Journal of Korea Multimedia Society Vol. 14, No. 11, November 2011(pp. 1478-1490)

다중 멀티미디어 스트리밍을 위한 멀티코어 시스템 기반의 실시간 스케줄링 기법

박 상 수†

요 약

멀티코어 프로세서는 멀티미디어 스트리밍에 있어서 비교적 낮은 비용에 높은 성능을 보인다는 장점 때문에 스마트폰, 스마트 TV, 셋톱박스 등 관련 산업계에서 큰 관심을 받고 있다. 멀티미디어 데이터를 처리하기 위해서는 주기적인 태스크의 시간 제약성을 만족시킬 수 있는 스케줄링 알고리즘이 필요하다. Pfair 스케줄링 알고리즘은 이론상으로 멀티코어 상에서 모든 태스크의 시간 제약성을 만족하는 동시에 100%의 이용률을 달성할 수 있지만 코어간에 태스크의 빈번한 이동이 필요하고 매 스케줄링 시점에 시스템 전역에 대한 동기화가 필요하는 등 스케줄링을 위한 오버헤드가 매우 높다. 이러한 문제점을 개선하기 위해 본 논문에서는 코어 간의 이동이 꼭 필요할 경우에만 전체 코어의 스케줄링을 수행하고 평상시에는 각 코어별로 독립적인 스케줄링을 수행하도록 하는 HPGP 스케줄러를 제안한다. 시뮬레이터를 통한 실험 결과 기존의 Pfair 알고리즘에 비해 스케줄링 오버헤드가 현격히 감소하는 것을 알 수 있으며 80% 이하의 이용률을 갖는 태스크 집합에서는 스케줄링 오버헤드가 거의 발생하지 않는 것을 확인하였다.

A Real-Time Scheduling Technique on Multi-Core Systems for Multimedia Multi-Streaming

Sangsoo Park†

ABSTRACT

Recently, multi-core processors have been drawing significant interest from the embedded systems research and industry communities due mainly to their potential for achieving high performance and fault-tolerance at low cost in such products as automobiles and cell phones. To process multimedia data, a scheduling algorithm is required to meet timing constraints of periodic tasks in the system. Though Pfair scheduling algorithm can meet all the timing constraints while achieving 100% utilization on multi-core based system theoretically, however, the algorithm incurs high scheduling overheads including frequent core migrations and system-wide synchronizations. To mitigate the problems, we propose a real-time scheduling algorithm for multi-core based system so that system-wide scheduling is performed only when it is absolutely necessary. Otherwise the proposed algorithm performs scheduling within each core independently. The experimental results by extensive simulations show that the proposed algorithm dramatically reduces the scheduling overheads up to as negligible one when the utilization is under 80%.

Key words: Streaming(스트리밍), Multi-Core(멀티코어), Real-Time(실시간), Scheduling(스케줄링)

※ 교신저자(Corresponding Author) : 박상수, 주소 : 서울 서대문구 대현동 11-1 이화여자대학교 컴퓨터공학과, 전화 : 02)3277-6643, FAX : 02)3277-2306, E-mail : sangsoo.park@ewha.ac.kr
접수일 : 2011년 9월 8일, 수정일 : 2011년 10월 14일
완료일 : 2011년 11월 3일
† 정회원, 이화여자대학교 공과대학 컴퓨터공학과
※ 본 논문은 한국연구재단의 기초연구사업(과제번호: 2011-0013422), 세계수준의 연구중심대학 육성사업(과제번호: R33-10085)의 지원을 받아 수행된 것임.

1. 서 론

멀티코어 프로세서는 멀티미디어 처리에 있어서 비교적 낮은 비용에 높은 성능을 보인다는 장점 때문에 스마트폰, 스마트 TV, 셋톱박스 등 관련 산업계에서 큰 관심을 받고 있다[1]. 또한 이러한 멀티미디어 기기들에서 소비자들이 요구하는 멀티미디어 처리 능력은 점점 증가하고 있으며, N-스크린 [2] 등 최근에는 여러 컨텐츠를 동시에 스트리밍 할 수 있는 다중 멀티미디어 스트리밍의 필요성이 증대되었다. 이때 하나의 프로세서에 여러 코어가 탑재되어 동시에 여러 명령어를 수행할 수 있는 멀티코어 프로세서는 이러한 다중 멀티미디어 스트리밍의 처리에 있어서 최적의 선택이라고 할 수 있다[3]. 동영상 혹은 음성과 같은 멀티미디어 컨텐츠를 재생하기 위해서는 매 프레임 별로 멀티미디어 데이터를 주기적으로 처리할 수 있어야 한다. 이러한 멀티미디어 컨텐츠의 특성상 다중 멀티미디어 스트리밍을 멀티코어 프로세서에 구현하는데 있어서 가장 중요한 기법은 다중의 멀티미디어 데이터를 처리하는 프로그램 태스크들을 정해진 주기 이내에 수행 완료되어야 한다는 점이다[4].

이러한 여러 태스크를 정해진 주기 이내에 수행의 완료를 보장하기 위해서는 각 태스크들을 주어진 시간 제약조건에 맞도록 스케줄링 해주는 실시간 스케줄링 알고리즘을 사용해야 한다[5]. 하나의 코어만을 탑재한 싱글코어 프로세서를 위한 실시간 스케줄링 알고리즘은 지난 수년간 광범위 하게 연구되어 많은 수의 최적화된 알고리즘이 이미 개발되었고 이를 실제 시스템에 적용할 때 발생할 수 있는 여러 복잡한 이슈들 또한 이미 연구되었다[6]. 멀티코어 프로세서를 위한 실시간 스케줄링 알고리즘은 최근 활발히 연구되고 있으며 초기에는 기존의 싱글코어 기반으로 개발되었던 알고리즘을 멀티코어 프로세서에 적용하는 여러 연구들이 있어 왔지만 아직 뚜렷한 성과를 내고 있지 못하다[6].

현재 멀티코어 프로세서 기반의 실시간 스케줄링 알고리즘은 크게 (1) 코어 분할 스케줄링 방식과 (2) 시스템 전역 스케줄링 방식의 두 가지 기법으로 나눌 수 있다. 코어 분할 스케줄링 방식은 시스템에 주어진 시간 제약성을 갖는 태스크들을 시스템의 수행 전에 각 코어 별로 할당한 후 시스템 수행 중에는 각 코어 별로 할당된 태스크들을 다른 코어들로 이동을 금지하며 각 코어 별로 싱글코어 기반의 실시간 스케줄링 알고리즘을 이용하는 방식이다. 최근의 연구결과로는 이러한 코어 분할 스케줄링 방식에서 모든 태스크의 시간 제약을 만족하기 위해서 전체 프로세서 시간의 50% 이상을 유휴 시간으로 함이 증명되었다. 즉, M 이 프로세서 내부의 코어의 개수일 때 최악의 경우 모든 태스크의 시간 제약을 만족할 수 있는 최대 시스템 이용 가능률은 $U = (M+1) / 2$ 이다[6]. 반면 시스템 전역 스케줄링 방식은 코어 분할 스케줄링 방식에서 시스템 수행 중에 태스크가 코어 간에 이동이 금지되었던 것과는 달리 시스템 수행 중에 각 태스크가 한 코어에서 다른 코어로 이동할 수 있게 한다. 이를 통해 모든 태스크의 시간 제약성을 만족하면서도 동시에 코어 분할 스케줄링 방식에 비해 보다 높은 시스템 이용률 가능하게 한다 [7].

현재까지 알려진 바에 의하면 Pfair (Proportionate-fair) 실시간 스케줄링 알고리즘은 모든 태스크의 시간 제약성을 만족하면서도 이론적으로 100%의 시스템 이용률이 가능한 ($U = M$) 오픈 소스로 제공되는 유일한 멀티코어 기반의 실시간 스케줄링 알고리즘이다[8,10]. 그러나 Pfair 시스템 전역 스케줄러는 시스템 수행 중에 태스크를 한 코어에서 다른 코어로 이동함에 따라 동반되는 다음과 같은 문제점을 내포하고 있다. Pfair 스케줄러에서 시스템 수행 시에 태스크의 코어 이동이 빈번하게 될 경우 이동전의 코어와 이동 후의 코어에서 캐시(Cache)와 TLB (Translation Lookaside Buffer) 효율 면에서 메모리 입출력 성능을 크게 저해할 수 있다[12]. 또한, Pfair 스케줄러의 경우 매번 태스크를 스케줄링 할 때마다 시스템 전역 동기화가 필요하다. 이러한 과도한 스케줄링 부하를 발생시키는 Pfair 스케줄러는 실제 스마트폰이나 스마트TV와 같은 시스템에 적용되었을 때 멀티미디어 컨텐츠의 품질에 큰 영향을 미치는 지터(Jitter)의 증가 및 일시적인 수행 정지 등의 문제점을 발생시킬 수 있다.

본 논문에서는 위와 같은 Pfair 알고리즘을 실제 다중 멀티미디어 스트리밍 시스템에 적용시켰을 때 발생할 수 있는 문제점들을 극복하기 위해서 시스템 수행 시에 태스크들의 시간 제약 만족에 필요한 시점에서만 원래의 Pfair 시스템 전역 스케줄링을 수행하고 평상시에는 각 코어 별로 스케줄링을 수행하는

혼합형 스케줄러 HPGP (Hybrid Partitioned-Global Pfair) 스케줄러를 제안한다. 이러한 혼합형 스케줄러는 시스템 전역 스케줄링이 수행되는 시점을 최소화함으로써 태스크의 이동이나 시스템 전역 동기화의 횟수를 최소화하여 스케줄링 오버헤드를 크게 줄일 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 HPGP 스케줄러가 기반하고 있는 Pfair 스케줄링 알고리즘의 기본 개념과 구현상의 이슈들을 다루며 Pfair 스케줄러가 발생시키는 스케줄링 오버헤드의 정도를 하나의 예를 통하여 확인한다. 3장에서는 본 논문의 HPGP 스케줄러에 대해 기술하며 4장의 성능 평가에서는 제안된 HPGP 스케줄러에 의해 감소되는 태스크 이동, 시스템 전역 스케줄링 횟수 및 수행시간 면에서의 스케줄링 오버헤드를 한 프로세서에 4개의 코어를 탑재한 ARM11MPCore 시뮬레이터를 통해 보인다. 5장은 관련 연구에 대해서 기술하며 마지막으로 6장에서는 본 논문의 결론을 내린다.

2. Pfair 스케줄러의 스케줄링 오버헤드 분석

본 장에서는 본 논문의 시스템 모델을 정의하고, 본 논문에서 제안하는 HPGP 스케줄러가 기반으로 하고 있는 Pfair 스케줄러에 대해 하나의 예를 사용하여 Pfair 스케줄링 오버헤드를 분석한다.

2.1 시스템 모델

본 논문의 시스템 모델을 정의하기 위해 다음과 같은 표기와 가정을 사용한다. 본 논문의 시스템은 M개의 성능이 동일한 코어를 갖고 N개의 주기적으로 수행되는 시간 제약성을 갖는 실시간 태스크를 수행한다고 가정한다. 시스템에서 모든 프로그램 태스크는 절대 시간을 [t, t+1)의 범위로 나눈 시간 유닛 t 단위로 수행된다고 가정한다. 각각의 멀티미디어 스트리밍을 담당하는 태스크 T는 시간 유닛 t의 배수 단위인 수행시간 T_e와 수행주기 T_p를 갖으며 매 수행주기 T_p마다 수행시간 T_e를 수행하는 태스크 T의 새로운 태스크 인스턴스가 릴리즈되며 이 인스턴스는 T_p 이내에 수행이 완료되어야 한다. 예를 들어, 그림 1과 같이 T_e=2, T_p=10의 시간 제약성을 갖는 태스크 T는 그림에서 태스크 인스턴스가 t, t+10에서 릴리즈 되어 수행을 시작한 것과 같이, 매 10시간 유닛마다 릴리스 되어 다음 수행 주기가 돌아오기 전에 2시간 유닛을 수행해야 시간 제약성을 만족한다고 가정한다. 단, 릴리즈 된 인스턴스가 바로 수행이 시작될 필요는 없으며 t에서 릴리즈된 인스턴스가 t+5부터 수행을 시작하여 t+7이전에 수행이 완료되는 것과 같이 릴리즈 된 인스턴스의 수행이 다음 주기 이전에만 완료되면 된다.

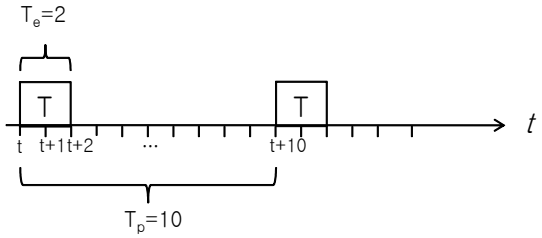


그림 1. 시스템에서 주기적인 태스크의 예

시스템의 임의의 태스크 집합의 한 태스크 T의 비중을 $W_T = \frac{T_e}{Tp}$라고 정의하면 전체 시스템의 이용률은 $U = \sum_N W_T$로 정의된다. Pfair 스케줄러의 경우 모든 태스크의 시간 제약성을 만족하는 경우에도 이론적으로 100%의 시스템 이용률이 가능한 최적의 시스템의 이용률을 가지므로, 하나의 시스템에서 임의의 태스크 집합에 대해 계산된 전체 시스템 이용률이 $U \le M$이라면 태스크에 주어진 모든 시간 제약성을 만족하며 이는 스케줄링이 가능하다고 정의한다 [9].

그림 1이 태스크 T의 한 인스턴스가 임의의 주기에서 수행이 될 때 [t, t+1)과 [t+1, t+2)의 두 시간 유닛에서 수행되는 것을 볼 수 있는데, 본 시스템 모델에서는 태스크 인스턴스가 수행되는 각각의 시간 유닛을 나누어 서브 태스크 (Sub-task) T_i로 구분한다. 시스템 모델에서는 각 서브 태스크도 가상(Pseudo)의 릴리즈 시간과 완료 시간이 존재한다고 가정하며, 이때 서브 태스크 T_i의 가상 릴리즈 시간은 $r(T_i) = \left\lfloor \frac{i-1}{W_T} \right\rfloor$로 정의되고 가상 완료 시간은 $d(T_i) = \left\lceil \frac{i}{W_T} \right\rceil - 1$로 정의된다.

2.2 Pfair 알고리즘의 동작 원리

본 논문의 시스템 모델에서 임의의 태스크 집합의 시간 제약성을 만족하는 실시간 스케줄링 문제는 매 스케줄링 지점 혹은 매 시간 유닛 t 마다 각 코어가

수행할 서브 태스크를 각 서브 태스크의 시간 제약성(혹은 완료 시간)을 만족하도록 전체 코어의 숫자인 M개만큼 선택하는 문제로 정의될 수 있다. 이때, Pfair 스케줄링 알고리즘의 기본 아이디어는 시간이 흘러감에 따라서 각 태스크의 시간 제약을 만족하기 위해는 반드시 해당 태스크의 비중만큼 즉 태스크의 수행시간 대비 수행주기의 비율에 비례하여 수행되어야 한다는 것이다. 이러한 비율은 [10]에서 Pfairness라고 정의된다. Pfair 알고리즘에서는 임의의 시간 t에 모든 태스크가 이러한 Pfairness를 만족하면 전체 시스템의 시간 제약성을 만족한다고 정의된다. 또한, Pfair 스케줄러에서 한 태스크 집합의 Pfairness를 만족하기 위해서는 가상 릴리즈 시간이 현재 시간 t 보다 작은 M개 이하의 서브 태스크를 선택하며, 항상 태스크 인스턴스 대비 실제 수행된 서브 태스크의 비율과 Pfairness에서 정의된 비율의 차이가 최대 하나의 시간 유닛 차이 이내가 되도록 한다. 이러한 수행된 비율과 Pfairness의 차이를 스케줄링 오차라 하며 한 태스크 T에 대해서 임의의 시간 t에서의 스케줄링 오류는 $lag(T,t)=t\times w_T-S(T,t)$로 정의된다. 이때 S(T,t)는 시간 범위 [0, t) 동안 실제 수행된 태스크 T의 시간 유닛 수를 의미한다.

위와 같이 한 시스템에서 임의의 태스크 집합에 대해 Pfairness의 비율은 $|lag(T,t)|< 1$과 같이 정의될 수 있으며, 시스템의 임의의 수행시간 t에서 태스크 집합의 모든 태스크의 Pfairness의 비율이 $|lag(T,t)|< 1$을 만족하도록 서브태스크를 각 코어에 할당하면, 이 시스템은 모든 시간 제약성을 만족하며 스케줄링 가능하다. 즉, Pfair 스케줄링 알고리즘은 매 시간 t마다 가상 릴리즈된 서브 태스크 Ti에 대해 우선순위 prio(Ti)를 매기고 전체 코어 개수인 상위 M개의 우선순위를 갖는 서브 태스크를 선택하는 과정을 반복하게 된다. 이때 각 서브 태스크의 우선순위를 정할 때는 기본적으로 가상 제한시간이 빠른 서브 태스크가 높은 우선순위를 갖도록 하는데, 같은 가상 제한시간을 갖는 여러 서브 태스크들 간에 우선순위를 정하는 방법은 PF [8], PD [11], PD2 [10] 알고리즘 등이 존재하는데, 본 논문에서는 가장 효율적으로 알려진 PD2 알고리즘을 사용한다.

2.3 스케줄링 오버헤드의 분석

시스템에서 Pfair 스케줄러를 구현하기 위해서는 그림 2와 같이 하나의 태스크가 이전 주기에서 이미 수행이 완료되어 다음 주기를 기다리는 상태(Suspended), 새로운 주기가 시작되어 수행이 준비 중인 상태(Ready), 해당 태스크의 서브 태스크가 가상 릴리즈되어 코어에 할당되어 수행될 수 있는 상태(Pseudo-ready), 그리고 이 서브 태스크의 우선순위가 전체 서브 태스크 상의 상위 M개 이내로 코어에 할당되어 수행이 시작된 상태 (Running)인 각 태스크와 서브 태스크별 스케줄링 상태 전환이 이루어져야 한다.

이러한 Pfair 스케줄링 알고리즘은 시스템 전역 스케줄러이기 때문에, 그림 2의 수행(Running) 상태, 가상 대기(Pseudo-ready) 상태, 대기(Ready) 상태, 중지(Suspended) 상태의 각 스케줄링 상태 별로 시스템 전역에 태스크 열(Queue)이 필요하다. 또한, 이러한 태스크 열의 무결성(integrity)을 유지하기 위해 매 시간 t마다 태스크 열의 자료구조에 대한 시스템 전역 동기화가 필요하게 되어, 매 시간 t마다 모든 코어에서의 태스크 수행이 일시로 중지되게 되는 오버헤드가 발생한다.

또한, Pfair 알고리즘은 매 시간 t에 수행할 M개의 서브 태스크를 선택하는데, 이때 선택된 서브 태스크들은 시스템 내의 임의의 코어에서 수행되어도 이론상으로는 여전히 시간 제약성을 만족할 수 있다. 그러나 실제 태스크가 수행되는 마이크로프로세서의 구조로 인하여 하나의 태스크가 이전에 수행되었던 코어가 아닌 다른 코어로 이동을 하여 수행을 계속하

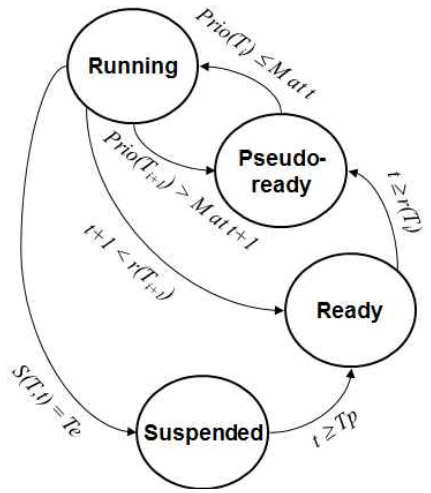


그림 2. Pfair 스케줄러에서 각 태스크의 상태 전환

게 되면 이에 따라 이동전의 코어와 이동 후의 코어에서 캐시(Cache)와 TLB (Translation Lookaside Buffer)에 저장된 내용이 무효화 되는 등 메모리 성능 저하가 불가피하다. 즉, Pfair 스케줄러를 구현할 때 있어 선택 된 서브 태스크들을 어느 코어에 할당하느냐에 따라서 스케줄링 오버헤드는 크게 차이 날 수 있다.

이때 Pfair 스케줄러를 구현하기 위한 가장 단순한 방법으로는 시스템 전역의 가상 대기 열의 가장 높은 우선순위를 갖는 서브 태스크 T_j 를 최초로 사용 가능한 코어에 할당하는 최초 적합 방식이 있다. 스케줄링을 위해 매 시간 t에 이 과정을 M번 혹은 가상 대기 열에 서브 태스크가 남아있을 때까지 반복한다. 본 논문에서는 Pfair 알고리즘의 스케줄링 오버헤드 분석을 위해 위의 방식을 알고리즘 1과 같이 구현하였으며 PFAIR-FF라고 명명하였다.

[알고리즘 1] PFAIR-FF: 최초 적합 코어 할당 방식의 Pfair 알고리즘

```
for i = 1 to M do
        Tj = 코어 i에서 수행 중인 서브 태스크
        if r(Tj) < t then
                Tj를 대기 열에 삽입
        else
                Tj를 가상 대기 열에 삽입
        end if
end for
대기 열과 가상 대기 열의 서브 태스크의 r(Ti), d(Ti),
prio(Ti)를 갱신
r(Ti) ≥ t인 대기 열의 서브 태스크 Ti를 릴리즈함
for i = 1 to M do
        Tj = 가상 대기 열의 가장 높은 우선순위를
갖는 서브 태스크
        Tj 를 코어 i에 할당하고 수행 상태로 변경
end for
```

이미 기술하였다시피 스케줄링 과정에서 한 코어에서 수행되었던 태스크를 다른 코어로 이동하면 캐시, TLB 등 메모리 성능을 저해할 수 있다. 이와 관련하여 Pfair 스케줄러의 과도한 태스크 이동이 필요하다는 것을 보여주기 위해서 본 절에서는 다음과 같은 6개의 태스크를 갖는 태스크 집합을 두 개의 코어를 갖는 듀얼코어 프로세서에서 PFAIR-FF에서 스케줄링 해본다. 단, 태스크의 표기법은 TaskT = (Te, Tp)과 같다.

Task0 = (2, 5), Task1 = (3, 15), Task2 = (3, 15),
Task3 = (2, 6), Task4 = (20, 30), Task5 = (6, 30)

각각의 태스크의 서브 태스크들은 매 시간 t에 코어 1과 코어 2에 스케줄링 되었으며 모든 태스크의 수행 주기의 최대 공약수인 30까지 스케줄링 하였다. (매 30시간 유닛 마다 같은 스케줄링이 반복된다.) 스케줄링 결과는 그림 3(a)와 같으며 각 칸의 숫자는 해당 코어에서 수행된 태스크의 번호를 나타낸다.

스케줄링 오버헤드를 분석하기 위해 태스크가 이전에 수행되었던 코어와 현재 수행 중인 코어가 다를 경우 태스크가 이동한 것으로 판별하는데 예를 들어 그림 3(a)의 시간 유닛 0에서 수행되었던 Task4가 시간 유닛 1에서는 코어 2에서 수행되는 경우 Task4는 코어 1에서 코어 2로 이동하였다고 판별한다. PFAIR-FF의 스케줄링 결과를 보면 태스크 이동 횟수가 총 33회인 것을 확인할 수 있으며, 이는 시스템이 수행되는 매 시간 t마다 한번 이상의 태스크를 이동시켰다는 것을 의미하는 것이다.

이러한 과도한 태스크 이동은 스케줄러를 구현할 때 서브 태스크가 이전에 할당되었던 코어에 먼저 할당을 시도하는 할당 방식을 사용함으로써 크게 줄일 수 있다. 본 논문에서는 Pfair 알고리즘의 스케줄링 오버헤드 분석을 위해 이러한 동일 코어 선호 기

코어 1	4	3	1	4	0	4	4	4	2	0	3	0	4	5	0	4	3	5	4	0	4	4	4	1	0	3	0	4	2	0
코어 2	0	4	2	5	3	0	3	1	5	4	4	1	2	3	4	0	4	1	2	3	0	3	5	2	4	4	5	1	3	4

(a) PFAIR-FF

코어 1	4	4	1	4	3	4	4	4	5	4	4	1	4	5	4	4	4	5	4	3	4	4	4	2	4	4	5	4	2	4
코어 2	0	3	2	5	0	0	3	1	2	0	3	0	2	3	0	0	3	1	2	0	0	3	5	1	0	3	0	1	3	0

(b) PFAIR-CA

그림 3. Pfair 스케줄러에 의한 태스크 스케줄링의 예

반의 할당 방식을 알고리즘 2와 같이 구현하였으며 PFAIR-CA라고 명명하였다. PFAIR-CA에 의한 태스크 집합의 예를 스케줄링 한 결과는 그림 3(b)와 같으며 이전에 할당되었던 코어에 먼저 할당을 시도함에 따라서 태스크 이동 횟수가 33회에서 11회로 크게 줄어든 것을 알 수 있다. 이와 같이 PFAIR-CA는 PFAIR-FF에 비해 태스크 이동 횟수를 줄여 이에 따른 메모리 성능 저하 문제를 상당 부분 해소할 할 수 있지만, 두 스케줄러 모두 시스템 전역 스케줄러이기 때문에 태스크 열을 시스템 전역에 유지해서 매 시간 t마다 시스템 전역 동기화를 수행하여 모든 코어의 태스크 수행을 중단하고 필요한 태스크 열을 접근하여야 하는 단점이 있다.

[알고리즘 2] PFAIR-CA: 동일 코어 할당 시도 방식의 Pfair 알고리즘

```
for i = 1 to M do
    Tj = 코어 i에서 수행 중인 서브 태스크
     t - 1에서 Tj가 수행된 코어 i의 이전 코어 할당
        결과를 저장함
    if r(Tj) < t then
        Tj를 대기 열에 삽입
    else
        Tj를 가상 대기 열에 삽입
    end if
end for
대기 열과 가상 대기 열의 서브 태스크의 r(Ti), d(Ti),
prio(Ti)를 갱신
r(Ti) ≥ t인 가상 대기 열의 서브 태스크 Ti를 릴리즈
함
for i = 1 to M do
    Tj = 가상 대기 열의 가장 높은 우선순위를 갖는
        서브 태스크
end for
for i = 1 to M do
    if 가상 대기 열의 Ti가 이전에 코어 j에서 수행되
       었으면 then
        Ti를 코어 j에 할당하고 수행 상태로 변경
        Remain++
    end if
end for
for i = 1 to Remain do
    for j = 1 to M do
        if 코어 j가 아직 가용하면 then
            Ti를 코어 j에 할당하고 수행 상태로 변경
        end if
    end for
end for
```

3. HPGP (Hybrid Partitioned-Global Pfair) 스케줄러

Pfair 스케줄러는 시스템 수행시간에 코어 간 태스크 이동이 많이 발생하고 매 시간 t마다 스케줄링 수행할 때 시스템 전역 동기화를 수행하여 모든 태스크가 일시로 정지되는 것이 필요하다. 이러한 태스크 이동 횟수와 시스템 전역 스케줄링 지점 횟수를 줄이기 위해서 본 논문에서는 시스템 전역 Pfair 스케줄링이 절대적으로 필요할 때만 이를 수행하는 코어 분할과 시스템 전역 스케줄링이 혼합된 새로운 형태의 스케줄러인 HPGP (Hybrid Partitioned-Global Pfair) 스케줄러를 제안한다. HPGP 스케줄러는 시스템 전역 Pfair 스케줄링이 필요하지 않은 경우 각 코어에 태스크를 할당하고, 각 코어 별로 독립된 Pfair 알고리즘을 수행한다. 또한, 스케줄링 시에 코어 분할 스케줄링으로 스케줄링을 할 때 전체 시스템의 태스크 중에서 시간 제약성을 만족하지 못할 가능성이 있다고 판단되는 경우 다시 코어 분할 스케줄링에서 Pfair 스케줄러와 동일한 시스템 전역 스케줄링으로 전환하여 스케줄링 알고리즘을 수행한다.

이를 위해 HPGP 스케줄러는 그림 4와 같이 임의의 시간 t에 시스템 내의 모든 태스크의 시간 제약성을 만족시키기 위해서 시스템 전역 스케줄링이 필요한지 혹은 코어 분할 스케줄링으로 충분한지 결정하는 결정 알고리즘을 수행한다. 또한 기존의 Pfair 스케줄러는 태스크의 스케줄링 상태 별로 시스템 전역에 태스크 열을 유지해야 했지만 HPGP 스케줄러는

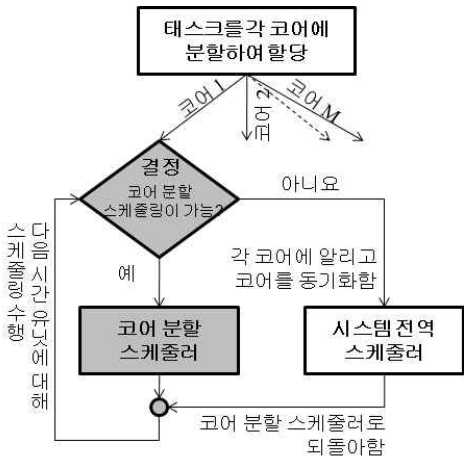


그림 4. 제안된 HPGP 스케줄러의 수행도

코어 분할 스케줄링이 수행될 때에는 각 코어 별로 지역적인 태스크 열만 유지하면 되므로 시스템 전역 동기화가 필요로 하지 않게 된다. 그림 4에서 회색으로 칠한 부분은 각 코어에 한정되어 내부의 태스크 열과 내부의 알고리즘만으로 수행이 가능하게 되어 전체 시스템의 스케줄링 오버헤드를 크게 줄일 수 있다.

3.1 시스템 시작 시 각 코어별 태스크 할당 방법

본 논문에서 제안된 HPGP 스케줄러는 그림 4에서 알 수 있듯이 시스템 초기에 태스크가 M개의 코어에 분할되어 할당된다. 구현의 간편성을 위해서 알고리즘 3과 같이 코어에 분할할 때 각 태스크를 순서대로 코어에 할당하는 비교적 단순한 방식을 채택하였다.

[알고리즘 3] HPGP 스케줄러의 초기 태스크 할당 알고리즘

```
j = 1
for i = 1 to N do
    코어 j에 Ti를 할당
j = (j + 1) % M
end for
```

3.2 시스템 전역과 코어 분할 스케줄러의 결정 알고리즘

HPGP 스케줄러는 매 시간 t에 각 코어 분할 스케줄러가 기본으로 동작한다고 가정한다. 이를 위해서 각 코어 j에서는 매 시간 t에 실제 스케줄링을 수행하기 전에 해당 코어에 할당된 태스크를 코어 분할 스케줄러로 스케줄링 했을 때도 여전히 전체 시스템의 Pfairness를 만족할 수 있는지 확인한다. 코어 j가 결정 알고리즘에 의해 코어 분할 스케줄링이 가능하다고 판정이 되면 Pfair 알고리즘을 코어 j에 할당된 태스크에 대해서만 독립적으로 수행한다. 즉, 코어 j 내부의 서브 태스크 중에서 Pfairness를 만족하기 위해 가장 높은 우선순위를 갖는 서브 태스크를 수행한다. (M=1인 시스템에 대해 Pfair 스케줄러를 수행하는 것과 동일하게 동작한다.

이때 프로세서 내부의 코어 중에서 단 하나의 코어라도 결정 알고리즘의 결과 코어 분할 스케줄링이 불가능하다고 판별되면 (혹은 코어 분할 스케줄링으로는 Pfairness를 만족하지 못하는 태스크가 존재할 경우) 해당 시간 t 에서는 프로세서 내부의 모든 태스크들을 기존의 시스템 전역 Pfair 스케줄러를 수행하여 Pfairness를 만족하도록 한다. 이 경우 시스템 전역 스케줄링을 하기 때문에 태스크가 하나의 코어에서 다른 코어로 이동할 수 있다.

결정 알고리즘은 각 코어에 할당된 태스크의 분할 스케줄링 가능성을 확인하는데 이러한 알고리즘의 구현을 위해서 본 논문에서는 각 코어별 태스크 집합의 비중과 가상 시간(Virtual time)의 개념을 도입하였다. 이때 코어 i의 비중이란 코어 i에 할당된 태스크의 비중의 합인 $W_{코어_i} = \sum_{T_j \in 코어_i} W_{T_j}$와 같이 정의된다.

Pfair 스케줄링 알고리즘은 임의의 태스크 집합에 대해서 U≤M이면 모든 시간 제약조건을 만족시키며 태스크를 수행할 수 있다. 이때 코어의 수가 1인 경우에도 마찬가지로 U≤1이면 코어 내의 모든 태스크의 시간 제약조건을 만족시키며 태스크를 수행할 수 있다. 이러한 $W_{코어_i} \le 1$인 경우 해당 코어는 저부하 (Under-loaded) 상태라고 하며, 코어 i 에서 수행되는 U ≤M=1의 모든 태스크들은 Pfair 알고리즘의 정의에 따라 최악의 경우에도 모든 태스크들의 시간 제약성을 만족시킬 수 있다. 만약 $W_{코어_i} > 1$ 인 경우 코어가 과부하 (Over-loaded) 되었다고 하며, 코어에 독립된 Pfair 알고리즘으로는 태스크들의 스케줄링이 불가능 할 수 있는데, HPHP 스케줄러의 결정 알고리즘은 코어가 과부하된 경우에도 가상 시간 개념을 사용하여 여전히 코어 분할 스케줄링이 가능한지 판별한다.

가상 시간이란 그림 5에서와 같이 CPU 수행속도는 동일하지만 시스템의 시간이 $W_{코어_i}$ 만큼 빠르게 간다고 가정하는 시간이다. 그림 5에서 Task4의 경우 실제 시간에서 t+6가 제한시간이지만 한 시간 유닛의 여유를 갖고 t+5에 완료가 되어 시간 제약을 완료하였다. 여기에서 시간이 1.2배 빨리 가는 가상

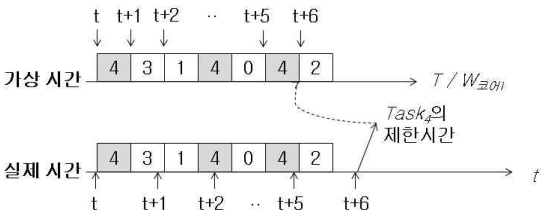


그림 5. T_e=3, T_p=6인 태스크 Task4의 가상 시간에서의 판별 예제

시간에서도 Task4는 t+6에 수행이 완료되어 시간 제약을 만족함을 알 수 있다. 이러한 개념을 코어가 과부화 된 상태에서 코어 분할 스케줄링이 가능한지의 여부를 판별하는데 적용하면, 코어가 과부화 된 만큼 비례하여 시간이 빨리 갈 때에도 여전히 Pfairness를 만족하는지 확인하여 판별할 수 있다.

본 논문에서는 이러한 가상 시간 개념을 도입하여 시스템 전역과 코어 분할 스케줄러의 결정 알고리즘을 작성하기 위해서 2.2절의 Pfairness를 정의하는 lag 값을 다음과 같이 [수식 1]의 각 코어 i 에서의 태스크 T의 lag로 확장하였다.

$$lag(T,t,i) = \begin{cases} t \times W_T - S(T,t) & \text{if } W_{\text{코어}_i} \le 1 \\ \lceil W_{\text{코어}_i} \times t \times W_T \rceil - S(T,t) & \text{if } W_{\text{코어}_i} > 1 \end{cases} \quad (1)$$

HPGP 스케줄러의 결정 알고리즘은 확장된 lag값을 이용하여 각 코어의 할당된 태스크가 매 시간 t마다 Pfairness를 만족하는지 [수식 2]를 사용하여 판별한다.

$$|lag(T,t,i)| < 1 \quad (2)$$

임의의 시간 t에 코어 내의 모든 태스크가 [수식 2]의 Pfairness를 만족한다고 하더라도 다음 스케줄링 시간 t+1에 시간 제약성을 만족시키지 못할 가능성이 있다. 예를 들어, 듀얼코어 프로세서에서 코어 1에 2개의 태스크가 할당되어 있고 시간 t에서 두 태스크 모두 lag 가 1에 가까울 경우 [수식 2]는 만족하지만 두 태스크 중 한 태스크만 시간 t에 스케줄이 가능하므로 이때 수행되지 않은 태스크는 시간 t+1에 Pfairness를 만족시키지 못할 수 있다. 즉, 한 코어 내의 두 태스크 중에 하나의 태스크의 lag가 1에 가까우면 (혹은 Pfairness 비율보다 태스크의 진행이 시간 유닛 1에 가깝게 지연되고 있을 경우) 다른 하나의 태스크는 lag 가 -1에 가까워야지 (혹은 Pfairness 비율보다 태스크의 진행이 시간 유닛 1에 가깝게 빠를 경우) 다음 스케줄링 시간 t+1에 두 태스크 모두 Pfairness를 만족시킬 수 있다. HPGP 스케줄러는 이러한 문제점을 해결하기 위해서 결정 알고리즘에서 [수식 3]과 같이 각 코어 별로 할당된 태스크의 lag의 밸런스가 맞는지 추가적으로 확인하여 최종적으로 임의의 시점 t에 코어 분할 스케줄링이 가능한지 판별하게 된다.

$$\sum_{T \in \text{코어}_i} lag(T,t,i) \le 0 \quad (3)$$

3.3 HPGP 스케줄러에 의한 스케줄링의 예

본 논문에서 제안된 HPGP 스케줄러의 효용성을 보이기 위해 2.3절의 태스크 집합의 예를 HPGP 스케줄러를 적용하였다. 그림 6에서 볼 수 있듯이 HPGP 스케줄러를 적용함으로써 태스크 이동은 PFAIR-FF의 33회, PFAIR-CA의 11회에서 HPGP에서 7회로 감소하였으며 PFAIR-FF와 PFAIR-CA는 매 시간 t에 시스템 전역 스케줄러가 수행되지만 (30회) HPGP 스케줄러는 총 12회만 수행하는 것으로 나타나 스케줄링 오버헤드를 크게 감소시킬 수 있음을 보여주었다.

4. 성능 평가

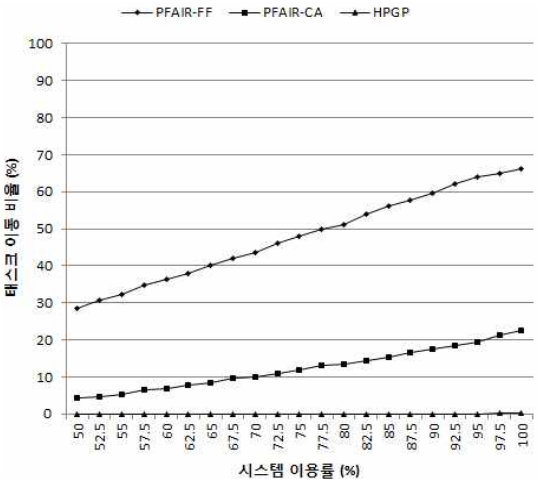
본 논문에서는 시스템 수행 시에 태스크들의 시간 제약 만족에 필요한 시점에서만 원래의 Pfair 시스템 전역 스케줄링을 수행하고 평상시에는 각 코어 별로 스케줄링을 수행하는 하도록 하여 태스크의 이동이나 시스템 전역 동기화의 횟수를 최소화한다. 본 논문에서 HPGP 스케줄러를 실제 다중 멀티미디어 스트리밍 시스템에 적용시켰을 때 지터와 수행시간 지연 등에 영향을 미치는 스케줄링 오버헤드에 대한 양적인 성능 평가를 위해서 HPGP 스케줄러를 PFAIR-FF 스케줄러와 PFAIR-CA 스케줄러에 대해서 비교하는 성능 평가를 실시 하였다.

이를 위에 본 논문에서는 이산 시간 기반 시뮬레이터(Discrete-time Simulator)로 HPGP, PFAIR-

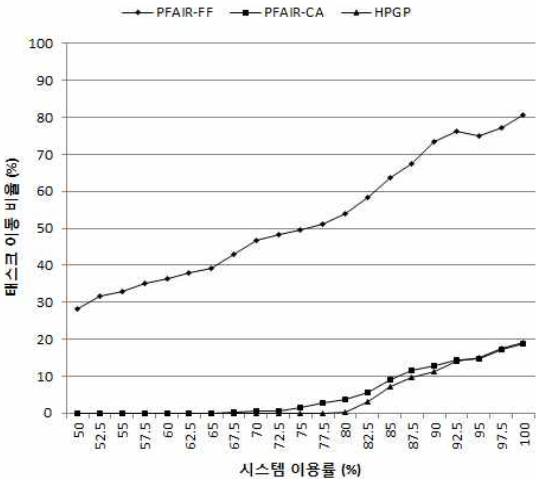
코어 1	4	0	4	4	0	4	4	0	4	0	3	0	0	3	2	0	3	2	0	3	0	3	2	0	4	4	0	4	4	0
코어 2	3	1	5	2	3	1	3	5	2	4	4	1	4	4	5	4	4	1	4	5	4	4	4	1	5	3	2	5	1	3
전역 스케줄러				V						V	V	V	V	V	V					V				V	V	V	V			

그림 5. HPGP 스케줄링의 예

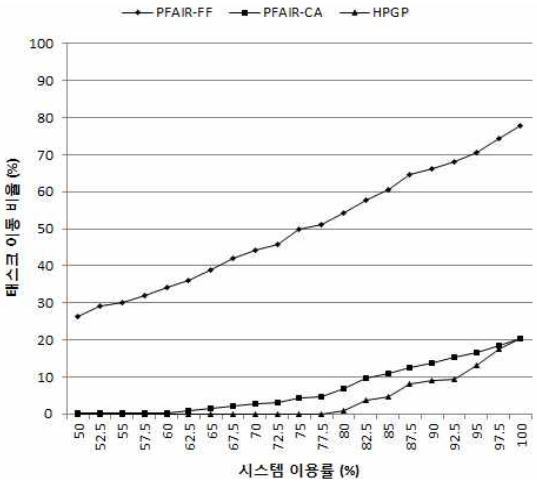
FF, PFAIR_CA를 구현하였다. 실험에서 사용된 태스크 집합은 GNU 과학 라이브러리 (GSL: GNU Scientific Library) [12]의 난수 발생기를 사용하여 무작위로 생성하였다. 또한, 기존의 Pfair 스케줄러와의 공정한 비교를 위해서 [13]에서 사용되었던 태스크 집합의 특성을 똑같이 적용하였다. 실험에서 태스크 집합은 크게 3종류로 분류되며 태스크 집합 유형 0은 낮은 비중을 갖는 태스크 만으로 구성되고, 태스크 집합 유형 1은 높은 비중을 갖는 태스크 만으로 구성되며, 마지막으로 태스크 유형 2는 낮은 비중의 태스크와 높은 비중의 태스크를 혼합한 태스크 집합으로 구성된다. 태스크 유형 0은 태스크의 비중이 [0.1, 0.5) 범위에, 태스크 유형 1은 [0.5, 0.9) 범위에 균일하게 분포하도록 하고, 태스크 유형 2는 낮은 비중의 태스크와 높은 비중의 태스크가 0.2대 0.8의 비율로 분포하도록 하였다. 실험에서는 태스크 집합의 전체 시스템 이용률이 50% 에서 100% 사이가 되도록 (혹은 총 4개의 코어에서 약 2개의 코어에서 4개의 코어가 사용 중 이도록) 유형별로 100개의 서로 다른 태스크 집합을 GSL을 이용하여 생성하였다. 또한, 현재 내장형 프로세서로 채택된 멀티코어 프로세서 중에 가장 많은 코어를 탑재하고 있는 쿼드코어인 ARM11MPCore를 모델로 하여 시뮬레이터를 쿼드코어 상에서 동작한다고 가정하고 설정하였다 [14].



(a) 태스크 집합 유형 0 (낮은 비중의 태스크)



(b) 태스크 집합 유형 1 (높은 비중의 태스크)



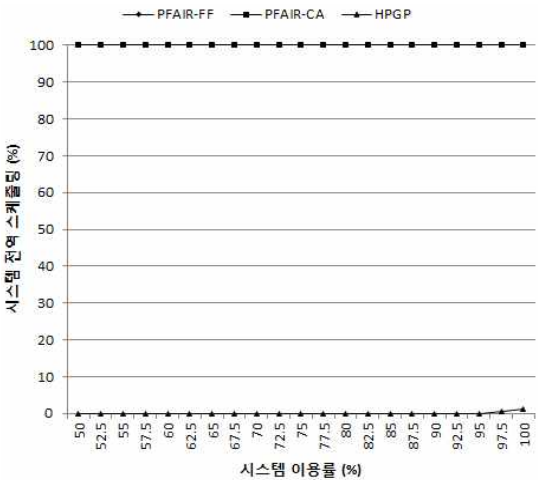
(c) 태스크 집합 유형 2 (혼합된 비중의 태스크)

그림 7. 시스템 이용률에 따른 태스크 이동 횟수의 비율

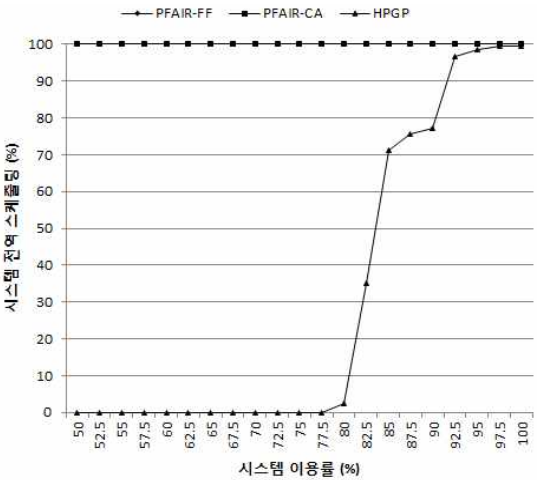
각 스케줄러의 스케줄링 오버헤드를 태스크 이동 횟수 및 시스템 전역 스케줄링 횟수로 비교한 실험 결과는 그림 7, 그림 8과 같다.

시뮬레이션과 측정 실험 결과를 보면 PFAIR-CA 스케줄러가 동일 코어 선호 서브 태스크 할당 방식을 채택함으로써 많은 수의 태스크 이동을 감소시켰지만 여전히 매 스케줄링 때마다 시스템 전역 스케줄링을 하는 오버헤드가 여전히 존재하였다. 반면, 본 논문에서 제안된 HPGP 스케줄러는 PFAIR-CA에 비해 더욱 많은 수의 태스크 이동을 감소시키는 것과 동시에 시스템 전역 스케줄링 횟수를 획기적으로 감소시켰다. 특히 HPGP 스케줄러는 시스템 이용률이 80% 이내인 임의의 태스크 집합에 대해서 태스크 이동과 시스템 전역 스케줄링 횟수를 최소화 혹은 발생시키지 않는 것으로 나타났다.

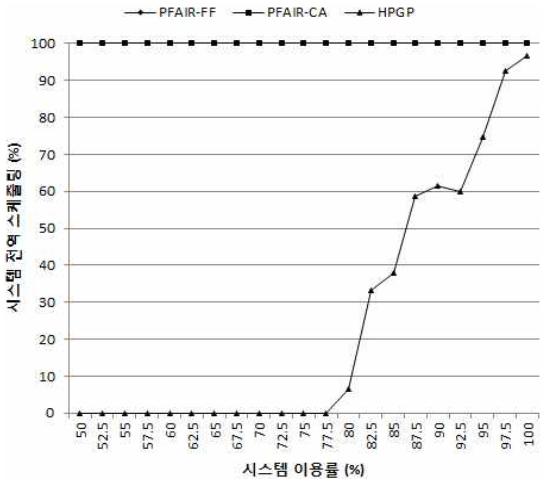
이러한 이산 시간 시간 시뮬레이터에 의한 스케줄링 오버헤드에 대한 성능 평가와 더불어 HPGP 스케줄러의 실제 CPU 수행시간 증가량을 측정하기 위해서 이를 내장형 멀티코어 프로세서에 이식하여 HPGP 스케줄러의 수행시간을 측정하였다. 측정을 위해서는 사이클 단위의 정밀도를 지원하는 VaST CoMET 하드웨어 시스템 시뮬레이터 [15] 상에서 ARM11MPCore 플랫폼을 4개의 코어를 지원하도록 설정하고 동작 주파수를 100MHz로 설정하였다. 이



(a) 태스크 집합 유형 0 (낮은 비중의 태스크)



(b) 태스크 집합 유형 1 (높은 비중의 태스크)



(c) 태스크 집합 유형 2 (혼합된 비중의 태스크)

그림 8. 시스템 이용율에 따른 시스템 전역 스케줄링 횟수의 비율

러한 환경에서 매 스케줄링 지점마다 결정 알고리즘의 수행시간을 측정한 결과 대부분 8μs 이내로 매우 미미하다는 것을 알 수 있다.

5. 관련 연구

최근 들어 멀티코어 프로세서를 위한 스케줄링 알고리즘에 대한 활발한 연구가 진행 중이다. Pfair 알고리즘에 대한 관련 연구들은 대부분 비주기적인 태스크의 지원 [8], 태스크 이동 금지 [5], 경성/연성 실시간 태스크의 동시 지원 [14] 등 새로 추가된 기능에 대한 이론적인 분석에 한정되고 있다.

Pfair 스케줄러의 스케줄링 오버헤드는 Pfair를 실제 시스템에 적용하고자 제안한 DFS (Deadline-Fair Scheduling) [15]에서 지적된바 있다. 이 연구에서는 Pfair 스케줄러의 과도한 태스크 선점과 작은 스케줄링 시간 유닛이 실제 시스템에 Pfair 스케줄러를 적용하기 어렵게 만들기 때문에 제안된 DFS 스케줄러에서는 시간 유닛을 확장하여 이를 공유할 수 있게 하였다. 이와 유사하게 [5]에서는 여러 태스크의 집합을 하나의 슈퍼 태스크로 묶어서 스케줄링 오버헤드를 줄이고자 하였다. 그러나, 이러한 연구들은 모두 스케줄링 오버헤드를 줄이기 위해 스케줄링 가능한 최대 시스템 이용률을 제한하는 단점을 갖고 있다.

반면, [7]의 연구는 태스크의 시간 제약성이 만족해야 할 지점은 각 태스크 별 제한시간 바운더리(Boundary)라는 점을 적용하여 Pfairness를 각각의 태스크들의 제한시간에만 만족시키도록 하여 스케줄링 오버헤드를 줄이는 BFS (Boundary-Fair Scheduling) [7]을 제안하였다. BFS는 최대 시스템 이용률을 그대로 유지하면서 동시에 효과적으로 태스크 선점, 태스크 이동 횟수 및 시스템 전역 스케줄링 횟수를 줄일 수 있지만, 각 스케줄링 지점 사이의 서브 태스크 할당을 McNaughton의 알고리즘 [16]을 사용하여 정적으로 할당해서 태스크의 실행 주기나 새로운 태스크의 추가나 기존 태스크의 제거 등이 발생할 수 있는 동적인 환경에 적합하지 않은 단점이 존재한다.

위와 같은 기존의 연구들과는 달리 HPGP 스케줄러는 코어간의 태스크 이동 및 시스템 전역 락이 필요한 경우를 크게 줄여줌과 동시에 스케줄링 오버헤드를 줄이기 위해 BFS에서와 같은 정적인 기법을 사용 하지 않음으로 인해서 동적인 시스템에도 적용될 수 있는 장점이 있다.

6. 결 론

하나의 프로세서에 여러 코어가 탑재되어 동시에 여러 명령어를 수행할 수 있는 멀티코어 프로세서는 다중 멀티미디어 스트리밍의 처리에 있어서 최적의 구조로 이러한 환경에서는 동영상 혹은 음성과 같은 멀티미디어 컨텐츠를 재생하기 위해서는 매 프레임별로 멀티미디어 데이터를 주기적으로 처리할 수 있는 실시간 스케줄링 알고리즘이 필요하다.

멀티코어 환경에서 스케줄링 오버헤드는 주로 메모리 성능을 저해시키는 코어 간 태스크 이동과 모든 태스크의 수행을 중단시키는 시스템 전역 동기화에 의해 발생한다. 본 논문에서는 시스템 이용률 면에서 최적으로 알려진 Pfair 알고리즘의 코어 분할 버전과 시스템 전역 버전을 혼합함으로써 Pfair 스케줄러의 단점이었던 높은 스케줄링 오버헤드를 크게 낮출 수 있었다. 즉, Pfair 스케줄러를 구현할 때 동일 코어 기반 서브 태스크 할당 방식을 도입함으로 인해서 태스크 이동 횟수를 크게 줄일 수 있음을 보여주었고, 또한 시스템 전역 스케줄링이 절대적으로 필요하지 않은 경우에는 코어 분할 스케줄링을 수행함으로써 시스템 전역 락의 횟수를 크게 감소시키는 것을 보여주었다.

쿼드코어 프로세서에서의 일련의 실험 결과는 HPGP 스케줄러를 사용함으로써 태스크 이동 및 시스템 전역 스케줄링 지점이 크게 줄어든 것을 확인할 수 있었으며, 성능 평가 결과에는 포함되지 않았지만 추가적인 실험을 통해 코어의 개수가 증가하더라도 이러한 경향은 여전히 유효함을 확인하였다. 특히, 태스크 집합의 전체 시스템 이용률이 80% 이내 일 때는 태스크 이동 및 시스템 전역 스케줄링이 거의 필요하지 않고 거의 대부분의 시간에 코어 분할 스케줄러로 동작하는 것을 확인하였다. 또한, 결정 알고리즘의 수행시간을 측정한 결과 약 0.8% 이내의 미미한 CPU 수행시간이 필요한 것으로 나타났다.

스케줄링 오버헤드는 멀티미디어 컨텐츠의 품질에 큰 영향을 미치는 지터(Jitter)의 증가 및 일시적인 수행 정지 등의 문제를 일으키기 때문에 높은 스케줄

링 오버헤드를 갖는 스케줄러의 경우 실제 다중 멀티미디어 스트리밍을 지원하기 위한 시스템에 도입되기 어렵다. HPGP 스케줄러는 이러한 두 가지 면에서 기존의 접근 방법들의 문제점들을 크게 개선하여 멀티코어 프로세서에 적용되었고 향후 보다 효율적인 다중 멀티미디어 스트리밍 시스템을 개발하는데 도움이 될 것이라고 기대된다.

참 고 문 헌

[1] 김용민, 김종면, "고성능 멀티미디어용 멀티코어 하드웨어 설계 및 구현," 제 18회 한국반도체학술대회, pp. 67-68, 2011.

[2] 윤장우, 전승현, 이호연, 허경우, 이현우, 류원, "N-스크린 서비스 분류와 표준화 기술," 한국정보과학회지, 제29권, 7호, pp. 23-31, 2011.

[3] Chung-Ming Huang, Chung-Wei Lin, and Wan-Ping Tsai, "A Multi-Core Based Parallel Streaming Mechanism for Concurrent Video-on-Demand Applications," *IEEE Communications Letters,* Vol.13, No.4, pp. 286-288, 2009.

[4] A. L. Narasimha Reddy, "Scheduling in Multimedia Systems," invited book chapter in *Design and Applications of Multimedia Systems*, Kluwer Academic Publishers, 1995.

[5] 전태건, 김창수, "임베디드 시스템에서 실시간성과 결함허용을 보장하는 스케줄러 설계," 한국멀티미디어학회논문지, 제14권, 1호, pp. 76-84, 2011.

[6] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, "A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms," in *Handbook of Scheduling: Algorithms, Models, and Performance Analysis,* Chapman and Hall/CRC, 2004.

[7] P. Holman and J. H. Anderson, "Using Supertasks to Improve Processor Utilization in Multiprocessor Real-time Systems," *Proceedings of Euromicro Conference on Real-Time Systems*, pp. 41-50, 2003.

[8] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate Progress: a Notion of Fairness in Resource Allocation," *Algorithmica,* Vol.15, No.6, pp. 600-625, 1996.

[9] P. Holman and J. H. Anderson, "Implementing Pfairness on a Symmetric Multiprocessor," *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 544-553, 2001.

[10] J. H. Anderson and A. Srinivasan, "Mixed Pfair/ERfair Scheduling of Asynchronous Periodic Tasks," Proceedings of Euromicro Conference on Real-Time Systems, pp. 76-85, 2001.

[11] S. K. Baruah, J. Gehrke, and C. G. Plaxton, "Fast Scheduling of Periodic Tasks on Multiple Resources," *Proceedings of International Parallel Processing Symposium*, pp. 280-288, 1995.

[12] GNU, "GNU scientific library (http://www.gnu.org/software/gsl/)."

[13] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson, "LITMUSRT : a Testbed for Empirically Comparing Real-time Multiprocessor Schedulers," *Proceedings of IEEE Real-Time Systems Symposium*, pp. 111-126, 2006.

[14] K. Hirata and J. Goodacre, "ARM MPCore; the Streamlined and Scalable ARM11 Processor Core," Proceedings of Asia and South Pacific Design Automation Conference, pp. 747-748, 2007.

[15] VaST Systems, "VaST tools and models for embedded system design (http://www.vast-systems.com/)."

[16] B. B. Brandenburg and J. H. Anderson, "Integrating Hard/Soft Real-Time Tasks and Best-Effort Jobs on Multiprocessors," Proceedings of Euromicro Conference on Real-Time Systems, pp. 61-70, 2007.

[17] A. Chandra, M. Adler, and P. J. Shenoy, "Deadline Fair Scheduling: Bridging the Theory

and Practice of Proportionate Fair Scheduling in Multiprocessor Systems," *Proceedings of IEEE Real Time Technology and Applications Symposium*, pp. 3-14, 2001.

[18] R. McNaughton, "Scheduling with Deadlines and Loss Functions," *Management Science*, 1959.

박 상 수

1998년 한국과학기술원 전산학과 졸업(학사)
2000년 서울대학교 컴퓨터공학과 졸업(공학 석사)
2006년 서울대학교 컴퓨터공학과 졸업(공학 박사)
2006년~2008년 Univ. of Michigan Research Fellow
2008년~2008년 삼성전자 종합기술원 전문연구원
2009년~현재 이화여자대학교 컴퓨터공학과 전임강사
관심분야: 임베디드 소프트웨어, 실시간 시스템, 시스템 시뮬레이션 등