경성 실시간 다중프로세서 시스템에서 주기적인 태스크를 위한 스케줄링 알고리즘

신동훈, 김용석
강원대학교 컴퓨터, 정보통신공학부

A Scheduling Algorithm for Periodic Tasks on Hard-Real Time Multiprocessor Systems

Dong-Hoon Shin, Yong-Seok Kim
Div. of Computer, Information and Communications Engineering,
Kangwon National University

요 약

n개의 동일한 프로세서 상에서 m개의 주기적인 태스크들의 적합한 스케줄을 구하기 위한 알고리즘을 제시한다. 정수이하의 작은 시간으로 태스크의 실행시간이 잘라질 수 있다는 가정 하에 태스크 집합은 기본 스케줄링 알고리즘을 통해서 부분적인 스케줄을 얻고 정수 화를 위한 다중프로세서 스케줄링 알고리즘으로 적합한 스케줄을 구성한다. 또한 태스크들에 대한 활용도의 합이 n보다 작거나 같다는 조건은 실시간 다중프로세서 시스템에서 주기적인 태스크 집합에 대한 적합한 스케줄을 구성하기 위한 필요·충분조건임을 보여준다.

1. 개 요

실시간 시스템에서 제어나 감시기능을 위해 다중프로세서의 사용이 최근 증가하고 있으며, 저가의 마이크로프로세서가 보급되면서 실시간 응용에 많은 프로세서의 적용이 실행 가능해졌다. 이러한 시스템의 효율적인 사용을 위해서는 적절한 스케줄링이 요구되는데, 다중프로세서 시스템에서는 해결하기 어려운 문제점이 나타난다. 특히 시간에 대해 중요성을 가지는 태스크들은 물리적으로 표현되는 특정한 마감시간이 미리 할당되어야 하고 실시간 다중프로세서 시스템에서는 단일프로세서와는 달리 태스크들에 대한 정보가 미리 알려지지 않으면 적합한(feasible) 스케줄을 위한 알고리즘을 찾을 수 가 없다는 것을 보여 주고 있다.[2]

주어진 주기적인 태스크 집합에 대한 활용도(utilization factor) U에 대하여 단일 프로세서 시스템의 경우는 조건 $U \le 1$은 적합한 스케줄을 찾기 위한 필요·충분조건임은 알려져 있다.[3] n개의 동일한 프로세서로 구성된 다중프로세서 시스템에서 조건 $U \le n$은 주어진 태스크 집합을 스케줄링하기 위한 필요조건이다. 그러나 이 조건이 충분 조건인지는 여전히 연구대상이다[2]. 따라서 본 논문에서는 조건 $U \le n$을 만족하는 주기적인 태스크 집합을 가지고 오프라인 방식에서 적합한 스케줄을 찾는 알고리즘을 제시하고, 이를 바탕으로 조건 $U \le n$이 주기적인 태스크 집합을 스케줄링하기 위한 충분조건임을 보여준다.

2.기본 알고리즘.

본 연구에서는 태스크의 특성이 미리 주어졌을 때, 오프라인 방식으로 경성실시간 다중프로세서 스케줄링 문제에 대하여 알아보는데, 태스크들은 서로 독립적이며 일정한 요청간격을 가지는 주기적인 태스크들로 구성되어 있고 주기가 끝나는 시점에서 새로운 수행에 대한 요청이 들어온다. 이러한 특성을 가지고 있는 모든 주기적인 태스크들의 수행시간이 비 정수시간으로 나누어 질 수 있다는 가정 하에서 주어진 알고리즘을 이용하여 적합한 스케줄을 찾는다. 주어진 태스크 집합 S는 m개의 태스크 T_i , $t=1,2,3,\ldots,m$,로 이루어지고 T_i의 수행시간은 C_i, 마감시간(주기)은 D_i라 가정한다. 또한 모든 태스크들은 동일한 시점에서 첫 번째 주기를 시작한다고 가정하며, 태스크 집합 S에 대한 활용도는 $U = \sum_{i=1}^{m} C_i/D_i$로 정의된다. 이와 같은 조건하에서 적합한 스케줄은 다음 두 조건을 반드시 만족해야 한다.[1]

(A1) 한 프로세서에서 많아야 하나의 태스크만 수행할 수 있다.
(A2) 어떠한 태스크도 동일시점에 동시에 2개이상의 프로세서에서 같이 수행 될 수 없다.

본 알고리즘은 제약조건(A1),(A2)을 만족하기 위해 단위시간당 수행할 수 있는 양인 C_i/D_i만 큼의 시간을 각 태스크에 할당하여 스케줄을 구한다. 우선 스케줄을 구하기 전에 적용 가능한 태스크 집단을 얻을 필요가 있다. 따라서 그림1에서 보여주는 바와 같이 적합성 검사(Feasibility Test)를 통해서 적용 가능한 태스크 집단을 얻는다.

Feasibility Test
if the sum of C_i/D_i for all tasks T_i is not greater than n
    the given set of tasks is feasible
else
    the given set of tasks is not feasible

[그림 1] 적합성 검사

기본 알고리즘에서는 임의의 작은 시간단위로 태스크들에게 프로세서 시간을 할당할 수 있다는 가정 하에 한 단위 시간인 구간 [0, 1] 사이에서 각 태스크 T_i에게 C_i/D_i만큼의 할당량을 부여한다.

만일 현재 주어진 프로세서에서 T_i를 위한 할당량이 구간 내에서 충분치 못하다면 다음 프로세서에서 나머지 시간을 할당받게 된다. 기본 적으로 모든 태스크는 $C_i \le D_i$와 같은 조건을 만족하므로 모

```
Basic Scheduling Algorithm For Interval [0,1]
Notations:
 U_i : the utilization of task T_i , C_i/D_i
 time : current time
 i : current task number
 j : current processor number
 l_i : length of the slot allocated for task T_i
Begin
for each task T_i , l_i ← U_i
 i ← 1;  j ← 1;
 u ← l_i ;  time ← 0;
while( i≤m and j≤n) {
  if( u≤(1−t)) then {
      allocate interval [time, time+u] of P_j to T_i
      time ← time+u ;
      i ← i+1 ;  u ← l_i ;
  } else {
      allocate interval [time, 1] of P_j to T_i
      l_i ← l_i−(1−t) ;
      j ← j+1 ;  time ← 0;
  }
}
End
```

[그림 2] 기본 알고리즘

자라는 시간을 다음 프로세서에서 할당하게 되면 제한조건(2)을 만족하는 스케줄을 얻을 수 있을 것이다. 그림 2는 이러한 과정을 보여주고 있다.

구간 [0, 1]에 대하여 제시된 기본 알고리즘으로 각 태스크, T_i에게 필요한 시간 C_i/D_i만큼을 할당할 수 있다면, 구간 [0, 1]에 적용된 스케줄을 전 시간구간에 반복적으로 적용하면 전체 스케줄을 얻을 수 잇다.

보조정리 1. 제안된 기본 스케줄링 알고리즘은 구간 [0,1]에서 태스크 집합 S를 위한 적합한 스케줄을 구성한다.

증명. 제안된 알고리즘은 각 태스크마다 수행시간을 구간 내에서 나누어서 시간을 할당하였다. 따라서 제한조건(A1)을 만족할 수 있다. 또한 알고리즘에서 보듯이 태스크의 수행시간이 남아 있는 시간보다 클 때 태스크의 잔여시간은 다음 프로세서에서 수행됨을 보여준다. 따라서 제안조건(A2)도 만족됨을 보여준다. 그러므로 제시된 기본 스케줄링 알고리즘은 적합한 스케줄을 구할 수 있다.

정리 1. 임의의 작은 시간 단위로 프로세서 할당이 가능하다면 조건 $U \le n$은 주어진 태스크집합을 스케줄하기 위한 필요충분조건 이다.

증명. k번째 태스크가 구간 [0,1]내에서 스케줄이 가능하지 않다면 다음과 같이 볼 수 있다. 1번째에서 $k-1$번째 프로세서에는 구간 [0,1]사이에 유용한 시간이 없는 상태이고 k번째 프로세서에서 오버플로가 발생함을 의미한다. 이는 식(1)과 같이 볼 수 있다.

$$\sum_{i=1}^{k} C_i/D_i > n \quad - (1)$$

그러나 모든 태스크들은 이미 적합도 검사를 통해 나온 필요조건 $U \le n$을 만족하는 집합이므로 (1)번식과 같이 나타나는 일은 없을 것이다. 즉 조건 $U \le n$는 충분조건이 됨을 알 수 있다.

3. 다중프로세서 스케줄링 알고리즘

기본 스케줄링 알고리즘은 비 정수시간에 태스크들이 선점될 수 있기 때문에 할당된 시간이 정수단위가 되도록 조정할 필요가 있다. 태스크가 정수시간에 선점 된다는 제약은 하드웨어 요구사항에 의해 제약을 받는다.[1] 따라서 기본 스케줄링 자체로는 적용할 수 없고 이를 바탕으로 하여 다중 프로세서 스케줄링 알고리즘을 구성한다.

그림3의 과정은 태스크들이 정수단위로 선점 될 수 있도록 시간을 할당한 것을 보여준다.

한 구간 내에서 모든 태스크들에게 할당하는 시간은 기본적으로 $\lfloor \delta_i + U_i * b \rfloor$ 만큼을 할당을 한다. 즉 정수부분은 기본적으로 취한다. 나머지 소수점 이하 부분은 그림3의 두 번째 while문에서 보듯이 현재 모든 태스크의 정수부분을 할당하고 나서 남는 여유시간이 있을 경우 나머지 소수점 이하의 부분을 각 태스크에 할당을 하게 된다.

각 태스크는 l_i만큼의 시간을 할당받고 아직 구간 내에 태스크에 할당 가능한 시간이 남아있을 경우에 나머지 소수점 아래 부분을 1만큼의 시간으로 할당을 받는다. 그러나 모든 태스크가 소수점 이하의 값을 1만큼 씩 할당받아서는 안 된다. 왜냐하면 이전 태스크의 소수점 이하부분을 모두 1로 할당하면 결국엔 남는 시간이 없게 되어서 뒤에 올 태스크가 소수점 이하부분을 할당받지 못함으로써 자신의 주기를 넘길 수 있다. 이러한 점을 고려하여 변수 δ_i를 사용하였다.

그림4-(b)는 제시된 알고리즘을 이용하여 두 개의 프로세서로 구성된 시스템에서 그림 4-(a)에서 제시된 태스크 집합의 스케줄을 보여주고 있다. 각 태스크는 모든 수행시간을 자신의 주기 내에 수행하였고, 또한 제한 조건(A2)에서 나타난 조건을 만족함을 볼 수 있다.

보조정리 2. 제시된 다중프로세서 스케줄링 알고리즘은 실시간 다중프로세서 시스템에서 주기적인 태스크들을 위한 적합한 스케줄을 구성한다.

증명. 제안된 알고리즘은 그림3의 알고리즘에서 보면 $\sum l_i \le b$을 만족함을 볼 수 있다. 알고리즘의 두 번째 while 문에서 보면 각 태스크는 b보다 크지 않는 범위 내에서 시간을 할당받고 많아야 b만큼 시간을 할당받는다. 즉 한 프로세서에서 두 개 이상의 태스크가 동시 수행되는 일은 없다. 따라서 제한조건(A1)을 만족할 수 있다. 그리고 이 알고리즘은 기본알고리즘을 블록크기를 기준으로 확장을 해놓은 모습을 보여주고 있다. 따라서 보조정리1에서 본바와 같이 제한조건(A2)도 만족하므로 본 알고리즘은 적합한 스케줄을 얻을 수 있다.

보조정리 3. 제시된 알고리즘에서 모든 태스크, T_i에 대해 항상 조건 $-1 < \delta_i < 1$을 만족한다.

증명. 알고리즘에서 이전의 δ_i를 δ_i^0라고 표현하면 시간 0에서는 $\delta_i^0 = 0$이고 그림3에서 제시된 for문에서는 δ_i값은 다음과 같이 나타낼 수 있다.

$$\delta_i = \delta_i^0 + U_i * b - \max(\lfloor \delta_i^0 + U_i * b \rfloor, 0)$$

경우 1. $\delta_i^0 + U_i * b \ge 0$일 때,

$\delta_i = \delta_i^0 + U_i * b - \lfloor \delta_i^0 + U_i * b \rfloor$ 이고 이는 소수점 이하 부분을 의미하므로 항상 $0 \le \delta_i < 1$이다.

경우 2. $\delta_i^0 + U_i * b < 0$일 때, $\delta_i = \delta_i^0 + U_i * b < 0$이다. 또한 $\delta_i^0 > -1$이므로, $\delta_i = \delta_i^0 + U_i * b > \delta_i^0$ 이다. 따

```
Multiprocessor Scheduling Algorithm
Notations:
 l_i : length of the slot allocated for task T_i
 δ_i : the amount of time to be allocated additionally
for T_i
 time : current time
 i : current task number
 b : length of the current interval
Begin
 time ← 0 ;
 lcm ← the LCM of all periods of tasks
 S ← the set of deadlines of all instance of tasks from
0 to LCM sorted in increasing order
for each task T_i , δ_i ← 0 ;
while(time < lcm) {
   d ← extract the first deadline from S
  /* the length of the current interval*/
   b ← d − time;
  /* the total processor time for the current interval*/
   a ← b*n ;
  for each task T_i {
        l_i ← max( ⌊ δ_i + U_i*b ⌋ ,0);
        δ_i ← δ_i + U_i*b − l_i ;
        a ← a − l_i ;
  }
  while( a>0) {
      if( δ_i>0 and l_i<b ) {
           l_i ← l_i + 1 ;
           δ_i ← δ_i − 1 ;
           a ← a − 1 ;
      }
  }
  apply the basic scheduling algorithm for the interval
  [time, time+b] using the calculated slot length l_i for
  each task T_i ;
  time ← time+b ;
}
End
```

[그림 3] 다중 프로세서 스케줄링 알고리즘

라서 $-1<\delta_i<0$이다. while문에서 δ_i가 조정이 될 경우는 $\delta_i>0$일 때 한해서 1만큼 줄어들기 때문에 조건 $-1<\delta_i<1$은 그대로 만족된다.

위에서 본 두 가지 경우는 한 태스크가 자신에게 할당받을 시간을 더 많이 받지도 적게 받지도 않으면서 자신의 주기내에서 수행시간을 종료할 수 있음을 나타낸다.

정리 2. 조건 $U\leq n$은 n개의 동일한 프로세서 상에서 태스크 집합 S의 적합한 스케줄을 위해 필요·충분조건이다.

증명. 기본 알고리즘의 정보를 이용한 다중프로세서 스케줄링 알고리즘은 보조정리2와 보조정리3에서 살펴본 바와 같이 필요조건 $U\leq n$을 만족하는 모든 주기적인 태스크 집합은 스케줄 가능하고 전체를 살펴보면 다음과 같이 나타낼 수 있다. 특히 보조정리3에서는 변수 δ_i의사용에 대한 경우를 들어 설명했다. 모든 태스크는 자신의 수행시간을 정수 단위로 구성되면서 주기 내에 완료하기 위해서 δ_i사용으로 보였다. 이렇게 함으로써 모든 태스크가 소수점이하에 대한 프로세서

태스크	수행시간	주기
태스크1	2	4
태스크2	4	6
태스크3	3	4

(a) 태스크 집합

시간	1	2	3	4	5	6	7	8	9	10	11	12
프로세서1	1	1	2	2	1	2	1	2	1	1	2	2
프로세서2	2	3	3	3	3	3	2	3	3	3	3	

(b) 알고리즘을 적용한 전체 스케줄

[그림 4] 다중 프로세서 스케줄링 알고리즘

시간 할당을 받을 수 있었다.

지금까지 살펴본 바와 같이 제시된 알고리즘을 반복 적용시켰을 때 조건을 만족하는 적합한 스케줄을 얻을 수 있었다.

따라서 조건 $U\leq n$은 n개의 동일한 프로세서 상에서 태스크 집합 S의 적합한 스케줄을 위해 필요·충분조건을 만족할 수 있다.

4. 결 론

본 논문에서 제시된 알고리즘은 주어진 가정 하에 실시간 다중프로세서 상에서 주기적인 태스크들을 위한 스케줄링 방법을 보여 주었다.

[1]에서 제시된 알고리즘은 최적인 스케줄을 제공하고 있으나, 주어진 태스크 집합이 특정 제한조건을 만족하지 않으면 조건 $U\leq n$을 만족하는 모든 태스크 집합에 대한 적합한 스케줄을 구할 수 있을지는 보장할 수 없다. 또한 [4]에서 제시된 알고리즘은 한 태스크 집합에 대해 GCD값을 이용하여, 그 단위로 스케줄을 구성하는데, 만일 GCD값이 1이라면, 너무 많은 스케줄링 오버헤드가 발생하는 문제점을 가진다.

위에서 살펴본 바와 같이 기본 스케줄링 알고리즘과 다중프로세서 알고리즘에서 제한조건(A1),(A2)을 모두 만족시키는 것을 보았고, 모든 태스크들의 마감시간을 보장할 수 있음을 보였다.

따라서, 제시된 두 가지 알고리즘은 모두 적합한 스케줄을 구성할 수 있었고, 조건 $U\leq n$이 다중프로세서 상에서 주기적인 태스크들에 대한 필요충분조건임을 또한 보여주었다.

5. 참 고 문 헌

[1] Ashok Khemka and R. K. Shyamasundar, "An Optimal Multiprocessor Real-Time Scheduling Algorithm," Journal of Parallel and Distributed Computing 43, 37-45(1997)

[2] M. L. Dertouzos, A. K. Mok, "Multiprocessor On-Line Scheduling of Hard-Real Time tasks," IEEE Transactions on Software Engineering, Vol. 15, No. 12, December 1989.

[3] Liu, C. L. and Layland, J. W. "Scheduling algorithms for multiprogramming in a hard-real time environment." J. Assoc. Comput. Mach 20,(Jan, 1973)

[4] Tse Lee and Albert Mo Kim Cheng, "Multiprocessor Scheduling of Hard-Real-Time Periodic tasks with task Migration Constraints", First International Workshop on Real-Time Computing Systems and Applications, Dec. 1994