실-시간 시스템의 결함 허용 태스크 스케줄링 전략에 관한 연구
A Study On The Fault-Tolerant Task Scheduling Strategy of Real-Time System

한 상 섭 *, 이 정 석 **, 박 영 수 ** 이 재 훈 ** 이 기 서 ***
Han. Sang-Seop., Lee. Jung-Suk., Park. Young-Soo., Lee. Jae-Hoon., Lee Kee-Seo.

ABSTRACT

Object of a real-time system, that performs exact information based on the real-time constraint. is required for an improvement of high reliability. The fault-tolerant task scheduling strategy of multiprocessor as using a distributed memory based on a hardware redundancy can be improved into a high reliability of the real-time system. Therefore, this paper is shown to analyze the reliability of the system by using the transfer parameter and make the modeling in reference to a minimization of the fault-tolerant task scheduling strategy which uses a percentage of task missing and deadline parameter based on optimization task size.

1. 서 론

실-시간 기반 결함 허용 시스템은 한번의 오 동작이 치명적인 결과를 초래할 수 있기 때문에 보다 높은 안전성 구현을 위해 신뢰 가능한 시스템의 구성이 필요하다. 이러한 시스템의 구성을 위해 소프트웨어 및 하드웨어의 여분 기법을 이용하게 되면서 분산형 메모리를 제어하는 멀티 프로세싱 기법을 사용하게 된다.[3] 단일 프로세서와 메모리를 사용한 시스템보다 높은 신뢰성과 안전성을 확보할 수 있지만 동기화 및 비 동기화에 대한 문제, 시스템 자원의 효율적인 활용과 주어진 시간제약에 따른 정보의 정확한 전달 문제가 발생하게 된다. 따라서, 실-시간 시스템 운영체제의 핵심인 태스크 스케줄링 기법의 선택은 인간의 머리에서 임의의 동작을 취하기 위해 순서를 결정하는 과정과 같은 것이기 때문에 매우 중요하다.[4] 실-시간 시스템의 최적화 태스크 스케줄링 알고리즘 구현은 신뢰성 향상을 위한 필수 조건이라 할 수 있다.[5] 스케줄링의 영역 중 동적 스케줄링 알고리즘은 프로세서의 실행 동안에 태스크를 할당하는데 작업부하 균형(Workload Balancing) 유지가 스케줄러의 오버헤드를 줄여주며 런-타임시 태스크 성능에 관한 최근의 정확한 정보를 가용할 수 있다. 작업 부하 균형 유지와 런-타임 스케줄링 오버헤드의 문제를 해결하기 위해서 동적 스케줄링 방법을 이용한 결함 허용 태스크 스케줄링 알고리즘을 모

* 광운대학교 석사과정
** 광운대학교 박사과정
** 철도청 신호 제어과
** 철도청 신호 제어과
*** 광운대학교 교수

델링 하고자 한다. 이 기법은 부하 균형을 유지하기가 쉽고 자체 스케줄링[4]을 통한 오버헤드의 최소화에 그 목적이 있다. 태스크 여분을 실현하기 위해서 적절한 태스크 재 할당 계획의 사용은 멀티프로세서 시스템에서 많은 위험요소들을 피할 수 있으며[3][4] 실-시간 시스템 기반 멀티 프로세싱의 태스크 재 할당에 대한 결합 허용 전략은 시스템의 높은 신뢰성이 확보 가능하다. 결함 허용 태스크 스케줄링 알고리즘에서 산출되는 영향 요인들에 대한 분석을 통해 향상된 신뢰성을 예견할 수 있다.

2. 본 론

결함 허용 태스크 스케줄링을 구현하기 위해서는 임의의 가정과 태스크 매개변수를 정의하고 해당 시스템을 고려하여 FTTS(Fault-Tolerant Task Scheduling)를 모델링 할 수 있다. 하드웨어의 여분기법인 TMR(Triple Modular Redundancy)구조의 분산형 메모리를 자원으로 하는 멀티프로세싱 시스템을 적용했다.

하드 실-시간 시스템의 동적 스케줄러는 온-라인 태스크 집합을 최적으로 스케줄링 하는 것을 결정하며 태스크들은 특정 기능의 수행을 위해 태스크를 야기(Invoke) 시킬 수 있는 소프트웨어 모듈로써 시스템 상 스케줄 되는 하나의 개체이다.[6][7] 태스크들은 처리 순서에 의해 각기 다른 결과를 발생하고 사이클을 형성하지 않도록 방향성 그래프를 사용하게 되며 이 때 시간적 제약 조건은 태스크의 도착시간과 준비시간, 실행시간, 완료시간들에 의해 스케줄러의 처리순서에 대한 선행제한과 선점 규칙을 변동시킬 수 있다. 위와 같은 일반적인 동적 스케줄러의 규칙과 함께 임의의 태스크 집합에 대하여 알고리즘이 모든 제약들을 만족하는 스케줄을 찾을 수 있고 발생 가능한 태스크에 대한 모든 정보를 가지고 정적 알고리즘을 사용하여 스케줄을 찾아 낼 때 동적 스케줄링 알고리즘을 사용하여 타당성 있는 스케줄을 찾아낸다면 최적 알고리

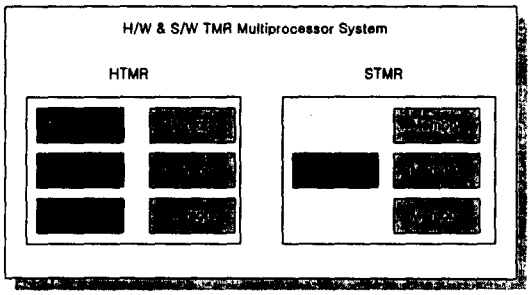

그림 1. H/W & S/W TMR System

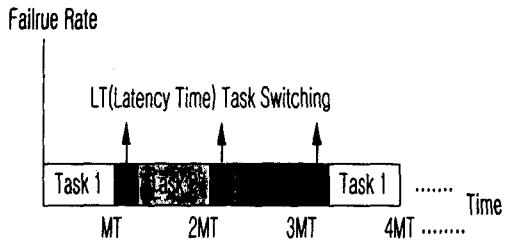

그림 2. 일반적인 TMR 태스크 진행

즘이라고 할 수 있다.[3][5][6] 또한 분산 시스템에 대한 동적 스케줄링 알고리즘은 네트워크 시스템에서 태스크의 보장률(Guarantee)을 최대화 해야하며 메모리에 대해서는 국부(Local) 스케줄링과 프로세서에 대해서는 분산 스케줄링이 만족되어야 한다. 그림 [1]과 같이 TMR 구조의 메모리는 국부적으로 구성되어 소프트웨어적인 제어를 할 수 있으며 프로세서의 경우 하드웨어적으로 분산 제어되는 형태를 취하고 있다.[2][3][6] 이 때 선점되는 메모리나 프로세서는 보터 논리의 방식에 따라 달라질 수 있게 된다. 그림[2]에서는 TMR구조의 일반적인 정적 스케줄링에 대한 태스크들의 진행을 나타냈다. 이 때 보터 고장으로 인한 메모리와 프로세서의 선점 오류, 그리고, 일정크기의 태스크들이 일정시간제약에 따라 움직이므로 메모리의 과대 낭비와 지연시간의 최대화가 시스템의 성능과 정확한 데이터가 시간제약 안에 실행되기 위해 최적 알고리즘이 구성되기 어렵고 해당 태스크에 대한 최대 보장률과 손실률 또한 보장되기 어렵다. 이것을 일반적인 동적 스케줄링을 하더라도 실행시간에 대한 제약과 자원요구 할당에 대한 선점에 있

어서 기존의 동적 스케줄링 방법으로는 많은 어려움이 야기된다.[4][5] 따라서, 결함 허용 태스크 스케줄링은 부하 균형 유지와 런-타임 오버헤드를 위해 임의의 태스크가 실행상태에서 최적의 시간제약을 이용하여 태스크가 결과 값을 도출하고 대기상태로 또는 준비 상태로 이동 할 수 있어야 하며 실행 태스크들이 최대한의 보장률을 가지고 최소한의 손실률을 나타내도록 해야 한다. 그리고 TMR 구조에서 국부 스케줄링에 의한 메모리 접근이 분산형 프로세서를 이용하여 동기 또는 비동기 적으로 구성되어야 한다. 따라서, 다음과 같은 가정과 정의에 의해 결함 허용 태스크 스케줄링 전략을 모델링 할 수 있다.

2. 1. 가정과 정의

해당 시스템의 멀티 프로세서 시스템은 그물망 위상 구조에 P(=3) 개의 프로세서로 구성되어 분산형 메모리 구조이고 전체 시스템은 하나의 스케줄러를 가지고 있다. 각 각의 작업노드들은 그들 자신의 국부 메모리를 할당하며 자료 구조 형태는 큐(QUEUE)의 형태를 가지고 있다.

2.1.1. 가정(ASSUMPTION)

A. 작업노드들의 국부 메모리 용량은 모든 태스크들을 할당하기에 충분해야 한다.
B. 태스크들간 선행제약이 없으며 독립적이다.
C. 시스템상의 임의의 결함은 하드웨어 메카니즘에 의해 고립되거나 검출할 수 있다.(Voter)
D. 스케줄러나 통신 네트웍 상에는 결함이 없다.

2.1.2. 태스크 파라메터와 결함 허용 태스크 스케줄링

A. 태스크 파라메터

태스크 양도 시간 : T_R , 태스크 실행시간 : T_E , 태스크 소멸시간 : T_a
태스크 전달 시간 : T_t , 스케줄러의 태스크 큐 상에서 기다리는 시간 : T_{wTQ}
국부 메모리 큐 상에서 기다리는 시간 : T_{wLMQ}
정상적인 시스템 실행에 의해 태스크가 작업을 마치는 시간 : T_{ND}

$$T_D \geq T_{ND} = T_t + T_{wLMQ} + T_{wTQ} + T_E \quad \text{- 식 (1)}$$

식(1)과 같이 정상적인 스케줄러에 의해 태스크 작업을 마치는 시간 각 파라메터들을 합한 것과 유사하며 각 파라메터들의 동적 수행 능력은 스케줄러의 동작에 영향을 받게 된다. 따라서, 다음과 같은 결함허용 태스크 스케줄링을 모델링 할 수 있다.

B. 결함 모델 및 결함 허용 태스크 스케줄링

시스템 결함 발생이 의미하는 것은 각 작업 노드들이 태스크를 제어할 수 없고 더 이상 다른 태스크들을 수용할 수 없게 된다. 즉 스케줄러에 의한 메모리의 재 할당이 어렵게 됨을 의미하므로 다음과 같은 결함 허용 태스크 스케줄링 규칙을 모델링 할 수 있다.

. 스케줄러는 작업 노드 상에 동작하는 모든 태스크들의 매개변수에 대해 기록을 유지하고 검사하며 끝나지 않은 태스크들은 순환하여 재 할당시키고 태스크가 성공적으로 끝났을 때 매개변수는 제거(Clear) 시킨다
. 스케줄러의 두 번째 큐는 자원 재 할당을 지원하기 위해서 항상 준비상태에 있어야 한다.
. 작업 노드는 스케줄러에게 각 시간에 따라 시스템이 유휴상태이거나 고장(Break Down) 상태일 때를 알려준다.
. 스케줄러는 작업 노드들의 우선 순위에서 프로세서 집합을 나타내며 검사한다.
. 스케줄러가 임의의 노드의 고장을 알고 고장으로 인해 재 할당을 필요로 하면 두번째 큐로

부터 태스크를 재 할당한다. 따라서, 두 번째 큐는 첫 번째 보다 우선 순위가 높으며 새로운 태스크 보다 준비 상태에 있는 태스크가 우선 순위가 높다.

$$T_D \geq T_{AND} = T_{ND} + T_{wTQ} \quad \text{- 식(2)}$$

위의 결함 허용 태스크 스케줄링 전략은 식(2) 번과 같이 태스크 소멸 즉 마감시간이 정상 마감시간과 스케줄러의 태스크 큐 상에서 기다리는 시간의 합으로 유추할 수 있다. 따라서, 일반 태스크 스케줄링과 비교할 때 동적 스케줄링 상태에서 많은 태스크 매개변수에 대한 부담이 감소하게되고 부정확한 스케줄링 파라메터의 동작이 없으므로 전달 데이터에 대한 신뢰성 또한 높을 것이다. 하지만, 대기 큐 상의 태스크에 대한 유휴상태와 재 할당으로 인한 정확한 시간적

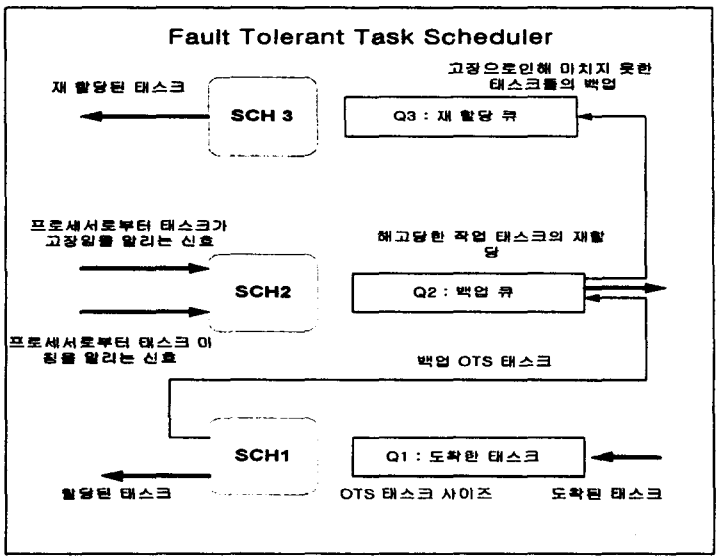

그림 3. 결함 허용 태스크 스케줄링 모델

제약 문제가 야기 될 수 있다. 그림 3과 같은 결함 허용 태스크 스케줄링은 최적화 태스크 사이즈의 구현이 선행되어야 한다. 최적 태스크 사이즈는 다음과 같다.

$$OTS = \frac{Telegram\text{실행수}}{\text{주어진임무시간}} \times \frac{Telegram\text{의메모리할당량}}{Processor\text{의사이클당동기수}} \quad \text{-식(3)}$$

식(3)과 같이 나타낼 수 있다. 최적 태스크 사이즈를 바탕으로 그림(3)과 같은 스케줄러는 할당된 태스크가 작업 큐(Queue)상에 들어오면 태스크를 할당하고 최적 태스크 사이즈가 아니면 백업 태스크를 수행하기 위해 백업 큐로 이동하게 되며 프로세서로부터 태스크 신호를 마치거나 재 할당 큐로 이동하게 된다. 재 할당되는 태스크는 백업 태스크의 실패나 작업 큐 상의 고장 또는 Telegram 데이터 파손 등으로 일어날 수 있다. 이러한 단계에서 태스크 분실율을 알아보면 다음과 같다. $\text{태스크분실률} = \frac{\text{데드라인에서분실되는태스크의수}}{\text{전체시스템에서종결되는태스크의수}} \times 100$ -식(4)이다. 또한 태스크의 최대 보장률은 다음과 같이 구할 수 있다. $\text{최대보장률} = \frac{\text{보장된전체태스크수}}{\text{도착된전체태스크수}} \times 100$ - 식(5)이다

최적 태스크 사이즈와 최대 보장률 태스크 분실률등은 결함 허용 태스크 스케줄링을 구현하는 신뢰성 지표가 된다.

2. 3. 시스템 성능 분석

본 논문의 결함 허용 태스크 스케줄링 전략은 멀티 프로세서 시스템의 신뢰성을 향상시키기 위한 전략이며 태스크의 재할 당 동적 결함 허용 규칙을 통해 보다 높은 신뢰성을 확보 할 수 있을 것이다. 따라서, 다음과 같은 수식적 전개를 통해 신뢰성 성능 분석을 할 수 있다. 스케줄링 특성에 따라 연속시간 신뢰성 함수보다는 식(6)과 같은 이산 시간에서의 신뢰성 함수를 고려해야 한다. 주기 T의 한 사이클 동안 에러가 없을 경우 $P=R(t) \mid_{t=T}= e^{-\lambda t}$ - 식(6)이 된다. 만일 N 사이클 동안 에러가 없다면 식(7)과 같은 수식을 도출해 낼 수 있다.

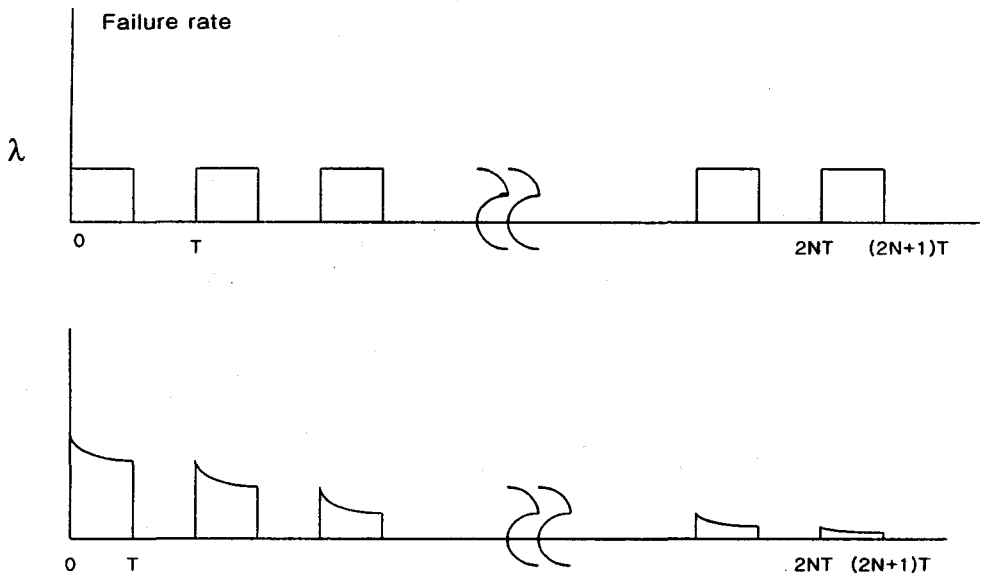

그림 4. 이산 시간 태스크 스케줄링의 시간 주기에 따른 고장률과 신뢰성 함수

$R[M]=P=\prod_{i=1}^{N}P=P^{N}$ - 식(7) 이때의 시간에 대한 결함률과 신뢰성은 그림 4와 같다. 이와 함께 고려해야할 사항은 태스크 스위칭 오버헤드이다. 그림2를 참조하면 태스크 1부터 3이 끝나고 그 다음 사이클의 태스크 1의 임무시간(Mission Time)을 결함 허용 태스크 스케줄링으로부터 부여받을 때까지의 잠복시간(Latency Time)이 태스크 스위칭 시간이 된다. 이때 스위칭 오버헤드 확률은 태스크의 성질만 다를 뿐 그 수식은 $P^{Latency}$로써 동일하게 된다. 이때 전체 스위칭 오버헤드 시간은 식(8)과 같이 구할 수 있다.

스위칭 오버헤드 전체시간 $=3NT+3\frac{NTL}{M}$ - 식(8)

N : 정상동작시간, T : 주기, M : 임무시간, L : 잠복 시간을 나타낸다. 따라서, 결함 허용 태스크 스케줄링 전략 기법의 전체 신뢰성 함수를 구하면 식(9)와 같은 수식을 얻을 수 있다.

R_{DFTTS}(동적 결함허용 태스크 스케줄링의 신뢰성)

$$= (3R^{2}_{OneTask} - 2R^{3}_{OneTask}) \prod_{k=1,2,\ldots,\frac{3N}{M}}^{P(\text{사이클동안에러가없을확률}[(M+L)k,(M+L)k+L]}$$

$$= (3R^{2}_{OneTask} - 2R^{3}_{OneTask}) \prod_{k=1,2,\ldots,\frac{3N}{M}}^{P^{L}} \quad \text{- 식 (9)}$$

식 (9)가 의미하는 것은 일반적인 태스크 스케줄링에 의한 신뢰성과 동적 결함 허용 태스크 스케줄링으로 인한 오버헤드의 신뢰성 함수의 곱으로 전체 스케줄링의 신뢰성 수식을 도출해낸 것이다.

3. 결 론

실-시간 기반 결함 허용 시스템은 요구되는 신뢰성,안전성,정확성을 만족시키기 위해 오늘날 많은 연구가 이루어지고 있다. 본 논문은 이러한 여분기법의 구성으로 메모리와 프로세서의 다중화 현상에 따른 시스템 운영체제에 따른 작업 스케줄링의 중요성을 인식하고 하드웨어/소프트웨어뿐만 아니라 작업 스케줄링에 결함 허용 기법을 적용하여 결함 허용 태스크 스케줄링 기법을 모델링하고 신뢰성 함수를 도출해 보았다. 본 연구의 중요한 의미는 시스템의 근간을 이루는 하드웨어와 소프트웨어 여분기법 적용에 따른 시스템 자원 다중화 현상에 따라 스케줄링의 과부하, 작업 분실 및 태스크 마감시간의 제어(OTS 크기)를 통해 신뢰 가능하고 안전성 있는 실-시간 기반 결함 허용 시스템 작업 스케줄러를 모델링 한데 있고 나아가서 해당 기능의 신뢰도를 스케줄러 특성에 따라 이산시간 연산을 통해 나타낼 수 있었다. 앞으로 실-시간 운영체제(Real-Time Operating System : RTOS)상에서 적용 시스템의 기능 특성에 맞는 작업 스케줄러의 개발은 시스템 개발의 핵심이라고 생각하며 인간의 생명과 한번의 오 동작이 치명적인 결과를 나타내는 발전소, 철도, 항공, 국방 분야에서는 보다 정형화되고 발전된 태스크 스케줄링 전략이 수립되어야 한다. 끝으로, 결함 허용 태스크 스케줄링 전략에 대한 검증 연구는 보다 힘든 연구가 될 것이며 시간 제약에 따른 매개변수의 제어가 핵심이 될 것으로 예상된다.

참고문헌

1. "Real-Time System Design and Analysis" IEEE Computer Society Press, 1993. Phillip Laplante.
2. "Safety-Critical Computer Systems" written by Neil Storey, published by ADDISON-WESLEY 1996.
3. "Real-Time Programming" Rick Grehan, Robert Moote, Ingo Cyliax ADDISON-WESLEY 1998.
4. "Fault-Tolerance in Real-Time Communication" IEEE 1993. Stephan Schultze. 0-7803-1227-9
5. "A Fault Tolerant Algorithm for Distributed Mutual Exclusion" IEEE. 1990. Ye-in Chang, Mukesh Singhal, Ming T Liu.
6. "An Empirical Comparison of Software Fault Tolerance and Fault Elimination. IEEE. 1991. Timothy J. Shimeall and Nancy G. Leveson. 0098-5589/91/0200-0173.
7. "Requirements Analysis of Real-Time Control System Using PVS" from NASA LARC. written by Bruno Duterte and Victoria Stavridou.