멀티프로세서에서 (m, k)-firm Deadline 을 가지는 태스크를 위한 실시간 스케줄링 알고리즘

공연화, 조현중
고려대학교 컴퓨터 정보학과
e-mail :{misticlime, raycho} @korea.ac.kr

A Guaranteed Real-time Scheduling Algorithm for (m,k)-firm Deadlines Constrained Tasks on Multiprocessors

Yeonhwa Kong, Hyeonjoong Cho
Dept. of Computer Information Science, Korea University

요 약

본 논문은 동종의 멀티코어에서 (m,k)-firm Deadline 을 가지는 태스크를 위한 실시간 시스템의 스케줄링 기법을 제안한다. 본 논문에서 제안된 알고리즘의 목적은 (m, k)-firm Deadline 을 만족시키는 확률을 증가시켜 최대의 Quality of Service 를 제공하는 것이다. 본 논문에서는 제안된 알고리즘이 QoS 를 보장함을 분석적으로 보이고 실험을 통해 알고리즘의 효율성을 검증한다.

1. 서론

실시간 시스템 중 경성 실시간 시스템은 각 태스크의 데드라인을 반드시 만족 시켜야만 하는 시스템인 반면 연성 실시간 시스템은 경우에 따라서 데드라인을 맞추지 못한 것이 허용된다. 연성 실시간 시스템에서 데드라인을 만족하지 못하는 정도를 나타내는 전형적인 방법 중 하나는 시스템이 허용하는 데드라인을 맞추지 못하는 최대의 비율을 표시하는 것이다. 그러나 그런 확률적인 표현은 [1]이 지적한 바와 같이, 주어진 시간 동안 데드라인을 맞추지 못하는 것이 고르게 발생하는 경우와 데드라인을 맞추지 못하는 것이 연속적으로 나타나는 경우를 구별할 수 없다.

Hamdaoui은 이러한 연성 실시간 시스템의 시간 제약을 정확하게 명시하기 위해 *(m, k)-firm Deadline* (0<m≤k)를 정의했다. *(m, k)-firm Deadline*은 시스템의 Quality of Service를 정확히 나타내기 위한 것으로 하나의 태스크에서 k개의 연속적인 Job 중에서 적어도 m개의 데드라인을 맞추어야 하는 것을 의미한다. 예를 들면, (9,10)-firm Deadline과 (90,100)-firm Deadline은 확률적으로는 데드라인을 놓치는 허용 비율이 모두 10%로 같지만, 두 경우에서 데드라인을 놓치는 Job들의 분산은 다르다.

어떤 태스크가 주어진 *(m, k)-firm Deadline* 를 위반하는 것, 즉, k개의 연속적인 Job에서 m보다 적게 데드라인을 맞추는 경우를 Dynamic Failure라 한다. 그러므로 Dynamic Failure가 일어나는 비율은 얼마나 자주 Quality of Service가 요구 수준 아래로 떨어지는지를 측정하는 방법으로 사용할 수 있다.

(*m, k*)-firm Deadline으로 제한된 다수의 실시간 태스크를 스케줄링하는 문제에 대한 접근 방법은 정적인 방법과 동적인 방법으로 분류된다. 정적인 방법은 태스크 스케줄링이 미리 결정된 순서를 따르는 반면, 동적인 방법은 시스템이 운영되는 동안 동적으로 스케줄링을 결정된다. Hamdaoui은 동적인 방법으로 *Distance-based Priority* (DBP) 개념을 제안하였는데, DBP는 태스크의 데드라인 만족 여부를 누적하고 이를 바탕으로 Dynamic Failure로 확률이 높은 태스크에게 높은 우선순위를 주는 스케줄링 방법이다 [1]. Off-line에서 미리 스케줄링을 해야 하는 정적인 방법에 비해 동적인 스케줄링 방법이 환경적인 제한에 대해 좀더 유연하기 때문에 본 논문에서는 동적인 방법에 초점을 둔다.

Proportionate fair (Pfair) 알고리즘은 멀티프로세서를 위한 최적 실시간 스케줄링이다[2]. Pfair 이후 Pfair의 오버헤드를 줄이기 위해 여러 가지 향상된 알고리즘들이 제안되었다 [3,4,5,6,7]. 특히 [6,7]은 *Time and Local Remaining Execution-time Plane* (T-L 평면)을 제안했는데, T-L 평면은 멀티프로세서에서 주기적인 태스크들의 실행 상태를 시각적으로 추론할 수 있는 추상화 도구로 이를 기초로 멀티프로세서 스케줄링의 분석적인 이해를 가능하게 해준다.

본 논문에서는 동종의 멀티프로세서에서 (*m, k*)-firm Deadlines으로 제한된 주기 태스크를 스케줄링하는 기법을 제안한다. 스케줄링 목표는 다음과 같다

(1) 시스템의 부하가 언더로드인 상태에서는 (m, k)-firm Deadline이 주어진 태스크들의 Dynamic Failure 확률이 제한됨을 보장해야 한다.
(2) 시스템의 부하가 오버로드인 상태에서는 데드라인을 만족하는 확률을 가능한 향상시켜 시스템이 제공하는 QoS를 최대화해야 한다.

본 논문에서 제안하는 (m, k)-firm Deadline을 가지는 태스크들을 위한 스케줄링 알고리즘(GMRTS-MK)은 멀티프로세서에서 (m, k)-firm 데드라인을 가진 태스크의 QoS를 보장하는 첫 번째 동적 실시간 스케줄링 알고리즘이다. GMRTS-MK는 (1,1)-firm 데드라인을 가지는 일반적인 경성 실시간 태스크들이 주어졌을 때, 시스템의 부하가 언더로드일 경우 최적으로 알려진 Pfair로 동작하여 태스크의 데드라인을 100% 만족시킨다.

2. 관련 연구

단일 프로세서에서 (m,k)-firm Deadline으로 제한된 실시간 태스크를 스케줄링하는 연구는 적지 않다. 그러나 비교적 최근에서야 멀티프로세서에서 (m,k)-firm Deadline으로 제한된 태스크를 스케줄링 하는 연구가 있었다. Wu는 DBP를 멀티프로세서로 확장한 MPDBP를 제안하였다. [8] MPDBP는 태스크의 Dynamic failure까지의 논리적 거리를 구해서 우선 순위를 정하고 스케줄링 이벤트가 발생될 때 가장 높은 우선 순위를 가진 태스크를 선택한다. 그러나 MPDBP는 Dynamic Failure 확률의 제한을 통해 QoS를 보장하지 않는다. 또, MPDBP는 시스템 부하가 언더로드인 상태에서도 100% 데드라인을 만족시키지 못한다.

3. 알고리즘

GMRTS-MK는 상위 스케줄링과 하위 스케줄링으로 이루어져있는 계층적인 알고리즘이다. 상위 스케줄링은 모든 태스크의 각 주기마다 실행되는데, 하위 스케줄링으로 넘겨줄 태스크를 선택한다. 반면에 하위 스케줄링은 매 시간 퀀텀마다 일어나며 상위 스케줄링에서 선택된 태스크들을 좀더 세밀하게 스케줄링 한다. 따라서 각 태스크의 주기마다 상위 스케줄링이 실행될 때에는 반드시 하위 스케줄링이 따라 실행되며, 각 시간 퀀텀마다 하위 스케줄링이 실행될 때 태스크의 주기가 아닐 경우는 상위 스케줄링이 일어나지 않는다. 본 논문에서는 멀티프로세서에서 최적 스케줄링으로 알려진 Pfair를 시간 퀀텀마다 실행되는 하위 스케줄링으로 사용한다. GMRTS-MK는 알고리즘1에서 설명한다.

알고리즘 1. GMRTS-MK 알고리즘

```
1: Input: Three Queues that contain tasks
2: ζ^A :a queue containing tasks admitted for the low-level scheduling
3: ζ_U^NA :a queue containing non-admitted and urgent tasks
4: ζ_NU^NA :a queue containing non-admitted and non-urgent tasks
5: Output: Three Updated Queues
6: Variables: U = 0, initialize U at every boundary
7:
8: At every boundary of each task τ_k:
9: τ_k = getTask( );
10: updateDist( τ_k );
11: if τ_k is urgent then insert( τ_k , ζ_U^NA );
12: else insert( τ_k , ζ_NU^NA );
13: for all τ_i ∈ ζ^A do U = U + τ_i.r; end for
14: if U < M then
15: for all τ_i ∈ ζ_U^NA do updateTER(τ_i); end for
16: for all τ_i ∈ ζ_NU^NA do updateTER(τ_i); end for
17: sortLEF(ζ_U^NA);
18: sortLDF(ζ_NU^NA);
19: for all τ_i ∈ ζ_U^NA do
20:   if U + τ_i.r ≤ M then
21:     move( τ_i , ζ^A , ζ_U^NA );
22:     U = U + τ_i.r;
23:   else break;
24:   end if
25: end for
26: for all τ_i ∈ ζ_NU^NA do
27:  if U + τ_i.r ≤ M do
28:     move( τ_i , ζ^A , ζ_NU^NA );
29:     U = U + τ_i.r;
30:  end if
31: end for
32: end if
33: At every time quantum:
34:     PfairScheduling(ζ^A );
```

GMRTS-MK 는 다음의 변수와 함수들을 사용한다.

- ζ^A , ζ_U^{NA} , ζ_{NU}^{NA} : 하위 스케줄링을 받게 될 태스크를 누적할 큐, 하위 스케줄링으로 넘겨지지 않지만 Dynamic Failure 까지의 거리가 1 이하인 태스크가 있는 큐, 하위 스케줄링으로 넘겨지지 않지만 거리가 1 이상인 태스크가 있는 큐.
- U : 하위 스케줄링을 받을 태스크들의 Execution Rate(실행시간/한 주기)의 합.
- $\tau_i.r$: τ_i 의 Execution Rate.
- getTask( ) 태스크를 꺼내는 함수.
- updateDist(τ_k) τ_k 가 최근에 데드라인 히스토리를 확인하여 τ_k 의 Distance 를 업데이트함.
- insert(τ_k , ζ) τ_k 를 큐 ζ에 넣음.
- updateTER(τ_i) τ_i 의 Execution Rate 를 업데이트하는 함수

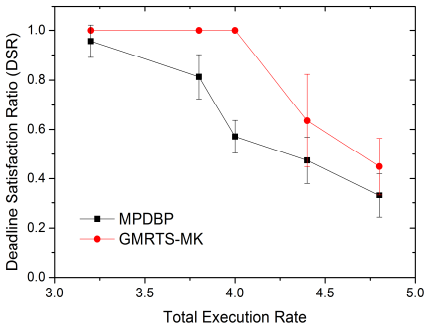
- sortLEF(ζ) 큐 ζ 에 있는 태스크들을 Execution Rate가 적은 순서대로 정렬한다..
- sortLDF(ζ) 큐 ζ 에 있는 태스크들을 Distance가 짧은 순서대로 정렬함.
- move(τ_i , ζ_1, ζ_2) 큐 ζ_2 에 있는 태스크 τ_i를 꺼내서 queue ζ_1에 넣음.
- PfairScheduling(ζ) 큐 ζ 을 매 시간 퀀텀마다 PFair 알고리즘으로 스케줄링. [2, 3]

줄 8 에서 32 까지는 상위 스케줄링이며 이것은 각 태스크의 주기마다 실행되고 줄 33 에서 34 까지 하위 스케줄링이며 매 시간 퀀텀마다 실행된다.

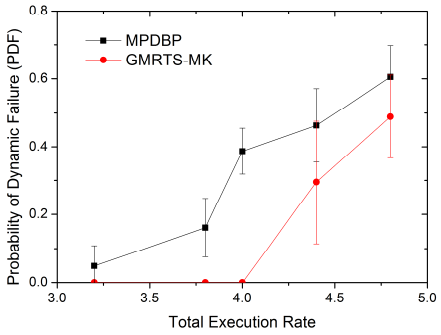
4. 실험 및 결과

본 실험에서는 GMRTS-MK가 Dynamic Failure 확률의 제한으로 QoS를 보장함을 입증한다. 또한 다양한 시스템 로드에서 데드라인을 만족하는 비율을 구하여 GMRTS-MK의 성능을 평가했는데, 이러한 비율은 실시간 스트림 서비스의 실제 QoS을 나타낸다. 실험에는 MPDBP[8]을 본 논문에서 제안하는 GMRTS-MK의 비교대상으로 선정했다. MPDBP는 멀티프로세서에서 (m, k)-firm Deadline으로 제한된 실시간 태스크들을 스케줄링하기 위한 알고리즘이다. GMRTS-MK의 오버헤드를 측정하는 비교대상으로는 멀티프로세서에서 최적 실시간 스케줄링으로 알려진 Pfair를 사용하였다. GMRTS-MK 및 비교할 알고리즘들은 Linux 테스트베드인 LITMUS에 구현하였고, 4개의 코어를 지닌 Intel® Core™i7 Quad Processor에서 운영하였다.

본 논문에서는 (2, 3)-firm Deadline으로 제한된 태스크를 무작위적으로 만들었다. 태스크의 주기는 30~ 100msec 이고 태스크의 실행 시간은 주기보다 작거나 같다. 그리고 8~10개의 태스크를 이용하여 총 Execution Rate의 합을 3.2~4.8로 하였다. 실험에 사용한 멀티프로세서의 코어 개수는 4개이므로 위와 같은 총 Execution Rate는 시스템 부하의 언더로드와 오버로드 모두를 포함하고 있다. 각각의 총 Execution Rate에서 10회 실험하여 평균값을 구하였다. 모든 실험 결과에서 각각의 데이터 포인트 주변의 에러 막대는 95%의 신뢰구간을 시각적으로 나타낸다.



(a)

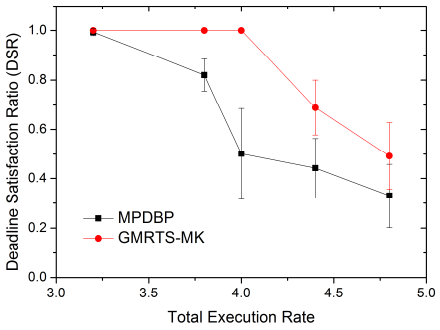


(b)

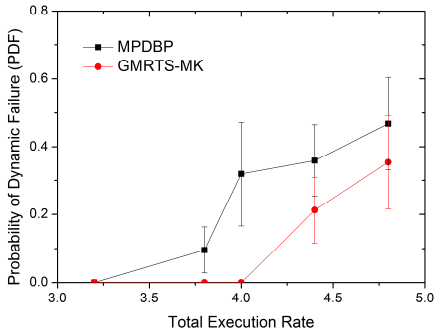
(그림 1) (2,3)-firm Deadline으로 제한된 Task의 DSR 과 PDF

(그림 1)은 MPDBP 과 GMRTS-MK의 Deadline Satisfaction Ratio (DSR)와 Probability of Dynamic Failure (PDF)를 보여준다. 총 Execution Rate가 4보다 작은 언더로드인 상황에서 GMRTS_MK는 100%의 DSR과 0%의 PDF를 보여준다. 게다가 GMRTS-MK는 오버로드인 상황에서 DSR를 최대화하고 있고 PDF를 최소화하여 결국 실시간 스트리밍 서비스의 QoS를 최대화할 수 있음을 보였다. 반면에 MBDBP는 실행 결과에 있어서 어떠한 보장도 하지 않으며 GMRTS-MK보다 DSR과 PDF가 나쁘다.

(그림 2)은 (2,5)-firm Deadlines를 가진 태스크를 스케줄링 했을 때의 DSR과 PDF를 보여준다. 이전 실험과 같이, GMRTS-MK는 언더로드에서 DSR과 PDF의 성능이 보장되고 오버로드에서도 MBDBP보다 성능이 우수하다.



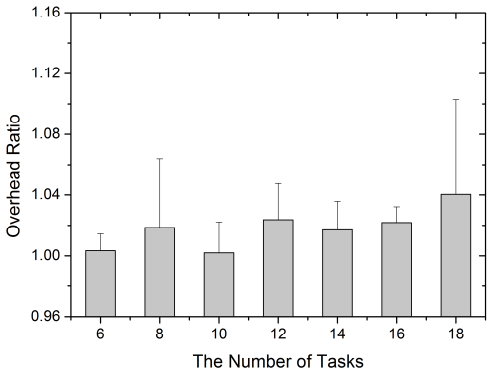
(a)



(b)

(그림 2) (2,5)-firm Deadline으로 제한된Task의 DSR 과 PDF

GMRTS-MK의 실용성을 살펴보기 위해 (그림 3)과 같이 Overhead Ratio를 정의하였다. Overhead Ratio는 일정 시간 동안에 GMRTS-MK의 계산 Overhead를 Pfair의 계산 Overhead으로 나눈 값이다. 실험은 고정된 Execution Rate인 3.6 근처에서 (2, 5)-firm Deadline을 가진 태스크 여러 개를 랜덤하게 생성했다. (그림 3)와 같이 Overhead Ratio는 1.0과 1.1사이이다. 비록 Overhead Ratio가 태스크가 늘어남에 따라 증가하는 경향이 있지만 단일 시스템에서 동시에 운영되는 멀티미디어 스트림 플레이어의 개수는 많지 않으므로 실험에서 확인된 값은 실제 시스템에 적용될 정도로 실용적이라 할 수 있다.



(그림 3) GMRTS-MK 과 Pfair의 수행 시간

5. 결론

본 논문은 멀티프로세서에서 (*m*, *k*)-firm Deadline 을 가진 태스크의 수행 성능을 보장하는 실시간 스케줄링 알고리즘인 GMRTS-MK 를 소개했다. GMRTS-MK 는 두 가지 스케줄링 목적이 있다.(1) 시스템 부하가 언더로드인 상태에서 (*m*, *k*)-firm Deadline 을 가진 여러 태스크들의 Dynamic Failure 확률을 제한한다. (2) 태스크들의 데드라인을 만족하는 확률을 가능한 만큼 향상시킨다. 본 논문에서 제안하는 GMRTS-MK 는 멀티프로세서에서 (*m*, *k*)-firm Deadline 를 가진 태스크의 수행 성능을 보장하는 첫 번째 스케줄링 알고리즘이다. 또 GMRTS-MK 가 데드라인을 만족하는 비율을 최대화하여 멀티미디어 스트리밍 서비스의 QoS 를 향상시킬 수 있음을 보였다. 실험을 통해 GMRTS-MK 의 이론적 결과를 확인하였고 기존의 알고리즘에 비해 효과적인 성능을 가짐을 보였다.

“ 본 연구는 2010 년 한국연구재단의 기본연구지원사업으로 수행 되었음 [2010-009251]”

참고 문헌

[1] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. In IEEE Transactions on Computers, pages 1443-1451, 1995.

[2] S. Baruah, N. Cohen, C.G. Plaxton, D. Varvel, Proportionate progress: A notion of fairness in resource allocation, Algorithmica 15 (1996) 600.

[3] S. K. Baruah, J. Gehrke, and C. G. Plaxton, “Fast scheduling of periodic tasks on multiple resources,” Proceedings of the 9th International Symposium on Parallel Processing, April 1995

[4] J. Anderson and A. Srinivasan, “Mixed pfair/erfair scheduling of asynchronous periodic tasks,” in Proceedings of the 13th Euromicro Conference on Real-Time Systems, 2001, pp. 76–85.

[5] D. Zhu, D. Moss´e, and R. Melhem, “Multiple-resource periodic scheduling problem: how much fairness is necessary?”, Proceedings of the 24th IEEE Real-Time Systems Symposium, 2003, pp. 142–151.

[6] H. Cho, B. Ravindran, and E. D. Jensen, “An optimal real-time scheduling algorithm for multiprocessors,” Proceedings of the 27th IEEE International Real-Time Systems Symposium, 2006.

[7] H. Cho, B. Ravindran, E. D. Jensen, “T-N Plane-Based Real-Time Scheduling for Homogeneous Multiprocessors”, Journal of Parallel and Distributed Computing, 2010, To appear

[8] T. Wu and S. Jin. Weakly hard real-time scheduling algorithm for multimedia embedded system on multiprocessor platform. In IEEE International Conference on Ubi-Media Computing, 2008.