경성 실시간 멀티프로세서 시스템을 위한 이중토큰 기반의 고장허용 스케줄링

(Dual Token-based Fault-Tolerant Scheduling for Hard Real-Time Multiprocessor Systems)

우 철 희 † 이 승 룡 †† 이 종 원 ††† 오 삼 권 ††††

(Chul-Hee Woo) (Sungyoung Lee) (Jong Won Lee) (Sam Kweon Oh)

요 약 지금까지 연구된 실시간 멀티프로세서 시스템 환경에서 경성 비주기적인 태스크들에 대한 고장허용 스케줄링 알고리즘들은 태스크를 할당하는 중앙 분배(central dispatcher) 프로세서는 실패하지 않는다고 가정하였다. 그러나 이같은 가정은 모든 프로세서가 고장날 수 있다는 관점에서 볼때 현실성이 결여되어있을 뿐만 아니라, 중앙 분배 프로세서에서 고장이 발생했을 때 시스템 전체가 실패한다는 문제점을 가지고 있다. 이러한 문제를 해결을 위해 본 논문에서는 중앙 분배 프로세서가 없는 실시간 멀티프로세서 시스템 하에서 이중토큰(dual token)에 기반한 고장허용 스케줄링 알고리즘을 제안한다. 제안된 알고리즘은 중앙 분배 프로세서의 역할을 시스템내의 주토큰(primary token)을 가진 임의의 프로세서가 수행하고 그 프로세서가 고장 시 복구토큰(backup token)을 가진 프로세서가 즉시 주토큰을 가진 프로세서의 역할을 대체 수행하므로써 시스템 전체의 실패를 방지할 수 있으며, 중앙 분배 프로세서를 태스크 처리 업무에 활용함으로써 대략 1/(프로세서의 수)만큼의 이용률을 향상시킬 수 있다. 또한 제안된 알고리즘은 복구 프로세서에서 마감시간에 기반한 태스크 스케줄링 정책을 사용함으로써, 기존의 경험적(heuristic) 스케줄링 방법과는 달리 알고리즘 구현 시 오버헤드가 적으며 스케줄링 예측성을 향상시킬 수 있다. 시뮬레이션 연구 결과 제안된 알고리즘은 고장발생 시에도 태스크들의 마감시간을 보장할 뿐만 아니라 모든 프로세서가 태스크를 처리하는데 참여하였기 때문에 기존의 중앙 분배 프로세서가 존재하는 경우보다 더 많은 태스크를 수용함을 보여주었다.

Abstract Real-time multiprocessor systems frequently assume that there exists a dedicated central dispatch processor for task allocation that never fails. This assumption is, however, too strong in the sense that all the physical objects are subject to failure. Moreover, once the dedicated processor fails, the whole multiprocessor system will fail. As a way to solve this problem, we propose a fault-tolerant scheduling algorithm based on moving dual-token. While the primary processor holding a primary token performs task allocation, the backup processor holding a backup token, in case that the primary processor has failed, does primary processor creation. Since no dedicated processor for task allocation exists in this scheme, failure of the whole multiprocessor system due to that of the dedicated processor can be avoided. In addition, the system utilization can be improved by a factor of roughly 1/(number of processors) since all the processors can be used for task execution. Lastly the deadline-friendly scheduling policy used for backup task allocation, compared to heuristic scheduling, allows easier implementation and improved scheduling predictability. Simulation results show that the proposed dual-token based algorithm yields low rejection rates over those with dedicated processor for task allocation.

· 이 논문은 한국학술진흥재단 자유공모(과제 번호 01E0541)의 지원과, 정보통신부 '97 대학기초연구지원사업(과제번호: AB-97-G-0655)의 일부지원에 의한 것임

† 비 회 원 · LG정보통신
airwoo@chollian.net
†† 종신회원 : 경희대학교 전산공학과 교수
sylee@oslab.kyunghee.ac.kr
††† 비 회 원 : 경희대학교 교육대학원 교수
jwlee@coral.kotel.co.kr
†††† 종신회원 · 호서대학교 컴퓨터학부 교수
ohsk@ologsuri.hoseo.ac.kr

논문접수 : 1997년 11월 19일
심사완료 : 1998년 7월 30일

1. 서 론

실시간 시스템이 가지는 시간 제약과 신뢰성의 요구사항 때문에 멀티프로세서 시스템은 병렬 어플리케이션을 수행하는 실시간 시스템에 널리 사용되어왔다. 멀티프로세서 시스템에서는 프로세서의 수가 증가할수록 고장 발생 가능성도 증가하기 때문에, 고장허용은 이러한 시스템과 관련이 깊다. 특히 엄격한 시간제약을 만족해야하는 경성 실시간 시스템에서 태스크의 마감시간을 지키지 못하는 경우, 치명적인 결과를 초래할 수 있기 때문에 프로세서의 고장이 존재함에도 불구하고 마감시간 전에 태스크의 실행이 완료되도록 보장하는 것은 필수적인 요구사항이다.

지금까지 연구된 실시간 멀티프로세서 환경 하에서 경성 비주기적 태스크에 대한 고장허용 문제는 태스크들을 실행하는 프로세서에서 발생하는 고장을 다루고 있다. 이 경우 고장이 발생한 프로세서에서 처리되던 태스크들의 사본 태스크들은 다른 프로세서에서 고장복구를 위해 재실행된다. 그런데, 태스크들의 전체 스케줄을 유지하면서 고장발생 시 이를 감지하여 복구카피를 갖고 있는 프로세서에게 고장복구를 수행하도록 통보하는 중앙 분배 프로세서는 고장이 발생하지 않는다고 가정하고 있다[2],[3]. 그러나 이는 태스크를 실행하는 시스템 내의 어떤 프로세서에서도 고장이 발생할 수 있다는 현실적인 상황을 간과한 것이고, 더우기 중앙 분배 프로세서에서 고장이 발생했을 때에는 전체 시스템 실패라는 치명적인 문제를 가지고 있다. 또한, 중앙 분배 프로세서는 새로 도착하는 태스크를 어떤 프로세서에다 할당할 것인지를 결정하는 스케줄링만을 수행하고 태스크를 처리하는데는 사용되지 못하기 때문에 대략 1/(프로세서 수) 만큼의 시스템 전체 이용률이 떨어진다. 그리고, 태스크들의 복구카피를 할당함에 있어 [3]과 이를 개선한 [1]에서는 경험적 알고리즘을 사용하고 있는데, 이는 스케줄링 구현비용이 비싸며, 실시간 시스템이 가져야 할 예측성 확보에 장애요인이 될 수 있다.

따라서, 본 논문에서는 이러한 문제들을 해결하기 위하여 실시간 멀티프로세서 시스템을 위한 이중토큰 기반 경성 비주기적 태스크 고장허용 스케줄링 알고리즘을 제안한다. 제안된 알고리즘은 중앙 분배 프로세서의 역할을 주토큰을 가진 프로세서가 수행하고 이 프로세서에 고장이 발생 시 복구토큰을 가진 프로세서가 즉시 대체 수행함으로써 중앙 분배 프로세서의 역할을 시스템내의 전체 프로세서로 분산시켰다. 이를 위하여 프로세서의 여유시간(free-slot) 상태에 따라 구성된 프로세서 큐에서 중앙 분배 프로세서의 역할을 해야하는 프로세서를 선택하여 주토큰을 할당한다. 여기서 말하는 여유시간이란 각각의 프로세서에서 사용되는 비선점(non-preemptive) EDF(Earliest Deadline First) 스케줄링에 의해 태스크에 할당될 수 있는 가용한 시간이다. 이같은 스케줄링 정책은 임의의 주토큰을 가진 프로세서에 고장이 발생하더라도 그 영향을 자신에만 국한시켜(복구토큰을 가진 프로세서가 이를 즉시 대체하여 수행하기 때문에) 시스템 전체가 실패하는 사태를 방지할 수 있어 신뢰도를 증진시킬 수 있을 뿐만 아니라 중앙 분배 프로세서가 태스크 처리를 수행함으로써 이용률을 향상시킬 수 있다. 그리고, 태스크의 복구카피를 할당할 때 마감시간을 기반으로 해서 스케줄 함으로써 구현비용을 감소시키고 시스템의 예측성을 향상시킬 수 있다.

본 논문의 구성은 다음과 같다. 2 장에서는 본 논문과 관련된 연구에 대하여 소개하고, 3 장에서는 본 논문에서 제안하는 시스템 모델 및 가정에 대하여 기술한다. 4 장에서는 제안된 알고리즘을 설명하고, 5 장에서 시뮬레이션 결과를 설명하며, 6 장에서 결론을 맺는다.

2. 관련연구

실시간 고장허용 스케줄링 알고리즘에 관한 연구분야는 [그림 1]에서 보는 바와 같이 크게 단일프로세서 환경과 멀티프로세서 환경으로 나누어지며, 이는 다시 주기적 태스크들과 비주기적 태스크들로 구분되어진다.

연구자	시스템 환경	태스크 성격	알고리즘의 특징
Liestman and Campbell [4]	단일 프로세서	주기적 태스크	실행되는 태스크의 수를 최대로 하려고 시도한다. 태스크들의 주기가 서로 배수여야 함.
Krishna and Shin [5]	멀티 프로세서		복구 스케줄을 생성하기 위해 최적의 스케줄이 존재한다고 가정하는데 이는 너무 낙관적임.
Oh and Son[6],[7]			하나의 실패를 허용하는 스케줄을 제공하기 위해 요구되는 프로세서의 수는 non-fault tolerant 스케줄의 두 배.
Ghosh, Melhem and Moss [2],[3]		비주기적 태스크	복구 overloading과 deallocation 방법 사용. 충분한 laxity를 갖는 태스크에만 적용가능.
Tsuchiya, Kakuda and Kikuno [1]			Ghosh의 알고리즘을 개선하여 충분한 laxity가 없는 경우에도 고장허용을 보장했지만 여전히 경험적임.

그림 1 실시간 고장허용 스케줄링 알고리즘 분류 및 특성

Linesman과 Campbell은 [4]에서 일시적인 고장을 다루기 위한 단일프로세서 환경 하에서 실시간 고장허

용 스케줄링 알고리즘을 제안하였다. 이때 태스크들은 주기적이라고 가정하고, 각 태스크들에 대한 고장을 허용하면서 시스템의 성능을 보장하기 위해 모든 태스크들의 복구카피들을 단일 프로세서에서 스케줄 한다. 그런 다음, 실행될 수 있는 태스크의 수를 최대로 하려고 시도한다. 이 알고리즘이 가지는 제한점은 태스크의 주기가 서로 배수가 되어야 한다는 것이다.

Krishna와 Shin[5]은 멀티프로세서 시스템 하에서 주기적 태스크들에 대한 고장허용 스케줄링 전략을 제안하였다. 이들은 주어진 최대수의 프로세서 실패에도 불구하고 경성 마감시간을 보장하기 위해 원래의 주(primary) 스케줄 내에 복구(contingency) 스케줄을 효과적으로 수행하는 동적 프로그래밍 알고리즘을 제안했다. 이 알고리즘은 복구 스케줄을 생성하기 위해 최적의 스케줄이 존재하며, 대기 태스크들의 역할을 하는 ghost의 추가로 스케줄을 할 수 있다고 가정한다. 그러나, 모든 스케줄들에 그런 ghost의 추가를 허용할 수 없기 때문에 이 같은 가정은 너무 낙관적이다.

Oh와 Son은 [6]과 [7]에서 멀티프로세서 시스템 하에서 주기적 태스크들에 대한 고장허용 스케줄링 알고리즘을 제안하였다. 이 알고리즘에서는 주 스케줄에 각 태스크에 대한 복구 스케줄이 생성된다. 그 다음에 태스크들은 주 스케줄과 복구 스케줄들이 다른 프로세서에 존재하고 겹치지 않도록 교체된다. 따라서, 최악의 경우에 하나의 프로세서 실패까지 허용할 수 있으며, 그때 요구되는 프로세서들의 수는 non-fault-tolerant 스케줄의 두 배이다.

Ghosh, Melhem, 그리고 Moss는 비주기적인 태스크 고장허용 스케줄링 알고리즘[3]을 제안하였고, 이는 [2]에서 더 개선되었다. 이 알고리즘들은 어떤 순간에 하나의 프로세서가 실패했을 경우 이 프로세서의 복구 전에 다른 프로세서가 실패하지 않는다면 시스템에 스케줄된 태스크들의 수행 완료를 보장한다. 또한 더 많은 태스크를 수용하기 위해 복구 *overloading*과 primary가 성공적으로 완료했을 때 복구 태스크를 위해 예약된 자원을 회수(deallocation)하는 메카니즘을 사용한다. 그러나 중앙 분배 프로세서가 고장나는 경우 시스템 전체가 실패한다는 점과, 프로세서의 고장 시 복구 태스크가 실행되어 마감시간을 보장하기 위해서 태스크는 재 수행될 수 있을 만큼의 충분한 laxity를 가져야 한다는 제약이 있다.

Tsuchiya, Kakuda, Kikuno[1]는 [2]와 [3]에서 제안된 알고리즘을 통합하여 태스크들이 실행을 반복할 만큼의 충분한 laxity가 없는 경우에도 고장허용을 보장하는 알고리즘을 제안하였다. 그러나 이 알고리즘도 여전히 경험적 스케줄링이 가지는 예측성 결여와 구현상의 오버헤드 문제점을 가지고있다.

3. 시스템 모델

제안된 시스템에서는 n개의 프로세서 중에서 중앙 분배 프로세서의 역할을 대신하는 임의의 한 프로세서가 주토큰을 가지고있다. 주토큰을 가진 프로세서에서 고장이 발생할 경우 이를 복구하기 위해서 나머지 $n-1$개의 프로세서 중에서 임의의 한 프로세서가 복구토큰을 가지고있다. 이러한 이중토큰 개념은 고장허용 시스템의 하드웨어 및 소프트웨어의 이중구조와도 자연스럽게 일관성을 (주토큰을 가진 프로세서에서 고장 발생 시 복구토큰을 가진 프로세서가 이를 대신 수행함) 유지할 수 있다. 다음은 이중토큰의 정의이다.

- **정의 1** : 주토큰 (Tok_P : primary token)은 시스템에 새로 도착하는 태스크들에 대한 스케줄링을 수행하는 프로세서가 가지며, 주토큰을 가진 프로세서는 새로운 태스크가 도착하였을 때 시스템의 공유메모리에 있는 프로세서 큐를 검사하여 여유시간이 가장 많은 프로세서에게 이를 할당한다.
- **정의 2** : 복구토큰 (Tok_B : backup token)은 주토큰을 갖고 있는 프로세서에서 고장이 발생했을 때 이를 복구하기 위한 프로세서가 갖는다. 주토큰을 갖고 있는 프로세서에서 고장이 발생하면, 복구토큰을 갖고 있는 프로세서는 프로세서 큐를 검사하여 여유시간이 가장 많은 프로세서에게 주토큰을 할당한다.

본 논문에서 사용하는 용어 및 표기는 다음과 같다.

P : 프로세서 (P_{Tok_P}, P_{Tok_B}, P_{PC_T}, P_{BC_T})

T : 태스크

T_a : 태스크의 도착시간

T_d : 태스크의 마감시간

T_c : 태스크의 실행시간

Q_T : 태스크 큐(Task Queue)

Q_P : 프로세서 큐 (idle($IdleQ_P$), ready($ReadyQ_P$), active($ActiveQ_P$))

PC_T : 태스크의 주카피 (primary copy)

BC_T : 태스크의 복구카피 (backup copy)

$BC_T(RP)$: 복구카피의 중복 부분 (주카피와 같이 수행되는 부분)

$BC_T(BP)$: 복구카피의 복구 부분 (복구카피에서 중복 부분을 제외한 나머지)

$T_C(BP)$: 복구 부분의 실행시간

FS_{PF} : 주 프로세서의 여유시간

FS_{BF} : 복구 프로세서의 여유시간

제안된 이중토큰기반 고장허용 시스템의 구조는 [그림 2]와 같다. 여기서 태스크 큐는 시스템 외부에서 발생한 태스크가 도착하는 곳이고 지역 큐는 태스크 큐에 도착한 태스크들 중 해당 프로세서가 처리할 수 있는 태스크들의 주카피나 복구카피를 임시 저장하는 곳이다. 프로세서 큐는 프로세서의 상태에 따라 active, ready, idle로 구분되며 그 역할은 다음과 같다.

- active 프로세서 큐의 프로세서들은 태스크의 최단 종료시간(shortest completion time)이 빠른 순으로 정렬되는데, 최단 종료시간이란 태스크의 도착시간과 실행시간의 합이다. 즉, 현재 실행 중인 태스크들 중에서 도착시간과 실행시간의 합이 가장 작은 태스크를 갖는 프로세서가 active큐의 헤드에 존재한다.
- ready 프로세서 큐에는 현재 태스크가 실행되고 있지 않으나 지역 태스크 큐에서 곧 실행될 태스크가 있는 프로세서들이 존재한다. 이 큐에서 프로세서들은 여유시간이 큰 순서로 정렬되고, 여유시간은 지역 큐의 헤드에 있는 태스크 도착시간이 가장 늦은 프로세서가 가장 많다.
- idle 프로세서 큐에는 지역 태스크 큐가 비어있는 프로세서들이 존재한다.

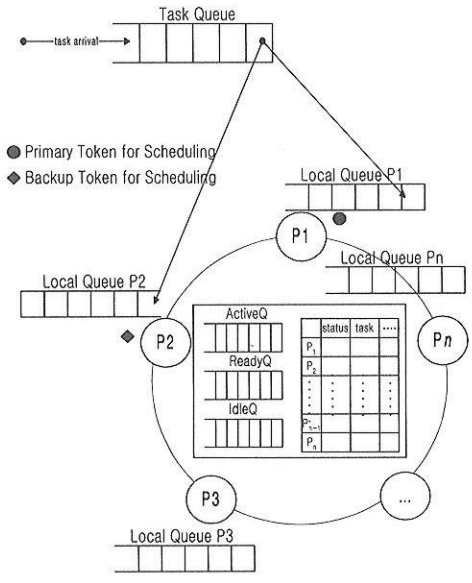


그림 2 이중토큰 기반 멀티프로세서 고장허용 시스템 모델

본 논문에서 사용되는 가정은 다음과 같다.

A1) 시스템은 상호 연결된 n개의 동일한 프로세서로 구성되어 있으며 공유메모리를 갖는 경성 실시간 시스템이다.

A2) 모든 태스크들은 비주기적이고 도착했을 때 비선점으로 스케줄 된다.

A3) 모든 태스크들은 서로 독립적이다. 그러나, 선행제약을 가진 태스크들은 선행 그래프를 준비 시간과 마감시간을 갖는 독립적인 노드로 변형함으로써 스케줄 될 수 있다.

A4) 주카피와 복구카피의 실행시간은 같다.

A5) 자원의 제약은 없다.

A6) 고장은 프로세서 상에만 일어나고 두 개의 연속적인 고장사이의 시간간격은 하나의 고장을 복구하기 위해 복구카피의 실행이 완료되는데 필요한 시간보다 크다. 즉 고장이 발생한 프로세서에서 수행되고 있던 태스크의 윈도우 내에서는 두 개 이상의 고장이 동시에 존재하지 않는다.

A7) 프로세서의 고장을 즉시 감지하는 메카니즘이 있다.

A8) 소프트웨어 고장이나 서로 관련된 구성요소의 실패는 고려하지 않는다.

A9) 토큰은 전달되는 동안에는 잃어버리지 않는다.

A10) 모든 프로세서는 태스크를 실행할 뿐 만 아니라 프로세서에 토큰을 할당하는 스케줄링도 가능하다. 또한, 태스크를 처리한 후 토큰을 전달하기 직전 에는 고장이 없다.

4. 이중토큰 기반 Fault-Tolerant 스케줄링

4.1 기본개념

제안된 알고리즘은 두 개의 종속 알고리즘으로 구성된다. 하나는 프로세서에 토큰을 할당하는 스케줄링으로, 주토큰을 할당하는 경우와 복구토큰을 할당하는 경우가 있다. 다른 하나는 도착한 태스크의 복구카피를 어떤 식으로 스케줄 할 것인가 이다.

첫 번째 토큰 할당 스케줄링에서 주토큰을 할당하는 경우는 프로세서의 상태에 따라 구성된 idle, ready, active 프로세서 큐의 순으로 검사하여 idle하고 여유시간이 가장 많은 프로세서에게 주토큰을 할당한다. Idle한 프로세서가 없으면, 최단 종료시간이 가장 빠른 프로세서에게 토큰을 할당한다. 이때, 주토큰을 가진 프로세서는 중앙 분배 프로세서의 역할을 수행하기 때문에 도착한 태스크에 대한 수용여부를 결정할 수 있다. 복구토큰을 할당하는 방법은 주토큰을 할당하는 방법과 같다.

복구토큰을 할당받은 프로세서는 주토큰을 가진 프로세서에 고장이 발생하면 이를 즉시 복구하며, 동시에 토큰 할당 정책에 따라 다른 프로세서에게 주토큰을 할당한다. 이렇게 함으로써 주토큰을 가진 프로세서에 고장이 발생하여도 도착한 태스크의 스케줄링을 계속 수행하여 시스템 전체의 실패를 방지할 수 있다.

두 번째 알고리즘은 마감시간 기반 복구카피 스케줄링이다. 이는 주토큰을 가진 프로세서가 도착 태스크의 주카피를 수용할 수 있을 때, 복구토큰을 가진 프로세서에서 복구카피를 마감시간에 가능한 근접하게 할당하는 스케줄링을 말한다. 이렇게 함으로써 주토큰을 가진 프로세서에 고장이 발생하여도 복구카피는 다른 프로세서에서 재실행되어 마감시간 내에 성공적으로 수행을 완료 할 수 있고, 주카피의 실행이 성공했을 때 복구카피에 할당된 자원을 회수하여 더 많은 태스크를 수용할 수 있다. 이러한 마감시간 기반 복구카피 할당 방법은 기존의 경험적인 복구카피 할당 방법[1]에 비하여 구현이 단순하다는 장점을 가진다.

한편, 도착하는 새로운 태스크를 어떤 프로세서에게 분배할 것인지를 결정하는 스케줄링은 [1],[2]에서 사용하는 방법을 그대로 적용하였다.

본 논문에서 제안한 알고리즘의 논리적 흐름도는 [그림 3]과 같다. 초기에는 모든 프로세서가 idle 상태이므로 임의의 두 프로세서에 주토큰과 복구토큰을 각각 할당한다. 그런 다음 프로세서를 상태에 따라서 4 가지 부류로 나눈다. 첫째는 주토큰을 갖고 중앙 분배 프로세서의 역할을 수행하는 프로세서(P_{Tok_t})이다. 이 프로세서는 태스크 큐를 검사하여 스케줄링 해야 할 태스크가 있는지를 조사한다. 태스크가 있다면 자신의 여유시간과 비교하여 태스크의 실행시간보다 큰 여유시간이 있는 경우($FS_{PP} \geq T_c$)에만 (이 때 복구토큰을 가진 프로세서도 동일한 조건을 비교하여 복구카피의수용 여부를 조사한다) 태스크의 주카피를 수용하고 그렇지 않으면 거부한다. 태스크를 수용한 경우에는 자신의 여유시간을 태스크의 실행 시간만큼 감소($FS_{PP} = FS_{PP} - T_c$)시킨 다음, 토큰할당 알고리즘을 수행하여 여유시간이 가장 많은 프로세서에게 주토큰을 전달한다. 만약 태스크 큐가 비어 있는 경우에는 태스크 도착을 기다리다 프로세서 큐의 상태가 변하였을 때는 (예를 들면, 어떤 프로세서가 태스크 수행을 완료하여 idle 상태가 된 경우) 토큰할당 알고리즘을 수행한다.

두 번째 부류는 복구토큰을 갖고 있는 프로세서(P_{Tok_b})이다. 이 프로세서는 주토큰을 갖고 있는 프로세서에서 고장이 발생한 경우에 주토큰을 복구하고 태스크의 복구카피를 비선점 마감시간 기반 알고리즘을 사용해 스케줄 한다. 주토큰을 가진 프로세서가 태스크의 주카피를 수용하면 복구토큰을 가진 프로세서가 복구카피를 수용할 수 있는 여유시간이 있는지($FS_{BP} \geq T_c$)를 검사한다. 수용할 수 있으면 자신의 여유시간을 태스크의 실행시간만큼 감소($FS_{BP} = FS_{BP} - T_c$)시키고 그렇지 않으면 태스크를 거부한다. 그리고 마감시간 기반 태스크 할당 알고리즘을 통해 복구카피를 가능한 마감시간에 근접하게 할당한 다음 토큰할당 알고리즘을 구동하여 복구토큰을 선택된 프로세서에게 전달한다.

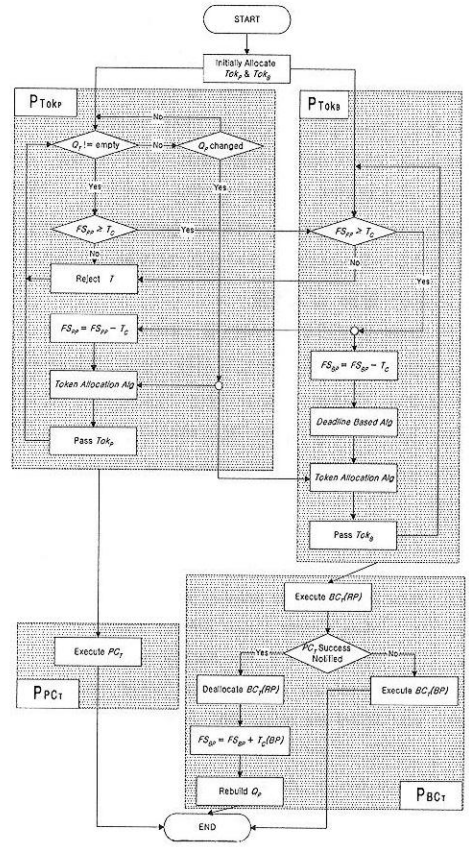


그림 3 제안된 이중토큰 기반 고장허용 스케줄링의 논리적 흐름도

세 번째 부류는 주카피를 수행하고 있는 프로세서(P_{PC_T})이다. 이 프로세서는 자신의 지역 큐로부터 가져온 태스크의 주카피를 실행한다. 실행이 성공한 경우에는 복구카피를 가지고 있는 프로세서에게 성공하였음을 알려주어 복구카피의 복구부분을 회수 하도록 한다.

마지막 부류는 복구카피를 수행하고 있는 프로세서

(P_{BC_T})이다. 이 프로세서에서는 지역큐에서 가져온 태스크의 복구카피 중 중복 부분이 있다면 이를 수행하다가 주카피를 수행하는 프로세서가 성공했음을 통보 받으면 복구부분을 회수하고 그만큼을 다시 자신의 여유시간에 복원해준다. 만약 성공여부를 통보 받지 못하면 주카피를 가진 프로세서에 고장이 발생한 것이므로 나머지 복구부분을 모두 수행한다. 이렇게 함으로써 고장이 발생한 경우에도 태스크의 마감시간을 보장할 수 있다.

4.2 토큰 할당 알고리즘

토큰 할당 알고리즘은 시스템 내에 중앙 분배 프로세서의 역할을 수행하는 프로세서를 찾아 주토큰을 할당하는 정책과, 주토큰을 갖고 있는 프로세서에 고장이 발생했을 경우 복구를 위한 프로세서에게 복구토큰을 할당하는 정책을 말한다. 알고리즘의 의사코드는 [그림 4]와 같다.

```
1: if F that has a token failed
2:    then if F has the primary token
3:           then Create a new primary token;
4:           else Create a new backup token;
5:           // in case of P has the backup token //
6:    else Select the token that F has;
7: If IdleQ_F is not empty
8:    then Get P' from the head of IdleQ_F;
9:    else if ReadyQ_F is not empty
10:          then Get P' from the head of ReadyQ_F;
11:          else Get P' from the head of ActiveQ_F;
12: If P' has already had a token
13:    then Ignore P' and Goto 6;
14:    else allocate the token to P';
```

그림 4 토큰할당 알고리즘

[그림 4]에서 보는 바와 같이 토큰할당 알고리즘은 크게 두 단계로 되어있다. 첫 번째 단계는 전달할 토큰을 생성하는 경우로 1~5 라인에 해당한다. 이 단계에서는 토큰을 갖고 있는 프로세서의 고장여부에 따라 선택되는 토큰이 다르다. 1번 라인에서는 토큰(주토큰 또는 복구토큰)을 갖고 있는 프로세서에 고장이 발생했는지 여부를 검사하며, 만일 고장이 발생했다면 2번 라인에서 고장이 발생한 프로세서가 갖고있는 토큰의 종류를 파악하여 새로운 주토큰이나 복구토큰을 각각 3번과 4번 라인에서 생성한다.

그러나, 토큰을 갖고 있는 프로세서에 고장이 발생하지 않았다면 5번 라인에서와 같이 그 프로세서가 가지고 있던 토큰을 그대로 선택한다.

두 번째 단계에서는 프로세서 큐의 상태를 조사하여 첫 번째 단계에서 생성된 토큰을 임의의 프로세서에 할당한다. 이 때 프로세서 큐를 idle, ready, active 순으로 검사하여 여유 시간이 가장 많은 프로세서를 선택한다(6~10 라인). 6번 라인과 같이 idle 큐가 비지 않았다면, 즉 idle 프로세서가 있다면 큐 헤드에서 프로세서를 선택하고, 그렇지 않고 7~8번 라인처럼 idle 큐는 비어있고 ready 큐가 비지 않았다면 ready 큐의 헤드에 있는 프로세서를 선택한다(9번라인). 그리고 모든 프로세서가 현재 태스크를 처리 중이라면 10번 라인처럼 active 큐의 헤드에서 프로세서를 선택한다. 그 다음에는 선택된 프로세서가 주토큰과 복구토큰을 동시에 모두 가져서는 안되기 때문에 그 프로세서가 이미 토큰을 가지고 있는가를 검사해야 한다 (11번 라인). 만약 이미 토큰을 가지고 있다면 그 프로세서는 제외하고 다시 선택한다 (12번 라인). 그렇지 않다면 13번 라인처럼 생성된 토큰을 선택된 프로세서에 할당한다. 위와 같은 과정을 거치는 토큰할당 알고리즘의 순서도는 [그림 5]와 같다.

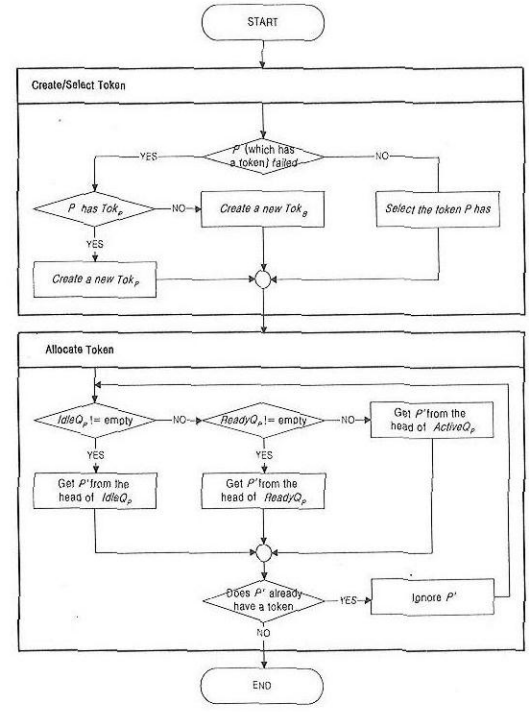


그림 5 토큰 할당 알고리즘 순서도

한편, 토큰 할당 알고리즘은 토큰을 갖고 있는 프로세서의 고장여부에 따라 차이가 있다. 만약 주토큰을 가진 프로세서에 고장이 발생하면, 복구토큰을 가진 프로세서가 할당 알고리즘을 수행하여 선택된 프로세서에게 주토큰을 부여하고, 그 프로세서로 하여금 중앙 분배 프로세서의 역할을 수행하도록 한다.

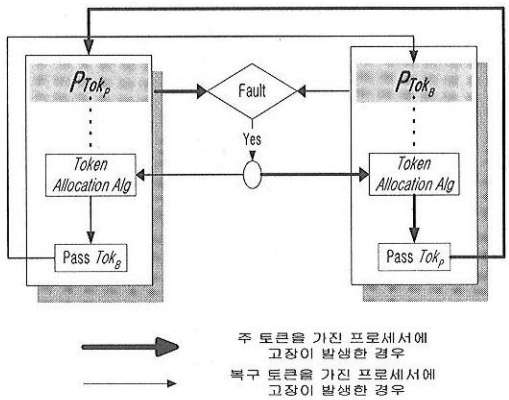


그림 6 고장 발생시 토큰의 복구 과정

또, 복구토큰을 가진 프로세서에 고장이 발생했을 때는 주토큰을 가진 프로세서가 할당 알고리즘을 수행하여 선택된 프로세서에게 복구토큰을 부여하여 주토큰을 가진 프로세서가 고장이 나는 경우를 대비하도록 한다. 이 과정은 [그림 6]과 같다.

4.3 마감시간 기반 태스크 할당 알고리즘

마감시간 기반 태스크 할당 알고리즘은 복구토큰을 갖고 있는 프로세서가 태스크의 복구카피를 수용할 수 있을 때, 이 프로세서가 갖고 있는 여분의 시간에 태스크의 복구카피를 어떻게 스케줄 할 것인가를 다루는 알고리즘이다. 기본적인 개념은 복구카피를 가능한 한 마감시간에 가깝게 스케줄 하는 것이다. 그렇게 함으로써 주카피가 성공적으로 완료되었을 때 복구카피를 회수하여 프로세서는 더 많은 여유시간을 가질 수 있다. 알고리즘의 의사코드는 [그림 7]에 나타나 있으며, 1번라인에서는 복구카피의 도착시간을 마감시간으로부터 실행시간을 뺀 시간으로 재 설정 한다. 이 때 새로운(수정된) 도착시간으로부터 태스크가 수행될 수 있는 여유시간이 없을 경우 (2번 라인)에는 마감시간보다 일찍 태스크가 완료되어도 무방하므로 재 설정한 도착시간을 앞으로 당기면서 실행될 수 있는 여유시간을 찾는다 (3번라인). 그 다음에는 발견한 여유시간 부분으로 도착시간을 조정한 후 (4번 라인), 지역 큐에 삽입한다 (5번라인).

```
1: Update  T_a by  T_a - T_c ;
2: If  FS_BF which can hold  BC_T not exist
3:       then Find the proper  FS_BF;
4:            Adjust  T_a to the start time of the found  FS_BF;
5: Insert  BC_T into local  Q_T;
```

그림 7 마감시간 기반 태스크 할당 알고리즘

알고리즘의 수행도는 [그림 8]과 같다.

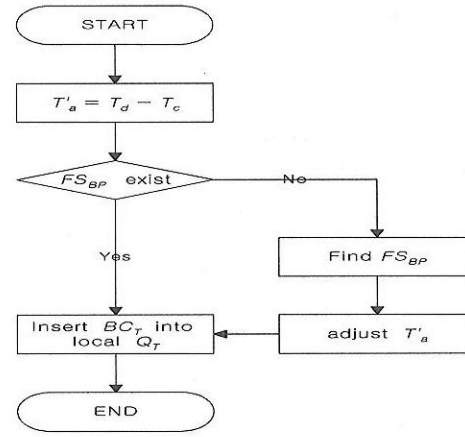


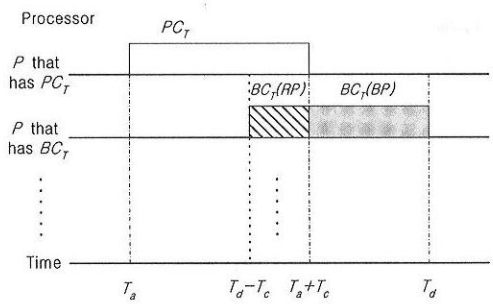
그림 8 마감시간 기반 태스크 할당 알고리즘의 수행도

이 알고리즘을 수행한 결과는 [그림 9]에 나타나 있는데 주카피는 가능한 일찍, 복구카피는 가능한 늦게 할당한다. 이렇게 함으로써 laxity가 적은 경우, 주카피와 복구카피의 중복부분을 병행 실행하다가 고장이 발생했을 때에는 복구 부분을 계속 실행함으로써 고장허용을 보장할 수 있다.

또한, 주카피가 성공적으로 실행됐을 때에는 복구 부분 (태스크를 재실행할 수 있을 정도로 충분한 laxity가 있는 경우에는 복구카피 전체)을 회수할 수 있으므로 프로세서의 효율성을 높일 수 있고 더 많은 태스크를 수용할 수 있다.

이 알고리즘은 [1]이나 [3]에서 사용되고 있는 알고리즘들과 유사하지만 프로세서 큐를 프로세서의 상태와 여유시간의 크기에 따라 분류하고 다른 고려사항 없이 마감시간만을 참조하여 복구카피를 할당하기 때문에 구현이 단순한 알고리즘이 된다.

그림 9 마감시간 기반 태스크 할당 알고리즘의 실행 예



5. 시뮬레이션

시뮬레이션은 동일한 태스크 집합에 대하여 본 논문에서 제안된 알고리즘과 [2],[3]에서 제안된 중앙 분배 프로세서가 존재하는 시스템의 고장허용 알고리즘과 비교하였다. 시뮬레이션에 사용된 파라미터들은 다음과 같다.

- 프로세서의 수(P) : 시뮬레이션에 사용된 프로세서의 수는 6개인데, 중앙 분배 프로세서가 존재하는 멀티프로세서 시스템에서는 하나의 프로세서가 전체 스케줄을 유지하는 중앙 분배 프로세서가 되어야 하므로 실제로 태스크를 처리하는 어플리케이션 프로세서는 5개이고 이중토큰에 기반한 멀티프로세서 시스템에서는 중앙 분배 프로세서가 존재하지 않으므로 6개 모두를 어플리케이션 프로세서로 사용하였다.
- 평균연산시간(c) : 도착하는 태스크의 연산시간은 평균 5로 uniform하게 분포한다고 가정한다.
- 부하(γ) : 태스크가 실시간 제약이나 고장허용의 요구사항이 없는 경우 이용하는 프로세서 시간의 평균율이다. 부하가 클수록 더 작은 태스크들의 도착간 시간을 갖는다. 태스크들의 도착간 시간은 평균 α ($=c/\gamma*P$) 인 uniform하게 분포한다고 가정한다.
- 수행 태스크는 1000개로 구성되어있다.
- 태스크의 윈도우(w_i)는 마감시간(d_i) - 준비시간(r_i) 로 정의된다.
- β : 평균 c $\times$ β로 uniform하게 분포하는 윈도우의 크기를 조절한다.

[그림 10]은 다른 윈도우 크기에 따라 부하를 변화시키면서 중앙 분배 프로세서가 존재하는 멀티프로세서 환경에서의 실시간 고장허용 스케줄링 알고리즘과 이중토큰에 기반한 멀티프로세서 환경에서의 실시간 고장허용 스케줄링 알고리즘에 의한 태스크 거부율을 비교하였다. 거부율은 부하가 증가함에 따라, 윈도우의 크기가 작아짐에 따라 커지는 것을 알 수 있다. 그리고 본 논문에서 제안된 알고리즘이 중앙 분배 프로세서가 존재하는 멀티프로세서 시스템에서 작동하는 알고리즘보다 더 많은 태스크를 수용하는 것을 알 수 있다.

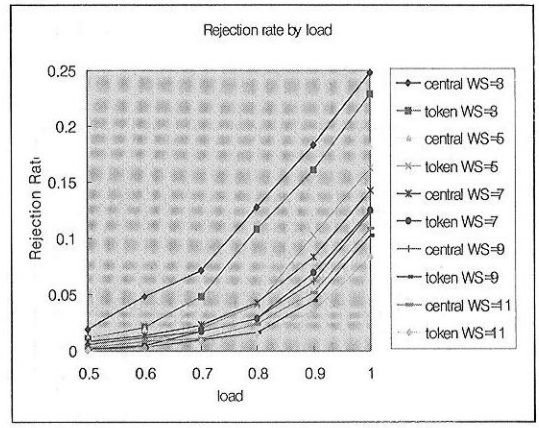


그림 10 윈도우 크기에 따른 중앙 분배 프로세서 시스템과 토큰 방식 시스템과의 거부율 비교

한편, 본 논문에서 다루는 시스템에서는 정상적으로 토큰을 전달해야 하는 경우와 토큰을 가진 프로세서에 고장이 발생하는 경우 (고장이 발생한 경우에는 다른 토큰을 가진 프로세서가 토큰을 복원하여) 모두다 토큰 할당 알고리즘을 통해 선택된 다른 프로세서에게 전달해야 하기 때문에 기존의 고장이 발생하지 않는다고 가정한 중앙 분배 프로세서가 존재하는 시스템보다는 더 많은 연산을 반복적으로 수행해야 하는 오버헤드와 문맥교환이 발생하게 된다. 따라서 시스템의 부하가 많은 경우에 태스크를 처리하던 프로세서가 스케줄링에 참가하게 되면 태스크의 마감시간을 위배할 수 도 있다. 그러므로, 다양한 토큰 할당정책에서 기인되는 오버헤드, 고장허용 정도, 그리고 시스템 이용률간의 득실(trade-off)에 대하여 더 연구를 진행할 예정이다.

6. 결 론

본 논문에서는 중앙 분배 프로세서가 존재하지 않는 경성 실시간 멀티프로세서 시스템에서 이중토큰에 기반한 고장허용 스케줄링 알고리즘을 제안하였다. 이는 기존의 실시간 멀티프로세서 시스템 환경에서 중앙 분배 프로세서는 실패하지 않는다는 가정을 제거하므로써 중앙 분배 프로세서에서 고장이 발생했을 때 시스템 전체가 실패하는 문제를 해결하였으며, 시스템내의 모든 프로세서들이 태스크 처리에 참여함으로써 이용률을 증진시켰다. 그리고, 주토큰을 가진 프로세서에서 고장 발생

시 복구토큰을 가진 프로세서가 이를 복구 해주는 이중 토큰 구현 메카니즘은 고장허용 시스템의 하드웨어 및 소프트웨어의 이중 구조와 개념적으로 일관성을 잘 유지한다. 또한, 비주기적 태스크의 복구카피를 수행할 때 마감시간에 기반한 비선점 알고리즘을 사용함으로써 고장 발생 시 스케줄링의 예측성을 높이고 구현 오버헤드를 감소시켰다.

향후 연구로는 실용성을 고려한 합리적인 가정들을 제시하고 새로 도착하는 태스크들을 프로세서에 분배하는 알고리즘과 각 프로세서의 큐에 큐잉 된 태스크의 처리방식에 대한 정확한 모델을 제시할 예정이다. 그리고, 다양한 토큰 할당정책 등을 비교분석하고 각각의 응용에 따른 득실을 고려하여 보다 견고하고 실용적인 고장허용 스케줄링 알고리즘을 개발할 예정이다.

참 고 문 헌

[1] T. Tsuchiya, Y. Kakuda, and T. Kikuno, "A New Fault-Tolerant Scheduling Technique for Real-Time Multiprocessor Systems," International Workshop on Real-Time Computing Systems and Applications, pp. 197-202, 1995.

[2] D. Mossé, R. Melhem, and S. Ghosh, "Analysis of a Fault-Tolerant Multiprocessor Scheduling Algorithm," Proc. 24th International Symposium on Fault-Tolerant Computing, pp. 16-25, 1994.

[3] S. Ghosh, R. Melhem, and D. Mossé, "Fault-Tolerant Scheduling on a Hard Real-Time Multiprocessor System," Proc. 8th International Parallel Processing Symposium, pp. 775-782, 1994.

[4] A. L. Liestman and R. H. Campbell, "A Fault-Tolerant Scheduling Problem," IEEE Trans. Software Eng., vol. 12, no. 11, pp. 1,089-1,095 Nov. 1986.

[5] C.M. Krishna and K.G. Shin, "On Scheduling Tasks with a Quick Recovery from Failure," IEEE Trans. on Computers, vol. 35, no. 5, pp. 448-455, May 1986

[6] Y. Oh and S. Son, "Multiprocessor Support for Real-Time Fault Tolerant Scheduling," Proc. IEEE 1991 Workshop Architectural Aspects of Real-Time Systems, pp. 76-80 San Antonio, Tex., Dec. 1991

[7] Y. Oh and S. Son, "Fault-Tolerant Real-Time Multiprocessor Scheduling," Technical Report TR-92-09, University of Virginia, April 1992

우 철 희

1996년 2월 경희대학교 전자계산공학과 (학사). 1998년 2월 경희대학교 전자계산공학과 (석사). 1998년 3월~현재 LG 정보통신.

이 승 룡

제 25 권 제 1 호(A) 참조

이 종 원

1985년 전북대학교 수학과 (학사). 1987년 미국 Illinois Institute of Technology 전산학과 (석사). 1991년~현재 한국통신 멀티미디어연구소 연구원. 1994년~1995년 경희대학교 산업정보대학원 강사. 1996년~1997년 미국 UniSQL사 파견근무(ORDBMS 개발). 1998년~현재 경희대학교 교육대학원 강사. 관심분야는 운영체제, 실시간 컴퓨팅, 데이타베이스, GIS

오 삼 권

1980년 한국항공대학교 항공전자공학 (학사). 1980~1984년 삼성전자 통신연구소 (연구원). 1986년 University of South Florida (U.S.A) 컴퓨터 과학 (석사). 1994년 Queen's University (Canada) 컴퓨터과학 (박사). 1994년~1995년 한국전자통신연구원 (선임연구원). 1995년~현재 호서대학교 컴퓨터학부 (조교수). 관심분야는 Real-time and distributed systems, Operating systems, Communications protocol, Multimedia systems, Fault tolerance