1999年度 大韓電子工學會 秋季綜合學術大會 論文集 第 22 卷 第2號 99/11

경성 실시간 멀티프로세서 환경에서 고장허용을 위한 토큰할당 알고리즘

최 장 홍, 이 승 룡
경희대학교 전자계산공학과
전화 : (0331) 201-2950 / 팩스 : (0331) 202-4730

Token Allocation Algorithm for Fault Tolerant in Hard Real-Time Multiprocessor Systems

Jang Hong Choi, Sungyoung Lee
Computer Engineering Kyung Hee University
E-mail : {jhchoi, sylee}@oslab.kyunghee.ac.kr

Abstract

Woo[8] proposed dual-token based fault-tolerant scheduling algorithm in multiprocessor environment for resolving the problem of old systems that have a central dispatcher processor. However, this algorithm does not present token allocation algorithm in detail when central dispatcher processor has failed. In this paper, we propose a fault detection algorithm and processor selection algorithm for token allocation when central dispatcher processor has failed.

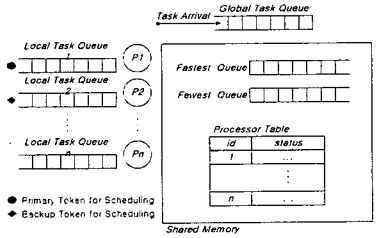
1. 서론

지금까지 연구된 실시간 멀티프로세서 환경 하에서 경성 비주기적 태스크에 대한 고장허용 문제는 태스크를 실행하는 프로세서에서 발생하는 고장을 다루고 있다. 이 경우 고장이 발생한 프로세서에서 수행되던 태스크의 사본 태스크는 다른 프로세서에서 고장복구를 위해 재실행된다. 그런데, 태스크들의 전체 스케줄을 담당하면서 고장이 발생했을 때 이를 감지하여 복구 카피를 가진 프로세서에게 고장복구를 하도록 통보하는 중앙분배 프로세서는 고장이 발생하지 않는다고 가정[2],[3] 하였지만, 이는 시스템내의 어떤 프로세서에서도 고장이 발생할 수 있다는 현실적인 상황을 간과한 것이고, 중앙분배 프로세서의 고장은 전체 시스템의 실패라는 치명적 결과를 초래할 수 있다. 이러한 문제를 해결하기 위해 Woo[8]는 실시간 멀티프로세서 시스템 하에서 이중 토큰 기반의 고장허용 스케줄링 알고리즘을 제안하였다. 그러나, 이 알고리즘은 토큰을 소유한 프로세서에 고장이 발생했을 때, 고장을 감지하여 토큰 할당을 위한 적절한 프로세서를 선정하여 토큰을 할당하는 알고리즘은 제시하지 않았다.

따라서, 본 논문에서는 토큰 소유 프로세서에서 고장이 발생했을 경우, 이를 감지하는 알고리즘과 토큰 할당 프로세서 선정 알고리즘을 제안한다. 제안한 시스템은 다수의 프로세서가 존재하는 환경으로 태스크의 스케줄링은 토큰을 소유한 두 개의 프로세서가 담당한다. 토큰이 할당된 두 프로세서는 상대 프로세서에서 고장이 발생하는지 상호 모니터링 한다. 토큰은 모든 프로세서가 할당받을 수 있으며, 토큰은 조건이 만족될 때 한 프로세서에서 다른 프로세서로 전달된다. 토큰이 전달되는 경우는 토큰을 소유한 프로세서에 고장이 발생했을 때이다. 토큰을 소유한 프로세서에서 고장이 발생하면 또 다른 토큰을 소유한 프로세서가 이것을 감지하여 새로운 프로세서를 선정하여 토큰을 할당한다.

2. 시스템 모델

제안하는 시스템은 태스크 수행을 위한 n개의 동일한 프로세서들로 구성된 경성 실시간 멀티프로세서 시스템이다. [그림 1]은 시스템 모델을 나타낸 것이다.



[그림 1] 시스템 모델

• 전역 태스크 큐(Global Task Queue)
수행 요청되는 태스크들을 저장하기 위한 큐이다. 전역 태스크 큐의 정렬 방식은 Laxity가 가장 작은 태스크가 먼저 스케줄 되는 Minimum Laxity First(MLF) 방식을 사용한다. 이렇게 함으로써 데드라인이 빠른 태스크나 laxity가 작은 태스크가 나중에 도착할 경우 태스크 수용율을 높일 수 있다

• 지역 태스크 큐(Local Task Queue)
글로벌 태스크 큐에 도착된 태스크들 중 해당 프로세서가 처리할 수 있는 태스크들이 스케줄링된 큐이다. 여기에는 스케줄링된 태스크들에 대한 정보(start time, end time, execution time, etc)도 함께 저장된다.

• Fastest Queue
스케줄링된 마지막 태스크의 종료시간이 가장 빠른 프로세서의 순으로 정렬된 큐이다. 토큰을 소유한 프로세서에 고장이 발생하면, 이를 복구하기 위해 새로 토큰을 할당할 프로세서를 찾게 된다. 이때에 Fastest Queue의 헤드에 위치한 프로세서를 선정하여 토큰을 할당한다.

• Fewest Queue
스케줄링된 태스크의 숫자가 가장 적은 프로세서의 순으로 정렬된 큐이다. 토큰을 소유한 프로세서에 고장이 발생하면, 이를 복구하기 위해 새로 토큰을 할당할 프로세서를 찾게 된다. 이때, Maximum Search Time 이내에 토큰 할당을 위한 새로운 프로세서를 찾지 못할 경우, 임의의 프로세서를 선정하여 토큰을 할당해야 한다. 이때에 Fewest Queue에서 프로세서를 선정하여 토큰을 할당한다.

• 프로세서 테이블(Processor Table)
Fewest Queue에 정렬된 프로세서의 상태 정보를 가진다. 상태정보는 프로세서에 스케줄링된 태스크가 실행중인지 아닌지를 나타낸다. 프로세서 테이블을 두는 이유는 Fewest Queue에서 프로세서를 하나 선정했을 때, 현재 스케줄링된 태스크가 실행중이면 토큰을 할당할 수 없다. 왜냐하면, 실행중인 태스크는 kill 할 수 없기 때문에 Maximum Search Time 이내에 토큰을 할당 할 수 없다. 따라서 프로세서 테이블에서 상태정보를 보고 현재 프로세서가 실행중인지 아닌지를 판단하여, 실행중이면 Fewest Queue에서 다음 프로세서를 선정하여 실행중이 아니면 토큰을 할당한다.

• 주토큰 프로세서(Primary Token Processor)
주태스크 스케줄링을 하면서 복구토큰 프로세서의 고장을 모니터링 한다. 고장을 감지하면 복구토큰을 소유하게 되어, 주태스크과 복구태스크 스케줄링을 함께 수행한다. 또한, 복구토큰 할당을 위해 새로운 프로세서를 찾는다.

• 복구토큰 프로세서(Backup Token Processor)
복구태스크 스케줄링을 하면서 주토큰 프로세서의 고장을 모니터링 한다. 고장을 감지하면 주토큰을 동시에 소유하게 되어, 주태스크과 복구태스크 스케줄링을 함께 수행한다. 또한, 주토큰 할당을 위해 새로운 프로세서를 찾는다.

본 논문에서 사용되는 가정은 다음과 같다.

A1) 모든 태스크들은 비주기적이고 도착했을 때 비선점으로 스케줄 된다.
A2) 고장은 프로세서에서만 발생한다.
A3) 주태스크와 복구태스크의 할당 구간에서 동시에 고장이 발생하지 않는다.
A4) 시스템은 태스크의 고장허용을 위해 primary-backup 스케줄링 방식을 사용한다.
A5) 주태스크와 복구태스크는 서로 동일하며, 같은 worst case execution time를 갖는다.

3. 고장감지 및 토큰할당 알고리즘

3.1 기본 개념

제안하는 알고리즘은 두 개의 알고리즘으로 구성된다. 하나는 토큰을 소유한 프로세서의 고장을 감지하기 위한 고장감지 메카니즘이고, 다른 하나는 고장이 발생했을 때, 토큰을 할당하기 위한 새로운 프로세서의 선정 알고리즘이다. 토큰을 소유한 프로세서는 프로세서에 고장이 발생한 경우에만 토큰을 다른 프로세서에게 전달한다.

첫 번째는 고장감지 메카니즘은 토큰을 소유한 프로세서에 고장이 발생했는지를 감지하는 메카니즘이다. 이 메카니즘은 주토큰을 소유한 프로세서와 복구토큰을 소유한 프로세서가 고장 감지를 위해 상호 고장을 모니터링을 한다. 상대 프로세서의 고장 감지를 위해서 고장 감지 태스크를 둔다.

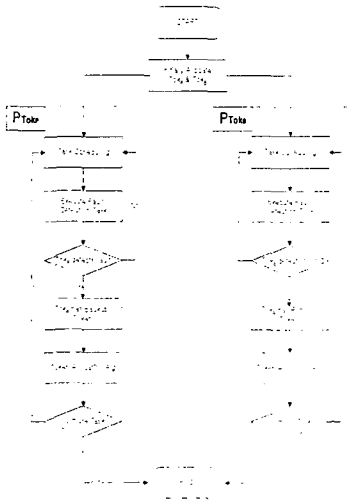
두 번째는 토큰 할당을 위해 새로운 프로세서를 선정하는 알고리즘이다. 토큰을 할당할 경우 새로운 프로세서를 선정해야 한다. 토큰을 할당하기 위한 프로세서의 선정은 다음의 두 가지 방법을 따른다.

- 스케줄링된 모든 태스크 중 마지막 태스크의 종료시간이 가장 빠른 프로세서의 선정.
- 스케줄링된 태스크의 숫자가 가장 적은 프로세서 선정.

또한, 토큰을 할당하기 위한 프로세서를 찾기 위해 Maximum Search Time을 두어 이 시간 이내에 적절한 프로세서를 찾지 못할 경우, 임의의 프로세서를 선정하여 토큰을 할당한다.

제안된 중앙분배 프로세서의 토큰 할당 알고리즘의 핵심 아이디어는 다음과 같다. 주토큰을 소유한 프로세서는 주 태스크 스케줄링을 담당한다. 그러면서, 복구토큰을 소유한 프로세서의 고장을 주기적으로 모니터링 한다. 같은 방식으로, 복구토큰을 소유한 프로세서는 복구 태스크 스케줄링을 담당하면서 주토큰을 소유한 프로세서의 고장을 주기적으로 모니터링 한다. 이때, 토큰을 소유한 어느 한 프로세서에 고장이 발생하면 토큰을 소유한 다른 프로세서가 고장이 발생한 프로세서의 토

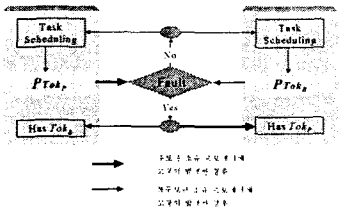
큰을 동시에 소유한다. 두 개의 토큰을 소유한 프로세서는 주태스크 스케줄링과 복구태스크 스케줄링을 모두 담당하게된다. 그리고, 토큰을 할당하기 위해 새로운 프로세서를 찾는다. 새로운 프로세서를 찾으면, 토큰을 할당하여 주토큰을 소유한 프로세서는 주태스크 스케줄링을, 복구토큰을 소유한 프로세서는 복구태스크 스케줄링을 한다. 이러한 방식을 사용함으로써, 토큰을 소유한 프로세서의 고장을 감지할 수 있을 뿐 아니라, 중앙 분배 프로세서의 고장으로 인해 시스템 전체가 실패하는 것을 방지할 수 있다. 본 논문에서 제안한 알고리즘의 논리적 흐름도는 [그림 2]와 같다.



[그림 2] 제안된 중앙분배 프로세서의 토큰 할당 알고리즘의 논리적 흐름도

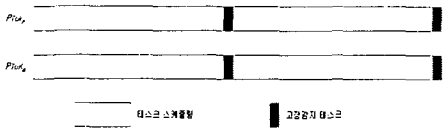
3.2 고장감지 메카니즘

토큰을 소유한 프로세서에서의 고장을 감지하기 위해 주토큰을 소유한 프로세서와 복구토큰을 소유한 프로세서가 고장을 상호 모니터링 한다. 토큰을 소유한 어느 한 프로세서에 고장이 발생하면 다른 하나의 프로세서가 이를 감지하여 두 개의 토큰을 동시에 소유하게 된다. [그림3]은 고장을 상호 모니터링 하는 고장감지 메카니즘을 나타낸 것이다



[그림 3] 고장감지 메카니즘

토큰을 소유한 프로세서의 고장을 감지하기 위해서 고장을 감지하는 고장감지 태스크를 둔다. [그림4]는 고장감지 태스크를 나타낸 것이다. 고장감지 태스크는 주기적 태스크의 성격을 가진다. 즉, 일정 시간마다 상대 프로세서의 고장을 감지하기 위해 주기적으로 태스크가 실행된다. 고장감지 태스크가 실행되는 동안 스케줄링이 불가능하지만, 태스크가 실행되는 시간이 매우 짧기 때문에 실제로 토큰을 소유한 프로세서의 태스크 스케줄링에 거의 영향을 미치지 않는다. 따라서, 고장감지 태스크 실행에 따른 스케줄링 지연은 무시한다.



[그림 4] 고장감지 태스크

고장감지 태스크의 실행에 대한 핵심 아이디어는 다음과 같다. 먼저, 토큰을 소유한 프로세서는 주기적으로 태스크를 발생시킨다. 이 태스크는 상대 프로세서에게 특정한 비트 값을 전송한다. 비트 값을 전송 받은 프로세서는 상대 프로세서에 고장이 발생하지 않았음을 감지한다. 만약, 프로세서에 고장이 발생하였다면, 고장감지 태스크는 실행되지 않으며, 비트 값을 전송할 수 없게 된다. 따라서, 비트 값을 전송 받지 못한 프로세서는 상대 프로세서에 고장이 발생했음을 감지하게 되고, 상대 프로세서의 토큰을 소유하게 된다. 그 후에 토큰 할당을 위한 프로세서 선정 알고리즘을 수행한다. [그림 5]는 고장감지 태스크의 실행에 대한 의사코드이다.

```
1 : P that has a primary(backup)token executes Fault Detection Task;
2 : If P that has a primary(backup) token does not detects Fault Detection Task
3 : {
4 :   then has backup(primary) token;
5 :        executes Token Allocation Algorithm;
6 : }
```

[그림 5] 고장감지 태스크의 실행

3.3 토큰 할당을 위한 프로세서 선정 알고리즘

토큰을 소유한 프로세서에 고장이 발생하면, 토큰을 재할당하기 위해 새로운 프로세서를 선정해야 한다. 토큰 재할당 프로세서의 선정은 다음의 두 가지 방법을 따른다.

- 할당된 모든 태스크 중 마지막 태스크의 종료시간이 가장 빠른 프로세서의 선정.
- 할당된 태스크의 숫자가 가장 적은 프로세서 선정.

주토큰을 가진 프로세서에서 고장이 발생하였다고 가정하자. 복구토큰을 가진 프로세서가 이를 감지하여 복구를 위한 주토큰을 새롭게 할당할 프로세서를 선정해야 한다. 복구토큰을 가진 프로세서는 주토큰을 소유하고 주토큰을 가진 프로세서가 수행했던 주태스크 스케줄링과 복구태스크 스케줄링 모두를 담당한다. 복구토큰을 가진 프로세서는 태스크 스케줄링을 하면서 스케줄링을 하지 않는 시간에 새롭게 토큰을 할당할 프로세서를 선정한다. 새로운 프로세서의 선정은 위의 방법에 따라 프로세서를 선택하여 주토큰을 할당시키고 이 프로세서에는 더 이상의 태스크를 스케줄링하지 않는다. 스케줄링된 태스크가 모두 실행되면 주태스크의 스케줄링을 시작한다.

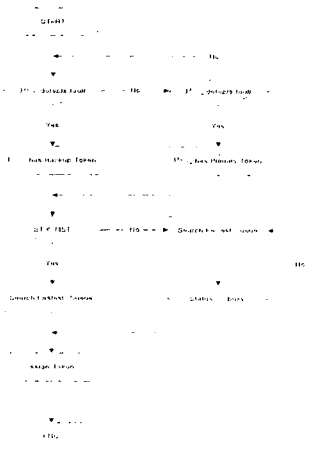
또한, 토큰을 할당하기 위한 프로세서를 찾기 위해

Maximum Search Time(MST)을 두어 이 시간 이내에 적절한 프로세서를 찾지 못할 경우 임의의 프로세서를 선정하여 토큰을 할당해야 한다. 그렇지 않으면, 복구를 위해 동시에 두 개의 토큰을 소유한 프로세서에 고장이 발생할 수도 있기 때문이다. 이 경우 시스템 전체가 실패하기 때문에 치명적 결과를 가져올 수 있다. 그림[6]은 토큰 할당 프로세서 선정 알고리즘의 의사코드이다.

```
1 : If P that has a primary token failed
2 :    then P that has a backup token has primary token;
3 : else P that has a primary token has backup token;
4 : If Maximum Search Time is greater than Search Time
5 :    then Get P' from Fastest Queue;
6 : else {
7 :    while (Status != busy)
8 :        Get P' from Fewest Queue;
9 : }
10 : Assign token to P'
```

[그림6] 토큰 할당 프로세서 선정 알고리즘

[그림6]에서 보는 바와 같이 주토큰을 소유한 프로세서에 고장이 발생(1번 라인)하면, 주토큰 소유 프로세서가 백업 토큰을 소유한다(2번 라인). 그렇지 않으면, 백업 토큰 소유 프로세서가 주토큰을 소유한다(3번 라인). 다음으로 토큰을 할당하기 위한 프로세서를 찾는데, Maximum Search Time 이내에 새로운 프로세서를 찾으면(4번 라인), Fastest Queue의 헤드에 위치한 프로세서를 선정한다(5번 라인). 그렇지 않으면, Fewest Queue에서 프로세서를 선정해야 하는데, 만약 선택한 프로세서가 태스크를 수행중이면 Fewest Queue의 다음에 위치한 프로세서를 선택하여 태스크가 수행중이 아니면 토큰 할당 프로세서로 선정한다 (6번 라인 ~9번 라인). 마지막으로 선정한 프로세서에 토큰을 할당한다 (10번 라인). 그림[7]은 위의 과정을 순서도로 나타낸 것이다.



[그림7] 토큰 할당 프로세서 선정 알고리즘 순서도

4. 결론 및 향후 연구방향

본 논문에서는 경성 실시간 멀티프로세서 환경에서 중앙분배 프로세서의 토큰 할당 알고리즘을 제안하였다. 이는 중앙 분배 프로세서의 역할을 하는 프로세서에 고장이 발생했을 때, 이를 감지하여 토큰을 할당함으로써 중앙 분배 프로세서에 고장이 발생했을 때 시스템 전체가 실패하는 문제를 해결하였으며, 기존 알고리즘의 문제점인 토큰 전달에 따른 오버헤드를 토큰을 소유한 프로세서에 고장이 발생했을 경우에만 토큰을 전달하도록 제한함으로써 오버헤드를 감소시켰다. 그리고, 글로벌 태스크 큐의 정렬 방식을 laxity 가장 작은 태스크가 먼저 스케줄 되는 Minimum Laxity First(MLF) 방식을 사용함으로써, 마감시간이 빠른 태스크나 laxity가 작은 태스크가 나중에 도착할 경우 태스크의 수용율을 높였으며, 토큰을 소유한 프로세서에 고장이 발생했을 경우 이를 감지하는 메카니즘을 개발하여 시스템에 적용하였다. 또한 제안한 알고리즘은 이중토큰 방식을 사용함으로써 중앙분배 프로세서가 항상 쌍을 이루어 동작하는 장점을 가진다.

향후 제안한 알고리즘을 확장하여 태스크 스케줄링의 효율성 향상을 위한 Free slot search 알고리즘의 개발이 필요하다.

참고문헌

[1] 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.

[2] 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.

[3] 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.

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

[5] 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

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

[7] T. Tsuchiya, Y. Kakuda, and T. Kikuno, "A New Fault-Tolerant Scheduling Technique for Real-Time Multiprocessor Systems," Proc. '95 Real-Time Conference System Application, pp. 197-202, 1995.

[8] C.H. Woo, S.Y. Lee S.K. Oh, J.W. Lee, "Dual Token-Based Fault-Tolerant Scheduling for Hard Real-Time Multiprocessor Systems", Journal of KISS(A): Computer Systems and Theory, Vol. 25, No. 10, 1998, pp. 997-1005