Journal of Digital Contents Society Vol. 8 No. 3 Sept. 2007(pp. 279-288)

Mode Change 환경을 위한 개선된 동적 퀀텀 크기 Pfair 스케줄링

차성덕*, 김인국**

요 약

최근 다중프로세서 실시간 시스템 환경에서 최적인 Pfair 스케줄링(PF) 알고리즘[1]이 Baruah 등에 의해 제안되었고, 이를 기반으로 하는 몇 가지 스케줄링 알고리즘들이 제안되었는데 이들은 모두 고정된 퀀텀 크기를 가정하고 있다. 전역 스케줄링 기법인 Pfair 기반 스케줄링 알고리즘에서 퀀텀 크기는 태스크 교환과 캐시 재적재와 같은 스케줄링 오버헤드에 직접적인 영향을 미치게 된다. 이에 따라 태스크 집합에 대한 최적 퀀텀 크기를 결정하기 위한 방법이 제안되었으며[2], 모든 태스크들의 주기와 실행 요구 시간이 $e \leq p/3+1$의 성질을 만족하는 제한적 특성의 태스크 집합에 대해서 보다 효율적으로 최적 퀀텀 크기를 결정할 수 있는 방법[3]이 제안된 바 있다. 그런데 이들 방법에서는 최적의 퀀텀 크기를 결정하기 위해 반복적으로 프로세서의 이용률을 계산하였다. 본 논문에서는 이러한 제한적 특성의 태스크 집합에 대해서 프로세서 이용률 계산을 반복적으로 수행하지 않고 상수 시간에 최적 퀀텀 크기를 결정할 수 있는 보다 효율적인 방법을 제안한다.

An Improved Dynamic Quantum-Size Pfair Scheduling for the Mode Change Environments

Seong-Duk Cha*, In-Guk Kim**

Abstract

Recently, Baruah et. al. proposed an optimal Pfair scheduling algorithm in the real-time multiprocessor system environments, and several variants of it were presented. All these algorithms assume the fixed unit quantum size. However, under Pfair based scheduling algorithms that are global scheduling technique, quantum size has direct influence on the scheduling overheads such as task switching and cache reload. We proposed a method for deciding the optimal quantum size[2] and an improved method for the task set whose utilization e is less than or equal to $e \leq p/3+1$ [3]. However, these methods use repetitive computation of the task's utilization to determine the optimal quantum size. In this paper, we propose a more efficient method that can determine the optimal quantum size in constant time.

Keywords : 실시간 스케줄링, 다중프로세서 스케줄링, Pfair 스케줄링

1. 서론

다중프로세서 시스템 환경에서 실시간 스케줄링 기법은 크게 분할(partitioning) 기법과 전역(global) 스케줄링 기법으로 분류될 수 있다[4]. 분할 기법에서 태스크들은 각각의 준비 큐를 갖는 프로세서에 고정적으로 할당되며 각 프로세서는 RMS와 EDS[5]와 같은 단일프로세서 스케줄링 알고리즘에 따라 할당된 태스크들을 실행하게 된다. 이러한 분할 기법은 다중프로세서의 스케줄링 문제를 단일프로세서의 스케줄링 문제로 단순화 시킬 수 있다는 장점을 갖지만 최적의 태스크 할당 방법은 NP-hard한 bin-packing 문제[6,7]이므로 태스크의 할당에 RMFFS나 RMNFS와 같은 휴리스틱 방법들이 사용되어 왔다[8].

※ 제일저자(First Author) : 차성덕
접수일자:2007년07월12일, 심사완료:2007년07월13일
* 단국대학교대학원 전자계산학과
chastone@dku.edu
** 단국대학교 컴퓨터학부 교수

이와 달리 전역 스케줄링 기법에서 태스크들은 모든 프로세서들이 공유하는 단일 준비 큐에 저장되고, 스케줄링 시점마다 단일 준비 큐로부터 최상위 우선순위의 태스크가 선택되어 실행된다. 따라서 한 태스크의 작업(job)들은 이전에 실행되었던 프로세서와는 다른 프로세서에서 실행 될 수 있다. 전역 스케줄링 기법의 중요한 이점은 태스크들의 프로세서 간 이동(migration)이 허용된다면 최적의 스케줄링이 가능하다는 것이다. 그러나 전역 스케줄링 기법에 RMS나 EDS와 같은 기존의 최적 단일 프로세서 스케줄링 알고리즘을 그대로 적용하게 되면 스케줄링에 실패하거나 낮은 프로세서 이용률을 초래하게 된다[8].

그러나 Baruah 등은 다중 프로세서 환경에서 프로세서의 이용률을 극대화시킬 수 있는 최적의 스케줄링 알고리즘인 PF 알고리즘[1]과 이를 효율적으로 개선한 PD[9] 알고리즘을 제안하였다. 이들은 프로세서의 수가 M일 때 태스크 집합이 스케줄링 가능하기 위한 필요충분조건은 다음과 같음을 증명하였는데, 여기서 T는 태스크 집합 τ에 속한 태스크를 의미하며, $T.e$와 $T.p$는 각각 태스크 T에 대한 실행 요구 시간과 주기를 의미한다.

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

이후, PD 알고리즘의 tie-breaking 파라미터 수를 2개로 감소시킨 PD2[10]가 Anderson 등에 의해 제안되었으며, 이 밖에도 ERfair[11], EPDF[12], QRfair(PDQ)[13], BF[14] 등 Pfair를 기반으로 하는 다수의 알고리즘들이 제안되었다. 그런데, 이러한 퀀텀 기반 스케줄링 알고리즘들에서는 태스크의 주기와 실행 요구 시간을 고정적인 퀀텀 크기의 배수로 가정하고 있다.

Pfair 기반 스케줄링 방법에서는 고정된 퀀텀 크기의 슬롯(slot) 마다 실행될 서브태스크가 선택되며, 서브태스크의 실행은 프로세서 간에 이동될 수 있음을 가정하고 있다. 따라서 너무 작은 퀀텀 크기는 빈번한 스케줄링으로 인한 빈번한 태스크의 선점과 프로세서 간 이동을 초래하게 되므로 태스크 문맥 교환과 캐시 재적재와 같은 스케줄링 오버헤드를 증가시키게 된다[4]. 따라서 해당 태스크 집합이 스케줄링 가능한 최대의 퀀텀 크기를 적용한다면 이러한 오버헤드를 감소시킬 수 있다. 이러한 최대의 퀀텀 크기를 최적 퀀텀 크기라 하는데, mode change가 발생하여 태스크 집합이 변경[15]되면 최적 퀀텀 크기는 다시 결정되어야 한다. 따라서 변경된 태스크 집합에 대한 최적 퀀텀 크기는 이전 퀀텀 크기에 비해 증가되거나 감소될 수 있다[2].

임의의 태스크 집합에 대해 최적의 퀀텀 크기를 결정할 수 있는 다항식 함수는 알려진 바 없다. 이에 대해, 반복적인 프로세서 이용률 계산을 통해 최적의 퀀텀 크기를 결정할 수 있는 방법[2]이 제안되었으며, 모든 태스크의 이용률이 $e \leq p/3+1$을 만족하는 제한적 특성의 태스크 집합에 대해서 이용률 계산의 반복 횟수를 이전 방법보다 감소시킬 수 있는 방법[3]이 제안되었다. 그러나 이 방법들은 여전히 프로세서 이용률 계산의 반복을 통해 최적의 퀀텀 크기를 결정하였다. 본 논문에서는 이러한 제한적 특성을 갖는 태스크 집합에 대해서 상수 시간에 최적의 퀀텀 크기를 결정할 수 있는 추가적인 조건을 제시하고 이를 증명함으로서 기존 방법에 비해 보다 효율적으로 최적 퀀텀 크기를 결정할 수 있는 방법을 제안하고자 한다.

2. 태스크 모델

본 논문의 대상이 되는 태스크 모델은 기존 방법[2,3]과 동일하다. 즉, 퀀텀 크기가 증가함에 따라 태스크 본연의 실행 요구 시간은 증가될 수 있으며, 주기는 감소될 수 있는 주기 가변 태스크를 대상으로 한다. 따라서 퀀텀 크기 Q에 대해 재계산되는 태스크의 실행 요구 시간 e_n과 주기 p_n은 각각 식 (2)와 식 (3)과 같다.

$$e_n = \left\lceil \frac{e}{Q} \right\rceil \qquad (2)$$

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

따라서 재계산되는 태스크의 이용률 w'은 식 (4)과 같으며 태스크 집합의 프로세서 이용률 U는 식 (5)와 같다.

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

$$U = \sum_{T \in \tau} T.w' \tag{5}$$

3. 일반 태스크 집합에 대한 최적 퀀텀 크기 결정

본 장에서는 태스크의 이용률에 제한이 없는 ($T.w \leq 1$) 일반 태스크 집합에 대한 최적 퀀텀 크기 결정 방법[2]에 대해 기술하기로 한다.

3.1 퀀텀 크기 변경에 따른 프로세서 이용률 변화 특성

식 (4)에 의해 Q를 증가시킴에 따라 태스크의 w'는 부분적으로는 감소와 증가를 반복하다가 특정 퀀텀 크기로 부터는 항상 1의 값을 갖게 된다. 이렇게 w'가 연속적으로 1이 되는 최초의 퀀텀 크기를 실제 도달점 RRP라 하는데 일반 태스크 집합에 대해 $RP \geq RRP$를 만족하는 도달점 RP를 결정하는 함수 $Reach(p, e)$는 [정리 1]과 같다.

[정리 1] 도달 함수 $Reach(p, e)$는 다음과 같이 정의한다.

$$Reach(p, e) = \begin{cases} \left\lfloor \frac{p}{2} \right\rfloor + 1 & \left(\text{if } \frac{e}{p} \leq \frac{1}{2}\right) \\ \left\lfloor \frac{p}{3} \right\rfloor + 1 & \left(\text{if } \frac{e}{p} > \frac{1}{2}\right) \end{cases} \tag{6}$$

이때, $Q \geq Reach(p, e)$이면

임의의 e와 p에 대해 $w' = \dfrac{\left\lceil \frac{e}{Q} \right\rceil}{\left\lfloor \frac{p}{Q} \right\rfloor} \geq 1$이다.

[정리 1]에 대한 증명은 [2]에 나와 있다.

3.2 최적 퀀텀 크기 결정 방법

N개의 태스크에 대한 RP가 오름차순으로 저장된 리스트를 도달 순위 리스트 $Rank[N]$이라 하고 프로세서의 수를 M이라 할 때, 도달 순위 리스트의 순서에 따라 M개의 태스크가 도달 구간($Q \geq RP$)에 있다면 스케줄링은 실패하게 되므로, 최적의 Q값은 적어도 M번째 태스크의 도달점 보다 작은 구간에서 결정될 수 있다. 따라서 도달 순위 리스트에 $RP-1$의 값이 저장된다고 하면, 최적의 퀀텀 크기는 다음 식을 만족하는 구간에 존재하게 된다.

$$1 \leq Q \leq Rank[M-1] \tag{7}$$

(알고리즘 1)은 일반 태스크 집합에 대해 도달 순위 리스트를 기반으로 항상 최적의 퀀텀 크기를 결정하기 위한 방법인 $FindQ()$를 보여주고 있다.

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

(알고리즘 1) Rank[M-1]로부터 반복을 시작하는 FindQ() 함수

4. 제한된 이용률을 갖는 태스크 집합의 최적 퀀텀 크기 결정

본 장에서는 제한된 이용률을 갖는 태스크 집합에 대해 최적 퀀텀 크기 결정하기 위한 기존

의 방법[3]에 대해 기술한다.

4.1 제한적 태스크 집합의 정의와 퀀텀 크기 증가에 따른 이용률 증가 특성

본 논문에서 제한적 태스크 집합은 태스크의 주기와 실행 요구 시간에 대해 다음의 성질을 만족하는 태스크 집합으로 정의한다.

$$e \leq \frac{p}{3}+1 \qquad (8)$$

즉, 제한적 태스크 집합은 w가 대략 1/3 이하인 태스크들로만 구성되어 있는데, 이러한 태스크들에 대해 다음이 항상 성립한다.

[정리 2] $e \leq \frac{p}{3}+1$인 태스크에 대해

$$\frac{p}{3} < Q \leq \frac{p}{2} \qquad (9)$$

이면 $w' = \dfrac{\left\lceil \frac{e}{Q} \right\rceil}{\left\lfloor \frac{p}{Q} \right\rfloor} = \frac{1}{2}$이다.

[정리 2]에 대한 증명은 [3]에 나와 있다.

4.2 최적 퀀텀 크기 결정 방법

제한적 이용률을 갖는 태스크들에 대한 도달점은 두 가지로 정의된다[3]. 첫 번째는 [정리 1]에 의한 도달점인 FRP(Full-Reach Point)로서 $w'=1$이 연속으로 나오기 시작하는 Q의 값이며, 두 번째는 [정리 2]에 의한 도달점 HRP(Half-Reach Point)로서 $w'=0.5$가 연속으로 나오기 시작하는 Q의 값이다. 이 두 가지의 도달점들을 계산하기 위한 도달 함수 $FReach(p, e)$와 $HReach(p, e)$는 [정리 1]과 [정리 2]에 의해 다음과 같이 정의된다.

$$\begin{cases} FReach(p,e) = \left\lfloor \frac{p}{2} \right\rfloor + 1 \\ HReach(p,e) = \left\lfloor \frac{p}{3} \right\rfloor + 1 \end{cases} \qquad (10)$$

FRP만을 이용해서 최적 퀀텀 크기를 결정하기 위한 방법을 *FindQF*()라 하고, FRP와 HRP를 기반으로 최적 퀀텀 크기를 결정하기 위한 방법을 *FindQHF*()라 하자. 태스크의 수를 N이라 할 때, *FindQF*()에서는 각 태스크의 FRP-1 값이 오름차순으로 기록된 도달 순위 리스트 $Rank[N]$만을 기반으로 전체 이용률 계산의 반복에 대한 초기 Q값을 결정하였다. 즉, 프로세서의 수를 $M(M<N)$이라 하면 태스크 집합의 프로세서 이용률 계산을 위해 초기 Q 값은 $Rank[M-1]$이 되고 이 Q 값을 1씩 감소시켜 가면서 전체 이용률이 $U \leq M$을 만족하는 최초(최적)의 퀀텀 크기를 찾아나가는 방식이다. 그런데 이 방법에서는 도달점에 도달하지 않은 태스크들의 이용률의 합도 1보다 클 수 있다는 점을 감안하지 않았다.

<표 1> τ_1에 대한 w'와 프로세서 이용률 변화

Q	U	w'					
		T1	T2	T3	T4	T5	T6
10	1.468	*0.500*	0.333	0.200	0.167	0.143	0.125
11	1.593	*0.500*	0.333	0.200	0.200	0.167	0.143
12	1.617	*0.500*	0.333	0.250	0.200	0.167	0.167
13	1.700	*0.500*	0.333	0.250	0.250	0.200	0.167
14	1.983	*0.500*	*0.500*	0.250	0.250	0.200	0.200
15	2.533	**1.000**	*0.500*	0.333	0.250	0.250	0.200
16	2.617	**1.000**	*0.500*	0.333	0.333	0.250	0.200
17	2.667	**1.000**	*0.500*	0.333	0.333	0.250	0.250
18	2.667	**1.000**	*0.500*	0.333	0.333	0.250	0.250
19	2.917	**1.000**	*0.500*	*0.500*	0.333	0.333	0.250
20	3.417	**1.000**	**1.000**	*0.500*	0.333	0.333	0.250
21	3.667	**1.000**	**1.000**	*0.500*	*0.500*	0.333	0.333
22	3.667	**1.000**	**1.000**	*0.500*	*0.500*	0.333	0.333
23	3.667	**1.000**	**1.000**	*0.500*	*0.500*	0.333	0.333
24	3.667	**1.000**	**1.000**	*0.500*	*0.500*	0.333	0.333
25	3.833	**1.000**	**1.000**	*0.500*	*0.500*	*0.500*	0.333
26	3.833	**1.000**	**1.000**	*0.500*	*0.500*	*0.500*	0.333
27	3.833	**1.000**	**1.000**	*0.500*	*0.500*	*0.500*	0.333
28	4.500	**1.000**	**1.000**	*0.500*	*0.500*	*0.500*	*0.500*
29	4.500	**1.000**	**1.000**	**1.000**	*0.500*	*0.500*	*0.500*
30	4.500	**1.000**	**1.000**	**1.000**	*0.500*	*0.500*	*0.500*
31	5.000	**1.000**	**1.000**	**1.000**	**1.000**	*0.500*	*0.500*

이에 반해, *FindQHF*()는 [정리 2]의 특성에 따라 전체 이용률 계산의 횟수를 감소시키기 위해 각 태스크의 FRP와 HRP 값 모두를 고려하여 최적의 퀀텀 크기를 찾도록 하고 있다. 즉, *FindQF*()에서는 도달점에 도달한 하나의 태스크만을 고려하여 이용률 계산의 시작점을 결정하였지만, *FindQHF*()은 HRP에 도달한 두 태스

크의 이용률 합도 1에 도달할 수 있다는 점을 감안하여 이용률 계산의 시작점을 결정하게 된다. 따라서 *FindQHF*()에서는 각 태스크의 FRP와 HRP가 오름차순으로 기록된 $Rank[2N]$으로 도달 순위 리스트를 재정의 한다.

<표 1>은 태스크 집합 $\tau_1 = \left\{\frac{6}{29}, \frac{5}{39}, \frac{7}{56}, \frac{7}{61}, \frac{4}{74}, \frac{9}{83}\right\}$에 대한 w'의 증가 모습을 보여주고 있으며, 이 태스크 집합에 대한 도달 순위 리스트는 <표 2>와 같다.

<표 2> τ_1에 대한 도달 순위 리스트

i	0	1	2	3	4	5	6	7	8	9	10	11
$Rank[i]$	10	14	15	19	20	21	25	28	29	31	38	42
$U_{\min}$	0.5	1	1.5	2.0	2.5	3	3.5	4	4.5	5	5.5	6
T	T1	T2	T1	T3	T2	T4	T5	T6	T3	T4	T5	T6

<표 2>에서 $U_{\min}$은 해당 도달점에서 가질 수 있는 이용률 합의 최소값을 의미하며 $Rank[i]$에서의 $U_{\min}$을 $Rank[i].U_{\min}$으로 표기한다. 도달 순위 리스트 내에 중복된 도달점이 존재하지 않는다면 인덱스 i가 1 증가함에 따라 $U_{\min}$도 0.5씩 증가하게 된다. 그런데 도달 순위 리스트 내에 동일한 도달점들이 존재한다면 도달 순위 리스트는 다음과 같은 성질을 갖는다. 여기서 태스크의 수를 N이라 하면 m은 $(1 \le m \le N-1)$인 정수이다. 또한 M은 프로세서의 수를 나타낸다.

[보조정리 1] $Rank[i]$로부터 시작하여 k개의 동일한 HRP 또는 FRP가 존재한다면($k \ge 2$인 경우 중복이 발생), $Rank[i]$부터 $Rank[i+k-1]$의 $U_{\min}$은 $Rank[i-1].U_{\min} + \frac{k}{2}$이다.

[보조정리 1]에 대한 증명은 [3]에 나와 있다.

[보조정리 2] $Rank[2m-1] \neq Rank[2m]$이면, 즉, $Rank[2m-1]$과 동일한 도달점이 존재하지 않는다면, 다음이 항상 성립한다.

$$m = Rank[2m-1].U_{\min} \tag{11}$$

[보조정리 2]에 대한 증명은 [3]에 나와 있다.

[보조정리 3] $Rank[2m-1]$ 이후로 중복된 도달점이 적어도 하나이상 존재하면, 다음이 항상 성립한다.

$$m < Rank[2m-1].U_{\min} \tag{12}$$

[보조정리 3]에 대한 증명은 [3]에 나와 있다.

최적 퀀텀 크기를 Q_{opt}라 하자. Q_{opt}를 찾기 위해서는 일단 임의의 $Rank[i]$를 기준으로 삼아야 하는데, 프로세서의 수가 M일 때 Q_{opt}를 찾기 위한 반복의 시작점을 sp_M이라 하면 sp_M은 일단 다음 식과 같이 선택된다.

$$sp_M = Rank[2M-1] \tag{13}$$

식 (13)에서 $Rank[i]$의 인덱스를 $2M-1$로 하는 이유는 프로세서의 수가 정수이고, 도달 순위 리스트 내에 중복된 도달점이 없다면 [보조정리 2]에 의해 정수 값을 갖기 때문이다.

[보조정리 4] $e \le \frac{p}{3}+1$인 태스크에 대해서 $Rank[2M-1] = Rank[2M]$이면, $Q_{opt} < Rank[2M]$이다.

[보조정리 4]에 대한 증명은 [3]에 나와 있다.

[보조정리 5] $e \le \frac{p}{3}+1$인 태스크에 대해서 $Rank[2M-1] \neq Rank[2M]$이면, $Q_{opt} < Rank[2M]$이다.

[보조정리 5]에 대한 증명은 [3]에 나와 있다.

[정리 3] $e \le \frac{p}{3}+1$인 태스크에 대해서 도달점의 중복에 상관없이 $Q_{opt} < Rank[2M]$이다.

[정리 3]에 대한 증명은 [3]에 나와 있다.

[정리 3]을 기반으로 하는 *FindQHF*()에 대한 알고리즘은 (알고리즘 2)와 같다.

```
int FindQHF() {
   int Q=1, sp;
   for(sp=Rank[2*M]-1; sp>0; sp--){
      if(U(sp) <= M) {
         Q = sp;
         break;
      }
   }
   return Q;
}
```

(알고리즘 2) FindQHF()

5. 개선된 최적 퀀텀 크기 결정 방법

본 장에서는 제한된 이용률을 갖는 태스크 집합에 대해서 경우에 따라 상수 시간에 최적 퀀텀 크기를 결정할 수 있는 추가적인 조건을 제시하고 이를 증명함으로써 기존 연구보다 향상된 효율을 보이는 최적 퀀텀 크기 결정 방법을 제안하도록 한다.

특정 HRP 이후에 재계산된 w'를 w_R(0.5 또는 1)이라하고 HRP 이전에 계산된 w'를 $w_E(<1)$이라 하면, $Rank[i].U_{\min}$이 일부 태스크의 w_R들로만 계산되는 경우와 모든 태스크들의 w_R들로만 계산되는 경우가 있다. <표 2>에서 $Q<28$인 경우가 전자에 해당하고 $Q \geq 28$인 경우는 후자에 해당한다. 이때 이러한 기준점은 HRP의 최대값인 $HRP_{\max}$임을 쉽게 알 수 있다. 이러한 기준점은 sp_M 또는 Q_{opt}를 결정하는데 중요한 영향을 미치게 되는데 이 두 가지의 경우, 즉, $Rank[i] < HRP_{\max}$인 경우와 그렇지 않은 경우에 대해서 sp_M 또는 Q_{opt}를 결정하기 위한 방법을 제시하도록 한다.

$Rank[i]$에서의 실제 이용률의 합, 즉, 모든 태스크들의 w_R과 w_E이 모두 계산된 실제 이용률의 합 $Rank[i].U$는 다음과 같이 정의한다. 여기서 $\sum w_E$은 해당 도달점에 존재하는 w_E의 총합이다.

$$Rank[i].U = Rank[i].U_m + \sum w_E \qquad (14)$$

또한 $Rank[i].U$에 대해 다음이 성립한다.

[보조정리 6] $Rank[i] < HRP_{\max}$이면 다음이 성립한다.

$$Rank[i].U_{\min} < Rank[i].U \qquad (15)$$

(증명) $Rank[i] < HRP_{\max}$이면 w_E가 적어도 하나 이상 존재하고 $w_E > 0$이므로 $\sum w_E > 0$이 된다. 따라서 $Rank[i].U$ 정의에 따라 정리가 성립한다. ■

[정리 4] $sp_M = Rank[2M-1]$이라 할 때, $sp_M < HRP_{\max}$이면 $Q_{opt} < Rank[2M-1]$이다.

(증명) [보조정리 3]에 의해 $M \leq sp_M.U_{\min}$이고 [보조정리 6]에 의해 $sp_M.U_{min} < sp_M.U$이므로, $M < sp_M.U$이 된다. 프로세서의 수가 M일 때 Pfair 스케줄링 알고리즘에 의해 스케줄링 가능하기 위한 필요충분조건은 $U \leq M$이다. 그런데 $Q \geq sp_M$이면 $Q.U > M$이므로 스케줄링에 실패하게 된다. 따라서 Q_{opt}는 sp_M 보다 작은 값으로 존재하게 된다. ■

[보조정리 7] $Rank[i] \geq HRP_{\max}$이면 다음이 성립한다.

$$Rank[i].U_{\min} = Rank[i].U \qquad (16)$$

(증명) $Rank[i].U$의 정의에 따라 $\sum w_E = 0$이므로 분명하다. ■

[정리 5] 프로세서의 수가 M이고 $sp_M = Rank[2M-1]$일 때, $sp_M \geq HRP_{\max}$이고 $sp_M = Rank[2M]$이면, $Q_{opt} < Rank[2M-1]$이다.

(증명) sp_M와 동일한 도달점이 적어도 하나 이상 존재한다면 [보조정리 7]과 [보조정리 1]에

의해 다음이 성립한다.

$$sp_M U = sp_M U_{\min} > M$$

따라서 [정리 4]와 마찬가지로 Q_{opt}는 sp_M 보다 작은 값으로 존재하게 된다. ■

[정리 6] 프로세서의 수가 M이고 $sp_M = Rank[2M-1]$일 때, $sp_M \geq HRP_{\max}$이고 $sp_M \neq Rank[2M]$이면, $Q_{opt} = Rank[2M]-1$이다.

(증명) [보조정리 7]과 [보조정리 2]에 의해 $sp_M U = M$이고 [보조정리 1]에 의해 $Rank[2M].U = M+0.5$이다. 그런데 $Rank[2M-1]$과 $Rank[2M]-1$ 사이에는 w_E가 존재하지 않으므로 U도 증가하지 않으며, $U_{\min}$은 $Rank[2M]$에서 증가되므로 $Rank[2M-1].U = (Rank[2M]-1).U$이다. 따라서 $sp_M U = Rank[2M-1].U = M$이므로 $Q_{opt} = Rank[2M]-1$이 된다. ■

```
int FindQHF2() {
   int Q = 1;
   int i;

   if(Rank[2*M-1] >= HRP_max
      && Rank[2*M-1] != Rank[2*M]){
      Q = Rank[2*M]-1;
   }
   else {
      for(i=Rank[2*M-1]-1; i>0; i--){
         if(U(Rank[i]-1)<=M){
            Q = i;
            break;
         }
      }
   }
   return Q;
}
```

(알고리즘 3) FindQHF2()

특히, [정리 6]은 특정 태스크 집합에 대해 최적의 퀀텀 크기를 찾기 위해 프로세서 이용률을 계산하지 않고도 상수 시간에 최적의 퀀텀 크기를 결정할 수 있다는 점에서 중요한 의미를 갖는다.

[정리 4]와 [정리 5] 및 [정리 6]을 기반으로 하는 최적 퀀텀 크기 결정 방법 (알고리즘 3)과 같다. (알고리즘 3)에서 HRP_max가 $HRP_{\max}$를 의미하며, if 절은 [정리 6], else 절은 [정리 4]와 [정리 5]에 해당한다.

6. 실험 및 분석

제한된 이용률 특성을 갖는 태스크 집합에 대한 최적 퀀텀 크기 결정을 위해 기존에 제안된 방법[3]을 *FindQHF*1()이라 하고, 본 논문에서 제안한 방법을 *FindQHF*2()라 하자. 본 장에서는 *FindQHF*1()과 *FindQHF*2()의 비교 실험을 통해 *FindQHF*2()가 항상 최적의 퀀텀 크기를 결정할 수 있는 지 확인해 보고, 평균 이용률 계산 횟수에서 기존 방법에 비해 어느 정도의 성능이 향상 되었는지 측정해 본다. 또한 제안된 알고리즘이 프로세서의 수에 따라 나타내는 특성을 분석해 보도록 한다.

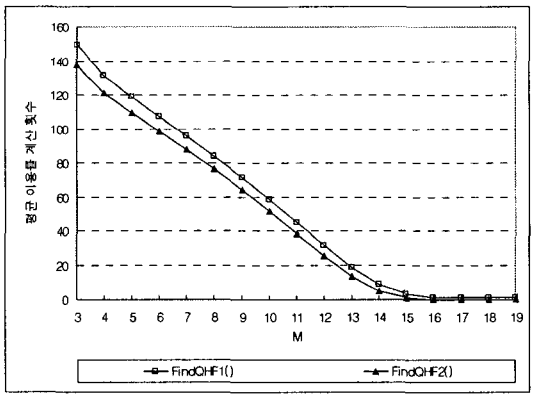
실험을 위해 생성된 태스크 집합은 모든 태스크들이 e와 p에 대해 $e \leq \frac{p}{3}+1$을 만족하도록 임의로 생성된 20개의 태스크들로 구성되며, 각 태스크 집합의 프로세서 이용률이 3을 넘지 않도록 무작위로 생성된 100,000개의 태스크 집합을 프로세서의 수가 3에서 19인 경우에 대해 실험하였다. 이때 생성된 태스크 집합의 평균 프로세서 이용률은 2.7637이다. 먼저, <표 3>은 프로세서의 수가 4인 경우에 대한 결과를 보여주고 있다. 이 실험의 평가 항목에서, 평균 이용률 계산 횟수는 최적 퀀텀 크기를 결정하기 위해 반복된 이용률 계산의 평균 횟수를 나타내며, 실패 횟수는 1보다 큰 최적 퀀텀 크기가 존재함에도 불구하고 그 값을 1로 결정한 평균 횟수를 의미하며, 차이 횟수는 최적 퀀텀 크기는 아니지만 1보다 큰 값으로 최적 퀀텀 크기를 결정한 평균 횟수를 나타낸다. 따라서 실패 횟수와 차이 횟수가 모두 0인 경우 모든 태스크 집합에 대해 최적 퀀텀 크기를 결정했다고 판단할 수 있다. 마지막으로 각 항목에서의 비율은 *FindQHF*1() 결과에 대한 *FindQHF*2() 결과의 비율을 나타낸다.

<표 3> M=4인 경우의 실험 결과

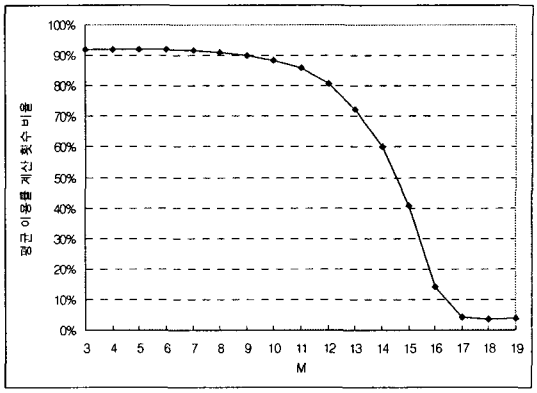
항목 / 방법	평균 이용률 계산 횟수 (비율)	실패 횟수 (비율)	차이 횟수 (비율)
FindQHF1()	131.32	0	0
FindQHF2()	120.85 (92.03%)	0 (0.0%)	0 (0.0%)

위의 실험 결과를 보면 모든 방법에서 실패와 차이 횟수가 0이므로 *FindQHF2*()는 항상 최적 퀀텀 크기를 결정했음을 볼 수 있다. 또한 *FindQHF1*()에 대한 *FindQHF2*()의 평균 이용률 계산 횟수의 비율은 92.03%로서 기존 방법에 비해 성능이 향상되었음을 볼 수 있었다. 프로세서 이용률의 계산이 필요 없이 상수 시간에 최적 퀀텀 크기를 결정할 수 있도록 추가로 제시한 조건인 [정리 6]에 의해 *FindQHF2*()는 추가적으로 약 8%의 평균 이용률 계산 횟수 감소 효과를 보이고 있다.

다음으로 프로세서의 수에 따른 *FindQHF1*()과 *FindQHF2*()의 결과를 분석해보도록 한다. (그림 1)은 프로세서의 수에 따른 *FindQHF1*()과 *FindQHF2*()의 평균 이용률 계산 횟수를 보여주고 있으며, (그림 2)는 그 비율을 보여주고 있다.



(그림 1) 프로세서 수에 따른 FindQHF1()과 FindQHF2()의 평균 이용률 계산 횟수



(그림 2) 프로세서의 수에 따른 FindQHF1()에 대한 FindQHF2()의 평균 이용률 계산 횟수 비율

실험 결과 *FindQHF1*()과 *FindQHF2*()는 프로세서의 수에 상관없이 항상 최적의 퀀텀 크기를 결정할 수 있었다. (그림 1)과 (그림 2)를 보면 *FindQHF2*()는 *FindQHF1*()보다 항상 작은 평균 프로세서 이용률 계산 횟수의 결과를 보이고 있다.

<표 4> (16≤M≤19)인 경우 FindQHF1()과 FindQHF2()의 평균 이용률 계산 횟수와 비율

M / 방법	16	17	18	19
FindQHF1()	1.34	1.02	1	1
FindQHF2()	0.19 (14.06%)	0.04 (4.11%)	0.04 (3.50%)	0.04 (3.80%)

<표 4>는 프로세서의 수가 $(16 \le M \le 19)$인 경우 *FindQHF1*()과 *FindQHF2*()의 평균 프로세서 이용률 계산 횟수와 그 비율을 보다 자세하게 보여주고 있다. *FindQHF1*() 방법에서는 최적의 퀀텀 크기를 결정하기 위해서 각 태스크 집합 당 적어도 1번 이상의 프로세서 이용률 계산을 수행해야 한다. 따라서 *FindQHF1*() 방법에서의 평균 이용률 계산 횟수는 1보다 작을 수 없기 때문에 <표 4>에서 보면 *FindQHF1*()은 프로세서의 수가 18과 19인 경우에 최소의 평균 프로세서 이용률 계산 횟수인 1의 결과를 보이고 있다. 이것은 모든 태스크 집합에 대해 단 한 번의 이용률 계산으로 최적의 퀀텀 크기를 결정할 수 있었음을 의미한다.

그런데 *FindQHF2*()는 프로세서의 수가 16인 경우부터 평균 이용률 계산 횟수가 1보다 작음을 볼 수 있다. 이것은 *FindQHF2*()가 [정리 6]에 의해서 프로세서 이용률 계산을 한 번도 수행하지 않고도 최적의 퀀텀 크기를 결정할 수 있었기 때문이다. 따라서 특정 태스크 집합에 대해서 프로세서 이용률 계산 횟수가 0일 수도 있으므로, 평균 이용률 계산 횟수는 1보다 작아질 수 있다. 즉, 최저값인 0.04는 태스크 집합 100개 중에서 4개의 태스크 집합은 한 번의 이용률 계산으로, 나머지 96개의 태스크 집합은 [정리 6]에 의해 이용률 계산 없이도 곧바로 최적의 퀀텀 크기를 결정할 수 있음을 의미하게 된다.

7. 결론 및 향후 연구

$e \leq \frac{p}{3}+1$를 만족하는 태스크 집합에 대해 최적의 퀀텀 크기를 결정하기 위한 방법[3]이 제안된 바 있다. 이 방법에서는 정의된 이용률 제한의 특성을 갖는 태스크들은 퀀텀의 크기를 증가시킴에 따라 재계산된 이용률이 1에 도달하는 기존의 도달점 외에도 재계산된 이용률이 0.5에도 도달하는 점을 이용해 새로운 도달 순위 함수를 정의하고, 이를 기반으로 최적의 퀀텀 크기를 찾기 위한 이용률 계산의 반복 횟수를 감소시킬 수 있었다.

본 논문에서는 이러한 태스크 집합에 대해 프로세서 이용률의 계산 없이도 상수 시간에 최적 퀀텀 크기를 결정할 수 있는 추가적인 조건을 제시함으로써 기존 연구보다 향상된 방법을 제시하였다.

실험 결과, 제안된 방법은 프로세서의 수에 관계없이 항상 최적 퀀텀 크기를 결정할 수 있었으며, 기존 연구에 비해 항상 향상된 결과를 보였다. 프로세서의 수가 증가 할수록 보다 향상된 성능을 보였는데, 특히, 프로세서의 수가 태스크의 수와 비슷할 경우 약 96%의 태스크 집합에 대해서는 상수 시간에 최적의 퀀텀 크기를 결정할 수 있었으며, 나머지 4%의 태스크 집합에 대해서도 단 한 번의 이용률 계산을 통해 최적의 퀀텀 크기를 결정할 수 있었다.

본 논문에서 제시한 방법들에서는 최적의 퀀텀 크기를 결정하기 위해 임의의 도달점으로부터 프로세서 이용률 계산을 시작하였는데, 이 도달점은 최적 퀀텀 크기를 결정하기 위한 충분조건에 불과하다. 따라서 이러한 충분조건으로서의 도달점 이외에도 최적 퀀텀 크기 결정을 위한 프로세서 이용률 계산의 반복을 감소시킬 수 있는 추가적인 태스크 집합의 특성이 존재할 것이라고 판단된다. 예를 들어, 각 태스크의 이용률 특성이나 경량/중량 태스크들의 구성 비율, 태스크들의 주기 및 실행 요구시간에 대한 공통성, 태스크 수와 프로세서의 수의 관계 등과 같은 추가적인 특성들에 대한 분석과 정의를 통해 본 논문에서 제안된 방법에 대한 성능을 보다 향상시킬 수 있을 것이다.

또한, 본 논문에서는 퀀텀 크기의 증가에 따라 태스크의 주기를 감소시키도록 했는데, 재계산된 주기를 원래의 주기보다 제한적으로 증가시킬 수 있다면, 퀀텀 크기의 변경에 따른 주기 가변 태스크의 문제는 연성 실시간 시스템의 스케줄링 문제에도 적용될 수 있을 것이라 본다.

참 고 문 헌

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

[2] 저자, "Mode change 환경에 적합한 동적 퀀텀 크기 스케줄링", 한국콘텐츠학회논문지, 제6권, 제9호, pp. 28-41, 2006.

[3] 저자, "태스크 집합의 특성을 고려한 동적 퀀텀 크기 Pfair 스케줄링", 한국콘텐츠학회논문지, 제7권, 제7호, 2007. (게재 예정)

[4] Srinivasan, A., Holman, P., Anderson, J., and Baruah, S., "The case for fair multiprocessor scheduling", Proceedings of the 11th International Workshop on Parallel and Distributed Real-Time Systems, April 2003.

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

[6] Davari, S. and Liu, C. L. "An On-Line Algorithm for Real-Time Tasks Allocation", In Proceedings of 7th Real-time Systems Symposium, pp. 194-200, 1986.

[7] Garey, M. and Johnson, D., "Computers and Intractab

ility: A Guide to the Theory of NP-Completeness", W.H. Freeman and Company, 1979.

[8] Dhall, S. K. and Liu, C. L., "On a Real-Time Scheduling Problem", Operations Research, Vol .26, No. 1, pp. 127-140, Jan-Feb. 1978.

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

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

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

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

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

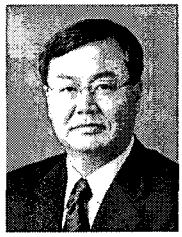
[14] Zhu, D., Mosse, D., and Melhem, R., "Multiple-Resource Periodic Scheduling Problem: how much fairness is necessary?", Real-Time Systems Symposium, Proceedings of the 24th IEEE Real-time Systems Symposium, pp. 142-151, Dec. 2003.

[15] Sha, L., Rajkumar, R., Lehoczky, J., and Ramamritham, K., "Mode Change Protocols for Priority-Driven Pre-emptive Scheduling", Real Time Systems 1(3), pp. 244-264, 1989.

차 성 덕

1999년 : 단국대학교 전자계산학과 (이학사)
2001년 : 단국대학교대학원 전자계산학과(이학석사)
2007년 : 단국대학교대학원 전자계산학과(이학박사)

관심분야: 운영체제, 실시간시스템, 임베디드 시스템



김 인 국

1982년 : 단국대학교(학사)
1985년 : 미국 에모리대학교(석사)
1995년 : 아주대학교(박사)
2001년~2003년 : 미국 뉴멕시코공과대학 방문교수
1986년~현 재 : 단국대학교 컴퓨터학부 교수
관심분야 : 운영체제, 실시간시스템, 임베디드 시스템