논문 2011-1-24

개선된 도달 함수를 이용한 동적 Pfair 스케줄링

Dynamic Pfair Scheduling Using an Improved Reach Function

박현선*, 김인국*

Hyun-Sun Park, In-Guk Kim

요 약 Pfair 알고리즘은 다중 프로세서 환경에서 경성 실시간 태스크 집합을 스케줄링 하는 최적 알고리즘인데, 고정된 퀀텀 크기를 기반으로 한다. 최근 mode change 환경에서 도달 함수를 이용하여 최적 퀀텀을 동적으로 결정하는 방법들이 제안되었는데, 이 방법들에서는 최적 퀀텀을 구하기 위해 순차 탐색을 해야 되는 경우가 많이 발생하였다. 본 논문에서는 개선된 도달 함수를 이용하여 최적 퀀텀을 보다 빠르게 구할 수 있는 새로운 방법을 제안하였다.

Abstract The Pfair scheduling algorithm, which is an optimal algorithm in the hard real-time multiprocessor environments, is based on the fixed quantum size. Recently, several methods that can determine the optimal quantum dynamically are developed in the mode change environments. These methods are based on the reach function and in many cases, we have to do the sequential search to find the optimal quantum. In this paper, we propose a new scheduling method, based on the improved reach function, that can determine the optimal quantum more quickly.

Key Words : Real-time Scheduling, Multiprocessor Scheduling, Pfair Algorithm

Ⅰ. 서 론

실시간 시스템에서 각 태스크의 종료시한(deadline)을 지키는 일은 최우선의 목표이며 따라서 종료시한에 맞추어 태스크들을 스케줄링 하는 것은 매우 중요한 일이다. 단일 프로세서 환경에서는 Liu와 Layland가 제안한 Rate-Monotonic Scheduling(RMS) 알고리즘과 Earliest Deadline Scheduling(EDS) 알고리즘이최적의 알고리즘으로 증명되었다[3]. 그러나 RMS 알고리즘이나 EDS 알고리즘은 다중프로세서 환경에서는 최적의 알고리즘이 되지 못한다.

Baruah 등은 다중 프로세서 환경에서 주기적 경성 실시간 태스크들에 대한 최적의 스케줄링 알고리즘인 Pfair 스케줄링 알고리즘[10]과 이것을 개선한 PD 알고리즘[9]을 제안하였다.

이들은 다중 프로세서 환경에서 프로세서의 수가 M 일 때 태스크 집합이 스케줄링 가능하기 위한 필요충분 조건이 다음과 같음을 증명하였다. 식 (1)에서 $T.e$와 $T.p$는 태스크 집합 τ에 속한 태스크 T 의 실행 요구 시간과 주기를 각각 나타낸다.

$$\sum_{T \in \tau} \frac{T.e}{T.p} \leq M \quad (1)$$

Baruah 등이 Pfair 스케줄링 알고리즘을 제안한 이후 Pfair를 기반으로 하는 다수의 알고리즘들이 제안되었는데, 이러한 퀀텀 기반의 스케줄링 알고리즘들은 각 태스크의 주기 및 실행 요구 시간이 퀀텀 크기의 배수가 됨을 가정하고 있다[4-8].

최근 mode change 환경에서 최적의 퀀텀 크기를 결정하는 스케줄링 알고리즘이 제안된 바 있다[1-2]. 제안된 방법들에서는 도달 함수를 정의하고 이를 이용하여 최적의 퀀텀 크기를 구할 수 있음을 보였다. 그러나 최적의 퀀텀

*정회원, 단국대학교 컴퓨터과학과
접수일자: 2010. 12. 17. 수정일자: 2011.1.18
게재확정일자: 2011.2.11

크기를 구하는데 시간이 많이 소요되는 경우가 빈번히 발생하였다. 이에 본 논문에서는 새로운 도달 함수를 정의하고 이를 이용하여 최적의 퀀텀 크기를 보다 빠르게 구하는 방법을 제안하고자 한다.

본 논문의 구성은 다음과 같다. Ⅰ장에서는 서론 및 관련연구를 기술하였고, Ⅱ장에서는 스케줄링 환경 및 태스크 모델이 기술되며, Ⅲ장에서는 최적의 퀀텀 크기를 결정하기 위한 개선된 방법이 제안되고 증명된다. 끝으로 Ⅳ장에서는 결론이 기술된다.

Ⅱ. 스케줄링 모델

본 논문에서 제안하는 스케줄링 방법은 전역(global) 스케줄링을 기반으로 하고 있다. 따라서 태스크들은 모든 프로세서들이 공동으로 사용하는 단일 준비 큐에 저장된다. 스케줄링 시점이 되면 단일 준비 큐에서 가장 높은 우선순위 태스크가 선택되어 실행된다. 따라서 태스크들은 프로세서 간 이동이 허용된다.

주어진 태스크 집합은 구 스케줄(old schedule)의 대주기(major period) 끝에서 새로운 태스크 집합으로 변경됨을 가정한다. 대주기는 태스크들의 주기의 최소공배수로서 모든 태스크들은 대주기의 시작점에서 동시에 실행준비가 됨을 가정한다.

태스크들은 실행 요구시간 및 주기의 변동 가능성을 가진다. 즉 태스크들은 대주기의 끝에서 mode change된 후 실행요구 시간이 이전보다 감소되거나 증가될 수 있으며 이에 따라 주기도 이전보다 감소되거나 증가될 수 있다.

각 태스크 T는 주기 p와 실행요구 시간 e를 속성으로 갖는다. e_n과 p_n은 주어진 퀀텀 크기 Q에 따라서 정의되는 실행요구 시간과 주기를 각각 나타낸다. $Q_{\min}$은 시스템에서 표현할 수 있는 최소 퀀텀 크기로서 단위 시간 1을 의미한다. 또한 $p_{\max}$는 태스크들의 주기의 최대값을 나타낸다.

이때, 주어진 퀀텀 크기 Q의 변경에 따라 태스크의 실행요구 시간과 주기는 증가되거나 감소될 수 있으므로 변경된 실행요구 시간 e_n과 주기 p_n은 다음과 같이 표시될 수 있다.

$$e_n = \left\lceil \frac{e}{Q} \right\rceil \text{ or } \left\lfloor \frac{e}{Q} \right\rfloor \tag{2}$$

$$p_n = \begin{cases} \left\lfloor \frac{p}{Q} \right\rfloor \text{ or } \left\lceil \frac{p}{Q} \right\rceil & (\text{if} \quad Q < p) \\ 1 & (\text{if} \quad Q \geq p) \end{cases} \tag{3}$$

따라서 w를 $Q_{\min}$에 의해 계산된 프로세서 이용률(utilization)이라 하면, Q에 의해 변경된 태스크의 이용률 w'은 다음과 같이 표시된다.

$$w' = \frac{e_n}{p_n} = \begin{cases} \dfrac{\left\lceil \frac{e}{Q} \right\rceil}{\left\lfloor \frac{p}{Q} \right\rfloor} \text{ or } \dfrac{\left\lceil \frac{e}{Q} \right\rceil}{\left\lceil \frac{p}{Q} \right\rceil} \text{ or } \dfrac{\left\lfloor \frac{e}{Q} \right\rfloor}{\left\lfloor \frac{p}{Q} \right\rfloor} \text{ or } \dfrac{\left\lfloor \frac{e}{Q} \right\rfloor}{\left\lceil \frac{p}{Q} \right\rceil} & (\text{if } Q < p) \\ 1 & (\text{if } Q \geq p) \end{cases} \tag{4}$$

Ⅲ. 최적 퀀텀 크기 결정

1. 도달점

퀀텀 크기가 증가할 때 태스크의 이용률은 단조 증가하지는 않는다. 그러나 퀀텀 크기 Q를 계속 증가시키면 어느 시점부터는 태스크의 이용률이 계속 1의 값을 유지하게 되는데, 이렇게 태스크의 이용률이 계속 1이 되는 최초의 퀀텀 크기 Q를 실제 도달점 RRP라 하고, 실제 도달점을 포함한 이후의 Q값들을 도달 구간이라 하며 도달 구간 내의 Q값들을 도달점이라 부른다.

태스크의 실제 도달점을 구하기 위해서는 $p_{\max}-1$로부터 Q값을 1만큼씩 감소시키면서 해당 태스크의 이용률이 1보다 작은 값이 나오는 지를 반복적으로 찾으면 되는데, 이렇게 태스크의 실제 도달점을 구하는 것은 $O(p_{\max})$의 시간복잡도를 갖는다. $p_{\max}$의 값이 $n!$이 된다면 이 방법은 지수 시간복잡도를 갖게 된다.

참고문헌 [2]에서는 주어진 태스크들의 실행요구 시간이 증가하지 않고 주기도 증가하지 않는다고 가정한 경우의 도달 함수를 정의하고 이를 이용하여 도달점을 구

할 수 있음을 증명하였다. 이 경우 어떤 태스크의 p와 e에 대해 $w' = \frac{\lfloor \frac{e}{Q} \rfloor}{\lfloor \frac{p}{Q} \rfloor}$ 이다. 이때 $\lfloor \frac{e}{Q} \rfloor$ 나 $\lfloor \frac{p}{Q} \rfloor$ 의 최소값은 1이라 정의한다.

[정리 1]

실행요구 시간이 증가하지 않고 주기도 증가하지 않는 태스크의 도달 함수 $Reach(p,\ e)$는 다음과 같이 정의한다.

$$Reach(p,\ e) = p \tag{5}$$

이때, $Q \geq Reach(p,\ e)$이면

임의의 p와 e에 대해 $\frac{\lfloor \frac{e}{Q} \rfloor}{\lfloor \frac{p}{Q} \rfloor} \geq 1$이다.

본 논문에서는 [정리 1]의 내용을 개선하여 태스크의 실제 도달점을 구하는 도달함수를 정의하고 이를 이용하여 최적의 퀀텀 크기를 보다 빠르게 구할 수 있는 방법을 다음과 같이 제안한다.

[정리 2]

실행요구 시간이 증가하지 않고 주기도 증가하지 않는 태스크의 도달 함수 $Reach(p,\ e)$는 다음과 같이 정의한다.

$$Reach(p,\ e) = \left\lfloor \frac{p}{2} \right\rfloor + 1 \tag{6}$$

이때, $Q \geq Reach(p,\ e)$이면

임의의 p와 e에 대해 $w' = \frac{\lfloor \frac{e}{Q} \rfloor}{\lfloor \frac{p}{Q} \rfloor} \geq 1$이다.

[증명]

위에서 $1 \leq e \leq p$이므로 $e = 1$일 때 $\frac{\lfloor \frac{e}{Q} \rfloor}{\lfloor \frac{p}{Q} \rfloor} \geq 1$ 이면 나머지 모든 값에 대해서도 식이 성립한다. 먼저 $Q \geq 1$이므로 $e = 1$이면 $\lfloor \frac{e}{Q} \rfloor = 1$이다. 따라서 위 식이 성립함을 증명하는 것은 $Q = \left\lfloor \frac{p}{2} \right\rfloor + 1$일 때 $w' = \frac{\left\lfloor \frac{1}{\lfloor \frac{p}{2} \rfloor + 1} \right\rfloor}{\left\lfloor \frac{p}{\lfloor \frac{p}{2} \rfloor + 1} \right\rfloor} \geq 1$이 성립하는 것을 보이는 것으로 충분하다. 그런데 $\left\lfloor \frac{1}{\lfloor \frac{p}{2} \rfloor + 1} \right\rfloor = 1$ 이므로 $\left\lfloor \frac{p}{\lfloor \frac{p}{2} \rfloor + 1} \right\rfloor = 1$임을 보이면 된다.

1보다 크거나 같은 양의 정수 p에 대해,

$$\frac{p}{2} < \frac{p+1}{2}$$
$$\Leftrightarrow \ \frac{p}{2} < \left\lceil \frac{p+1}{2} \right\rceil$$
$$\Leftrightarrow \ p < 2\left\lceil \frac{p+1}{2} \right\rceil$$
$$\Leftrightarrow \ \frac{p}{\left\lceil \frac{p+1}{2} \right\rceil} < 2$$
$$\Leftrightarrow \ \left\lfloor \frac{p}{\left\lceil \frac{p+1}{2} \right\rceil} \right\rfloor = 1$$
$$\Leftrightarrow \ \left\lfloor \frac{p}{\lfloor \frac{p}{2} \rfloor + 1} \right\rfloor = 1$$

따라서 $w' = \frac{1}{1} \geq 1$이다. ■

위 증명의 과정에서 $\left\lceil \frac{p+1}{2} \right\rceil = \left\lfloor \frac{p}{2} \right\rfloor + 1$이 성립함은 참고문헌 [1]에서 증명하였다.

주어진 태스크 집합은 τ이고 사용 가능한 프로세서의 수는 M이라 가정하자. M개의 태스크들이 이미 그들의 도달 구간에 있다면, 그들의 이용률의 합은 M이 되고, 나머지 태스크들의 이용률의 합은 0보다 크므로 스케줄링은 불가능하게 된다. 따라서 최적의 Q값은 적어도 $M-1$번째 태스크 또는 그 이전에 있는 태스크의 도달 구간에서 구할 수 있게 된다. 이를 위해 각 태스크의 도달점을 오름차순으로 저장하는 도달 순위 리스트 $Rank[n]$을 생성한다. 또한 실제 도달점보다 작은 값으로부터 Q값을 찾아야 하므로 도달 순위 리스트의 각 원소에는 해당 태스크의 $Reach(p,e)-1$의 값이 저장된다.

참고문헌 [1]에서 제안된 알고리즘(그림 1)에서는 먼저 프로세서의 수 M에 의거하여 도달 순위 리스트에서 $Rank[M-1]$을 찾는다. 다음으로는 퀀텀 크기가 $Rank[M-1]$인 경우에 태스크들의 이용률의 합을 계산하는데 그림 1.내의 Calculate_U 함수가 이 일을 수행한다. 계산된 이용률의 합이 M보다 작거나 같다면 주어진 태스크 집합은 Pfair 스케줄링 알고리즘에 의해 스케줄링 가능한 것이고 이때 최적의 퀀텀 크기는 $Rank[M-1]$이 된다.

```
int FindQ() {
    int Q = 1;
    int base, i;
    for(i=M-1; i>=0; i--) {
        if(calculate_U(ts, Rank[i]) <= M) {
            Q = Rank[i];
            base = i;
            break;
        }
    }
    if(calculate_U(ts, Q+1) <= M) {
        for(i=Rank[base+1]; i>0; i--){
            if(calculate_U(ts, i) <= M) {
            Q = i;
                break;
            }
        }
    }
    return Q;
}
```

그림 1. 최적 퀀텀 크기를 결정하는 알고리즘
Fig. 1. Algorithm that determines the optimal quantum size

이용률의 합이 M보다 크다면 도달 순위 리스트에서 $Rank[M-2]$를 찾고, 퀀텀 크기가 $Rank[M-2]$인 경우에 태스크들의 이용률의 합을 계산한다. 계산된 값이 M보다 작거나 같다면 주어진 태스크 집합은 스케줄링 가능한 것이고, 이 경우 최적 퀀텀 크기는 $Rank[M-2]$가 된다.

이러한 과정은 태스크들의 이용률의 합이 M보다 작거나 같게 될 때까지 반복 실행되는데 경우에 따라서는 계산된 Q가 최적이 아닐 수도 있다. Q가 최적이 아니라는 것은 퀀텀 크기가 $Q+1$일 때 주어진 집합이 스케줄링 가능한지의 여부를 보고 판단할 수 있다. 계산된 Q가 최적이 아닌 경우에는 바로 이전 단계의 $Rank$ 값으로부터 시작하여 퀀텀의 크기를 1만큼씩 감소시키면서 Q를 찾으면 된다.

2. 예제

$\tau=\left\{T_1=\frac{7}{41}, T_2=\frac{2}{4}, T_3=\frac{29}{48}, T_4=\frac{8}{39}, T_5=\frac{15}{22}\right\}$

가 주어진 태스크 집합이고 프로세서의 수가 M이라 가정하자. τ는 다섯 개 태스크들의 집합이며 T_1은 실행요구 시간이 7이고 주기가 41인 태스크이다.

이제 참고문헌 [2]에서 제안된 방법(이하 방법 1)과 본 논문에서 제안된 방법(이하 방법 2)을 이용하여 주어진 태스크 집합의 스케줄링 가능성을 검사하고 최적의 퀀텀 크기를 구하는 알고리즘을 비교 분석해 본다.

가. 방법 1

이 경우에 주어진 태스크 집합 τ에 대한 도달 순위 리스트는 표 1.과 같다.

표 1. τ에 대한 도달 순위 리스트(방법 1)
Table 1. Reach Rank List for τ(Method 1)

i	0	1	2	3	4
Rank[i]	3	21	38	40	47
Task 번호	T_2	T_5	T_4	T_1	T_3

$M=4$이면 $Rank[3]=40$이고 이 경우 이용률의 합은 5이어서 주어진 태스크 집합은 Pfair 스케줄링 알고리즘에 의해 스케줄링이 불가능하다. 계속해서 $Rank[2]=38$인 경우와 $Rank[1]=21$인 경우도 이용률의 합이 4를 넘으므로 스케줄링이 불가능하다. $Rank[0]=3$이고 이 경우 이용률의 합은 2.5846이어서 Pfair 스케줄링이 가능하다. 이 경우에 퀀텀 크기는 3이다.

여기서 퀀텀 크기를 1만큼 증가시켜 이용률을 계산하면 2.5055이어서 역시 스케줄링이 가능하다. 따라서 3은 최적의 퀀텀 크기가 아니다. 최적 퀀텀 크기는 $Rank[1]$로부터 시작하여 퀀텀의 크기를 1만큼씩 감소시키면서 구하게 된다. 이렇게 구해진 최적 퀀텀 크기는 20이다.

$M=3$이면 $Rank[2]=38$인 경우와 $Rank[1]=21$인 경우가 모두 이용률의 합이 4를 넘어서 스케줄링이 불가능하다. $Rank[0]=3$이고 이 경우 이용률의 합은 2.5846이어서 스케줄링이 가능하다. 이때 퀀텀 크기를 1만큼 증가시키면 $U=2.5055$이 되어 스케줄링이 가능하므로 퀀텀 크기 3은 최적이 아니다. 이 경우 최적 퀀텀

크기는 $Rank[1]$로부터 시작하여 퀀텀의 크기를 1만큼씩 감소시키면서 구하면 된다. 이렇게 구해진 최적 퀀텀 크기는 11이다.

종합하여 보면 방법 1은 도달 순위 리스트의 rank값으로부터 직접 최적 퀀텀 크기를 구하지는 못하고 있음을 볼 수 있다. 최적 퀀텀 크기는 rank값으로부터 순차적으로 탐색한 끝에 구할 수 있었다.

나. 방법 2

이 경우에 주어진 태스크 집합 τ에 대한 도달 순위 리스트는 표 2.와 같다.

표 2. τ에 대한 도달 순위 리스트(방법 2)
Table 2. Reach Rank List for τ(Method 2)

i	0	1	2	3	4
Rank[i]	2	11	19	20	24
Task 번호	T_2	T_5	T_4	T_1	T_3

$M=4$이면 $Rank[3]=20$이고 이 경우 이용률의 합은 4이어서 주어진 태스크 집합은 Pfair 스케줄링 알고리즘에 의해 스케줄링이 가능하다. 이 경우에 퀀텀 크기를 1만큼 증가시키면 $U=4.5$가 되어 스케줄링이 불가능하게 된다. 따라서 20은 최적의 퀀텀 크기이다.

$M=3$이면 $Rank[2]=19$이고 이 경우 이용률의 합 U는 4이어서 스케줄링이 불가능하다. 계속해서 $Rank[1]=11$이고 이 경우 이용률의 합은 2.5833이어서 스케줄링이 가능하게 된다. 이때 퀀텀 크기를 1만큼 증가시키면 $U=3.1666$이 되어 스케줄링이 불가능하게 된다. 따라서 이 경우의 최적 퀀텀 크기는 11이다.

위에서 보듯이 방법 2는 방법 1의 경우에 비해 보다 빠르게 최적 퀀텀 크기를 구할 수 있다. 그러나 방법 2가 항상 rank 값으로부터 직접 최적의 퀀텀 크기를 구할 수 있는 것은 아니다.

Ⅳ. 결 론

일반적으로 스케줄링 가능한 퀀텀의 크기가 증가하게 되면 문맥 교환을 하기 위한 시간이나 캐시 재적재 시간 등이 감소하게 되어 시스템 성능이 향상되는 요인이 된다. 따라서 Pfair 스케줄링에서 최적의 퀀텀 크기는 스케줄링 가능한 최대의 퀀텀 크기이다.

이전 방법에서 제안된 도달 함수는 태스크의 도달점을 구할 수는 있었으나, 구해진 도달점이 항상 실제 도달점인 것은 아니었다. 이로 인해 이전 방법에서는 최적의 퀀텀을 구하기 위해 이용률 리스트를 순차 탐색해야 하는 경우가 많이 발생하였다

본 논문에서는 개선된 도달 함수를 제안하고 이를 이용하여 태스크의 실제 도달점을 구할 수 있음을 증명하였다. 이렇게 구해진 실제 도달점은 최적의 퀀텀 크기를 보다 빠르게 구하는데 이용될 수 있다.

참 고 문 헌

[1] 차성덕, 김인국, “Mode Change 환경에 적합한 동적 퀀텀 크기 스케줄링”, 콘텐츠학회논문집, 제6권, 제9호, pp.28-41, 2006.

[2] 김남진, 김인국, “개선된 동적 퀀텀 크기 Pfair 스케줄링의 구현”, 한국산학기술학회논문지, 제10권, 제10호, pp.2760-2765, 2009.

[3] C. L. Liu and J. W. Layland, "Scheduling Algorithm for Multiprogramming in a hard real-time environment," JACM, Vol.20. pp.46-61, 1973.

[4] D. Zhu, D. Mosse, and R. Melhem, "Multiple-Resource Periodic Scheduling Problem: how much fairness is necessary?" Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE, pp.142-151, Dec. 2003.

[5] J. Anderson and A. Srinivasan, “A New Look at Pfair priorities”, Technical report, Dept of Computer Science, Univ. of North Carolina, 1999.

[6] J. Anderson and A. Srinivasan, "Early-release fair scheduling," Proceedings of the 12th Euromicro Conference on Real-time Systems, pp.35-43, June. 2000.

[7] J. Anderson and A. Srinivasan, "Pfair Scheduling: Beyond Periodic Task Systems," Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications, pp.297-306, Dec. 2000.

[8] J. Anderson, A. Block, and A. Srinivasan, "Quick-release Fair Scheduling," Proceedings of the 24th IEEE Real-time Systems Symposium, pp.130-141, Dec. 2003.

[9] S. Baruah, J. Gehrke, and C. G. Plaxton. "Fast Scheduling of Periodic Tasks on Multiple Resource," Proceedings of the 9th International Parallel Processing Symposium, pp.280-288, Apr. 1995.

[10] S. Baruah, N. Cohen, C. G. Plaxton, and D. Varvel, "Proportionate Progress: A notion of fairness in resource allocation," Algorithmica, Vol.15, pp.600-625, 1996.

※ 이 연구는 2010년도 단국대학교 대학연구비의 지원으로 연구되었음.

저자 소개

박 현 선(정회원)

- 2000년 2월 : 단국대학교 전자계산학과 (이학사)
- 2003년 2월 : 단국대학교 전자계산학과 (이학석사)
- 2006년 2월 : 단국대학교 전자계산학과 (박사수료)

<주관심분야 :운영체제, 실시간시스템, 임베디드시스템>

김 인 국(정회원)

- 1982년 2월 : 단국대학교(학사)
- 1985년 5월 : 미국 에모리대학교 (석사)
- 1995년 8월 : 아주대학교(박사)
- 1986년~현재 : 단국대학교 컴퓨터과학과 교수
- 2001년~2003년 : 미국 뉴멕시코 공과대학 방문교수

<주관심분야 : 운영체제, 실시간시스템, 임베디드시스템>