멀티미디어 응용을 위한 수행률에 기반한 실시간 스케쥴링 알고리즘*

박문주○ 조유근
서울대학교 컴퓨터공학과

Execution-Rate Based Real-Time Scheduling Algorithm for Multimedia Applications

Moonju Park Yookun Cho
Dept. of Computer Engineering, Seoul National University

요 약

멀티미디어 응용은 시간 제약 조건을 가지며, 이를 만족시키는 실시간 스케쥴러는 멀티미디어를 지원하는 운영체제의 필수적인 요소이다. 본 논문에서 제안하는 실시간 스케쥴링 기법은 시간 제약 조건의 명세를 이용하여 멀티미디어 응용들의 수행 시간과 지연 시간 변동폭의 상한을 보장한다. 응용은 그 주기와 수행 시간을 명세하며, 스케쥴러는 수행률에 기반하여 응용들을 스케쥴한다. 또한 본 논문에서는 제안된 스케쥴링 기법을 RT-MACH[6] 운영체제 상에서 구현하여 실제로 수행 시간과 지연 시간 변동폭의 상한이 보장됨을 보인다.

1. 서론

멀티미디어 응용을 지원하기 위한 운영체제는 동적이며 선점가능한 실시간 스케쥴링 기능을 제공해야 한다. 현재 수행중인 태스크보다 긴급한 태스크가 도착했을 때 더 긴급한 태스크를 수행시키기 위하여 선점이 필요하며, 동적인 스케쥴링은 현재의 부하 상태를 고려하여 태스크의 스케쥴을 결정하기 위하여 필요하다. 또한 멀티미디어 응용의 스케쥴링은 시간제약 조건의 만족뿐 아니라 목적지(destination)로 전송되는 데이타의 전송시간까지 고려해야 한다. 지터(Jitter)는 데이타의 전송 시간의 변화 정도(variation)을 나타낸다. 대부분의 멀티미디어 응용이 연성 실시간 특성을 가짐을 고려할 때, 서비스의 질(QoS : Quality of Service)은 지터를 중요한 인자로 하게 된다.

일반적으로 사용자는 정확한 응용의 수행시간을 알 수 없으며, 또한 시스템에 예기치 않은 과부하 조건(overload condition)이 발생할 수 있다. 이러한 상황에서 멀티미디어 응용의 수행을 보장하기 위해서는 다른 응용이 명세에 따라 행동하는가(well-behave) 또는 그렇지 않은가(badly-behave)에 상관없이 명세된 수행 시간을 보장해 주는 방법이 필요하다. 고전적인 RM(Rate-Monotonic)이나 EDF(Earliest Deadline First) 알고리즘[8]들은 QoS에 대한 고려가 없기 때문에 이러한 경우 수행 시간을 보장할 수 없다. 다른 응용의 수행에 상관없이 일정한 비율로 수행하도록 강제하는 알고리즘에 대한 연구는 [10], [11], [2] 등이 알려져 있다.

본 논문에서 제안하는 스케쥴링 알고리즘은 다른 태스크의 수행에 영향을 받지 않고 태스크의 수행 시간을 보장하는 데에 중점을 두며, 지연 지터 (delay jitter)의 상한 또한 보장하도록 한다.

본 논문은 다음과 같이 구성되어 있다. 2.장에서는 멀티미디어 응용을 위한 스케쥴링 알고리즘을 제안하고 그 특성을 설명한다. 3.장에서는 제안된 알고리즘의 구현에 관하여 설명하고, 4.장에서는 구현된 알고리즘의 실험과 그 성능을 설명한다. 마지막으로 5.장에서 결론을 맺는다.

* 이 연구는 특정목적기초연구 사업의 지원을 받아 이루어 졌습니다

2. 수행률 기반 스케쥴링

2.1 스케쥴링 알고리즘

모든 태스크는 수행하기 전에 주기와 상대적인 데드라인(relative deadline) 및 계산 요구량을 명세하도록 한다. 상대적 데드라인은 주기보다 작거나 같다. 커널은 태스크의 명세를 받아 그 태스크의 수행률을 계산한다. 한 태스크 Q의 수행률 erate(Q)는 다음과 같이 계산한다.

$$erate(Q) = \frac{computation\ time}{relative\ deadline}$$

주기적으로 수행하는 멀티미디어 응용의 수행 시간 및 지연의 상한을 보장하기 위하여, 스케쥴러는 다음과 같은 알고리즘을 수행한다. 여기에서 현재 처리기를 차지하고 수행중인 태스크를 T_i라고 하자.

스케쥴러에서 {
  runtime(T_i) = 현재 수행중인 태스크가 마지막으로 스케쥴되어 수행한 시간량
  P = 대기큐에서 $vstart$가 가장 작은 태스크
  $vstart(T_i) = vstart(T_i) + \frac{runtime(T_i)}{erate(T_i)}$
  if $\left(\lceil \frac{vstart(T_i)}{P_i} \rceil == \lfloor \frac{vstart(T_i)-D_i}{P_i} + 1 \rfloor\right)$
    $vstart(T_i) = vstart(T_i) + (P_i - D_i)$
  if($vstart(P) \leq curtime$ and $vstart(T_i) > vstart(P)$)
    T_i를 선점하고 P를 수행
}
태스크를 대기큐에 넣을 때 {
  P = 대기큐에 넣을 태스크
  if($vstart(P) < curtime$)
    vstart(P)=curtime
}

표1. 스케쥴링 알고리즘

curtime은 현재 시간을 나타내며, P_i, D_i는 각각 태스크 T_i의 주

기와 상대적 데드라인을 나타낸다. vstart는 태스크의 다음 수행될 시간을 의미하며, 태스크들의 우선순위로서 사용된다. 따라서 vstart가 빠른 태스크가 먼저 수행되도록 스케쥴된다. 초기에 vstart는 그 태스크가 최초로 수행된 실제 시간으로 설정된다. vstart가 설정된 태스크들은 대기 큐로 들어가게 된다. 대기큐에 새로운 태스크가 들어오면 스케쥴러는 대기 큐의 태스크들을 vstart가 작은 순서로 정렬한다. 따라서 매 스케쥴링 시점에서 스케쥴러는 대기큐의 가장 앞에 있는 태스크의 vstart와 현재 수행중인 태스크의 갱신된 vstart를 비교하여 수행시켜야할 태스크를 결정한다.

2.2 승인 제어

새로운 태스크가 생성되어 실시간 요구조건을 명세하면, 시스템은 요구조건을 만족시키는 스케쥴이 가능하지를 검사한다. 제안된 알고리즘은 주기보다 짧은 데드라인을 가지는 것을 허용한다. 주기와 데드라인이 다른 태스크 집합의 스케쥴 가능성을 검사하는 것은 NP 문제로 알려져 있으나[1], pseudo-polynomial 시간에 수행이 가능한 알고리즘들이 존재한다[1][9]. 또 부정확하나 효율적인 알고리즘을 사용할 수도 있다[7].

2.3 주기 응용의 실행 시간 보장

제안된 알고리즘은 클럭 주기의 크기와 문맥교환의 오버헤드를 무시할 수 있는 경우 주어진 데드라인이전에 명세된 수행시간만큼 수행할 수 있도록 보장한다. 다른 태스크가 명세를 지키지 않고 수행을 할 경우에도, 그 응용의 vstart는 더 큰 값으로 설정되며 알고리즘에 의해 수행을 할 수 없게 된다. 문맥교환의 오버헤드가 무시될 수 있는 이상적인 경우 이 알고리즘은 비트단위의 멀티플렉싱을 하는 것처럼 보이며, 명세된 수행시간을 완전히 보장한다. 그러나 실제의 경우 문맥교환의 오버헤드는 무시할 수 없으며, 제안된 알고리즘에서 하나의 태스크 T_i는 그 데드라인 D_i까지 최악의 경우 $\frac{D_i}{time\ quantum}$번 선점당할 수 있다. 따라서 시간 할당량을 조절하여 선점의 횟수를 줄일 수 있으나 시간 할당량을 줄이면 수행 시간은 편차가 더 커질 수 있다.

2.4 지연 지터의 상한 보장

지연 지터는 응용의 QoS에 영향을 미치며 QoS에 대한 보장은 멀티미디어 응용의 지원에 가장 중요한 측면이므로, 응용의 수행에 있어 그 지연 시간의 크고 작음은 매우 중요한 성능 요소(performance factor)이다. 주기와 데드라인이 같은 경우 최대 지터의 크기는 $P_i - C_i$로 주어진다. 제안된 알고리즘에서는 주기보다 짧은 데드라인(D_i)을 명세함으로써 지연 지터는 $(D_i - C_i)$보다 작게 된다. 그러나 실제 구현에 있어서는 클럭의 크기가 유한하고, 커널에 대한 시스템 호출에서 선점불가능한 부분이 있으므로 지연 지터의 상한은 $(D_i - C_i) + 2 \times (time\ quantum)$+(선점불가능시간) 으로 주어진다.

3. 구현

본 논문에서 제안된 스케쥴링 기법은 RT-MACH상에서 하나의 서버로서 구현되었다. RT-MACH는 "Processor Capacity Reserves[3]"라는 추상(abstraction)을 제공하며, 이를 기반으로 주기와 처리기 사용시간을 명세할 수 있다. 또한 스레드의 수행시간에 대한 정보를 제공한다. 그러나 'reserves 추상'만으로 스레드의 수행시간을 보장할 수는 없으며, 이는 스케쥴러가 담당해야 한다[4]. 제안된 스케쥴링 알고리즘을 적용하기 위해서, 실제로 어떤 스레드의 작업이 도착하는 것을 감시하는 것은 커널에 새로운 작업을 부과하므로, 클럭 인터럽트가 발생할 때마다 csw_needed를 호출하고, csw_needed는 태스크들의 우선순위를 계산하여 문맥교환이 필요한지를 검사한다.

비실시간 스레드를 스케쥴하는 스케쥴러는 일반적인 UNIX의 시분할(time-sharing) 우선순위 알고리즘에 따라 스케쥴된다[5]. 한 프로세서에 실시간 스레드와 비실시간 스레드가 존재할 경우, 실시간 스케쥴러는 실시간 스레드를 위한 큐와 비실시간 스레드를 위한 큐를 따로 가진다. 비실시간 스레드는 실시간 스레드가 수행되지 않을 경우에만 수행이 가능하며, rem_runq가 불릴 때 실행가능한 스레드가 없음을 알려주게 되므로 그러한 경우에 비실시간 스케쥴러를 호출하여 비실시간 스레드들을 스케쥴하도록 한다.

thread_enqueue에 의해 스레드가 실행 큐로 진입할 때, 스케쥴러는 실행 큐의 스레드들을 그들의 vstart에 따라 정렬한다. csw_needed가 불리면 스케쥴러는 실행 큐의 맨 앞에 있는 스레드 즉 vstart가 가장 작은 스레드와 현재 수행중인 스레드의 vstart를 비교하여 문맥 교환이 필요한지를 결정한다. 이 때 사용자 수준 스케쥴러는 커널 수준 스케쥴러의 실행 큐도 같이 관리하여 커널 수준 스케쥴러와의 통신과 같은 복잡한 문제들을 회피하도록 하였다.

4. 실험 결과 및 분석

실험은 두 대의 컴퓨터를 LAN으로 연결하여 수행하였다. 하나의 컴퓨터는 서버로서 멀티미디어 데이타를 송신하고, 다른 하나의 컴퓨터는 수신된 데이타의 도착 시간 간격(inter-arrival time)을 측정하도록 한다. 서버는 인텔 486DX2-66 처리기와 8MB RAM, 3C503 LAN NIC를 가지며, 본 논문에서 제안한 스케쥴링 기법이 구현된 RT-MACH 운영체제를 사용한다. 수신측의 컴퓨터는 SPARC-10 처리기와 8-bit LAN NIC를 가지며, 운영체제로는 Solaris 2.5 를 사용한다. 수신측의 컴퓨터에는 본 논문에서 제안한 스케쥴링 기법이 구현되어 있지 않다. LAN은 1Mbps Ethernet으로 구성된다.

실험을 위하여 주기 200ms, 수행시간 20ms(erate = 0.1)인 한 번에 6KB의 데이타를 보내는 주기 스레드와 gsin-30000이라는 한 번에 30000번의 sin 계산을 수행하는 태스크를 같이 수행하였다. gsin-30000은 주기 50ms, 수행시간 10ms(erate = 0.2)로 명세하나, 실제로는 한 번 수행에 약 8ms의 처리기 시간을 요구하며, 주기에 상관없이 하나의 계산이 끝나면 바로 다음 계산을 수행한다.

그림 1은 기존에 RT-MACH에 구현된 스케쥴러가 하나의 주기 스레드와 하나의 gsin-30000을 스케쥴했을 때의 목적지에서의 데이타 도착시간 간격 분포를 나타낸다. x축은 데이타 전송 횟수를 나타내며, y축은 도착 시간 간격을 밀리초(ms) 단위로 나타낸 것이다. 그림 2는 그림 1의 1000ms 이하 부분을 확대한 것이다. gsin-30000은 주기 스레드가 수행한 후에 수행되었다. 그림 1에서 볼 수 있는 바와 같이, 스케쥴러는 우선순위가 높은 태스크인 gsin-30000을 수행시켜 주기 스레드의 수행을 보장하지 못함을 알 수 있다. 이 때 최대의 지연 시간은 약 232초 정도이다.

그림 3는 하나의 주기 스레드와 5개의 gsin-30000을 수행했을 때의 목적지에서의 데이타 도착 시간 간격을 측정한 것이다. 5개의 gsin-30000중 하나의 스레드는 주기 스레드 수행 이전에 수행되었다. 그림 3에서 볼 수 있는 바와 같이, 주기 스레드 수행 이전에 수행된 스레드는 주기 스레드의 수행에 영향을 미치지 못한다. 또한 주기 스레드 수행 이후에 도착한 잘못 행동하는 태스크들도 수행시간에는 영향을 미치지 못한다. 그러나 목적지에서의 지터 지연은 발생할 수 있으며, 이 경우 최대 319ms, 최소 81ms의 도착 간격시간을 보인다. 그림 4은 주기 응용에 주기보다 짧은 데드라인을 주어 실험한 결과이다. 데드라인은 50ms로 주어졌으며 따라서 지터는 (50ms-20ms)=30ms로 예상할 수 있다. 실제로는 이 값보다 큰 값을 나타내는데, RT-MACH에서 프로세스를 생성할 때 커널이 선점불가능하므로 gsin-30000을 생성하는 데 드는 시간(메모리 적재 시간) 동안 지연되기 때문이다. 그 시간은 약 50ms로써 실험 결과와 일치한다.

5. 결론 및 향후 과제

멀티미디어 응용을 지원하는 운영체제는 멀티미디어 응용의 시간 제약 조건을 만족시켜야 하며, 여러 응용이 수행되는 환경하에서도 멀티미디어 응용의 수행을 보장할 수 있어야 한다. 그러나 고전적인 스

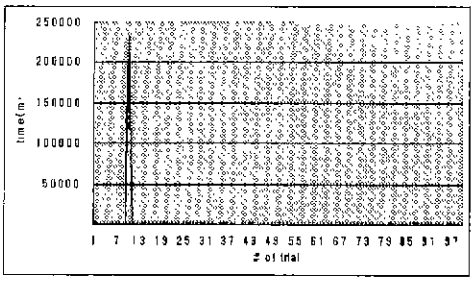

그림 1: RM 스케쥴링 – 주기 태스크와 gsin-30000을 수행시켰을 때의 도착시간 간격 분포

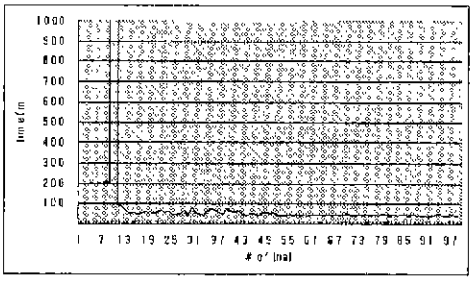

그림 2: 그림 1의 확대

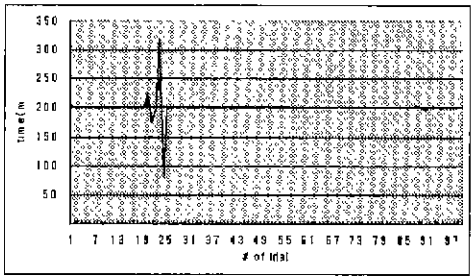

그림 3: 제안된 스케쥴링 – 주기 태스크와 5개의 gsin-30000을 수행시켰을 때의 도착시간 간격 분포

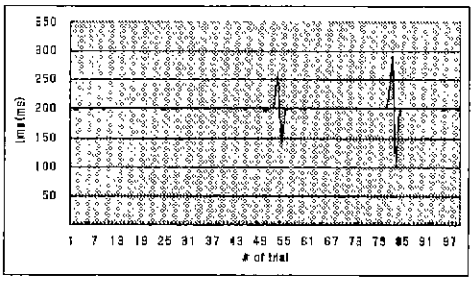

그림 4: 제안된 스케쥴링 – 주기보다 짧은 데드라인을 설정한 결과

케쥴링 알고리즘을 이용해서는 이러한 요구조건들을 만족시키지 못한다. 특히, 잘못 행동하는(badly behave) 태스크가 존재하는 경우 대부분의 시스템에서 채용하고 있는 EDF나 RM과 같은 알고리즘은 실시간 응용의 수행을 전혀 보장해 줄 수 없다.

본 논문에서는 멀티미디어 응용의 수행에 요구되는 조건들을 만족시킬 수 있는 수행률에 기반한 스케쥴링 알고리즘을 제안하였다. 본 논문에서 제안된 스케쥴링 알고리즘은 태스크가 잘못 행동하는 상황에서도 주기적으로 수행하는 멀티미디어 응용의 예측 가능한(predictable) 수행을 보장한다. 또한 네트워크를 이용하는 응용에 데이타를 전송할 경우, 네트워크 지연이 일정하다면 지연 지터의 변동폭을 제한시켜 목적지에서 데이타의 수신과 처리를 예측할 수 있도록 한다.

본 논문에서 제안된 스케쥴링 알고리즘은 RT-MACH상에서 사용자 수준 스케쥴러로 구현되었다. 실험 결과는 제안된 스케쥴링 알고리즘이 태스크들의 수행시간을 보장하며, 특별한 폴리싱(policing) 기법이 없이도 다른 태스크의 영향을 받지 않도록 수행할 수 있음을 보여준다.

References

[1] S. K. Baruah, L. E. Roiser, and R. R. Howell, "Algorithms and Complexity Comcerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor", Real-Time Systems, 2(4), Nov. 1990

[2] S K. Baruah, J. E. Gehrke, C. G. Plaxton, I. Stoica, H. Abdel-Wahab, and K. Jeffay, "Fair On-Line Scheduling of a Dynamic Set of Tasks on a Single Resource", Information Processing Letters, 64(1), Oct 1997

[3] C. Lee, K. Yoshida, C. Mercer, and H. Tokuda, "Processor Capacity Reserves: Operating System Support for Multimedia Applications", Proceedings of the Real-Time Technology and Applications Syposium, Jun. 1996

[4] C. W. Mercer, R. Rajkumar, and H. Tokuda, "Applying Hard Real-Time Technology to Multimedia Systems", Proceedings of the Workshop on the Role of Real-Time im Multimedia/Interactive Computing Systems, Nov. 1993

[5] D. B. Golub, "Operating System Support of Coexistence of Real-Time and Conventional Scheduling", Technical Report, CMU-CS-94-212, Carnegie Mellon University, May 1994

[6] H. Tokuda, T. Nakajima, and P. Rao, "Real-Time Mach: Towards a Perdictable Real-Time System", Proceedings of USENIX Mach Workshop, Oct. 1990

[7] L. George, N. Rivierre, and M. Spuri, "Preemptive and Non-Preemptive Real-Time Uniprocessor Scheduling", Research Report, INRIA, Sep 1996

[8] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", Journal of ACM, 20(1), Jan. 1973

[9] I. Ripoll, A. Crespo, and A. L. Mok, "Improvements in Feasibility Testing for Real-Time Tasks", Real-Time Systems, 11(1), Jul. 1996

[10] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton, "A Proportional Share Resource Allocation Algorithm For Real-Time, Time-Shared Systems", Proceedings of 17th IEEE Real-Time Systems Symposium, Dec. 1996

[11] C. A. Waldspurger and W. E. Weihl, "Lottery and Stride Scheduling· Flexible Proportional Share Resource Management", PhD Thesis, Technical Report, MIT/LCS/TR-667, Laboratory of CS, MIT, Sep. 1995