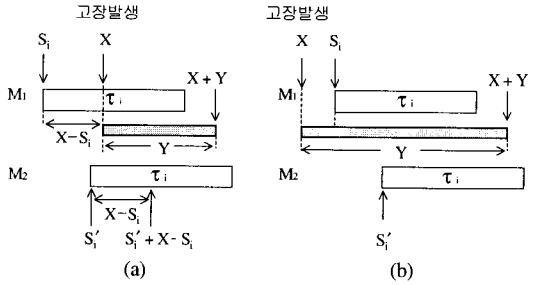
3.2 두 모듈의 작업 수행이 거리를 두고 진행

(그림 3)에서와 같이 모든 작업들은 두 모듈에서 될 수 있는 한 시간적 간격을 많이 두고 수행된다. 모듈 1에서는 가능한 한 우선적으로 작업수행을 시작하며, 모듈 2에서는 마감시한에 임박하여 작업이 수행된다.



(그림 3) 시퀀싱 기법이 적용된 이중시스템의 작업 수행

영역 A에서 고장이 발생할 경우는 (그림 4 (a))와 같으며, $Y > TD_i$가 되는 순간 작업들이 동시 발생적 고장의 영향을 받는다. 영역 B에서 고장이 발생할 경우는 (그림 4 (b))와 같으며, $X+Y > S_i' = S_i + TD_i$가 되는 경우 작업들이 동시 발생적 고장의 영향을 받게된다. 고장의 지속이 충분히 길 때에만 동시 발생적 고장의 영향을 받게 되므로 스케쥴링 기법을 적용하기 전보다 고장의 영향을 덜 받게 된다.

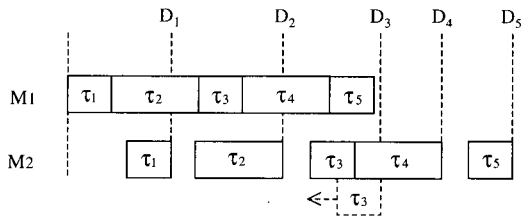


(그림 4) (a) 영역A (b) 영역B 에서의 고장발생

따라서 작업 전체가 수행되는 동안 최소한 한 개 이상의 작업이 동시 발생적 고장의 영향을 받게 될 확률은 다음과 같이 계산된다.

$$
\begin{aligned}
P_c = & \sum_{i=1}^{N} e^{-\lambda_X S_i}(1-e^{-\lambda_X C_i}) e^{-\lambda_Y TD_i} \\
& + \frac{\lambda_X}{\lambda_Y-\lambda_X} e^{-\lambda_Y S_1'}(e^{S_1(\lambda_Y-\lambda_X)}-1) \\
& + \sum_{i=2}^{N} \frac{\lambda_X}{\lambda_Y-\lambda_X} e^{-\lambda_Y S_i'}(e^{S_i(\lambda_Y-\lambda_X)} - e^{(S_{i-1}+C_{i-1})(\lambda_Y-\lambda_X)})
\end{aligned}
$$

만일, 모듈 2에서 두개의 작업이 중복 될 경우에는 마감시한이 늦은 작업의 수행 시작 시간을 앞당겨 주어 마감시한 이내에 작업 수행을 마칠 수 있도록 한다(그림 5).

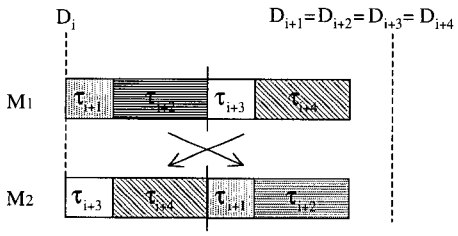


(그림 5) 작업 3과 작업 4의 수행 시간이 겹치는 경우

따라서 작업이 중복되는 경우를 고려한 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 그림 6과 같이 임의로 정해진 모듈 1의 작업 수행 순서에 대해 모듈 2의 작업 순서를 바꾸어 수행된다.



(그림 6) 동일한 마감시한을 갖는 네 개의 작업

이때의 작업 수행 시작 시간은 다음과 같이 정해진다.

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

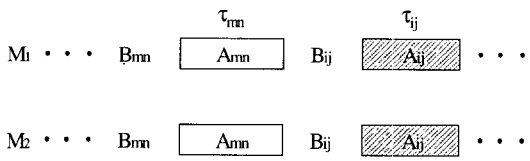
$$S_{i+q}' = \begin{cases} D_i - \sum_{p=k+1}^{M} C_{i+p} & \text{if } q=1 \\ D_i + \sum_{p=k+1}^{M} C_{i+p} + \sum_{p=1}^{q-1} C_{i+p} & \text{if } 2 \le q \le M \\ D_i & \text{if } q=k+1 \\ D_i + \sum_{p=k+1}^{q-1} C_{i+p} & \text{if } k+2 \le q \le k+M \end{cases}$$

여기서 M은 동일한 마감시한을 갖는 작업의 개수이고, $C_{\max}$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.

4. 주기적 작업의 스케쥴링

주기적인 작업에 대해, 작업 수행시간(C_i)과 주기(T_i)에 대한 정보가 모두 주어져 있다고 가정한다. 이 정보를 이용하여 작업의 수행 시작 시간(S_{ij})과 작업거리(TD_{ij}^k)를 결정하여 모든 작업의 수행 순서를 정하게 된다. 여기서 S_{ij}는 j 번째 수행하는 작업 i의 수행 시작 시간을 의미하며 작업 i의 주기는 T_i이다. 주기적인 작업의 스케쥴링은 기본적으로 RM(Rate Monotonic) 방식을 따르며, 작업 i의 마감시한과 주기는 동일한 것으로 가정한다. RM 방식에 따르면 빈번한 수행을 요하는 작업의 우선 순위가 높으므로 작업 수행과정에서 인터럽트를 고려해야 한다.

4.1 두 모듈의 작업 수행이 동시에 진행



(그림 7) 작업의 수행영역(영역A)과 비수행영역(영역B)

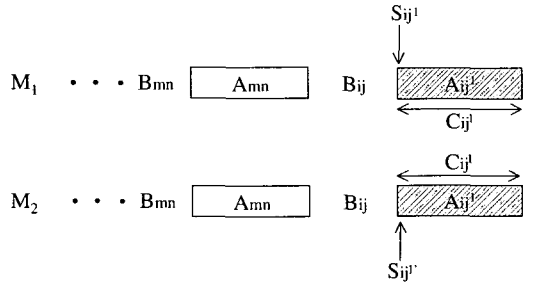
두 모듈의 작업들은 RM방식에 따라 (그림 7)과 같이 동시에 수행된다. 작업 τ_{ij} 바로 이전 작업을 τ_{mn}이라 한다. 작업 τ_{ij}가 동시 발생적 고장의 영향을 받을 확률값(P_c^{ij})을 유도하고, $1 \le i \le N$ 사이의 모든 정수 i와 $1 \le j \le \lceil \frac{t}{T_i} \rceil$ 사이의 모든 정수 j에 대해 P_c^{ij} 값을 구함으로써 전체 작업이 완료되기까지의 고장발생확률을 다음 식과 같이 구할 수 있다.

$$P_c = \sum_{i=1}^{N} \sum_{j=1}^{\lceil \frac{t}{T_i} \rceil} P_c^{ij} \qquad \lceil x \rceil : x\text{보다 큰 정수중 가장 작은 정수}$$

작업 τ_{ij}에 인터럽트가 한 번도 안 일어날 때, 영역 A_{ij}와 영역 B_{ij}에서 동시 발생적 고장이 일어날 확률이 다음과 같다.

$$\begin{aligned} P_c^{ij} &= P(X \in A_{ij}) + P(X \in B_{ij} \text{ and } X+Y > S_{ij}) \\ &= e^{-\lambda_X S_{ij}}(1 - e^{-\lambda_X C_i}) \\ &\quad + \frac{\lambda_X}{\lambda_Y - \lambda_X} e^{-\lambda_Y S_{ij}} \left(e^{S_{ij}(\lambda_Y - \lambda_X)} - e^{(S_{mn} + C_m)(\lambda_Y - \lambda_X)} \right) \end{aligned}$$

(그림 8)은 작업 τ_{ij}에 인터럽트가 한 번 발생하는 경우이며, 주기 T_q가 주기 T_i보다 작아서 작업 τ_{pq}가 작업 τ_{ij}보다 높은 우선 순위를 갖게 된다.

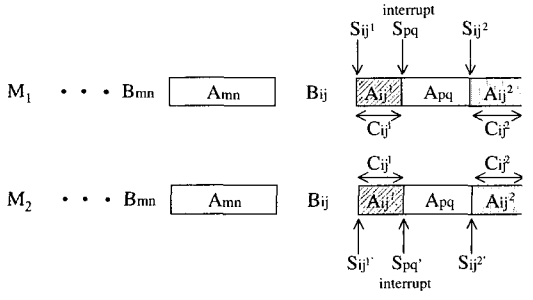


(그림 8) 인터럽트가 한 번도 발생하지 않을 경우

따라서 이 경우 동시 발생적 고장이 일어날 확률이 다음과 같이 구해진다.

$$P_c^{ij} = e^{-\lambda_X S_{ij}^1}(1-e^{-\lambda_X C_{ij}^1}) + e^{-\lambda_X S_{ij}^2}(1-e^{-\lambda_X C_{ij}^2}) + \frac{\lambda_X}{\lambda_Y-\lambda_X} e^{-\lambda_Y S_{ij}^1}\left(e^{S_{ij}^1(\lambda_Y-\lambda_X)} - e^{(S_{mn}+C_m)(\lambda_Y-\lambda_X)}\right)$$

(그림 9)는 작업 τ_{ij}에 인터럽트가 한 번 발생하는 경우이며, 이때 작업 τ_{ij}는 수행도중 작업 τ_{pq}와 작업 τ_{uv}에 의해 선점된다.



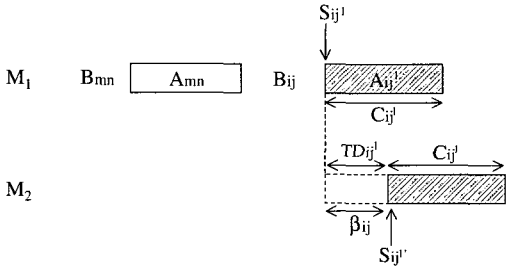
(그림 9) 인터럽트가 한 번 발생할 경우

따라서 동시 발생적 고장이 일어날 확률은 다음과 같이 구해진다.

$$P_c^{ij} = e^{-\lambda_X S_{ij}^1}(1-e^{-\lambda_X C_{ij}^1}) + e^{-\lambda_X S_{ij}^2}(1-e^{-\lambda_X C_{ij}^2}) + \frac{\lambda_X}{\lambda_Y-\lambda_X} e^{-\lambda_Y S_{ij}^1}\left(e^{S_{ij}^1(\lambda_Y-\lambda_X)} - e^{(S_{mn}+C_m)(\lambda_Y-\lambda_X)}\right)$$

4.2 두 모듈의 작업 수행이 거리를 두고 진행

모듈 1의 작업들은 일반적인 RM방식에서와 같이 수행되며, 모듈 2의 작업들은 β 만큼의 시간지연을 두고 릴리스 된다. 두 모듈에서의 작업 τ_{ij}에 인터럽트가 한 번도 일어나지 않을 경우는 그림 10과 같다.

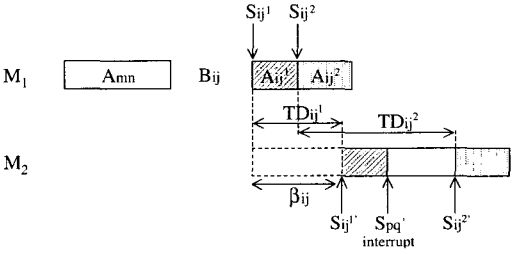


(그림 10) 인터럽트가 한 번도 발생하지 않을 경우

따라서 이 경우 동시 발생적 고장이 일어날 확률이 다음과 같이 구해진다.

$$P_c^{ij} = e^{-\lambda_X S_{ij}}(1-e^{-\lambda_X C_{ij}})e^{-\lambda_Y TD_{ij}} + \frac{\lambda_X}{\lambda_Y-\lambda_X} e^{-\lambda_Y S_{ij}'}\left(e^{S_{ij}(\lambda_Y-\lambda_X)} - e^{(S_{mn}+C_m)(\lambda_Y-\lambda_X)}\right)$$

또한 모듈 1에서 인터럽트가 발생하지 않고 모듈 2에서 인터럽트 한 번 발생하는 경우로서 그림 11과 같은 경우를 예로 들 수 있다.

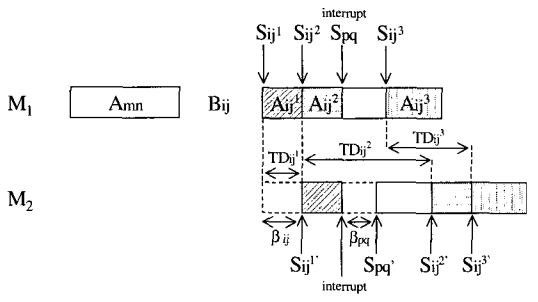


(그림 11) 모듈 2에서만 인터럽트가 한 번 발생할 경우

이 경우에, 영역 A_{ij}와 영역 B_{ij}에서 동시 발생적 고장이 일어날 확률은 다음과 같이 구해진다.

$$P_c^{ij} = e^{-\lambda_X S_{ij}^1}(1-e^{-\lambda_X C_{ij}^1})e^{-\lambda_Y TD_{ij}^1} + e^{-\lambda_X S_{ij}^2}(1-e^{-\lambda_X C_{ij}^2})e^{-\lambda_Y TD_{ij}^2} + \frac{\lambda_X}{\lambda_Y-\lambda_X} e^{-\lambda_Y S_{ij}^{1'}}\left(e^{S_{ij}^1(\lambda_Y-\lambda_X)} - e^{(S_{mn}+C_m)(\lambda_Y-\lambda_X)}\right)$$

(그림 12)는 양 모듈에서 인터럽트가 한 번씩 발생하는 경우이다.



(그림 12) 양 모듈에서 인터럽트가 한 번 발생할 경우

이 경우에, 영역 A_{ij}와 영역 B_{ij}에서 동시 발생적 고장이 일어날 확률은 다음식과 같이 유도된다.

$$P_c^{ij} = \sum_{k=1}^{3} e^{-\lambda_X S_{ij}^k}(1-e^{-\lambda_X C_{ij}^k})e^{-\lambda_Y TD_{ij}^k} + \frac{\lambda_X}{\lambda_Y-\lambda_X} e^{-\lambda_Y S_{ij}^{1'}}\left(e^{S_{ij}^1(\lambda_Y-\lambda_X)} - e^{(S_{mn}+C_m)(\lambda_Y-\lambda_X)}\right)$$

이 이외에도 인터럽트의 발생 횟수와 발생 시간 및 작업수행 시간에 따른 경우의 수가 많이 있으나, 본 논문에서는 전형적인 몇 가지 예를 들었으며, 인터럽트의 발생이 빈번하지 않다고 가정한다.

RM 방식에서 작업들의 스케쥴 가능성(schedulibility)을 검사하는 방법이 제안된 바 있다[5]. 본 논문에서 제안된 주기적 작업의 스케쥴 기법에 대해 스케쥴 가능성 검사를 위한 검사 식으로써 다음 식을 이용한다.

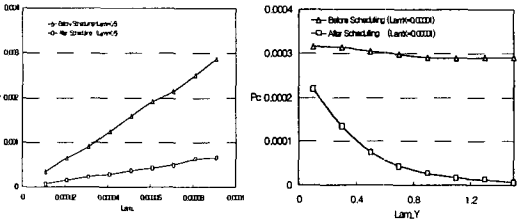
$$\forall i=1,2,3,\ldots,N,\ \min_{\{0<t\le T_i\}} \sum_{j=1}^{i} \frac{C_j+\beta_j}{t} \lceil \frac{t}{T_j} \rceil \le 1$$

위 식은 모듈 2에서의 작업에 대한 스케쥴 가능성 검사 식으로써, 모든 작업의 수행시간 C_i이 $C_i+\beta_i$으로 증가했다고 생각할 수 있다. 모듈 1에서의 작업 수행시간은 모듈 2에서의 작업 수행시간보다 항상 작으므로, 모듈 2에서의 작업들이 스케쥴링이 가능하다면, 전체 모듈에서의 모든 작업들은 스케쥴링이 가능하다고 결론지을 수 있다.

5. 수치예제

5.1 비주기적 작업

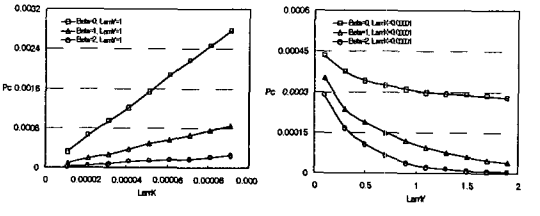
스케쥴링 이전 작업들의 마감시한과 수행시작 시간(S_i)이 각각 {3, 6, 9, 13, 18, 20, 24, 26, 29, 32}와 {3, 6, 9, 13, 18, 20, 24, 26, 29, 32}이며, 실행시간(C_i)은 {2, 3, 1, 4, 3, 3, 2, 5, 2, 2}이고, 스케쥴링 이후 모듈 1의 작업의 수행시작 시간(S_i)과 모듈2의 작업 수행시작시간(S_i')이 각각 {0, 2, 5, 6, 10, 13, 16, 18, 23, 25}와 {1, 3, 8, 9, 13, 16, 19, 21, 27, 30}인 경우를 살펴본다. 동시 발생적 고장의 발생비율(λ_X)과 지속비율(λ_Y)을 변화시켜가며 동시 발생적 고장이 발생할 확률을 구하였다. (그림 13 (a))와 (그림 13 (b))는 각각 고장의 발생비율과 지속비율을 변화시킬 때, 스케쥴링기법 적용 이전과 적용 이후의 고장 발생 확률 값을 비교한 것이다. 고장의 지속이 길어지더라도 시간차를 둔 스케쥴링 기법이 효과적으로 고장 발생 확률을 감소 시킨다.



(그림 13) (a) λ_X의 변화에 따른 고장발생 확률
(b) λ_Y의 변화에 따른 고장발생 확률

5.2 주기적 작업

작업들의 주기(T_i)가 {10, 15, 20}이고, 세 종류의 작업에 대한 α가 {3, 2, 0}이며, 작업들의 수행 길이(C_i)가 {2, 2, 2}로 주어질 경우에 고장 발생 확률을 알아본다. 이때 전체 작업의 종료시간은 52라 가정한다. (그림 14 (a))와 (그림 14 (b))는 각각 고장의 발생비율과 지속비율을 변화시킬 때, 스케쥴링기법 적용 이전과 적용 이후의 고장 발생 확률 값을 비교한 것이다. 두 그래프에서 알 수 있듯이, β를 크게 할수록 고장 발생 확률이 줄어들며, 고장의 지속이 길어지더라도 시간차를 둔 스케쥴링 기법이 효과적으로 고장 발생 확률을 감소시킴을 알 수 있다.



(그림 14) (a) λ_X의 변화에 따른 고장발생확률
(b) λ_Y의 변화에 따른 고장발생확률

6. 결 론

이중시스템은 동시 발생적 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. 이 같은 고장에 의한 피해를 줄이기 위해 두 모듈의 작업 수행에 시간차를 두는 스케쥴링 방법을 제시하였다. 또한 수치예제를 통해 본 결과 제시된 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

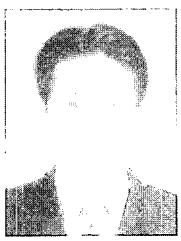
[1] Parang K. Lala, *Fault Tolerant and Fault Testable Hardware Design*, Prentice/Hall Press. 1985.

[2] Hagbae Kim and Kang G. Shin, "Task Sequencing to Minimize the Effect of Near-Coincident Faults in TMR Controller Computers," *IEEE Trans. on Computers*, Vol.5, No.11, pp.1331-1337, November, 1996.

[3] Hagbae Kim and Kang G. Shin, "Modeling of Externally-Induced/Common-Cause Faults in Fault-Tolerant Controller Computers," *Proc. IEEE Digital Avionics Systems Conf.*, pp.402-407, October, 1994.

[4] Jie Xu and Brian Randell, "Roll-Forward Error Recovery in Embedded Real-Time Systems," *Proc. IEEE Parallel and Distributed Systems Conf.*, pp.414-421, 1996.

[5] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm : Exact Characterization and Average Case Behavior," *Proc. 10th Real-Time Systems Symp.*, pp.166-171, 1989.



임 한 승

e-mail : hslim75@hanmail.net

1998년 연세대 전기공학과 졸업

1998년~현재 연세대 전기·컴퓨터공학과 석사

관심분야 : 실시간시스템, 자동화공학, 고장포용시스템, 작업스케쥴링

김 학 배

e-mail : hbkim@yonsei.ac.kr

1988년 서울대 전자공학과 졸업

1990년 미국 미시간대 대학원 전기공학과(EECS)(석사)

1994년 동 대학원 졸업(박사)

1994년~1996년 미국 National Research Council(NRC) Research Associate st NASA Langley Research Center

1996년~현재 연세대학교 전기·컴퓨터공학과 조교수

관심분야 : 실시간제어, 자동화공학, 고장포용기법 및 신뢰도 평가