다중처리기에서 주기적인 실시간 태스크 스케줄링

조성제°
단국대 정보컴퓨터학부
sjcho@dankook.ac.kr

Scheduling Periodic Real-Time Tasks on Multiprocessors

Seongje Cho
Division of Information and Computer Science, Dankook University

요 약

다중처리기 상에서 실시간 태스크 스케줄링에 대한 연구가 현재 많이 수행되고 있지만, 주로 Pfair (P-Fairness)와 EDF(Earliest Deadline First) 알고리즘에 대한 연구이다. Pfair는 이론적인 연구에 초점을 두고 있으며, EDF는 처리기들을 효율적으로 이용하지 못하는 문제점을 가지고 있다. 본 논문에서는 다중처리기 상의 주기적인 실시간 태스크 집합을 대상으로, LLA(Least Laxity Algorithm) 알고리즘이 높은 스케줄링 가능성(schedulability)을 가짐을 먼저 보인다. 다음으로 기존 알고리즘들의 문제점을 보완하기 위해 EDF와 LLA를 결합한 ED/LL(Earliest Deadline/Least Laxity)이라는 알고리즘을 제시한다. ED/LL은 LLA보다 문맥교환 횟수는 적고 EDF보다 스케줄링 가능성이 높으며, 구현 오버헤드도 크지 않다.

1. 서론

하드웨어 기술이 발전하고 실시간 태스크들의 작업부하가 커짐에 따라, 다중처리기 구조가 실시간 시스템에 점차 일반화되고 있다. 현재 다중처리기 시스템에서 실시간 스케줄링에 대한 연구는 많이 이루어지고 있지만[1-6], 실용적인 온라인 스케줄링 알고리즘에 대한 연구가 거의 없다. 다중처리기에서 EDF(Earliest Deadline First) 알고리즘은 낮은 처리기 이용률(utilization)로 인해 데드라인(deadline) 만족시키지 못하는 비율이 높다[1]. LLA(*Least Laxity Algorithm*)는 좋은 스케줄링 조건을 보이나 문맥교환 오버헤드가 높다[2, 7]. 데드라인과 여유시간(laxity)을 결합한 EDZL(Earliest Deadline until Zero Laxity) 알고리즘은 좋은 스케줄링 조건과 낮은 문맥교환 오버헤드를 가지며 실시간 태스크 스케줄링에 효율적이다[7].

본 논문에서는 먼저 LLA가 어떤 주기 태스크 집합에 대해서 효율적인 알고리즘임을 보이고, 다음으로 EDF와 LLA의 문제를 일부 해결한 실용적인 ED/LL(Earliest Deadline/Least Laxity) 알고리즘을 제안한다. ED/LL은 데드라인과 여유시간에 기반한 알고리즘으로 모든 작업(job)의 여유시간이 양수일 경우에는 EDF를 사용하고, 그렇지 않을 경우에는 LLA를 사용한다. ED/LL은 모든 주기 태스크가 동일한 도착시간(arrival time, release time), 수행시간(execution time), 데드라인을 가지는 단순한 실시간 시스템에서 최적(*optimal*)이다.

본 논문의 구성은 다음과 같다. 2장에서는 태스크 모델과 온라인 스케줄링 알고리즘들의 특성을 기술한다. 3장에서는 LLA가 다중처리기에서 상당한 성능을 나타냄을 보이며, 4장에서는 ED/LL 알고리즘과 특성을 제시한다. 5장에서 결론을 내리고 향후 연구과제를 설명한다.

2. 관련 연구

2.1 시스템 모델

대상 시스템은 m개 처리기로 구성되며 n개의 주기 태스크들을 가진다. 각 주기 태스크 τ_i는 3개의 파라미터 (R_i, C_i, D_i)로 표현되는데, 이는 각각 태스크의 초기 시작시간(initial start time), 최악의 수행시간, 상대적 데드라인을 의미하는데, 모든 태스크의 초기 시작시간이 0일 경우에는 τ_i는 간단히 (C_i, D_i)로 표현된다. 시간은 정수로 표현되며, 상대적 데드라인은 주기 P_i와 동일하다. 즉, $C_i \le D_i = P_i$ 이다. 태스크는 일련의 작업(job)들을 생성하며, 작업은 한 순간에 요청되는 수행 개체(*instance*)를 말한다. 태스크 τ_i의 한 작업 J_i는 (r_i, c_i, d_i)로 표시되며, 이는 각각 그 작업의 도착시간, 남아있는 수행시간(remaining execution time), 데드라인(remaining time to deadline)을 의미한다. 작업이 처음 도착했을 때 $c_i = C_i$, $d_i = D_i$이며, c_i와 d_i는 시간에 따라 변한다. 각 태스크 τ_i의 처리기 이용률은 $u_i = C_i/D_i$로 주어지며, 작업 J_i의 '현재부하'(remaining load)는 $s_i = c_i/d_i$로 주어진다. 태스크 집합 τ의 전체 처리기 이용률은 $U = \sum_{i=1}^{n} u_i$ 인 상수로 주어지며, 현재 시스템부하는 $S(t) = \sum_{i=1}^{n} s_i$로 주어진다. s_i와 $S(t)$는 시간에 따라 변한다. 작업 J_i의 실행 비율 e_i는 $(C_i - c_i)/(D_i - d_i)$로 주어지며, 도착 후 그 작업이 사용한 처리기 시간의 비율을 나타낸다.

태스크 τ_i의 여유시간은 $L_i = D_i - C_i$로, 작업 J_i의 여유시간은 $l_i = d_i - c_i$로 정의되며, 여유시간은 태스크/작업의 긴급한 정도를 나타낸다. 여유시간이 음수이면 작업의 데드라인을 만족할 수 없음을 의미한다. 여유시간이 0이면 그 작업은 즉시 수행되어야 함을 의미하는데, 즉시 수행되지 않으면 데드라인을 만족할 수 없게 된다. 작업의 여유시간이 0이면 현재부하는 1이다.

준비 태스크(ready task)란 이전에 도착하였는데 아직 완료되지 않은 작업을 가진 태스크로 정의되며, 현재 태스크(*current task*)란 처리기에서 수행 중인 준비 태스크로 정의된다. 태스크 집합 τ의 모든 태스크가 데드라인을 만족할 수 있

다면 τ는 '실행가능'(feasible)하다라고 한다. 모든 실행가능한 태스크 집합이 스케줄링 알고리즘 Q 상에서 '스케줄가능하면' (schedulable), Q는 최적(optimal)이다. 온라인 스케줄링 시스템에서 주요 가정은 태스크의 도착 시간이 미리 알려지지 않는다는 것이다. 즉, 본 논문에서 스케줄링 판단은 미래에 도착될 태스크들이 아닌, 준비 태스크들에 근거하여 수행된다.

2.2 알고리즘들의 특성

처리기 수가 m일 때 EDF는 데드라인이 짧은 m개의 태스크를 먼저 실행하며, LLA는 여유시간이 짧은 m개의 태스크를 먼저 수행한다. Dertouzos 등은 다중처리기에서 EDF와 LLA의 성능을 분석하였다[2]. 본 논문은 Dertouzos 분석에 기반하여 알고리즘 특성을 연구하였다. 특별한 경우를 제외한 대부분의 경우에 LLA의 문맥교환 횟수가 EDF에서 보다 많다. 문맥교환 횟수 비교는 정리 1과 보조정리 1에 나타나 있다.

[정리 1] '실행가능한'(feasible) 주기적인 태스크 집합 τ이 있을 때, 전체 n개 태스크의 도착시간(R), 실행시간(C), 데드라인(D)이 동일하다고 하자. τ가 한 주기 동안 EDF에 의해 스케줄링 되면 n개의 문맥교환이 발생하고, LLA에 의해 스케줄링 되면 n×C-(C-1)번의 문맥교환이 발생한다.

증명: 참고문헌 [7] 참조. ■

[보조정리 1] n개의 태스크 $\{\tau_1, \tau_2, ..., \tau_n\}$로 구성된 집합이 있고, R_i, D_i가 각각 τ_i의 도착시간, 데드라인이고, 모든 태스크의 수행시간이 C로 동일하다고 하자. 또 $R_i=i$, $D_i=D-2i$ 라고 가정하자. 여기서 $i=1, 2,..., n$ 이고 $C>1$, $D \geq 2n+C$이다. 이때 EDF의 문맥교환 횟수는 (2n-1)로 LLA에서와 같다.

증명: EDF에서는 각 i 시간에 새로 도착하는 태스크의 데드라인이 짧기 때문에, LLA에서는 새로 도착하는 태스크의 여유시간이 짧기 때문에 총 (n-1)번의 선점(preemption)이 발생한다. 그리고, 각 태스크가 종료될 때마다 문맥교환이 발생하므로 총 n+(n-1)=(2n-1)번의 문맥교환이 있다. 문맥교환 횟수는 선점 횟수를 포함한다. ■

Baruah 등은 다중처리기 스케줄링에서 좋은 결과를 제시하였다[3-5]. Pfair[3]에서는 각 태스크 τ_i가 가중치 $w_i=C_i/P_i$에 비례하여 스케줄 되며 $\sum_{i=1}^{m} w_i \leq m$가 스케줄 가능성에 대한 필요조건이다. 즉, 태스크 τ_i가 [0, t) 시간간격에서 $\lceil w_i \cdot t \rceil$ 또는 $\lfloor w_i \cdot t \rfloor$ 시간만큼 실행되면 최적이다. EDF-US[m/(2m-1)] 알고리즘[4]에서는 m/(2m-1)보다 높은 이용률을 가진 태스크가 최고의 우선순위를 가지며, 나머지 태스크는 EDF에 따른 우선순위를 가진다. Srinivasan 등은 전체 이용률이 $U \leq m2/(2m-1)$을 만족시키기만 하면 EDF-US[m/(2m-1)]가 모든 작업을 스케줄할 수 있음을 보였다[4]. 또한 m개 처리기 시스템에서 $U > (m+1)/2$보다 클 경우에 어떠한 우선순위기반 스케줄러도 주기 태스크 집합의 스케줄 가능성을 보장할 수 없음을 보였다.

[정리 2] 전체 n개의 태스크로 된 집합 τ가 있고, C_i를 태스크 τ_i의 실행시간, $C=\sum_{i=1}^{n} C_i$를 τ의 전체 실행시간이라고 할 때, 이 집합을 스케줄링하기 위해 단일 처리기에서는 최소 n×C 단위시간이 필요하고, m개 처리기에서는 최소 (n×C)/m 단위시간이 필요하다.

증명: 태스크 집합 가 n×C 실행시간을 요구하는데, 처리기가 하나일 경우에는 n×C 시간이 소요되며, 처리기가 m개일 경우에는 (n×C)을 m으로 균등 배분하여 수행하면 된다. 이는 선점 등의 문맥교환 비용을 무시할 경우에 성립된다. ■

정리 2에 의하면, 문맥교환 비용과 하드웨어적인 연결 비용을 무시할 경우에 수행 시간 면에서 m개 다중처리기는 단일 처리기에 비해 최대 m배의 성능을 보인다.

3. LLA 알고리즘

그림 1은 4개의 주기 태스크 $\tau_1 \ldots \tau_4$를 가진 3개 처리기 시스템에서 LLA에 의한 스케줄을 보여준다. 4개 태스크의 파라미터는 각각 (0, 2, 3), (0, 2, 3), (0, 2, 3), (0, 7, 15)이며, 초기 시작시간이 모두 0이므로 이 경우는 {(2, 3), (2, 3), (2, 3), (7, 15)}로 표현된다. 그림에서 주기는 각각 3, 3, 3, 15이고, 초기 여유시간은 각각 1, 1, 1, 8이며 U=2.47이다. 이 태스크 집합은 EDF에서는 스케줄 가능하지 않다.

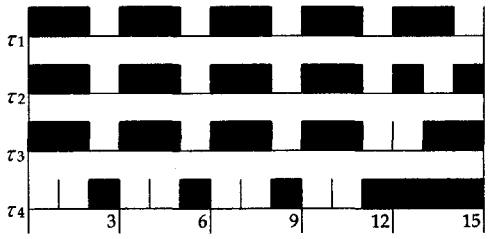

그림 1. 3개 처리기에서 LLA 스케줄

$t<11$ 일 때는 τ_4의 여유시간이 최대이기 때문에, 다른 태스크가 준비 작업을 가지고 있을 때 항상 τ_4는 대기해야 한다. τ_4 작업의 여유시간이 0이 되기 이전에 τ_4는 1/3 보다 높은 비율로 실행될 수 없다. t=11에 의 여유시간은 0이며 가장 높은 우선순위를 가진다. $12 \leq t \leq 15$ 사이에 시스템의 부하는 3이 되며, 이는 LLA가 실행할 수 있는 최대 스케줄링 바운드(*schedulability bound*)이기도 하다. 그림 1에서 선점은 t=3, 6, 9, 13에서 총 4번 발생한다.

3개 처리기 시스템에서 모든 태스크의 초기 시작시간이 0인 태스크 집합 {(2, 3), (2, 3), (2, 3), (5, 15)}은 U=2.33으로 EDF에서 스케줄될 수 있는데, $0 \leq t \leq 15$ 사이에서 선점이 t=3, 6, 9에서 총 3번 발생한다. 모든 태스크의 초기 시작시간이 0인 2개 처리기 시스템에서 태스크 집합 {(2, 3), (2, 3), (2, 3)}이나 {(2, 4), (2, 4), (5, 6)}는 EDF에서는 스케줄 될 수 없고 LLA에서는 스케줄 될 수 있다. LLA에 의해 스케줄 될 경우, $0 \leq t \leq 12$ 사이에 선점횟수는 각각 3회(주기마다 1회), 2회이다. 이처럼, 많은 경우에 LLA는 좋은 스케줄링 성능을 보이며 선점 오버헤드도 크지 않다. 작업 완료로 인해 발생하는 문맥교환 오버헤드는 모든 알고리즘에서 같다.

4. ED/LL 알고리즘

2개 처리기에서 3개의 주기 태스크로 구성된 집합 {τ_1, τ_2, τ_3}, 즉 {(5, 8), (5, 8), (5, 8)}이 LLA에 의해 스케줄 될 경우, 매 주기마다 4번의 선점이 발생한다. 이 경우에는 LLA가 좋은 성능을 보이지 않는다. 물론 EDF에 의해서는 스케줄 될 수 없다. 본 절에서는 데드라인과 여유시간을 동시에 고려한 ED/LL 알고리즘을 제안한다. ED/LL은 먼저 모든 작업들의 여유시간을 조사하여, 모든 여유시간 값이 0보다 클 경우에는 데드라인이 짧은 작업 순으로 스케줄하고, 여유시간이 0인 작업이 하나라도 생겨나면 여유시간이 짧은 작업 순으로 스케줄 한다. ED/LL 알고리즘이 그림 2에 나타나 있다. {(5, 8), (5, 8), (5, 8)}이 ED/LL에 의해 스케줄 될 경우 매 주기마다 2번의 선점이 발생하는데, 이는 LLA의 0.5배이다. ED/LL에 의한 스케줄이 그림 3에 나타나 있으며, t=3, 4에서 선점이 발생한다. 스케줄 시, 모든 작업의 여유시간이 양수인 비율이 높을수록 ED/LL가 LLA보다 낮은 문맥교환 오버헤드를 보인다.

```
Algorithm ED/LL

1  while TRUE do
2      if there is a job with negative laxity then
3          call overload handler
4      else if all jobs have positive laxity do
5          perform EDF
6      else
7          perform LLA
8      endif
9  endwhile
```

그림 2. ED/LL 알고리즘

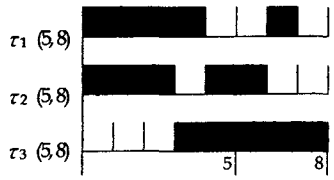

그림 3. 2개 처리기에서 ED/LL 스케줄

[정리 3] ED/LL에서의 문맥교환 수는 LLA보다 적거나 같다.

증명: 태스크 집합 τ의 작업들이 수행될 때 각 작업 J_i의 여유시간 $l_i \geq 0$이므로, 다음 두 가지 경우로 나누어서 증명할 수 있다. 첫 번째는 모든 작업의 $l_i > 0$인 경우로, 이 경우에 ED/LL은 EDF와 같이 동작하므로 정리 1과 보조정리 1에 의해 문맥교환 수가 LLA보다 적거나 같다. 두 번째는 $l_i = 0$인 작업이 하나라도 있는 경우로, 이 경우는 ED/LLL의 문맥교환 수가 LLA와 동일하다. 즉, ED/LL에서의 문맥교환 수는 LLA보다 적거나 같다. ■

[정리 4] n개의 주기 태스크로 구성된 실행 가능한 태스크 집합 τ에서 모든 태스크의 도착시간(R), 수행시간(C), 데드라인(D)이 동일할 때, m개 처리기 시스템에서 전체 처리기 이용률 $U \leq m$이면 ED/LL 하에서 τ는 스케줄 가능하다.

증명: $U = \sum_{i=1}^{n} C/D = n(C/D) = m$인 경우, 즉 nC=mD인 경우를 고려하면 된다. 수행시간이 C인 작업 n개를 한 주기 안에서 수행하기 위해서는 nC만큼의 수행시간이 요구된다. 또한 m개 처리기가 D 단위시간 동안에 가지는 전체 실행시간은 mD이며, 또한 초기 우선순위는 모두 동일하다. 따라서 nC=mD가 참인 경우에 τ는 ED/LL 하에서 스케줄 가능하다. ■

정리 3과 정리 4에서 알 수 있듯이 ED/LL은 LLA보다 문맥교환 오버헤드는 적고, 태스크들의 도착시간, 수행시간, 데드라인이 같을 때는 최적이다. ED/LL을 구현하는 방법은 각 작업을 데드라인에 기반한 Dl_Queue와 여유시간에 기반한 Lx_Queue를 동시에 유지시키면서, 모든 여유시간 값이 양수일 경우에는 Dl_Queue의 앞에 있는 작업 순으로 스케줄하고 그렇지 않을 경우에는 Lx_Queue의 앞에 있는 작업 순으로 스케줄한다. 시간이 지남에 따라 우선순위가 바뀌거나 새로운 작업이 도착하면 큐를 갱신시켜 주고, 작업이 끝나면 두 큐에서 동시에 제거시키면 된다.

5. 결론 및 향후 연구

다중처리기에서 LLA가 EDF보다 스케줄 가능성 면에서 더 효율적이며, 문맥교환 오버헤드도 크지 않은 경우가 많다. 제안한 ED/LL 알고리즘은 데드라인과 여유시간을 합성한 알고리즘으로 EDF보다 좋은 스케줄 가능성을 보이며 LLA보다 문맥교환 오버헤드가 크지 않다. m개 처리기에서 태스크 집합의 처리기 이용률이 (m+1)/2보다 높은 경우에는 LLA과 ED/LL이 EDF보다 더 좋은 성능을 보인다. 또한 ED/LL은 구현이 쉬운 실용적인 알고리즘이다. 향후, 다중처리기에서 ED/LL의 스케줄링 바운드를 계산하는 연구가 지속되어야 한다.

[참고문헌]

[1] D. Locke, "Best-Effort Decision Making for Real-Time Scheduling," Doctoral Dissertation, Computer Science Dept., CMU, 1986.

[2] M.L. Dertouzos and A.K. Mok, "Multiprocessor On-line Scheduling of Hard Real-Time Tasks," IEEE Trans. on Software Eng., Vol. 15, No. 12, pp. 1497-1506, 1989.

[3] 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.

[4] A. Srinivasan and S. Baruah, "Deadline-based Scheduling of Periodic Task Systems on Multiprocessors," Information Processing Letters (Accepted for publication).

[5] J. Goossens, S. Funk, and S. Baruah, "Priority-driven scheduling of periodic task systems on multiprocessors," *Real-time Systems (Accepted for publication)*.

[6] B. Anderson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," Proc. of the IEEE Real-Time Systems Symposium, pp 193-202, Dec. 2001.

[7] 조성제, 이석균, 유해영, "산발적인 경성 실시간 태스크를 위한 온라인 스케줄링 알고리즘," 정보과학회논문지(A), 제25권 제7호, pp. 708-718, 1998.