태스크 집합의 특성을 고려한 동적 퀀텀 크기 Pfair 스케줄링

Dynamic Quantum-Size Pfair Scheduling Considering Task Set Characteristics

차성덕, 김인국
단국대학교 컴퓨터학부

Seong-Duk Cha(chastone@dankook.ac.kr), In-Guk Kim(igkim@dankook.ac.kr)

요약

다중 프로세서 환경에서 경성 실시간 태스크 집합의 스케줄링 문제를 해결하는 최적의 방법인 PF 알고리즘[13]이 제안된 이후, 이를 기반으로 하는 여러 가지 스케줄링 알고리즘들이 제안되었다. 그러나 고정된 퀀텀 크기를 기반으로 태스크들을 스케줄링하는 이들 알고리즘은 mode change 하에서 문제점을 갖는다. 이러한 문제점들을 해결하기 위한 최적의 퀀텀 크기 결정 방법이 제안된 바 있다[2]. 본 논문에서는 모든 태스크들에 대한 이용률이 $e \leq p/3+1$의 성질을 만족하는 제한적 특성을 갖는 태스크 집합에 대해서 최적의 퀀텀 크기를 결정하기 위한 이용률 계산의 반복 횟수를 보다 감소시킬 수 있는 방법을 제안한다.

■ 중심어 : | 실시간 스케줄링 | 다중프로세서 스케줄링 | Pfair 스케줄링 |

Abstract

Since the PF scheduling algorithm[13], which is optimal in the hard real-time multiprocessor environments, several scheduling algorithms have been proposed. All these algorithms assume the fixed unit quantum size, and this assumption has problems in the mode change environments. To settle the problem, we already proposed a method for deciding the optimal quantum size[2]. In this paper, we propose improved methods considering the task set whose utilization e is less than or equal to $p/3+1$. As far as the numbers of computations used to determine the optimal quantum size are concerned, newly proposed methods are proved to be more efficient than our previous ones.

■ keyword : | Real-Time Scheduling | Multiprocessor Scheduling | Pfair Scheduling |

I. 서론

실시간 시스템의 태스크들은 각각의 종료시한 내에 태스크들을 종료해야 하는 시간적 제약을 갖고 있다. Liu와 Layland에 의해 제안된 RMS와 EDS 알고리즘은 단일 프로세서 시스템 환경에서 최적의 스케줄링 알고리즘으로 증명되었지만[4], 다중 프로세서 환경에서는 최적이 아니다. 이 후 발표된 RMNFS와 RMFFS와 같은 알고리즘[15] 등도 역시 다중 프로세서 환경에서 최적의 알고리즘은 아니었다.

그러나 최근 Baruah 등은 다중 프로세서 환경에서 주기적 경성 실시간 태스크들에 대한 최적의 스케줄링

접수번호 : #070411-001
접수일자 : 2007년 04월 11일
심사완료일 : 2007년 06월 04일
교신저자 : 김인국, e-mail : igkim@dankook.ac.kr

알고리즘인 PF 알고리즘[13]과 이것을 효율적으로 개선한 PD[12] 알고리즘을 제안하였는데, 프로세서의 수가 M일 때 태스크 집합이 스케줄링 가능하기 위한 필요충분조건은 다음과 같음을 증명하였다. 여기서 T는 태스크 집합 τ에 속한 태스크를 의미하고 있으며, $T.e$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, PD2[6]와 ERfair[7] 알고리즘 등이 Anderson 등에 의해 제안되었으며, 이 밖에도 EPDF[8], QRfair [10], BF[5] 등 Pfair를 기반으로 하는 다수의 알고리즘들이 제안되었는데, 이러한 퀀텀 기반의 스케줄링 알고리즘들에서 태스크의 주기와 실행 요구 시간은 고정적인 퀀텀 크기의 배수로 가정하고 있다. 하지만, 태스크 집합의 대주기에서 태스크 집합이 새로운 태스크 집합으로 변경(mode change)되는 상황이나 동적 태스크 집합에 대해서는 스케줄링의 효율성 또는 스케줄링 가능성을 위해 퀀텀의 크기가 증가되거나 감소될 필요성이 있다.

퀀텀 크기를 증가시키게 되면 스케줄링 횟수를 감소시킬 수 있는데, 스케줄링 시점의 감소는 작업들에 대한 선점 횟수를 감소시킬 수 있으며 이로 인한 작업의 이동 횟수도 감소시킬 수 있다. 빈번한 태스크 교환은 문맥 교환에 드는 시간을 증가시키고 빈번한 태스크의 이동은 캐시 메모리에 대한 적중률을 낮춤으로서 이에 대한 캐시 메모리 재적재 횟수를 증가시키게 되므로, 퀀텀 크기의 증가는 문맥 교환과 캐시 재적재 시간과 같은 오버헤드를 감소시킴으로서 전반적인 시스템 효율을 향상시킬 수 있다. 그런데, 이렇게 퀀텀 크기가 증가된 상태에서 mode change가 발생하여 태스크 집합이 변경되면 이 퀀텀 크기로는 더 이상 스케줄링이 가능하지 않게 될 수 있다. 이런 경우 새로운 태스크 집합이 스케줄링 가능하도록 퀀텀 크기를 적절하게 감소시켜야 하는데, 스케줄링이 실패하지 않는 한도에서의 최대값으로 최적의 퀀텀 크기가 결정될 수 있다. 최근 이러한 최적의 퀀텀 크기를 찾기 위한 방법[2]이 제안되었는데, 이 방법에서는 각 태스크들의 주기와 실행 요구 시간의 특성에 따라 퀀텀의 크기를 증가시키면 이용률 1이 연속적으로 나온다는 특성을 이용해서 반복적으로 최적의 퀀텀 크기를 찾을 수 있었다.

본 논문에서는 이 방법을 기반으로 제한된 특성을 갖는 태스크 집합에 대해, 태스크 집합의 총 이용률 계산 횟수를 보다 감소시키면서도 항상 최적의 퀀텀 크기를 결정 할 수 있는 새로운 방법을 제안하도록 한다.

II. 태스크 모델

본 논문에서 고려하고 있는 스케줄링 기법은 전역 스케줄링[7][9][12]을 기반으로 하고 있으며, 프로세서 간 이동이 허용되는 태스크 집합은 구 스케줄(old schedule)의 대주기(major period) 마지막에서 모드 변경됨을 가정한다.

또한, 최적 퀀텀 크기의 결정을 위한 이전 연구[2]에서는 주기의 변경 허용성에 따라 태스크를 두 가지로 분류하였다. 퀀텀 크기의 변경 이후에도 원래의 주기를 엄격히 지켜야하는 태스크인 주기 불변 태스크와 주기의 변경이 허용되는 태스크인 주기 가변 태스크로 분류된다. 주기 불변 태스크에서 주기는 종료시한뿐만 아니라 태스크가 발생되어야 하는 시간적 제약을 모두 포함하며, 주기 가변 태스크는 각 작업의 발생 간격이 모드 변경 시점 전보다 짧아질 수 있음을 의미한다[2]. 여기서, 주기 가변 태스크는 태스크의 특성상 태스크의 매 작업(job)에 대한 주기가 변할 수 있는 가변 주기의 의미가 아니라, 퀀텀 크기의 변경에 따라 재계산된 태스크의 주기가 변경될 수 있음을 의미한다. 따라서 주기 가변 태스크는 퀀텀 크기가 변경되지 않는 한 각 작업은 동일한 주기를 갖게 된다.

$q_{\min}$은 시스템에서 표현할 수 있는 최소 퀀텀 크기로서 단위 시간 1을 의미한다. q_c는 시스템에 현재 적용중인 퀀텀 크기이고, q_n은 새롭게 선택되는 퀀텀의 크기이다.

e와 p는 $q_{\min}$의 배수로 표현되는 실행 요구 시간과

주기로서 가장 정밀하게 표현될 수 있는 태스크 본래의 값들이다. 또한, e_c와 p_c는 q_c에, e_n과 p_n은 q_n에 비례해서 정의되는 실행 요구 시간과 주기를 나타낸다. 이 때, 퀀텀 크기의 변경에 따라 주기가 증가하면 안 되고, 실행 요구 시간은 감소되지 않아야 하므로 e_n과 p_n은 다음과 같은 식으로 표현된다.

$$e_n = \left\lceil \frac{e}{q_n} \right\rceil \tag{2}$$

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

또한, w를 $q_{\min}$에 의해 계산되는 태스크의 프로세서 이용률(utilization)이라 하면, q_n에 의해 증가된 태스크의 이용률 w'은 다음과 같이 표현된다.

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

그러므로 퀀텀 크기가 q_n일 때 재계산되는 전체 프로세서 이용률 U는 다음과 같다.

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

III. mode change 환경에서의 퀀텀 크기 결정

퀀텀 크기의 결정 문제는 퀀텀 크기의 증가 또는 감소 여부에 관계없이 태스크가 본래 갖고 있는 e와 p 및 $q_{\min}$만을 기반으로 mode change된 현재 태스크 집합에 대해 $U \leq M$을 만족하는 최대 크기의 q_n을 찾는 문제로 단순화 될 수 있으므로, 이후에는 q_c와 q_n 대신 기호 Q를 사용하기로 한다.

기본적인 퀀텀 크기의 결정 문제는 모든 태스크들의 주기와 실행 요구 시간에 대해서 1이 아닌 공약수가 존재한다면, 공약수들 가운데 $U \leq M$을 만족하는 최대 공약수가 퀀텀의 크기로 선택될 수 있다. 이 경우, 모든 태스크들에 대해 어떠한 주기도 변하지 않고 전체 이용률 U도 변하지 않으므로 스케줄링이 가능하지만, 실제로 모든 태스크들의 주기와 실행 요구 시간에 대해 1이 아닌 공약수가 존재할 가능성은 매우 희박하므로 이러한 공약수가 존재하지 않을 수 있다는 가정 하에 새로운 퀀텀 크기를 결정하는 방법을 찾아야 하며, 주기 변화를 허용할 경우 주기 변경 허용 여부에 따라 이 값보다 큰 값에서 Q가 존재할 수 있다는 점도 고려해야 한다.

그런데, 모든 태스크들의 주기와 실행 요구 시간들을 이용하여 $U \leq M$을 만족하는 Q의 최대값을 계산할 수 있는 다항식 함수가 알려진 바가 없기 때문에, 본 논문에서는 반복적인 방법으로 적절한 퀀텀 크기를 결정하도록 한다. 따라서 $U \leq M$을 만족하는 최대의 Q를 구하기 위해 각 Q에 대한 U의 계산 과정을 반복하게 되므로 반복의 대상이 되는 Q 값의 범위를 최대한 줄이는 것이 관건이다.

다음으로, 주기 불변 태스크에 대한 문제이다. 주기 불변 태스크가 둘 이상 존재할 경우 p_n은 모든 주기 불변 태스크들의 주기에 대한 공약수들 중 $U \leq M$을 만족하는 최대의 Q로 선택되어야 한다.

마지막으로, 퀀텀 크기의 변경으로 각 태스크들의 주기가 변경됨에 따라 mode change할 시점인 대주기가 변경되는 문제이다. 대주기는 태스크들의 주기 특성에 따라 변경되지 않을 수도 있고 늘어나거나 줄어들 수도 있지만, mode change는 재계산된 대주기의 증감 여부에 상관없이 원래의 대주기 시점에 발생되도록 해야 한다. 따라서 모든 태스크들이 원래의 대주기 내에서 p에 따라 발생되는 작업의 수만큼 p_n에 의해서 발생되는 작업의 수도 보장되어야 하고, 대주기 내에서 모든 태스크들은 스케줄링 가능해야 하는데, Pfair 스케줄링 알고리즘에서는 선택된 Q 값에 의해 원래의 대주기 내에서도 모든 태스크들은 항상 스케줄링 가능함이 보여 졌다[2].

IV. FindQ()

본 장에서는 태스크의 이용률에 대한 제한이 없는 일반 태스크 집합에 대해서 최적의 퀀텀 크기를 찾기 위해 이전에 제안된 방법[2]에 대해 간단히 살펴보기로 한다.

Q의 크기를 증가시킴에 따라 각 태스크들의 w'는 1에 도달하게 된다($w \leq 1$이어야 하므로 $w' > 1$인 경우 $w' = 1$). Q의 증가에 따라 w'는 부분적으로 감소될 수 있으며 p와 e에 따라 w'가 1에 도달하는 순서에 차이가 발생한다. 하지만, Q를 계속 증가시키다보면 어느 시점부터는 w'가 계속 1의 값을 유지하게 되는데, 이렇게 w'가 연속적으로 1이 되는 최초의 Q를 실제 도달점 RRP이라 하고 실제 도달점을 포함한 이 후의 Q값들을 도달 구간이라 하자. 그런데, e와 p만을 이용해서 실제 도달점과 같거나 도달 구간 내에 존재하는 Q값을 찾을 수 있는 도달 함수 $Reach(p, e)$는 [정리 1]과 같으며, $Reach(e, p)$에 의해 얻어진 Q값을 도달점 RP라 하면 다음이 성립한다.

$$RP \geq RRP \tag{6}$$

[정리 1]

도달 함수 $Reach(e, p)$는 다음과 같이 정의한다.

$$Reach(e, p) = \begin{cases} \left\lfloor \frac{p}{2} \right\rfloor + 1 & \left(\text{if } \frac{e}{p} \leq \frac{1}{2}\right) \\ \left\lfloor \frac{p}{3} \right\rfloor + 1 & \left(\text{if } \frac{e}{p} > \frac{1}{2}\right) \end{cases} \tag{7}$$

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[2].

[정리 1]에 대한 증명은 앞서 제안한 논문[2]에 나와 있다.

각 태스크의 RP가 오름차순으로 저장된 리스트를 도달 순위 리스트 $Rank[n]$라 한다. 도달 순위 리스트의 순서에 따라 M개의 태스크가 도달 구간에 있다면 스케줄링은 실패하게 되므로, 최적의 Q값은 적어도 M번째 태스크의 도달점 보다 작은 구간에서 구해진다. 도달 순위 리스트에 $Reach() - 1$의 값이 저장된다고 하면, 최적의 퀀텀 크기는 다음 식을 만족하는 구간에 존재하게 된다.

$$1 \leq Q \leq Rank[M-1] \tag{8}$$

[알고리즘 1]은 도달 순위 리스트를 기반으로 최적의 퀀텀 크기를 찾기 위한 방법을 보여준다.

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

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

V. $e \leq \frac{p}{3}+1$인 태스크들에 대한 퀀텀 크기

앞 장에서 태스크의 이용률 w에 상관없이 항상 최적의 퀀텀 크기를 찾을 수 있는 방법에 대해 살펴보았다. 그런데 태스크 집합을 구성하고 있는 모든 태스크들에 대한 이용률이 어떤 값보다 작다면, 이러한 제한적인 특성을 갖는 태스크 집합에 대해서는 FindQ()에서의 이용률 계산 횟수를 감소시킬 수 있는데, 본 장에서는 이러한 FindQ() 함수를 제안한다.

$e \leq \frac{p}{3}+1$을 만족하는 즉, $w \leq \frac{1}{3}$인 태스크들에 대해 다음이 성립한다.

[정리 2]

$e \leq \frac{p}{3}+1$인 태스크에 대해

$$\frac{p}{3} < Q \leq \frac{p}{2} \tag{9}$$

이면

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

[증명]

식 (9)의 조건에 의해 $\left\lfloor \frac{p}{Q} \right\rfloor$는 다음과 같이 정리된다.

$$\frac{p}{3} < Q \leq \frac{p}{2} \quad (Q,\ p \neq 0)$$

$$\Rightarrow \quad \frac{2p}{p} \leq \frac{p}{Q} < \frac{3p}{p}$$

$$\Rightarrow \quad 2 \leq \frac{p}{Q} < 3$$

$$\Rightarrow \quad \left\lfloor \frac{p}{Q} \right\rfloor = 2 \tag{10}$$

$p=1$인 경우 $\frac{e}{p}$는 항상 1이므로 식 (10)을 만족하는 Q는 존재하지 않는다. $p=2$이면, $\frac{2}{3} < Q \leq \frac{2}{2}$이므로 조건을 만족하는 Q는 1이다.

따라서 $w' = \frac{\left\lceil \frac{e}{Q} \right\rceil}{\left\lfloor \frac{p}{Q} \right\rfloor} = \frac{\lceil e \rceil}{\lfloor p \rfloor} = \frac{e}{p} = \frac{e}{2}$이 된다. 그런데 $e \leq \frac{p}{3}+1$이어야 하고 이 조건을 만족하는 e는 1뿐이므로 $w' = \frac{1}{2}$이다.

$p=3$인 경우 $\frac{3}{3} < Q \leq \frac{3}{2}$을 만족해야 하는데 이를 만족하는 Q가 존재하지 않으므로 이 후 $p \geq 4$인 경우 $\left\lceil \frac{e}{Q} \right\rceil = 1$이 항상 성립함을 보이면 된다.

우선 $\frac{e}{Q}$가 가질 수 있는 값의 범위를 보자. $\frac{e}{Q}$의 최소값은 $e=1$이고 $Q=\frac{p}{2}$인 경우에 발생하므로 $\frac{e}{Q}=\frac{1}{p/2}=\frac{2}{p}$이다. 그런데, $p \geq 4$이므로 $\frac{e}{Q}$의 최소값 범위는 다음과 같다.

$$0 < \frac{e}{Q} \leq \frac{1}{2} \tag{11}$$

$\frac{e}{Q}$의 최대값은 e가 최대값을 Q가 최소값을 갖는 경우에 발생한다. 여기서, $e \leq \frac{p}{3}+1$이므로 e는 $\frac{p}{3}+1$보다 작거나 같은 최대 정수이어야 하고, 따라서 $e = \left\lfloor \frac{p}{3} \right\rfloor + 1$이 된다. 마찬가지로, 식 (9)의 Q에 대한 조건에 의해 Q는 $\frac{p}{3}$ 보다 큰 최소 정수 이므로 $Q = \left\lfloor \frac{p}{3} \right\rfloor + 1$이 된다. 따라서 $\frac{e}{Q}$의 최대값은 항상 1이 되므로 식 (11)의 최소값에 의해 $\left\lceil \frac{e}{Q} \right\rceil$ 값은 다음과 같다.

$$0 < \frac{e}{Q} \leq 1 \quad \Leftrightarrow \left\lceil \frac{e}{Q} \right\rceil = 1 \tag{12}$$

따라서 식 (10)과 식 (12)에 의해 [정리 2]는 $p \geq 4$에 대해 항상 성립한다.

태스크들은 두 가지의 도달점을 갖게 된다. 첫 번째는 [정리 1]에 의해 얻어진 도달점으로서 재계산된 이용률 $w'=1$이 연속으로 나오기 시작하는 Q의 값이며, 또 하나의 도달점은 [정리 2]에 의해 얻어진 도달점으로서 $w'=0.5$가 연속으로 나오기 시작하는 Q의 값인데, 이 두 도달점을 각각 FRP(Full-Reach Point)와 HRP(Half -Reach Point)라 하자. 또한, 두 도달점들을 계산하기 위한 도달 함수를 각각 $FReach(e, p)$와 $HReach(e, p)$라 하면 $FReach(e, p)$는 [정리 1]의

식 (7)에서 태스크의 이용률이 $\frac{1}{2}$이하인 경우에 해당한다.

또한, [정리 2]의 식 (9)에서 $\frac{p}{3} < Q$인 최소의 Q는 $\left\lfloor \frac{p}{3} \right\rfloor + 1$이므로 $FReach(e, p)$와 $HReach(e, p)$는 [정리 1]과 [정리 2]에 의해 다음과 같이 정의된다.

$$\begin{cases} FReach(e,p) = \left\lfloor \frac{p}{2} \right\rfloor + 1 \\ HReach(e,p) = \left\lfloor \frac{p}{3} \right\rfloor + 1 \end{cases} \quad (13)$$

표 1. τ_1에 대한 w와 프로세서 이용률 변화

Q	U	w'				
		T1	T2	T3	T4	T5
1	1.349	0.2222	0.2667	0.2857	0.2571	0.3171
4	2.008	0.5000	0.3333	0.4000	0.3750	0.4000
5	2.494	1.0000	0.3333	0.5000	0.2857	0.3750
6	2.733	1.0000	0.5000	0.3333	0.4000	0.5000
7	2.633	1.0000	0.5000	0.3333	0.4000	0.4000
8	3.400	1.0000	1.0000	0.5000	0.5000	0.4000
9	3.333	1.0000	1.0000	0.5000	0.3333	0.5000
10	3.333	1.0000	1.0000	0.5000	0.3333	0.5000
11	4.000	1.0000	1.0000	1.0000	0.3333	0.6667
12	4.167	1.0000	1.0000	1.0000	0.5000	0.6667
13	3.833	1.0000	1.0000	1.0000	0.5000	0.3333
14	4.000	1.0000	1.0000	1.0000	0.5000	0.5000
15	4.000	1.0000	1.0000	1.0000	0.5000	0.5000
16	4.000	1.0000	1.0000	1.0000	0.5000	0.5000
17	4.000	1.0000	1.0000	1.0000	0.5000	0.5000
18	4.500	1.0000	1.0000	1.0000	1.0000	0.5000
19	4.500	1.0000	1.0000	1.0000	1.0000	0.5000
20	4.500	1.0000	1.0000	1.0000	1.0000	0.5000
21	5.000	1.0000	1.0000	1.0000	1.0000	1.0000

또한 FRP를 기반으로 최적의 Q 값을 찾기 위해 이전 연구에서 제안된 FindQ() 함수와 본 논문에서 새로 제안하고 있는 FindQ()를 구분하기 위해 이제부터 각각을 FindQF()과 FindQHF()라 하자. FindQF()에서는 태스크의 수를 N이라 할 때 각 태스크의 FRP-1 값이 오름차순으로 기록된 리스트 $Rank[N]$을 기반으로 전체 이용률 계산의 반복에 대한 초기 Q값을 지정하였다. 즉, 프로세서의 수를 M ($M < N$)이라 하면 최초 전체 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 사이즈를 찾아나가는 방식이다.

즉, FRP만을 이용하여 이용률 계산의 시작점을 결정하는데, 퀀텀의 크기가 증가함에 따라 어떤 태스크의 이용률이 1에 도달함으로서 태스크 집합이 스케줄링에 실패하게 된다면 이 도달점 이하에서 최적의 퀀텀 크기를 찾아나가게 된다.

태스크 집합 $\tau_1 = \left\{\frac{2}{9}, \frac{4}{15}, \frac{6}{21}, \frac{9}{35}, \frac{13}{41}\right\}$에 대한 w'의 증가 모습을 보여주고 있는 [표 1]의 예에서 보면, 프로세서의 수가 3인 경우 기존의 연구에 의하면 퀀텀 크기가 18일 때 T_1, T_2, T_3, T_4가 모두 1의 이용률을 가지므로 최적 퀀텀 퀀텀의 크기는 17로부터 찾아나가게 된다. 그러나 퀀텀 크기가 14에서 17인 구간에서 T_5와 T_6의 이용률은 0.5이고 이 두 태스크의 이용률 합은 1이므로 실제 최적 퀀텀 크기는 14보다 작은 구간에 존재함을 알 수 있다. 따라서 최적의 퀀텀 크기를 찾기 위해서는 퀀텀 크기가 13인 경우부터 시작되어야 한다.

이에 반해, 본 논문에서 제안하는 FindQHF()는 [정리 2]의 특성에 따라 전체 이용률 계산의 횟수를 감소시키기 위해 각 태스크의 FRP와 HRP 값 모두를 기반으로 최적의 퀀텀 크기를 찾도록 하고 있다. 즉, 도달점에 도달한 하나의 태스크만을 고려하여 이용률 계산의 시작점을 결정했던 이전 연구와는 달리, 본 연구에서는 두 태스크에 대한 이용률 합도 1에 도달한다는 점을 고려하여 이용률 계산의 시작점을 결정하게 된다.

따라서 본 논문에서는 모든 태스크의 FRP와 HRP 값이 오름차순으로 기록된 $Rank[2N]$으로 도달 순위 리스트를 재정의 한다. 여기서 특이할만한 점은 FindQF()에서와 같이 FRP-1과 HRP-1의 값이 기억되는 것이 아니라 FRP와 HRP 값이 기억된다는 것인데, 이는 설명의 이해를 돕기 위할 뿐이지 의미상의 차이는 없다. [표 2]는 τ_1에 대한 도달 순위 리스트를 보여준다.

표 2. τ_1에 대한 도달 순위 리스트

i	0	1	2	3	4	5	6	7	8	9
Rank[i]	4	5	6	8	8	11	12	14	18	21
Umin	0.5	1	1.5	2.5	2.5	3	3.5	4	4.5	5
Task	T1	T1	T2	T2	T3	T3	T4	T5	T4	T5

[표 2]에서 $U_{\min}$은 해당 도달점에서 가질 수 있는 총 이용률의 최소값을 의미하며 $Rank[i]$에서의 $U_{\min}$을 $Rank[i].U_{\min}$으로 표기한다. $Rank[0]$는 항상 최소의 HRP 값을 가지며, 도달 순위 리스트 내에 중복된 도달점이 존재하지 않는다면 인덱스 i가 1 증가함에 따라 $U_{\min}$도 0.5씩 증가하게 된다. 그러나 도달 순위 리스트 내에 동일한 도달점들이 존재한다면 도달 순위 리스트는 다음과 같은 성질을 갖는다.

[보조정리 1]

$Rank[i]$로부터(포함하여) k개의 동일한 HRP 또는 FRP가 존재한다면($k \geq 2$인 경우는 중복이 발생한 것을 의미한다.), $Rank[i]$부터 $Rank[i+k-1]$의 $U_{\min}$은 $Rank[i-1].U_{\min} + \frac{k}{2}$이다.

[증명]

$Rank[i]$에서 i가 1씩 증가함에 따라 $U_{\min}$은 0.5씩 증가하므로 i가 k만큼 증가하게 되면 $U_{\min}$은 $\frac{k}{2}$만큼 증가하게 되므로 정리가 성립한다.

[보조정리 2]

N은 태스크의 수이고 m은 $(1 \leq m \leq N-1)$인 정수라 할 때, $Rank[2m-1]$과 동일한 도달점이 존재하지 않는다면, 즉, $Rank[2m-1] \neq Rank[2m]$이면 다음이 항상 성립한다.

$$m = Rank[2m-1].U_{\min} \tag{14}$$

[증명]

$Rank[0].U_{\min} = 0.5$로부터 $Rank[2m-2]$ 까지는 도달점의 중복 여부에 관계없이 총 $2m-1$개의 도달점이 존재하므로, $Rank[2m-2].U_{\min} = m-0.5$이다. 그런데, $Rank[2m-1] \neq Rank[2m]$이므로, [보조정리 1]에 의해

$$\begin{aligned} Rank[2m-1].U_{\min} &= Rank[2m-2].U_{\min} + \frac{1}{2} \\ &= m - \frac{1}{2} + \frac{1}{2} = m \end{aligned}$$

이다. 따라서 정리가 성립한다.

[보조정리 3]

$Rank[2m-1]$ 이후로 중복된 도달점이 적어도 하나 이상 존재하면, 정수 $m(1 \leq m \leq N-1)$에 대해 다음이 항상 성립한다.

$$m < Rank[2m-1].U_{\min} \tag{15}$$

[증명]

[보조정리 2]에 의해 $Rank[2m-1]$과 중복된 도달점이 존재하지 않을 경우 $Rank[2m-1].U_{\min} = m$이다. 그러나 $Rank[2m-1]$ 부터 $k(k \geq 2)$개의 동일한 도달점이 존재하게 된다면

$$\begin{aligned} Rank[2m-1].U_{\min} &= Rank[2m-2].U_{\min} + \frac{k}{2} \\ &= m - \frac{1}{2} + \frac{k}{2} = m + \frac{k-1}{2} \end{aligned}$$

이다. 따라서 $m + \frac{k-1}{2} > m\,(k \geq 2)$이므로 정리가 성립한다.

최적의 퀀텀 크기를 Q_{opt}라 하면, 도달 순위 리스트를 기반으로 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 하고, 도달 순위 리스트에 중복된 도달점이 존재하지 않는다고 가정하면, [보조정리 2]에 의해 sp_M은 일단 다음 식과 같이 선택될 수 있다.

$$sp_M = Rank[2M-1] \quad (16)$$

이제 임의의 sp_M은 $Rank[2M-1] = Rank[2M]$인 경우와 $Rank[2M-1] \neq Rank[2M]$인 경우만 존재하므로 이 두 경우에 대해 다음이 성립한다.

[보조정리 4]

$e \leq \frac{p}{3}+1$인 태스크에 대해 프로세서의 수가 M일 때, $Rank[2M-1] = Rank[2M]$이면 Q_{opt}는 $Rank[2M]$보다 작은 값으로 존재한다.

[증명]

[보조정리 3]에 의해 $Rank[2M-1].U_{\min} > M$이고 $Rank[2M-1] = Rank[2M]$이므로 보조정리가 성립한다.

[보조정리 5]

$e \leq \frac{p}{3}+1$인 태스크에 대해 프로세서의 수가 M일 때, $Rank[2M-1] \neq Rank[2M]$이면 Q_{opt}는 $Rank[2M]$보다 작은 값으로 존재한다.

[증명]

[보조정리 2]에 의해 $Rank[2M-1].U_{\min} = M$이다. 그런데, $Rank[2M-1] \leq Q < Rank[2M]$인 Q의 구간에서 $Rank[2M-1].U_{\min}$에 포함되지 않은 w'를 갖는 태스크들이 존재할 수도 있고 존재하지 않을 수도 있다. $Rank[2M-1].U_{\min}$에 포함되지 않은 w'를 갖는 태스크들이 적어도 하나 이상 존재하게 된다면 w'는 적어도 0보다 크므로 $Rank[2M-1]$에서의 실제 이용률은 $Rank[2M-1].U_{\min}$보다 크다. 따라서 이런 경우 Q_{opt}는 $Rank[2M-1]$보다 작은 값으로 존재한다. 그러나 $Rank[2M-1].U_{\min}$에 포함되지 않은 w'가 하나도 없다면 $Rank[2M-1]$에서의 실제 이용률은 M이고 $Rank[2M]$보다 작은 구간에서 그 값이 유지된다. 또한, $Rank[2M].U_{\min} > M$이므로 Q_{opt}는 $Rank[2M]$보다 작은 값으로 존재하게 된다. 결국, 두 경우를 포함해야 하므로 Q_{opt}는 $Rank[2M]$보다 작은 값으로 존재한다.

[정리 3]

$e \leq \frac{p}{3}+1$인 태스크에 대해 프로세서의 수가 M이면 도달점의 중복에 관계없이 Q_{opt}는 $Rank[2M]$보다 작은 값으로 존재한다.

[증명]

[보조정리 4]와 [보조정리 5]로 부터 정리가 성립한다.

이제 [정리 3]을 기반으로 하여 FindQHF() 함수를 구현할 수 있는데 이에 대한 알고리즘은 다음과 같다.

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

알고리즘 2. FindQHF() 함수

VI. 실험 및 분석

본 장에서는 FindQF()와 FindQHF()에 의한 결과를 비교한다. 먼저, $p_{\max}$로부터 반복적으로 프로세서 이용률을 계산하여 얻어진 최적의 퀀텀 크기와 FindQHF()에 의해 구해진 퀀텀의 크기가 일치하는지 확인해보고, FindQF()와 FindQHF()에서 수행된 프로세서 이용률의 계산 횟수로 두 알고리즘에 대한 효율성을 비교해 본다. 또한, 제안된 FindQHF() 알고리즘이 프로세서의 수에 따라 갖는 특성을 분석해 본다.

본 실험은 Windows XP가 설치된 x-86 기반의 플랫폼에서 Visual C++ 6.0을 이용, C 언어로 작성된 프로

그램을 통해 수행되었다. 실험에서는 $e \leq \frac{p}{3}+1$이고 최대 주기가 1,000인 태스크들을 전체 이용률이 3을 넘지 않도록 태스크 집합 당 20개씩 무작위로 생성하였다. 이렇게 무작위로 생성된 100,000개의 태스크 집합들에 대해 프로세서의 수가 $3 \leq M \leq 19$인 경우[1]의 프로세서 이용률 계산 횟수를 비교하였다. 이 실험에서 생성된 태스크 집합의 평균 프로세서 이용률은 2.77이다.

표 3. $M=10$인 경우의 실험 결과

항목 / 방법	평균 이용률 계산 횟수	실패 횟수	차이 횟수	평균 퀀텀 크기
FindOpt()	816.944	0	0	530.38
FindQF()	103.307	0	0	530.38
FindQHF()	68.774	0	0	530.38

먼저, [표 3]은 프로세서의 수가 10인 경우의 결과를 보여준다. 표에서 FindOpt()는 $p_{\max}$로부터 반복적으로 Q_{opt}를 찾는 방법으로서 항상 최적의 퀀텀 크기를 구할 수 있기 때문에 FindQF()와 FindQHF()가 최적의 퀀텀 크기를 찾을 수 있는지 비교하는데 사용된다. 다음으로 평가 항목을 보면, 평균 이용률 계산 횟수는 각 방법에서 태스크 집합 당 Q_{opt}를 찾기 위해 반복된 프로세서 이용률 계산의 평균 횟수로서 0에 가까울수록 효율이 좋다고 판단된다. 다음으로 실패 횟수는 1보다 큰 Q_{opt}가 존재함에도 불구하고 그 값을 찾지 못한 경우의 총 횟수를 나타내고 있으며, 차이 횟수는 1보다 큰 Q는 찾아냈지만 $Q < Q_{opt}$인 경우의 총 횟수를 의미한다. 따라서 차이 횟수가 0이라는 것은 모든 태스크 집합에 대해 최적의 퀀텀 크기를 결정할 수 있었음을 의미하고, 차이 횟수가 0이면 실패 횟수는 당연히 0이 된다. 또한 차이 횟수가 0보다 크다는 것은 비록 최적 퀀텀 크기는 아니지만 스케줄링 시점을 감소시킬 수 있는 퀀텀 크기를 찾았다는 것을 의미하게 된다.

이 실험의 결과를 보면 FindQHF()는 차이 횟수의 결과가 0이므로 모든 태스크 집합에 대해 최적의 퀀텀 크기를 찾을 수 있으며, 평균 이용률 계산 횟수의 결과를 보면 FindQF()에 비해 약 33.5% 정도 이용률 반복 횟수가 감소된 것을 볼 수 있다.

또한 [그림 1]은 프로세서의 수가 $3 \leq M \leq 19$인 경우에 대해 실험한 평균 이용률 계산 횟수를 비교한 그래프이며, [그림 2]는 FindQF()에 대한 FindQHF()의 이용률 계산 횟수 비율을 보여주고 있다.

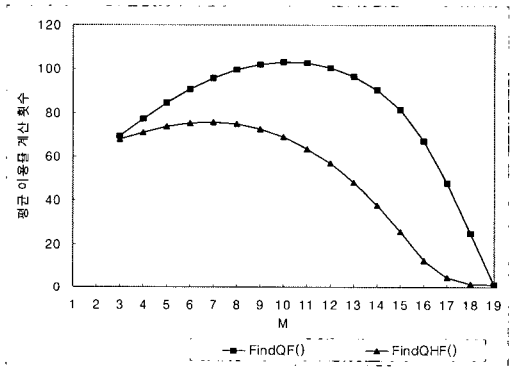


그림 1. 프로세서 수에 따른 평균 이용률 계산 횟수

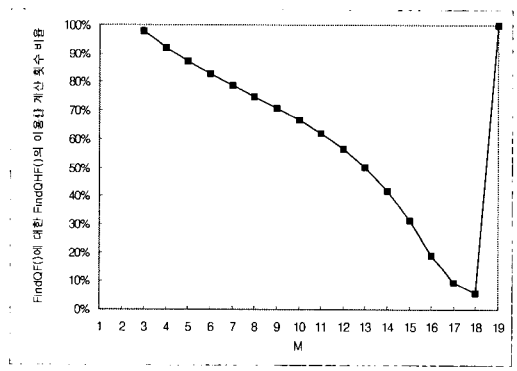


그림 2. FindQF()에 대한 FindQHF()의 이용률 계산 비율

이 실험 결과에서도 FindQHF()는 모든 프로세서의 수에 대해 항상 최적의 퀀텀 크기를 찾을 수 있었다. 위 그래프를 보면 프로세서의 수가 많을수록 FindQHF()는 FindiQF()에 비해 적은 이용률 계산 횟수의 결과를 보이고 있다. 특정 도달점으로부터의 이용률 계산의 반복 횟수에 영향을 미치는 것은 이 퀀텀 크기로 재계산된 이용률이 아직 도달점에 도달하지 않은 태스크들의

1) 생성된 태스크 집합의 프로세서 이용률은 $2 \leq U < 3$이므로 프로세서의 수가 3보다 작은 경우 태스크 집합은 퀀텀 크기에 상관없이 스케줄링에 실패하게 된다. 또한, 태스크 집합은 20개의 태스크로 구성되므로 프로세서의 수가 20 이상이면 태스크 집합은 항상 스케줄링 가능하기 때문이다.

이용률이다. 따라서 프로세서의 수가 증가할수록 도달점에 도달한 태스크들의 수도 많아지므로 이용률 계산 횟수는 감소될 수 있다. 또한, FindQHF()는 FindQF()보다 세밀한 HRP를 추가한 도달 순위 리스트를 기반으로 이용률 계산의 시작점을 결정하므로 도달점에 도달하지 않은 태스크의 수를 감소시킬 수 있기 때문에 FindQHF()는 FindQF()보다 이용률 계산 횟수를 감소시킬 수 있다. 그런데 실험 결과에서 프로세서의 수가 19인 경우와 같이 프로세서의 수가 태스크의 수보다 1이 적은 경우 FindQHF()와 FindQF()의 이용률 계산 횟수는 거의 유사한 결과를 보이게 된다. 이것은 두 알고리즘에 대한 도달 순위 리스트의 각 도달점에서 아직 도달점에 도달하지 않은 태스크의 수가 많아야 1이기 때문에 이런 결과를 보이게 된다.

VII. 결론 및 향후 연구

본 논문에서는 모든 태스크들이 $e \leq \frac{p}{3}+1$인 태스크 집합에 대해 최적의 퀀텀 크기를 찾기 위한 이용률 계산 횟수를 이전 논문에서 제안한 방법인 FindQF()에 비해 감소시킬 수 있는 방법을 제안하였다. 또한 실험을 통해 본 논문에서 제안한 방법이 모든 프로세서 수에 대해서도 이전 연구에서 제시되었던 방법에 비해 이용률 계산 횟수를 감소키면서 항상 최적의 퀀텀 크기를 결정할 수 있음을 확인했다.

따라서 이전 연구에 비해 본 연구가 갖는 의미는 다음과 같다. 첫째, 퀀텀 크기의 증가에 따른 이용률 증가의 특성을 분석하여 이용률 0.5에 대한 새로운 도달점 HRP가 존재함을 파악하고 이에 대한 수식을 제시하고 이를 증명하였다. 둘째, 하나의 태스크에 대한 도달점만을 고려하여 이용률 계산의 시작점을 결정했던 이전 연구와 달리, 도달 리스트에 HRP를 추가하여 이용률 계산의 시작점 결정에 둘 이상의 태스크에 대한 도달점을 고려했다는 점이다. 결론으로 반복적으로 구해질 수밖에 없는 최적 퀀텀 크기 결정 문제에서 이용률 계산의 반복 횟수를 기존 연구에 비해 감소시킬 수 있었다.

본 연구에서 도달점은 최적 퀀텀 크기를 결정하기 위한 기준점이 된다. 즉, 이 도달점은 최적 퀀텀 크기를 결정하기 위한 충분조건에 불과하다. [그림 2]의 실험 결과를 보면 프로세서의 수가 작은 경우 프로세서의 수가 비교적 큰 경우에 비해 많은 이용률 계산이 반복되었음을 볼 수 있다. 이것은 충분조건으로서의 도달점 이외에도 최적 퀀텀 크기 결정을 위한 프로세서 이용률 계산의 반복 횟수를 감소시킬 수 있는 추가적인 태스크 집합의 특성들이 존재할 수 있다는 것으로 판단될 수 있다.

따라서 도달 함수 이외에 추가적으로 프로세서 이용률 계산을 감소시킬 수 있는 태스크 집합의 특성에 대한 지속적인 분석이 필요하며, 이 밖의 제한적 특성을 갖는 태스크 집합에 대한 최적의 퀀텀 크기의 결정 문제도 연구의 대상이다.

참 고 문 헌

[1] 김인국, *흐름 공정 모델의 효율적인 실시간 스케줄링*, 아주대학교 박사학위 논문, 1995.

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

[3] A. Srinivasan, P. Holman, J. Anderson, and S. Baruah, "The case for fair multiprocessor scheduling," Manuscript, Nov. 2002.

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

[5] D. Zhu, D. Mosse, and R. Melhem, "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.

[6] J. Anderson and A. Srinivasan, "A New Look at

Pfair priorities," Technical report, Dept of Computer Science, Univ. of North Carolina, 1999.

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

[8] 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.

[9] J. Anderson and A. Srinivasan, "Mixed Pfair/ERfair scheduling of asynchronous periodic tasks," Proceedings of the 13th Euromicro Conference on Real-time Systems, pp.76-85, June 2001.

[10] 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.

[11] K. W. Tindell, A. Burns, and A. J. Willings, "Mode Change in Priority Preemptively Scheduled Systems," IEEE Real-Time Systems Symposium, 1992.

[12] 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.

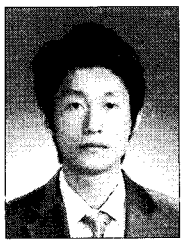
[13] S. Baruah, N. Cohen, C. G. Plaxton, and D. Varvel, "Proportionate Progress: A notion of fairness in resource allocation," Algorithmica, Vol.15, pp.600-625, 1996.

[14] S. Davari and C. L. Liu. "An On-Line Algorithm for Real-Time Tasks Allocation," IEEE Real Time Systems Symposium, pp.194-200, 1986.

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

저 자 소 개

차 성 덕(Seong-Duk Cha) 정회원



- 1999년 : 단국대학교 전자계산학과(이학사)
- 2001년 : 단국대학교대학원 전자계산학과(이학석사)
- 2004년 : 단국대학교대학원 전자계산학과(박사수료)

<관심분야> : 운영체제, 실시간시스템, 임베디드 시스템

김 인 국(In-Guk Kim) 정회원

- 1982년 : 단국대학교(학사)
- 1985년 : 미국 에모리대학교(석사)
- 1995년 : 아주대학교(박사)
- 1986년 ~ 현재 : 단국대학교 컴퓨터학부 컴퓨터과학전공 교수
- 2001년 ~ 2003년 : 미국 뉴멕시코 공과대학 방문교수

<관심분야> : 운영체제, 실시간시스템, 임베디드 시스템