Mode Change 환경에 적합한 동적 퀀텀 크기 스케줄링

Dynamic Quantum-Size Pfair Scheduling In the Mode Change Environments

차성덕, 김인국
단국대학교 공대 컴퓨터학부

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

요약

최근 다중 프로세서 환경에서 경성 실시간 태스크 집합의 스케줄링 문제를 해결하는 최적 Pfair 알고리즘이 Baruah 등에 의해 제안되었으며, 이를 기반으로 하는 여러 가지 스케줄링 알고리즘들이 제안되었다. 이들 알고리즘은 단위 크기의 고정된 퀀텀 크기를 기반으로 태스크들을 스케줄링하는데, 고정된 퀀텀 크기는 태스크 집합이 변경되는 mode change 하의 스케줄링에서 두 가지 문제점을 갖는다. 퀀텀이 너무 크면 프로세서의 이용률 저하로 인해 스케줄링이 실패할 수 있으며, 반대로 너무 작으면 스케줄링 빈도수의 증가에 따른 태스크 전환 오버헤드의 문제점을 갖게 된다. 본 논문에서는 mode change 환경에서 태스크 집합이 스케줄링 가능하도록 최대의 퀀텀 크기를 결정하기 위한 방법을 제안한다.

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

Abstract

Recently, Baruah et.al. proposed an optimal Pfair scheduling algorithm in the hard real-time multiprocessor environments, and several variants of it were presented. All these algorithms assume the fixed unit quantum size, and this assumption has two problems in the mode change environments. If the quantum size is too large, it results in the scheduling failure due to the decreased processor utilization. If it is too small, it increases the frequency of scheduling points, and it incurs the task switching overheads. In this paper, we propose several methods that determine the maximum quantum size dynamically such that the task set can be scheduled in the mode change environments.

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

I. 서 론

실시간 시스템의 태스크들은 각각의 종료시한 내에 태스크들을 종료해야 하는 시간적 제약을 갖고 있다. 단일 프로세서 시스템 환경에서 Liu와 Layland에 의해 제안된 Rate-Monotonic Scheduling(RMS) 알고리즘과 Earliest Deadline Scheduling(EDS) 알고리즘은 최적의 스케줄링 알고리즘으로 증명되었다[3].

이후 급속한 하드웨어와 네트워크 기술의 발전으로 인해 다중 프로세서 환경이 요구되었고, RMS나 EDS는

접수번호 : #060803-001
접수일자 : 2006년 08월 03일
심사완료일 : 2006년 09월 07일
교신저자 : 차성덕, e-mail : chastone@dankook.ac.kr

다중 프로세서 환경에서 최적이 아님이 쉽게 보여 졌다. 따라서, 다중 프로세서 환경에 적합한 실시간 스케줄링 알고리즘의 필요성이 제기되었고, RMNFS와 RMFFS와 같은 알고리즘[14] 등이 발표되었지만, 이 알고리즘들도 다중 프로세서 환경에서 최적의 알고리즘은 아니었다.

그러나 최근 Baruah 등은 다중 프로세서 환경에서 주기적 경성 실시간 태스크들에 대한 최적의 스케줄링 알고리즘인 Pfair scheduling(PF)[12] 알고리즘과 이것을 효율적으로 개선한 PD[11] 알고리즘을 제안하였다. 이 논문을 통해 Baruah 등은 프로세서의 수가 M일 때 태스크 집합이 스케줄링 가능하기 위한 필요충분조건은 다음과 같음을 증명하였다. 여기서, $T.e$와 $T.p$는 태스크 집합 τ에 속한 태스크 T의 실행 요구 시간과 주기를 나타낸다.

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

또한, Anderson 등은 기본 PF에 비해 태스크의 우선순위 결정에 필요한 파라미터의 수를 두 개로 감소시킨 PD2[5]와 Intra-sporadic 태스크를 고려한 ERfair[6] 알고리즘 등을 제안하였다. 이 밖에도 EPDF[7], QRfair[9], BF[4] 등 Pfair를 기반으로 하는 다수의 알고리즘들이 제안되었지만, 이러한 퀀텀 기반의 스케줄링 알고리즘들에서 주기 및 실행 요구 시간은 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 각 태스크의 실행 요구 시간 및 주기에 따른 퀀텀의 크기 변경이 요구될 수 있다. 예를 들어, 변경된 태스크 집합에 대한 모든 태스크들의 주기 및 실행 요구 시간이 퀀텀 크기의 정수 배수 n일 경우 퀀텀의 크기를 n으로 증가시킬 수 있다.

퀀텀 크기의 증가는 스케줄링 횟수를 감소시킬 수 있다. 태스크의 각 작업(job)에 대한 프로세서간 이동(migration)이 허용되는 시스템 하에서, 스케줄링 시점의 감소는 작업들의 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 된다. 결과적으로 퀀텀 크기의 증가는 문맥 교환과 캐시 재적재 시간과 같은 오버헤드를 감소시킴으로서 전반적인 시스템 효율을 향상시킬 수 있다. 하지만, 태스크 교환 간격의 증가로 인해 응답 시간이 증가될 수 있으며 프로세서 이용률 증가로 스케줄링이 실패할 수도 있다.

만약, 퀀텀 크기가 증가된 상태에서 mode change가 발생하여 더 이상 스케줄링이 가능하지 않게 된다면 퀀텀 크기를 이전 상태 또는 그 이하로 감소시켜야 한다. 주어진 태스크 집합에 대해 최적의 퀀텀 크기는 스케줄링이 실패하지 않는 한도 내에서 최대로 결정하는 문제인데 본 논문에서는 이러한 퀀텀 크기의 결정 방법을 제안하고자 한다.

II. 스케줄링 환경 및 태스크 모델

다중 프로세서 상에서 주기적 태스크 집합의 스케줄링 방법은 분할(partitioning)과 전역(global) 스케줄링으로 분류할 수 있다. 분할 스케줄링 방법에서 각 태스크는 한 프로세서에 고정 할당되어 실행된다. 이러한 스케줄링 방법은 다중 프로세서 스케줄링 문제를 단일 프로세서 스케줄링의 문제로 간소화시킨다는 장점을 갖지만, 프로세서들에게 태스크들을 최적으로 할당하는 방법은 NP-hard한 bin-packing 문제이다[2][13]. 전역 스케줄링에서는 스케줄링에 적합한 모든 태스크들이 단일 우선순위 큐에 저장되고 이 큐로 부터 최상위 우선순위의 태스크들이 전역 스케줄러에 의해 선택되어 실행된다. Pfair 스케줄링[12]을 기반으로 하는 스케줄러들이 이 범주에 포함된다[6][8][11]. 본 논문에서 고려하고 있는 스케줄링 기법은 전역 스케줄링을 기반으로 하고 있으며, 프로세서간 이동이 허용되는 태스크 집합은 구 스케줄(old schedule)의 대주기(major period) 마지막에서

모드 변경됨을 가정한다. 대주기는 태스크 주기들의 최소공배수로서 모든 태스크들은 스케줄링 순환 마지막 부분에서 동시에 시작될 수 있다.

실시간 시스템에서 태스크들은 일정한 주기를 가지고 스케줄링되며 일반적으로 각 태스크의 종료시한은 태스크의 주기와 동일함을 가정하고 있다. 태스크의 주기는 두 가지 의미를 갖게 되는데, 각 작업이 발생되어야하는 시간적 간격과 각 작업이 발생된 이후 실행을 완료해야 하는 시간적 제약을 의미한다.

본 논문에서는 주기의 변경 허용성에 따라 태스크를 두 가지로 분류한다. 퀀텀 크기의 변경 이후에도 원래의 주기를 엄격히 지켜야하는 태스크인 주기 불변 태스크와 주기의 변경이 허용되는 태스크인 주기 가변 태스크로 분류된다. 주기 불변 태스크에 대한 예로는 정확한 시간에 작동되어야 하는 센서 태스크나 동기적 통신을 수행해야 하는 태스크들이 있다. 이러한 태스크의 주기는 종료시한뿐만 아니라 태스크가 발생되어야 하는 시간적 제약을 모두 포함한다.

주기 가변 태스크는 각 작업의 발생 간격이 모드 변경 시점 전보다 짧아질 수 있음을 의미한다. 이러한 태스크들의 예는 멀티미디어 자료의 전송과 처리를 수행하는 태스크들을 들 수 있다. 동영상 재생기는 초당 33 프레임을 재생해야한다고 가정했을 때, 수신측에서 끊김 없이 동영상을 재생하기 위해서 송신측은 초당 최소 33 프레임을 전송해야 한다. 주기는 종료시한을 의미하게 되는데 이는 어떠한 작업이 처리될 수 있는 최소한의 시간적 보장이다. 이러한 시간적 보장을 만족시키기 위해 전송측 태스크의 주기는 1초, 실행 요구 시간은 33 프레임을 전송하기 위해 필요한 시간으로 결정될 수 있다. 만약, 수신측의 수신 버퍼 용량과 소비 속도에 대한 제한을 고려하지 않는다면, 전송측 시스템의 퀀텀 크기 증가로 인한 주기의 감소 문제는 수신측 동영상 재생에 아무런 영향을 미치지 않게 된다. 물론, 수신측 버퍼의 용량과 소비 속도를 감안한다면 송신측에서도 주기의 감소 한계를 결정해야 할 것이다. 하지만, 본 논문에서는 이러한 제한은 없다고 가정한다.

III. 기호 및 용어

본 장에서는 논문에서 이후에 사용하게 될 주요 기호와 용어 및 관련 식들에 대해 정의하기로 한다. 우선 퀀텀 크기와 관련하여 $q_{\min}$은 시스템에서 표현할 수 있는 최소 퀀텀 크기로서 단위 시간 1을 의미한다. q_c는 시스템에 현재 적용 중인 퀀텀 크기이고, q_n은 새롭게 선택되는 퀀텀의 크기로서 q_n이 결정되면 $q_c = q_n$이 된다. 이러한 퀀텀 기호들에 대해서 다음이 항상 성립한다.

$$q_{\min} \leqq q_c \ \wedge \ q_{\min} \leqq q_n \qquad (2)$$
$$q_n \leqq q_c \ \wedge \ q_n \geqq q_c$$

각 태스크는 주기와 실행 요구 시간으로 설명될 수 있다. 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.

이때, 퀀텀 크기의 변경에 따라 주기가 증가하면 안되고, 실행 요구 시간은 감소되지 않아야 하므로 e_n과 p_n은 다음과 같은 식으로 표현된다.

$$e_n = \left\lceil \frac{e}{q_n} \right\rceil \qquad (3)$$

$$p_n = \begin{cases} \left\lfloor \frac{p}{q_n} \right\rfloor & (\text{if} \quad q_n < p) \\ 1 & (\text{if} \quad q_n \geqq p) \end{cases} \qquad (4)$$

특정 퀀텀 크기 q에 비례해서 표현된 시간 x를 실제 시간으로 나타내는 함수를 $rt(x)$라 하면 e_c와 p_c 및 e_n과 p_n은 각각 다음과 같이 정의되고,

$$rt(e_c) = e_c \cdot q_c \text{ 과 } rt(p_c) = p_c \cdot q_c \qquad (5)$$
$$rt(e_n) = e_n \cdot q_n \text{ 과 } rt(p_n) = p_n \cdot q_n \qquad (6)$$

$rt(e_n)$과 $rt(p_n)$은 다음과 같다.

$$rt(e_n) = e_n \cdot q_n = \left\lceil \frac{e}{q_n} \right\rceil \cdot q_n \quad (7)$$

$$rt(p_n) = p_n \cdot q_n = \begin{cases} \left\lfloor \frac{p}{q_n} \right\rfloor \cdot q_n & (\text{if} \quad q_n < p) \\ q_n & (\text{if} \quad q_n \geqq p) \end{cases} \quad (8)$$

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

$$w' = \frac{e_n}{p_n} = \begin{cases} \frac{\left\lceil \frac{e}{q_n} \right\rceil q_n}{\left\lfloor \frac{p}{q_n} \right\rfloor q_n} = \frac{\left\lceil \frac{e}{q_n} \right\rceil}{\left\lfloor \frac{p}{q_n} \right\rfloor} & (\text{if } q_n < p) \\ 1 & (\text{if } q_n \geqq p) \end{cases} \quad (9)$$

그러므로 퀀텀 크기가 q_n일 때 재계산되는 전체 프로세서 이용률 U는 다음과 같다. 여기서 $T.w'$는 태스크 집합 τ에 속한 각 태스크 T, 즉, 각 태스크의 프로세서 이용률을 나타내며 U는 τ에 속한 모든 태스크들의 이용률 합으로서 태스크 집합의 프로세서 이용률을 의미한다.

$$U = \sum_{\tau \in T} T.w' \quad (10)$$

이 밖에 프로세서의 수를 M이라 하면, 태스크에 할당되지 않은 여분의 이용률 $\hat{U}$은 $\hat{U} = M - U$으로 표현된다. 이밖에 P는 태스크들의 대주기를, $p_{\max}$와 $p_{\min}$은 각각 태스크들에 대한 주기의 최대값과 최소값을 의미한다.

IV. mode change 환경에서의 퀀텀 크기 결정

본 장에서는 mode change에 따른 퀀텀 크기의 변경이 요구되는 상황에 대해 살펴보고, 이와 관련한 고려사항들을 통해 퀀텀 크기를 결정하기 위한 방법들을 논의하도록 한다. 퀀텀의 크기 변경은 이전 퀀텀 크기에 대한 상대적인 증가나 감소를 의미한다.

1. 퀀텀 크기의 감소

현재의 퀀텀 크기에 비해 퀀텀 크기를 줄여야하는 경우는 태스크 집합의 변경(mode change)에 따라 $U > M$가 되는 경우이다. 이때 퀀텀의 크기를 감소시키는 것은 스케줄링이 불가능한 태스크 집합을 스케줄링 가능하도록 하기 위함이다. 하지만, 다음의 두 가지 경우에 대해서 퀀텀의 크기 감소 여부가 결정될 수 있는데 이것은 결국 mode change된 태스크의 스케줄링 여부를 결정하게 되는 것이다.

첫 번째, 현재의 퀀텀 크기가 더 이상 작아질 수 없는 경우로서 이미 최소 퀀텀 크기인 경우($q_c = q_{\min}$)이다. 이러한 경우 퀀텀 크기는 더 이상 작아질 수 없고 $U > M$이므로, 태스크 집합은 Pfair 알고리즘에 의해 스케줄링 불가능하게 된다. 이 경우 추가로 고려해 볼만한 상황은 전체의 태스크 집합을 거절할 것인가? 아니면, mode change에 따라 새롭게 추가된 태스크의 진입만을 금지시킬 것인가 하는 문제인데 본 논문에서는 전체 태스크의 스케줄링 가능성 여부가 주요 관심 대상이므로 전체 태스크 집합은 스케줄링 불가능하다고 판단하기로 한다.

다음은 퀀텀 크기가 감소될 가능성이 있는 경우로서, $q_c > q_{\min}$인 경우이다. 만약 퀀텀 크기의 감소가 가능하다면 mode change된 태스크 집합은 스케줄링될 가능성이 존재하게 된다. 이때, 다음의 조건을 만족하는 최대 크기의 q_n을 구하는 것이 기본적인 퀀텀 크기의 결정 문제이다.

$$(q_{\min} < q_n < q_c) \quad \wedge \quad (U \leqq M) \quad (11)$$

만약, 위의 조건을 만족하는 최대 크기의 q_n이 존재하지 않으면 스케줄링은 불가능하게 된다.

2. 퀀텀 크기의 증가

태스크 집합이 변경됨에 따라 $U < M$이 될 수 있다. 이때는 스케줄링 빈도수를 감소시키기 위해 q_c를 증가시킬 수 있는데, 여기서는 다음의 조건을 만족하는 최대 크기의 q_n을 구하는 것이 퀀텀 크기의 결정 문제가 된다.

$$(q_c < q_n) \quad \wedge \quad (U \leq M) \tag{12}$$

하지만, 퀀텀 크기의 결정 문제는 퀀텀 크기의 증가 또는 감소 여부에 관계없이 mode change된 현재 태스크 집합이 스케줄링 가능한 최대 크기의 q_n을 찾는 문제로 단순화 될 수 있다. 결국, 이렇게 얻어진 q_n이 q_c보다 작게 된다면 퀀텀 크기의 감소가 발생된 것이고, q_n이 q_c보다 크다면 퀀텀 크기의 증가가 발생된 것을 의미한다. 따라서, 식 (11)과 식 (12)에서 보는바와 같이, q_n을 q_c보다 작은 범위에서 찾아야하는지 큰 범위에서 찾아야 하는지를 제외하고는 다음의 조건을 공통적으로 만족해야 한다. $q_{\min} \leq q_c$이므로 식 (12)로부터 식 (13)을 얻는다.

$$(q_{\min} < q_n) \quad \wedge \quad (U \leq M) \tag{13}$$

그러므로, q_n은 e_c와 p_c 및 q_c를 기반으로 하지 않아도 되며 태스크가 본래 갖고 있는 e와 p 및 $q_{\min}$만을 기반으로 구할 수 있다. 따라서, 이후에는 특별한 언급이 없는 한 선택되는 퀀텀 크기로서 q_c와 q_n 대신 기호 Q를 사용하기로 한다.

이제 각 태스크의 실행 요구 시간과 주기 등과 같은 태스크 집합의 특성으로부터 새로운 퀀텀 크기 Q를 결정하기 위해 필요한 고려 사항들을 살펴보도록 한다.

3. Q 결정 시 고려 사항

3.1 가장 기본적인 퀀텀 크기의 결정 문제

퀀텀 크기의 증가 및 감소 여부에 상관없이, 주어진 태스크 집합에 대한 최적인 퀀텀 크기는 모든 태스크들의 주기와 실행 요구 시간에 대해서 1이 아닌 공약수가 존재할 경우에 가장 확실하면서도 가장 간단하게 결정될 수 있다. 공약수들 가운데 $U \leq M$을 만족하는 최대 공약수 $CD_{\max}(\leq GCD)$가 퀀텀의 크기로 선택될 수 있는 데 이런 경우, 모든 태스크들에 대해 어떠한 주기도 변하지 않고 전체 이용률 U도 변하지 않으므로 스케줄링이 가능하다.

하지만, 실제로 모든 태스크들의 주기와 실행 요구 시간에 대해 1이 아닌 공약수가 존재할 가능성은 특별한 경우를 제외하고는 매우 희박하다. 따라서 이러한 공약수가 존재하지 않을 수 있다는 가정 하에 새로운 퀀텀 크기를 결정하는 방법을 찾아야 하며, 주기 변화를 허용할 경우 주기 변경 허용 여부에 따라 $CD_{\max}$보다 큰 값에서 Q가 존재할 수 있다는 점도 고려해야 한다.

태스크 집합이 스케줄링 가능한 최대 퀀텀 크기를 결정하는 데에 직접적으로 영향을 미치는 요소들에는 모든 태스크들의 주기와 실행 요구 시간 및 잉여 이용률 $\hat{U}$이 있다. 태스크의 실행 요구 시간은 전체 이용률 증가에 영향을 미칠 수 있고 주기는 주기의 변화와 이용률 증가에 영향(재계산된 주기의 감소는 상대적으로 이용률을 증가시킴)을 미치며, $\hat{U}$는 실행 요구 시간에 따라 증가될 수 있는 이용률의 정도에 영향을 미치게 된다.

그렇다면, 이러한 파라미터들을 이용하여 식 (13)을 만족하는 Q의 최대값을 계산할 수 있는 다항식 함수 가 과연 존재할 것인가? 각 태스크들의 주기에 대한 실행 요구 시간의 합에 의해 전체 이용률 U가 결정될 수는 있지만, 동일한 $\hat{U}$에 대해서도 태스크들의 주기 및 실행 요구 시간의 조합에 따라 최적의 퀀텀 크기는 서로 다르게 나올 수 있다. 간단한 예로 $M=1$일 때 태스크 집합 $\tau_1 = \left\{\frac{3}{12}, \frac{3}{12}, \frac{3}{12}\right\}$과 $\tau_2 = \left\{\frac{2}{12}, \frac{3}{12}, \frac{4}{12}\right\}$를 고려해 보자. 두 태스크 집합의 태스크들의 주기는 모두 12이고 실행 요구 시간만 차이가 있으며, $Q=1$일 때 U는 모두 0.75로서 스케줄링 가능하다. 다음으로 $Q=3$인 경우를 보자. τ_1과 τ_2에 대한 U는 각각 0.75와 1로서 여전히 스케줄링 가능하지만, Q의 변화에 따라 실행 요구 시간의 조합이 전체 이용률의 변화에 영향을 미치고 있

음을 볼 수 있다. 이제 $Q=4$일 때를 보면 재계산된 $\tau_1=\left\{\frac{1}{3},\frac{1}{3},\frac{1}{3}\right\}$과 $\tau_2=\left\{\frac{1}{3},\frac{1}{3},\frac{2}{3}\right\}$의 U는 각각 1과 1.33이 되므로 τ_1은 여전히 스케줄링 가능하지만 τ_2는 스케줄링에 실패하게 된다. 결국 τ_1과 τ_2에 대한 최적의 퀀텀 크기는 각각 4와 3임을 알 수 있다.

따라서, 본 논문에서는 이러한 다항식 함수 존재하지 않는다고 가정하고 반복적인 방법으로 적절한 퀀텀 크기를 결정할 수 있는 방법에 대해 연구하고자 한다. 반복적인 방법으로 Q를 결정하기 위한 함수를 FindQ()라 하자. 이 함수에서는 $U \leqq M$을 만족하는 최대의 Q를 구하기 위해 각 Q에 대한 U의 계산 과정을 반복하게 되는데, 최악의 경우 $p_{\max}$번 반복하게 될 것이다. 따라서, FindQ() 함수의 효율성 문제는 반복의 대상이 되는 Q 값의 범위를 최대한 줄이는 것이 관건이다.

3.2 주기 불변 태스크에 대한 고려

태스크 집합에 주기 불변 태스크가 포함되어 있을 경우 퀀텀 크기를 어떻게 결정할 것인가? Q에 대한 임의의 태스크의 실제 주기는

$$rt(p_n) = q_n \cdot Q = \left\lfloor \frac{p}{Q} \right\rfloor \cdot Q \qquad (14)$$

이다. 그런데, 변경된 Q에 의해 주기가 변경되지 않기 위해서는 $\left\lfloor \frac{p}{Q} \right\rfloor \cdot Q = p$이어야 하고, 이 등식이 항상 성립되기 위해서는 Q가 p의 약수이어야 한다. 따라서, 주기 불변 태스크가 둘 이상 존재할 경우 p_n은 모든 주기 불변 태스크들의 주기에 대한 공약수들 중 하나로 선택되어야 한다. 즉, 공약수들 가운데 $U \leqq M$을 만족하는 최대의 Q를 구해야 한다. 주기 불변 태스크에 대한 Q 결정 문제는 그것들에 대한 공약수 문제로 명확하므로, 이후로는 주기 가변 태스크들에 대한 Q 결정 문제에 대해서만 논의하기로 한다.

3.3 주기 변경에 따라 대주기가 변경되는 문제

퀀텀 크기가 변경됨에 따라 mode change할 시점인 대주기가 변경될 수 있다. 대주기는 태스크들의 주기 특성에 따라 변경되지 않을 수도 있고 늘어나거나 줄어들 수도 있다. 하지만, 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 한다. 대주기 전에 p_n에 의해 발생되는 작업의 수에 대한 보장은 모든 태스크들에 대해 다음 식이 항상 성립해야 한다.

$$\left\lfloor \frac{P}{rt(p_n)} \right\rfloor \geqq \left\lfloor \frac{P}{p} \right\rfloor \qquad (15)$$

식 (15)에서 $rt(p_n) \leqq p_{origin}$이므로 위 식은 모든 태스크들에 대해 항상 성립한다. 또한, Pfair스케줄링 알고리즘에서 스케줄링 가능성 조건은 특정한 시점인 대주기를 고려하지 않으며, Q는 모든 퀀텀 구간에 대해서도 $U \leqq M$을 만족하도록 선택되므로, 원래의 대주기 내에서도 모든 태스크들은 항상 스케줄링 가능하다.

3.4 퀀텀 크기의 상한

태스크의 수를 n이라 할때, $n \leqq M$이고 $Q=P$이면 모든 태스크들은 각각 하나의 프로세서를 전용으로 할당받을 수 있으므로 스케줄링 가능하게 된다. 하지만, $n > M$인 경우, 모든 태스크들이 스케줄링되기 위해서는 Pfair 알고리즘의 스케줄링 가능 필요충분조건과 같이 다음의 조건을 항상 만족해야 한다.

$$U = \sum_{1}^{n} \frac{\left\lceil \frac{e_i}{Q} \right\rceil \cdot Q}{\left\lfloor \frac{p_i}{Q} \right\rfloor \cdot Q} = \sum_{1}^{n} \frac{\left\lceil \frac{e_i}{Q} \right\rceil}{\left\lfloor \frac{p_i}{Q} \right\rfloor} \leqq M \qquad (16)$$

그런데, $n > M$이고 모든 태스크들의 주기에 대한 공약수가 존재하지 않는 경우, $Q \geqq p_{\max}$가 되면 모든 태스크들의 w'가 1이 되므로 $(n-M)$개의 태스크들은 스케줄링될 수 없다. 따라서, Q에 대해서 항상 다음이 성립해야 한다.

$$Q < p_{\max} \tag{17}$$

[그림 1]은 $\tau_3 = \left\{\frac{5}{11}, \frac{10}{21}, \frac{15}{31}, \frac{20}{41}, \frac{25}{51}, \frac{51}{103}\right\}$에 대해서 $M=4$일 때 퀀텀 크기의 증가에 따른 U의 값을 보여준다. [그림 1]의 그래프에서 보는 바와 같이 퀀텀의 크기가 증가함에 따라 U는 단조 증가하지 않음을 볼 수 있다. 이것은 $U \leqq M$을 만족하는 최대의 퀀텀 크기를 찾기 위해, Q를 $p_{\max}-1$에서부터 1씩 감소시키면서 $U \leqq M$이 될 때까지 반복해야함을 의미하게 되는데, 최악의 경우 $p_{\max}-1$번 반복해야 한다. 따라서, 본 논문에서는 반복의 시작점을 $p_{\max}-1$ 이하로 낮춤으로서 반복의 횟수를 줄이면서도 최적의 퀀텀 크기를 결정할 수 있는 방법을 제안하고자 한다.

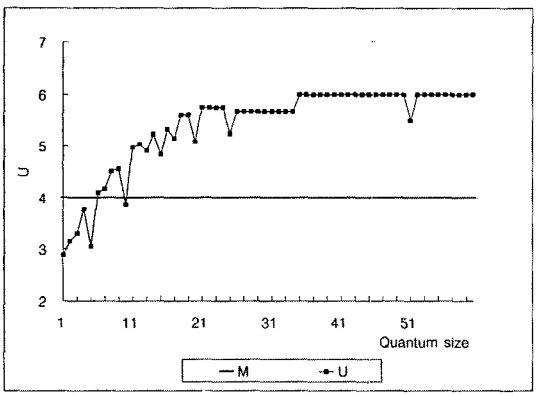


그림 1. Q의 증가에 따른 τ_3의 프로세서 이용률의 변화

V. FindQ()의 구현

본 장에서는 이용률 계산 시점을 낮춤으로서 반복 횟수를 줄일 수 있는 방법을 제안한다. 앞서 언급하였듯이 태스크의 수를 n이라 하면 $n > M$임을 가정한다.

Q의 크기를 증가시킴에 따라 각 태스크들의 w'는 1에 도달하게 된다($w \leqq 1$이어야 하므로 $w' > 1$인 경우 $w'=1$). [표 1]은 $\tau_4 = \left\{\frac{7}{41}, \frac{2}{4}, \frac{29}{48}, \frac{8}{39}, \frac{15}{22}\right\}$에 대한 w'의 증가 모습을 보여준다. 표에서 보면 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. 실제 도달점을 찾기 위해서는 $p_{\max}-1$로부터 Q값을 1씩 감소시키면서 이용률이 1보다 작은 값이 나오는지 반복적으로 찾아야하는데, 본 논문에서는 e와 p만을 이용해서 실제 도달점과 같거나 도달 구간 내에 존재하는 Q값을 찾을 수 있는 다항식 함수를 제안한다. 이 함수를 도달 함수 $Reach(p, e)$라 하는데 이 함수에서 얻어진 Q값을 도달점 RP라 하면 다음이 성립한다.

$$RP \geqq RRP \tag{18}$$

표 1. τ_4에 대한 w와 프로세서 이용률 변화

Q	U	w'				
		T_1	T_2	T_3	T_4	T_5
1	2.162	0.171	0.500	0.604	0.205	0.682
2	2.263	0.200	0.500	0.625	0.211	0.727
3	2.801	0.231	1.000	0.625	0.231	0.714
2	2.263	0.200	1.000	0.625	0.211	0.727
3	2.801	0.231	1.000	0.625	0.231	0.714
4	2.889	0.200	1.000	0.667	0.222	0.800
5	2.952	0.250	1.000	0.667	0.286	0.750
6	3.292	0.333	1.000	0.625	0.333	1.000
7	3.433	0.200	1.000	0.833	0.400	1.000
8	3.117	0.200	1.000	0.667	0.250	1.000
9	3.300	0.250	1.000	0.800	0.250	1.000
10	3.333	0.250	1.000	0.750	0.333	1.000
11	3.417	0.333	1.000	0.750	0.333	1.000
12	3.417	0.333	1.000	0.750	0.333	1.000
13	3.667	0.333	1.000	1.000	0.333	1.000
14	4.000	0.500	1.000	1.000	0.500	1.000
15	3.667	0.500	1.000	0.667	0.500	1.000
16	3.667	0.500	1.000	0.667	0.500	1.000
17	4.000	0.500	1.000	1.000	0.500	1.000
18	4.000	0.500	1.000	1.000	0.500	1.000
19	4.000	0.500	1.000	1.000	0.500	1.000
20	4.500	0.500	1.000	1.000	1.000	1.000
21	5.000	1.000	1.000	1.000	1.000	1.000

[보조정리 1]
정수 a와 b에 대해 다음이 항상 성립한다.

$$\left\lceil \frac{a+1}{b} \right\rceil = \left\lfloor \frac{a}{b} \right\rfloor + 1 \tag{19}$$

[증명]

a를 b로 나눈 몫을 n이라 하면 $n = \left\lfloor \frac{a}{b} \right\rfloor$ 이고, 나머지를 r이라 하면 $a = \left\lfloor \frac{a}{b} \right\rfloor \cdot b + r$로 표현되므로 $\frac{a}{b} = \left\lfloor \frac{a}{b} \right\rfloor + \frac{r}{b}$이다.

이때, 식 (19)의 좌변은 다음과 같다.

$$\left\lceil \frac{a+1}{b} \right\rceil = \left\lceil \frac{a}{b} + \frac{1}{b} \right\rceil$$

$$= \left\lceil \left\lfloor \frac{a}{b} \right\rfloor + \frac{r}{b} + \frac{1}{b} \right\rceil$$

$$= \left\lceil \left\lfloor \frac{a}{b} \right\rfloor + \frac{r+1}{b} \right\rceil = \left\lceil n + \frac{r+1}{b} \right\rceil$$

이때, $0 \leqq r \leqq b-1$이므로 $0 \leqq \frac{r}{b} \leqq \frac{b-1}{b}$이다. 따라서, $\frac{1}{b} \leqq \frac{r+1}{b} \leqq \frac{b}{b}$이므로

$\left\lceil n + \frac{r+1}{b} \right\rceil = n+1$이 다. 또한, 식 (19)의 우변은 명확하게 $n+1$이므로 식 (19)는 성립한다.

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

$$Reach(p, e) = \begin{cases} \left\lfloor \frac{p}{2} \right\rfloor + 1 & \left(\text{if } \frac{e}{p} \leqq \frac{1}{2}\right) \\ \left\lfloor \frac{p}{3} \right\rfloor + 1 & \left(\text{if } \frac{e}{p} > \frac{1}{2}\right) \end{cases} \tag{20}$$

이때, $Q \geqq Reach(p, e)$이면

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

[증명]

(i) $\left\lfloor \frac{p}{2} \right\rfloor + 1$의 증명

$\frac{e}{p} \leqq \frac{1}{2}$이면 $1 \leqq e \leqq \frac{p}{2}$이므로 $e = 1$일 때 $\dfrac{\left\lceil \frac{e}{Q} \right\rceil}{\left\lfloor \frac{p}{Q} \right\rfloor} \geqq 1$이면 $1 < e \leqq \frac{p}{2}$일 때도 성립한다.

이때, $\frac{2p}{p} = 2$

$$\Rightarrow 1 \leqq \frac{2p}{p+1} < 2 \qquad (p \geqq 1)$$

$$\Rightarrow 1 \leqq \frac{p}{\frac{p+1}{2}} < 2$$

$$\Rightarrow 1 \leqq \frac{p}{\left\lceil \frac{p+1}{2} \right\rceil} < 2$$

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

이므로, $\dfrac{1}{\left\lfloor \frac{p}{\left\lfloor \frac{p}{2} \right\rfloor + 1} \right\rfloor} = 1$ 이다.

따라서, $\frac{e}{p} \leqq \frac{1}{2}$이면 $\dfrac{\left\lceil \frac{e}{\left\lfloor \frac{p}{2} \right\rfloor + 1} \right\rceil}{\left\lfloor \frac{p}{\left\lfloor \frac{p}{2} \right\rfloor + 1} \right\rfloor} \geqq 1$ 이다.

(ii) $\left\lfloor \frac{p}{3} \right\rfloor + 1$의 증명

$\frac{e}{p} > \frac{1}{2}$이면 $\frac{p}{2} < e \leqq p$이다. 마찬가지로 e의 최소값에서 $\frac{\left\lceil \frac{e}{Q} \right\rceil}{\left\lfloor \frac{p}{Q} \right\rfloor} \geqq 1$이면 $\frac{p}{2} < e \leqq p$에서도 만족한다. 그런데, e는 $\frac{p}{2}$보다 큰 최소 정수이어야 하므로 e의 최소값은 $\left\lceil \frac{p+1}{2} \right\rceil$이다. 이때 다음식이 만족함을 보이면 충분하다.

$$w(p) = \frac{\left\lceil \frac{\left\lceil \frac{p+1}{2} \right\rceil}{\left\lceil \frac{p+1}{3} \right\rceil} \right\rceil}{\left\lfloor \frac{p}{\left\lceil \frac{p+1}{3} \right\rceil} \right\rfloor} \geqq 1 \qquad (21)$$

위 식에서 $x = \left\lceil \frac{p+1}{2} \right\rceil$, $y = \left\lceil \frac{p+1}{3} \right\rceil$라 하면 식 (21)의 분자 부분은 다음과 같이 정리될 수 있다.

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

$$= \left\lfloor \frac{\left\lfloor \frac{p}{2} \right\rfloor}{y} \right\rfloor + 1 = \left\lfloor \frac{p}{2y} \right\rfloor + 1 \qquad (22)$$

따라서, 식 (21)은 다시 다음과 같이 표현된다.

$$w(p) = \frac{\left\lfloor \frac{p}{2y} \right\rfloor + 1}{\left\lfloor \frac{p}{y} \right\rfloor} \qquad (23)$$

이때, $\frac{p}{3} - 1 < \left\lfloor \frac{p}{3} \right\rfloor \leqq \frac{p}{3}$ (24)

$$\Rightarrow \frac{p}{3} < \left\lfloor \frac{p}{3} \right\rfloor + 1 \leqq \frac{p+3}{3}$$

$$\Rightarrow \frac{3}{p+3} \leqq \frac{1}{\left\lfloor \frac{p}{3} \right\rfloor + 1} < \frac{3}{p}$$

$$\Rightarrow \frac{3p}{p+3} \leqq \frac{p}{y} < 3 \text{ 이므로}$$

식 (24)에서 $f(p) = \frac{3p}{p+3}$라 하면 $\lfloor f(p) \rfloor$의 값은 다음과 같다.

$$\lfloor f(p) \rfloor = \begin{cases} 0 & (\text{if } p = 1) \\ 1 & (\text{if } 2 \leqq p \leqq 5) \\ 2 & (\text{if } p \geqq 6) \end{cases} \qquad (25)$$

따라서, $\left\lfloor \frac{p}{y} \right\rfloor$의 값은 다음과 같다.

$$\left\lfloor \frac{p}{y} \right\rfloor = \begin{cases} 0, 1, \text{또는} 2 & (\text{if } p = 1) \\ 1 \text{ 또는} 2 & (\text{if } 2 \leqq p \leqq 5) \\ 2 & (\text{if } p \geqq 6) \end{cases} \qquad (26)$$

한편, $\frac{3p}{p+3} \leqq \frac{p}{y} < 3$ (27)

$$\Rightarrow \frac{3p}{2(p+3)} \leqq \frac{p}{2y} < \frac{3}{2}$$

$$\Rightarrow \frac{5p+6}{2(p+3)} \leqq \frac{p}{2y} + 1 < \frac{5}{2}$$

이므로,

식 (27)에서 $g(p) = \frac{5p+6}{2(p+3)}$라 하면 $\lfloor g(p) \rfloor$의 값은 다음과 같다.

$$\lfloor g(p) \rfloor = \begin{cases} 1 & (\text{if } 1 \leqq p \leqq 5) \\ 2 & (\text{if } p \geqq 6) \end{cases} \qquad (28)$$

따라서, $\left\lfloor \frac{p}{2y} \right\rfloor + 1$의 값은 다음과 같다.

$$\left\lfloor \frac{p}{2y} \right\rfloor + 1 = \begin{cases} 1 \text{ 또는 } 2 & (\text{if } 1 \leqq p \leqq 5) \\ 2 & (\text{if } p \geqq 6) \end{cases} \quad (29)$$

그러므로, p의 범위에 따른 $\left\lfloor \frac{p}{2y} \right\rfloor + 1$과 $\left\lfloor \frac{p}{y} \right\rfloor$의 값은 다음과 같다.

p	$\left\lfloor \frac{p}{2y} \right\rfloor + 1$	$\left\lfloor \frac{p}{y} \right\rfloor$
$p = 1$	1	1
$p = 2$	2	2
$p = 3$	1	1
$p \geqq 4$	2	2

위에서 보듯이 모든 p에 대해 $\left\lfloor \frac{p}{2y} \right\rfloor + 1 = \left\lfloor \frac{p}{y} \right\rfloor$이 성립하므로 $\frac{e}{p} > \frac{1}{2}$이고 $Q \geq \left\lfloor \frac{p}{3} \right\rfloor + 1$이면 $w(p) = 1$이다.

일반적인 반복에서는 Q를 찾기 위해 $p_{\max} - 1$로부터 반복이 시작되는데 τ_4의 경우 $Q = 47$로부터 시작된다. 하지만, [표 1]에서 보면 $Q = 21$일 때 모든 태스크가 이미 도달 구간에 존재하므로 w는 1이고 $n > M$이므로 스케줄링은 실패하게 된다. 또한, M개의 태스크가 도달 구간에 있다고 하면 이미 M개의 태스크에 의해 $U = M$이 되었고 나머지 태스크들의 이용률 합은 적어도 0보다 크므로 스케줄링은 실패하게 된다. 따라서, 최적의 Q값은 적어도 M번째 태스크의 도달점 보다 작은 구간에서 구해진다. 이를 위해 도달 순위에 따라 도달점을 기억하는 도달 순위 리스트 $Rank[n]$을 생성하는데 각 태스크의 도달점을 오름차순으로 저장한다. 실제 특정 도달점 보다 작은 값으로부터 Q값을 찾아야 하므로 리스트에는 $Reach() - 1$의 값이 저장된다. [표 2]는 τ_4에 대한 도달 순위 리스트를 보여주고 있으며 Q의 범위는 다음과 같이 정리될 수 있다.

$$1 \leqq Q \leqq Rank[M-1] \leqq p_{\max} \quad (30)$$

표 2. τ_4에 대한 도달 순위 리스트

i	0	1	2	3	4
Rank[i]	2	7	16	19	20
Task 번호	T_2	T_5	T_3	T_4	T_5

이제 위의 도달 함수와 도달 순위 함수를 기반으로 Q를 결정하기 위한 3가지 방법을 제안한다.

(1) 방법-1

태스크 집합 τ_4에 대해, $M = 4$일 때 $Rank[3] = 19$이고 $U \leqq 4$이므로 태스크 집합은 Pfair 알고리즘에 의해 스케줄링 가능하게 되므로 최적의 퀀텀 크기는 19임을 쉽게 찾을 수 있다. 하지만, $M = 3$인 경우를 보자. $Rank[2] = 16$에서 $U = 3.667 > 3$로 스케줄링은 실패한다. 도달점을 한 단계 내려 보면 $Rank[1] = 7$이지만 여전히 $U = 3.433 > 2$로 스케줄링이 실패한다. 마찬가지로 $Rank[0] = 2$를 보면 $U = 2.263$으로 스케줄링 가능하게 된다. 하지만, [표 1]에서 보듯이 실제 최적의 Q값 5와는 차이가 발생하게 된다. [알고리즘 1]은 이 방식을 이용한 FindQ() 함수의 구현을 보여준다. 여기서 calculate_U(int Q)는 퀀텀의 크기가 Q일 때 전체 프로세서 이용률을 계산하는 함수로서 식 (10)을 구현한 것이다.

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

알고리즘 1. Rank[] 만을 고려하는 FindQ() 함수

(2) 방법-2

(방법-1)에서는 실제 최적의 Q가 존재함에도 불구하고 이보다 작은 값으로 선택될 수 있다. 심지어 Q의 값이 1일 수도 있는데 이때를 FindQ() 함수가 실패했다고 한다. 두 번째로 제안하는 방법에서는 식 (30)에 따라 $Rank[M-1]$로부터 퀀텀의 크기를 1씩 감소시키면서 Q를 찾는 방법이다. 이 방법으로 실패 없이 항상 최적의 Q값을 찾을 수 있지만, 반복 횟수가 비교적 많아질 수도 있다. [알고리즘 2]는 (방법-2)에 대한 알고리즘이다.

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

알고리즘 2. Rank[M-1]로부터 반복을 시작하는 FindQ() 함수

(3) 방법-3

(방법-3)은 (방법-1)과 (방법-2)를 혼합한 방법이다. 우선 (방법-1)로 Q를 찾고 만약 실패한다면 다시 (방법-2)로 최적의 Q를 찾는 방법이다.

이 방법에서는 비록 최적의 Q보다 작은 값으로 Q가 결정될 수도 있지만, (방법-1)이 갖는 실패 확률을 없앴으며 (방법-2)에 비해 반복 횟수를 줄일 수 있다. [알고리즘 3]은 (방법-3)에 대한 알고리즘이다.

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

              Q = i;
              break;
          }
      }
  }
  return Q;
}
```

알고리즘 3. (방법-1)과 (방법-2)를 혼합한 FindQ() 함수

VI. 실험 및 분석

본 장에서는 앞서 제안한 세 가지 방법들에서 구한 Q의 값을 최적의 Q값과 비교하고, 이 들 방법에서 계산된 프로세서 이용률의 반복 횟수를 p_{max}로부터 반복하여 최적의 경우와 비교해 본다. 본 실험은 Windows XP가 설치된 x-86 기반의 플랫폼에서 C 언어로 구현된 프로그램을 통해 수행되었으며, 최대 주기가 1,000이고 전체 이용률이 4를 넘지 않도록 무작위로 생성된 100,000개의 태스크 집합을 대상으로 실험하였다. 또한, $w \leq 0.5$인 경량 태스크(light- task)만으로 구성된 태스크 집합과 $w > 0.5$인 중량 태스크(heavy-task)와 경량 태스크들이 혼합된 태스크 집합을 대상으로 실험하고 이를 비교해 본다. [표 3]은 경량 태스크 집합의 결과를 보여준다.

표 3. 경량 태스크 집합에 대한 실험 결과

항목 / 방법	평균 이용률 계산 횟수 (비율)	실패 횟수 (비율)	차이 횟수 (비율)	평균 퀀텀 크기 (비율)
최적 퀀텀	934.651	0	0	5.258
방법 1	3.0 (0.321%)	79928 (79.93%)	0 (0.000%)	1.00 (19.02%)
방법 2	120.168 (12.86%)	0 (0.0%)	0 (0.0%)	5.258 (100.0%)
방법 3	121.743 (13.07%)	0 (0.0%)	0 (0.0%)	5.258 (100.0%)

여기서, 평균 이용률 계산 횟수는 각 방법에서 각 태스크 집합에 대한 최대 Q를 찾기 위해 반복된 프로세서 이용률 계산의 평균 반복 횟수이고 각 비율은 최적 퀀텀의 반복 횟수에 대한 각 방법에서의 계산 횟수의 비율로서 0에 가까울수록 효율이 좋음을 의미하게 된다. 또한, 실패 횟수는 실제 1보다 큰 Q값이 존재함에도 불구하고 그 값을 찾지 못해 Q의 값이 1로 결정된 총 횟수를 의미하며, 차이 횟수란 최적의 퀀텀 크기는 아니지만 1보다 큰 퀀텀 크기를 찾은 횟수를 나타낸다. 마지막으로 평균 퀀텀 크기와 비율은 최적 퀀텀 크기에 대한 크기와 비율을 각각 나타내는데 차이 횟수와 실패 횟수가 모두 0인 경우 최적 퀀텀 크기와 같고 비율은 100%가 된다.

이 실험에서 생성된 태스크 집합의 평균 태스크 수는 15.773이며 평균 프로세서 이용률은 3.8352이다. (방법 1)의 결과에서는 이용률 계산의 횟수가 3이지만 최적의 퀀텀 결정 성공률이 약 20%에 불과했고 (방법 2)와 (방법 3)에서는 대략 13%의 반복으로 모두 100% 최적의 퀀텀 크기를 결정할 수 있었다.

[표 4]는 경량 태스크들과 중량 태스크들이 혼합된 태스크 집합에 대한 실험 결과이다. 이 태스크 집합의 평균 태스크 수는 12.473개이며 평균 프로세서 이용률은 3.7853이다. 위의 결과는 경량 태스크만을 이용한 실험 결과와 거의 유사한 결과를 보여 준다. 그러나, (방법 3)에서는 최적의 퀀텀 크기를 100% 결정하지 못했지만 결정된 퀀텀의 크기는 평균적으로 최적 퀀텀 크기의 99%에 이른다.

표 4. 경량 태스크와 중량 태스크가 혼합된 태스크 집합에 대한 실험 결과

항목 / 방법	평균 이용률 계산 횟수 (비율)	실패 횟수 (비율)	차이 횟수 (비율)	평균 퀀텀 크기 (비율)
최적 퀀텀	917.622	0	0	7.840
방법 1	3.0 (0.327%)	83545 (83.55%)	2 (0.002%)	1.008 (12.85%)
방법 2	129.081 (14.07%)	0 (0.0%)	0 (0.0%)	7.840 (100.0%)
방법 3	125.180 (13.64%)	0 (0.0%)	932 (0.932%)	7.763 (99.01%)

100,000개의 태스크 집합에 대한 실험은 99,999번의 mode change가 발생했음을 의미하는데 [표 4]의 실험 중 (방법 2)에서는 총 53,745번의 퀀텀 사이즈 증가가 발생되었으며 이때 이용률 계산의 평균 횟수는 133회이고 퀀텀 사이즈의 평균 증가 비율은 458%이었다. 즉, mode change 시점에서 133회의 이용률 계산으로 스케줄링 횟수를 1/4.58회로 감소시킬 수 있었다.

VII. 결론 및 향후 연구

본 논문에서는 퀀텀을 기반으로 하고 있는 다중 프로세서 스케줄링 알고리즘에서 고정 퀀텀 크기가 갖는 문제점을 해결하기 위해 최적의 퀀텀 크기를 동적으로 결정하기 위한 방법을 제안하였다. 임의의 태스크 집합에 대해서 스케줄링 가능한 최대의 퀀텀 크기는 스케줄링 횟수를 최대한 감소시킬 수 있는 최적의 퀀텀 크기이다. 최적의 퀀텀 크기를 결정하기 위해서는 태스크 집합 내에서 가장 큰 주기로부터 반복적으로 프로세서 이용률을 계산해야 하는데, 본 논문에서는 퀀텀 크기 증가에 따라 태스크의 이용률이 지속적으로 1이 되는 도달점을 찾는 다항식 도달 함수를 제시하고 이를 기반으로 프로세서 이용률 계산의 반복 횟수를 감소시키면서 동시에 최적의 퀀텀 크기를 결정할 수 있는 방법을 제안하였다.

두 번째 방법에서는 중량 태스크의 유무와 상관없이 약 12~14%의 반복으로 항상 최적의 퀀텀 크기를 결정할 수 있었다. 세 번째 방법에서는 두 번째 방법에 비해 약간의 반복 횟수 감소가 있었고 최적의 퀀텀크기를 결

정할 수 없는 결과를 보이기도 했지만 그 비율은 대략 1% 미만이었다.

두 번째 실험의 (방법-3)에서 보듯이 이용률이 0.5보다 큰 태스크가 존재하는 경우 최적의 퀀텀 크기를 찾지 못할 수도 있었다. 이것은 본 논문에서 제안한 도달 함수가 이용률이 0.5보다 큰 경우에는 실제 도달점에 근접하지 않았기 때문인 것으로 파악된다. 따라서, 향후에는 이 범위에서 보다 근접한 도달점을 도출하기 위한 도달 함수에 대한 연구가 필요하다. 또한, 본 논문에서는 수신측 버퍼 크기 에 대한 제한을 무시하였는데, 이러한 버퍼 크기를 고려한 퀀텀 크기의 증가에 대해서도 연구가 필요하다.

참 고 문 헌

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

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

[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, Proceedings of the 24th IEEE Real-time Systems Symposium, 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 nad Applications, pp.297-306, Dec. 2000.

[8] 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.

[9] 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.

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

[11] 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.

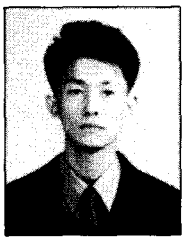
[12] S. Baruah, N. Cohen, C. G. Plaxton, and D. Varvel, "Proportionate Progress: A notion of fairness in resource allocation," Algorithmica, Vol.15, pp.600-625, 1996.

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

[14] 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 전자컴퓨터학부 컴퓨터과학전공 교수
- 2001년~2003년 : 미국 뉴멕시코공과대학 방문교수

<관심분야> : 운영체제, 실시간시스템, 임베디드 시스템