잉여 여유시간을 이용한 연성 비주기 태스크들의 효율적인 스케줄링

(Efficient Scheduling of Soft Aperiodic Tasks Using Surplus Slack Time)

김 희 헌 † 박 학 봉 † 박 문 주 ††
(Heeheon Kim) (Xuefeng Piao) (Moonju Park)
박 민 규 ††† 조 유 근 †††† 조 성 제 †††††
(Minkyu Park) (Yookun Cho) (Seongje Cho)

요 약 마감시간이 있는 주기 태스크와 마감시간이 없는 비주기 태스크가 공존하는 경성 실시간 시스템에서는 주기 태스크의 마감시간과 비주기 태스크의 빠른 응답시간을 보장하는 것이 중요하다. 본 논문에서는 비주기 태스크 처리에 효율적이면서 잘 알려져 있는 알고리즘인 *Total Bandwidth Server*(TBS) 보다 향상된 알고리즘인 *Enhanced TBS*(ETBS)를 제시한다. ETBS는 Earliest Deadline First(EDF) 스케줄링 알고리즘을 사용하는 단일처리기 시스템에서 주기 작업의 단위 수행시간마다 확보할 수 있는 잉여 여유시간을 이용해 온라인으로 비주기 태스크에 마감시간을 부여하는 알고리즘이다. 제시한 알고리즘은 주기 및 비주기 태스크들이 처리기의 이용률을 모두 이용할 수 있게 하며 주어진 주기 태스크들의 마감시간을 보장한다. ETBS 알고리즘은 TBS와 같은 계산 복잡도 $O(1)$을 가지면서도 TBS보다 좋은 응답시간을 가짐을 이론적으로 보였고, 정량적인 응답시간 차이는 모의실험을 통해 보였다.

키워드 : 실시간 스케줄링 알고리즘, 비주기 태스크, 응답시간, 마감시간

Abstract In a real-time system with both hard real-time periodic tasks and soft real-time aperiodic tasks, it is important to guarantee the deadlines of each periodic task as well as obtain fast response time for each aperiodic task. This paper proposes Enhanced Total Bandwidth Server (ETBS) with possibly shorter response time than Total Bandwidth Server (TBS), which is efficient and widely used for servicing aperiodic tasks. For uniprocessor system using Earliest Deadline First (EDF) scheduling algorithm, ETBS assigns an on-line deadline to each aperiodic task considering a surplus slack time which gained for every unit execution time of periodic job. The proposed method can fully utilize the processor while meeting all the deadlines of periodic tasks. We show that the proposed ETBS provides better response time of aperiodic tasks than TBS theoretically, but has the same computational complexity as TBS, $O(1)$. Simulation results show that the response time of aperiodic tasks with ETBS are shorter than one with TBS.

Key words : Real-time scheduling algorithm, Aperiodic task, Response time, Deadline

† 학생회원 : 서울대학교 전기컴퓨터공학부
hhkim@os.snu.ac.kr
hbpark@ssrnet.snu.ac.kr
†† 정 회 원 : 인천대학교 컴퓨터공학과 전임강사
mpark@incheon.ac.kr
††† 정 회 원 : 건국대학교 컴퓨터응용과학부 교수
minkyup@kku.ac.kr
†††† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
ykcho@snu.ac.kr
††††† 정 회 원 : 단국대학교 컴퓨터학부 교수
sjcho@dku.edu

논문접수 : 2008년 3월 27일
심사완료 : 2008년 11월 12일

Copyright©2009 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.
정보과학회논문지: 시스템 및 이론 제36권 제1호(2009.2)

1. 서 론

복잡한 실시간 시스템은 마감시간(deadline)이라 불리는 시간 제약을 가지는 태스크들로 구성된다. 태스크가 마감시간을 지키지 못했을 경우 인명 피해나 경제적 손실 등의 재앙이 발생하는 경우 태스크가 경성(hard) 마감시간을 가진다고 말한다. 마감시간을 지키는 것이 요구되지만 지키지 못했을 경우 커다란 피해가 발생하지 않고 계산 결과의 품질이 저하되는 경우 태스크가 연성(soft) 마감시간을 가진다고 말한다.

주기적으로 활성화 되어 작업을 수행하는 태스크를 주기(periodic) 태스크라고 하며, 산발적으로 활성화되는 태스크를 비주기(aperiodic) 태스크라고 한다. 일반적으로 주기 태스크들은 경성 마감시간을 가지며, 비주기 태스크들은 비실시간 태스크이거나 혹은 시간 제약을 가지더라도 연성 마감시간을 가진다. 이러한 주기 태스크와 마감시간이 없는 비주기 태스크가 시스템에 존재할 경우, 실시간 스케줄링의 목표는 주기 태스크들이 마감시간을 충족시키는 것을 보장하면서, 비주기 태스크들이 빠른 응답시간을 보이도록 하는 것이다.

주기 태스크와 비주기 태스크의 스케줄링 알고리즘은 주기 태스크를 스케줄 하는 알고리즘을 기준으로 고정 우선순위 서버(fixed-priority server)와 동적 우선순위 서버(dynamic-priority server) 두 그룹으로 나눌 수 있다. 주기 태스크를 스케줄할 때, 일반적으로 고정 우선순위 서버는 Rate Monotonic(RM)[1] 알고리즘을 사용하고 동적 우선순위 서버는 Earliest Deadline First (EDF)[1] 알고리즘을 사용한다.

고정 우선순위 서버 알고리즘으로는 Deferrable Server(DS)[2], Priority Exchange server(PE)[3], Sporadic server[4], 및 Slack Stealer[5] 등이 있다. DS 알고리즘은 비주기 태스크를 서비스하기 위한 주기 서버를 생성하고 주기마다 활성화 되어 비주기 태스크 요청이 있으면 주기 서버의 수행시간 이내에서 비주기 태스크를 실행한다. 비주기 태스크가 없을 때, PE 알고리즘은 비주기 태스크를 위한 서버의 수행 용량을 현재 수행하는 (현재 가장 높은 우선순위) 주기 태스크의 우선순위로 보존한다는 점에서 DS 알고리즘과 차이가 있다. PE 알고리즘과 DS 알고리즘은 서버의 주기가 시작되는 시점에서 서버의 수행 용량이 재충전되는데, Sporadic Server 알고리즘은 비주기 태스크에 의해 수행 용량이 사용되면 사용될 때부터 주기시간 후에 사용량만큼 충전된다. Slack Stealer 알고리즘은 주기 서버를 생성하지 않고 주기 태스크의 여유시간 (slack time)을 빼어내어 수행한다.

동적 우선순위 서버 알고리즘으로는 Dynamic Priority Exchange Server(DPE)[6,7], Dynamic Sporadic Server(DSS)[6,7], Total Bandwidth Server(TBS)[6,7], Constant Bandwidth Server(CBS)[8] 등이 있다. DPE와 DSS 알고리즘은 각각 PE 서버와 SS 서버를 동적 우선순위 시스템에서 사용할 수 있게 확장된 알고리즘이다. TBS 알고리즘은 각 비주기 태스크에 마감시간을 부여하고 EDF 알고리즘으로 스케줄 한다. TBS 알고리즘은 다른 비주기 태스크 처리 알고리즘에 비해 계산 복잡도가 $O(1)$로 매우 낮고, 비주기 태스크들의 응답시간도 빠르다. 최근에는 다중 처리기에서 비주기 태스크를 처리하기 위한 방법으로도 연구되었다[9]. CBS 알고리즘은 일정한 처리기 이용률을 비주기 태스크들에게 보장함으로써 비주기 태스크들이 초과 실행하는 경우를 방지한다. CBS는 다중 처리기에도 적용할 수 있지만 [10], 단일 처리기 및 다중 처리기에서 비주기 태스크에 대한 최적 응답시간은 확률적인 값으로 구하게 된다[11].

본 논문은 처리기 이용률을 최대로 활용하면서 경성 실시간 태스크들의 마감시간을 보장하고 비주기 태스크들의 빠른 응답시간을 제공하는 *Enhanced TBS* (ETBS) 알고리즘을 제안한다. ETBS는 비주기 태스크에 마감시간을 부여해 비주기 태스크를 주기 작업들과 같이 EDF 알고리즘으로 스케줄을 한다. 제안한 알고리즘에 의해 비주기 태스크에 부여되는 마감시간은 주어진 주기 태스크들의 스케줄 가능성을 해치지 않으며, 비주기 태스크의 응답시간 면에서 ETBS가 TBS보다 향상된 성능을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 시스템 모델에 대해 설명한다. 3장에서는 ETBS 알고리즘과 스케줄 가능성에 대해 기술한다. 4장에서는 성능평가를 수행하며 5장에서 결론을 맺는다.

2. 시스템 모델

본 논문은 단일 처리기에서 마감시간이 있는 주기 태스크와 마감시간이 없는 비주기 태스크 집합을 대상으로 하는 선점 가능(preemptive) 경성 실시간 스케줄링을 다룬다. 또한 태스크는 마감시간이 주기와 일치하고, 각각 독립적인 태스크로 구성된 태스크 집합을 대상으로 한다.

태스크 집합 $\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$은 n개의 주기 태스크로 구성되며, 각 태스크 τ_i는 (R_i,C_i,P_i)로 표현되는데, 이는 각각 태스크의 도착시간(release time), 최악 수행시간(worst-case execution time), 주기(period)를 의미한다. 모든 태스크의 도착시간이 동일할 때 동기적(synchronous) 태스크 집합이라고 하고, 도착시간이 동일하지 않은 경우 비동기적(asynchronous) 태스크 집합이라고 한다. 태스크 집합이 동기적이고 주기와 마감시간이

같은 경우 τ_i는 (C_i,P_i)로 표현한다. 각 태스크는 P_i의 정수 배 시간, $R_i+(k-1)P_i$, $(k=1,2,...)$ 마다 하나의 작업(job)을 생성하며, 이 작업은 마감시간, $R_i+k\cdot P_i$ 까지 완료되어야 한다. 주기 태스크 집합 τ의 전체 이용률은 U_p로 나타내고, $U_p=\sum_{i=1}^{n}\frac{C_i}{P_i}$이다.

본 논문에서 k번째 비주기 태스크는 J_k 로 나타낸다. 비주기 태스크 J_k의 도착시간 a_k와 마감시간 d_k는 알 수 없고, 최악 수행시간 e_k는 비주기 태스크가 도착하면 알 수 있다. 비주기 태스크들의 처리는 도착한 순서대로 처리하며, 이전에 도착한 작업이 끝나야 다음 비주기 태스크가 처리된다. 동시에 도착한 비주기 태스크들은 임의의 방식으로 순서가 부여된다.

태스크 집합의 어떤 스케줄이 각 태스크의 모든 작업의 마감시간을 충족할 때 그 스케줄을 유효(valid)하다고 한다. 스케줄링 알고리즘 Q가 주어진 태스크 집합 τ에 대해 유효한 스케줄을 생성해 낼 수 있으면, 태스크 집합 τ는 스케줄링 알고리즘 Q에 의해 스케줄 가능(schedulable)하다고 한다.

3. Enhanced TBS(ETBS)

ETBS는 어떤 구간 $[t',t]$에 주기 작업이 계속 있다면, 주기 작업들의 스케줄 가능성을 보장하기 위해 TBS처럼 해당 구간에서의 비주기 태스크들의 요구 수행시간이 항상 $(t-t')U_s$보다 크지 않도록 비주기 태스크에 마감시간을 부여한다. 이는 구간 $[t',t]$에서 주기 작업들이 적어도 $(t-t')U_p$ 시간을 수행하도록 보장한다. 그러나 구간 $[t',t]$에서 주기 작업들을 최대한 수행시키는 경우에도, 특정 구간에 주기 작업이 존재하지 않아 그 총 수행시간이 보장된 시간인 $(t-t')U_p$ 보다 작은 경우가 있을 수 있다. ETBS에서는 이런 경우를 고려하여 주기 작업이 존재하지 않는 구간이 발생하는 경우 비주기 태스크들의 요구 수행시간을 TBS 보다 크게 부여 할 수 있게 하여 응답시간을 단축시킬 수 있다.

3.1 알고리즘

주기 태스크 집합 $\tau=\{\tau_1=(C_1,P_1),\tau_2=(C_2,P_2),\cdots,\tau_n=(C_n,P_n)\}$의 이용률 U_p를 $0<U_p<1$이라고 하자. 그리고 $U_s=1-U_p$, $\rho=U_s/U_p$라 하고, 또 다른 주기 태스크 집합 τ'를 $\tau'=\{\tau_1'=(C_1\cdot\rho,\ P_1),\tau_2'=(C_2\cdot\rho,\ P_2),\cdots,\tau_n'=(C_n\cdot\rho,\ P_n)\}$라 하자. 태스크 집합 $\tau\cup\tau'$의 처리기 이용률은 다음과 같다.

$$\sum_{\tau_i\in\tau}\frac{C_i+C_i\cdot\rho}{P_i}=U_p+U_s.$$

$U_p+U_s=1$이므로 태스크 집합 $\tau\cup\tau'$는 EDF 알고리즘으로 스케줄 가능하다. 이때, τ의 한 작업을 먼저 단위시간 스케줄하고 그 작업과 마감시간이 같은 τ'의 한 작업을 ρ시간 스케줄 하는 방법으로 유효 스케줄 S를 만들 수 있다. 이를 응용하여 주기 태스크 집합 τ와 비주기 태스크 집합을 스케줄 할 수 있다. 즉, 비주기 태스크가 도착했을 때부터 S에서 τ와 τ'의 작업이 스케줄되는 것처럼 τ의 주기 작업을 단위시간 스케줄하고 수행 가능한 비주기 태스크가 있을 때 ρ시간 스케줄 할 수 있다. 본 논문에서 제시되는 ETBS 알고리즘은 이를 응용한다. 즉 주기 작업을 단위시간 스케줄하고 비주기 태스크를 ρ시간 스케줄하는 방법으로 스케줄 했을 때, 해당 비주기 태스크가 수행을 마칠 수 있는 시간으로 마감시간을 부여하게 된다.

ETBS에서의 비주기 태스크를 위한 이용률 U_s는 $1-U_p$로 정의되고, 여유 시간율 ρ는 U_s/U_p 로 정의된다. 여유 시간율은 주기 작업의 단위시간 수행 당 앞으로 수행될 주기 작업의 마감시간을 해치지 않으면서 활용될 수 있는 여유시간 비율이다. 즉, 주기 작업이 구간 $[t',t]$에서 수행했다면 $(t-t')\rho$만큼의 여유시간이 생기게 된다. 본 논문에서는 이렇게 계산된 여유시간을 잉여 여유시간 (surplus slack time) 이라 한다. ETBS는 잉여 여유시간에 비주기 태스크를 수행시켰을 경우 이 태스크의 수행을 완료할 수 있는 시간으로 마감시간을 부여한다. 예를 들어, 첫 번째 비주기 태스크 J_1가 e_1의 수행시간을 갖고 시간 t에 도착했다고 하자. 주기 작업이 시간 t부터 e_1/ρ 만큼 수행되었다면, 시간 $t+e_1/\rho$에서의 잉여 여유시간은 e_1이 된다. 따라서 J_1은 시간 $t+e_1/\rho$에 스케줄 될 수 있고 $t+e_1/\rho+e_1$에 수행을 마칠 수 있다. 그러나 이렇게 주기 작업의 수행을 기다려 비주기 태스크가 수행한다면 응답시간 측면에서 효율적이지 못하다. 어떤 유효한 스케줄에서 J_k가 시간 f_k에 수행을 마친다고 할 때, 다른 작업들의 마감시간은 바꾸지 않고 J_k의 마감시간을 f_k로 하여 EDF 방식으로 다시 스케줄 하면, EDF의 최적성(optimality)[12]에 의해 유효한 스케줄이 된다. 따라서 비주기 태스크 J_1이 시간 $t+e_1/\rho+e_1$에 작업을 마치는 유효한 EDF 스케줄이 존재하므로 J_1에 대해 $t+e_1/\rho+e_1$로 마감시간을 부여할 수 있다.

마감시간이 부여된 비주기 태스크는 잉여 여유시간이 충분해지기도 전에 미리 수행을 해서 자신의 마감시간보다 앞서서 최악 수행시간을 마칠 수 있다. 앞선 예에서 J_1이 $t+e_1/\rho+e_1$ 전에 수행을 마치고 J_2가 $t+e_1/\rho+e_1$ 전에 J_1과 같은 방법으로 마감시간을 부여 받는다면 앞으로 수행될 주기 작업들의 스케줄 가능성을 해칠 수

있다. 즉, 앞으로 생성될 잉여 여유시간을 미리 사용한 경우 이를 다음 비주기 태스크의 마감시간 계산에 반영해 주어야 주기 작업들의 스케줄 가능성을 해치지 않는다. 이를 위해 ETBS는 지연계수를 관리한다. 시간 t에서의 지연계수는 $R(t)$로 표현되고 $R(0)=0$이다. 지연계수는 그림 1의 알고리즘에 의해 계산된다.

지연계수는 비주기 태스크의 마감시간이 결정된 후부터 계산된다. 스케줄링 시점에만 지연계수를 계산하면 되나, t'가 t바로 이전의 스케줄링 시점이라면 $R(t')=0$이고 비주기 태스크가 없는 스케줄링 시점 t에서는 지연계수 계산이 필요 없다. 시간 t에 수행 가능한 비주기 태스크가 있거나, $R(t')<0$ 인 경우에 지연계수 계산 알고리즘을 수행 한다. 지연계수는 비주기 태스크가 수행할 경우 (ii)에서 그 수행시간 만큼 감소되고, 주기 작업이 수행할 경우 (iii)에서 단위 수행시간당 여유 시간을 ρ만큼 증가된다. 어떤 한 비주기 태스크의 마감시간이 결정된 후, 그 비주기 태스크의 마감시간이 현재 수행 가능한 주기 작업의 마감시간 보다 앞서 비주기 태스크가 먼저 수행한다면 지연계수는 음수 값을 가질 수 있다. 반면 비주기 태스크의 마감시간이 늦어 주기 작업이 먼저 수행할 경우에는 지연계수 값은 증가되고, 비주기 태스크가 수행하면 지연계수는 감소하게 된다. 지연계수가 음수인 상태에서 주기 작업이 계속 수행을 하면 양수가 될 수 있다. 지연계수 계산 알고리즘의 (i)과 (iv)의 설명은 뒤에서 하기로 한다.

ETBS에서 비주기 태스크 J_k의 마감시간은 아래와 같다. 이때, $U_s=1-U_p$ 이고, $\rho=\dfrac{U_s}{U_p}$이다.

$$d_k=r_k+\frac{e_k}{U_s}-\frac{R(r_k)}{\rho} \tag{1}$$

여기서 r_k는 J_k의 마감시간이 결정되는 시간이다. J_k의 마감시간은 J_k가 도착했을 때 선행 비주기 태스크 J_{k-1}가 있다면 J_{k-1}이 수행을 마쳤을 때 결정되고, J_k가 도착했을 때 선행 비주기 태스크가 없다면 도착 시간에 결정 된다(J_k의 도착시간이 a_k이므로 $a_k \le r_k$ 이다).

ETBS에서 마감시간을 부여받은 비주기 태스크는 수행 가능하다고 표현하며, 임의의 시간에 최대 하나의 수행 가능한 비주기 태스크만 존재할 수 있다. 즉 J_k는 J_{k-1}가 수행을 마친 이후에야 마감시간을 부여 받고 수행 가능하게 된다. 수행 가능한 비주기 태스크는 주기 태스크에서 생성된 주기 작업과 함께 EDF 방식으로 스케줄링 된다.

식 (1)은 다음에 의해 유도 된다. 지연계수가 음수인 시간 r_k일 때 J_k가 마감시간을 부여 받는다고 하자. J_k의 선행 비주기 태스크가 미리 사용한 잉여 여유시간을 채우기 위해 주기 작업은 $-R(r_k)/\rho$ 만큼 수행해야 하고, J_k의 수행시간 e_k의 시간을 처리하기 위해서는 주기 작업이 e_k/ρ 만큼 수행해야 한다. 따라서 J_k가 잉여 여유시간을 기다렸다 스케줄 된다면 $r_k+e_k/\rho-R(r_k)/\rho$에 스케줄이 가능하고 $r_k+e_k/\rho-R(r_k)/\rho+e_k$에 수행을 마칠 수 있다. $U_p+U_s=1$ 이므로 $r_k+\dfrac{e_k}{\rho}-\dfrac{R(r_k)}{\rho}+e_k = r_k+\dfrac{e_k(U_p+U_s)}{U_s}-\dfrac{R(r_k)}{\rho}=r_k+\dfrac{e_k}{U_s}-\dfrac{R(r_k)}{\rho}$이 된다. 앞에서도 언급하였듯이, 지연계수는 비주기 태스크 J_{k-1}이 부여된 마감시간보다 빨리 끝나게 되었을 때, 이를 고려해 앞으로 수행해야 하는 주기 작업들의 마감시간을 해치지 않게 비주기 태스크 J_k에 마감시간을 부여할 수 있게 한다.

이제 지연계수 계산 알고리즘의 (i)과 (iv)를 살펴보도록 하자. 주기 작업이 없는 경우에는 비주기 태스크가

```
/* t는 현재 스케줄링 시간, t'는 t 바로 이전의 스케줄링 시간 */

if (시간 t'에 수행 가능한 주기 작업이 없었고 R(t') ≤ 0)
        R(t) = 0                                                    (i)
else
        if (구간 [t',t]에서 비주기 태스크가 수행)
                R(t) = R(t') - (t - t')                             (ii)
        else if (구간 [t',t]에서 주기 작업이 수행)
                R(t) = R(t') + (t - t')ρ                            (iii)
                if (시간 t'에 수행 가능한 비주기 태스크가 없었고 R(t) > 0)
                        R(t) = 0                                    (iv)
                end if
        end if
end if
```

그림 1 지연계수 계산 알고리즘

수행되어도 (i)에 의해 지연계수는 0이 된다. 그 이유는 다음과 같다. 주기 작업이 수행하면 여유 시간율 ρ 만큼을 잉여 여유시간으로 계속 축적하고, 비주기 태스크가 수행하거나 유휴 시간인 경우에는 그 시간만큼 축적된 잉여 여유시간을 소비한다고 하자. 그런데 시간 t에 주기 작업이 없다고 하자. 구간 $[0,t]$에서 주기작업의 총 수행시간을 C_p라 하면 $C_p=\sum_{i=1}^{n}\left\lceil \frac{t}{P_i}\right\rceil C_i \geq \sum_{i=1}^{n}\frac{t}{P_i}C_i = t\cdot U_p$ 이므로, $C_p \geq t\cdot U_p$이다. 따라서 시간 t까지 주기 태스크의 수행을 제외한 시간은 최대 $t-t\cdot U_p$ 이다 ($U_p+U_s=1$ 이므로 $t-t\cdot U_p=t\cdot U_s$). 구간 $[0,t]$에서 주기 작업은 최소 $t\cdot U_p$ 만큼 수행을 했고, 이 수행시간에 대한 잉여 여유시간은 최소 $t\cdot U_p\cdot\rho=t\cdot U_s$이 된다. 즉 주기 태스크가 없는 시간에는 잉여 여유시간이 음수가 될 수 없다. 따라서 현재 주기 작업이 없는 경우 지연계수를 0으로 한다. ETBS에서 주기 작업이 없을 때 비주기 태스크가 수행한다면 이는 주기 태스크가 없을 때 유휴 시간을 이용하여 비주기 태스크를 스케줄 하는 후면(background) 스케줄처럼 된다. 결과적으로 어떤 비주기 태스크가 이러한 경우와 같이 스케줄 되면 다음 비주기 태스크의 마감시간이 빨라질 수 있다.

비주기 태스크가 없는데도 잉여 여유시간을 계속 축적해 이를 사용하면 주기 작업의 마감시간을 해칠 수 있다. 이는 각 주기 작업의 수행으로 축적되는 잉여 여유시간은 해당 작업의 우선순위로 사용되어야 하기 때문이다. 본 논문에서는 지연계수 계산 알고리즘이 복잡해지지 않게, 비주기 태스크가 없음에도 주기 작업의 수행으로 지연계수가 0보다 커지게 되면 (iv)에서 지연계수를 0으로 하고, 이후 새로운 비주기 태스크가 도착할 때까지 지연계수를 계산하지 않는다.

스케줄링 시점 t에서 ETBS 알고리즘은 다음과 같은 순서로 현재 수행할 작업을 선택한다.

1. 수행 가능한 (마감시간이 결정된) 비주기 태스크가 있거나, $R(t')<0$인 경우에 지연계수 계산 알고리즘을 수행한다. 이때 t'는 t바로 이전의 스케줄링 시점이다.
2. 마감시간이 결정된 비주기 태스크가 없고, 도착한 비주기 태스크가 있다면 식 (1)에 의해 이 비주기 태스크에 마감시간을 부여한다.
3. 수행 가능한 주기 및 비주기 태스크들 중 마감시간이 가장 빠른 작업을 선택해 수행시킨다. 만일 비주기 태스크와 주기 작업의 마감시간이 같은 경우 비주기 태스크를 선택한다.

그림 2는 동기적 주기 태스크 집합 $\tau=\{\tau_1=(3,6), \tau_2=(2,8)\}$에 대해 수행시간이 각각 1, 2, 1인 비주기 태스크 J_1, J_2, J_3가 시간 6, 15, 17에 도착했을 때의 ETBS와 TBS의 스케줄 예제이다(TBS의 마감시간 계산 수식은 4장 도입부에 있다). 그림 2의 (a)에서는 ETBS 스케줄과 함께 지연계수 값의 변화를 연속적으로 표시하였다. ETBS 스케줄에서 $U_p=3/4$, $U_s=1/4$, $\rho=1/3$이다. 시간 6에서의 지연계수는 0이고, 이때 도착한 J_1은 마감시간 10을 부여 받는다. J_1의 마감시간이 가장 빠르므로 J_1이 먼저 스케줄 되어 시간 7에 수

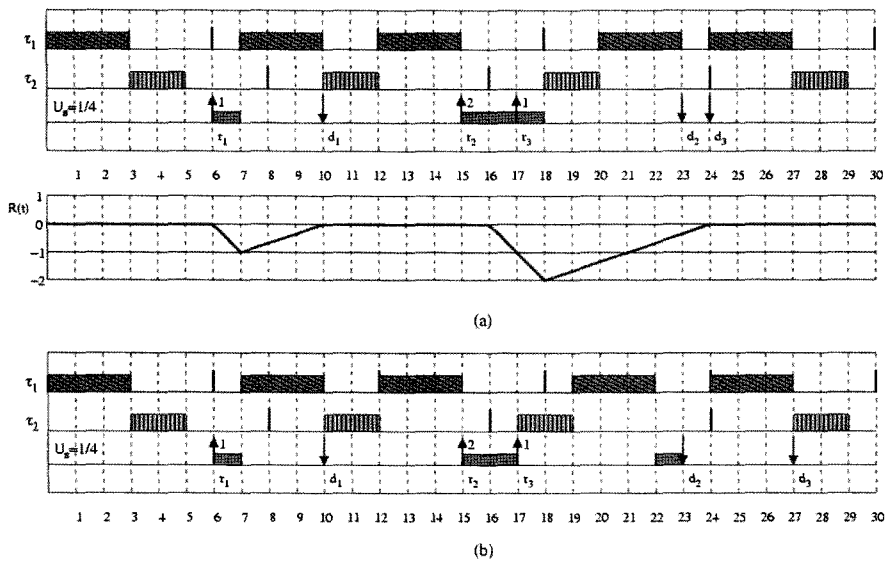

그림 2 (a) ETBS 스케줄 예제와 지연계수의 변화. (b) TBS 스케줄 예제

행을 마친다. 비주기 태스크가 단위 시간을 수행했으므로 $R(7)=-1$이다. 이후 주기 작업이 수행을 계속해, 시간 10에는 지연계수가 0이 된다(3ρ 시간을 $R(7)$에 더한다). 시간 10은 J_1의 마감시간임을 주의하자. 시간 10에 지연계수가 0이고 시간 15까지 비주기 태스크가 없으므로 구간 (10, 15]에서의 지연계수는 계산되지 않고 0이다. 시간 15에 도착한 최악 수행시간이 2인 J_2는 마감시간 23이 주어진다. 시간 15에는 주기 작업이 없으므로 J_2가 스케줄 된다. 시간 16에는 τ_2의 3번째 작업이 도착하여 스케줄링이 이루어진다. 시간 15에 수행할 주기 작업이 없었으므로 지연계수 계산 알고리즘 (i)에 의해 시간 16에서의 지연계수는 $R(16)=0$이다. 만일 시간 15에 비주기 태스크가 없었다면 구간 [15, 16]은 유휴시간이 됨을 주의하자. 시간 16에서도 J_2의 마감시간이 가장 빠르므로 J_2가 스케줄 되어 시간 17에서 J_2가 수행을 마치고 $R(17)=-1$이다. 시간 17에 도착한 비주기 태스크 J_3의 최악 수행시간은 1이고 $R(17)=-1$이므로 J_3의 마감시간은 24이다. 만일 J_3가 시간 16에 도착하여도 J_2가 수행을 마치는 시간까지 마감시간을 부여받지 않고 수행 가능하지 않음을 주의하자(이러한 경우에도 마감시간은 시간 17에서 24로 받게 된다).

J_1과 J_2의 마감시간은 ETBS 스케줄과 TBS 스케줄에서 동일하나, J_3의 마감시간은 ETBS 스케줄에서 24이고 TBS 스케줄에서는 27로, ETBS 스케줄에서 더 빠른 마감시간이 부여되어 J_3가 ETBS 스케줄에서 TBS 스케줄보다 빨리 끝나게 된다. TBS 알고리즘은 임의의 구간 $[t',t]$ 에서의 비주기 태스크들의 요구 수행시간이 항상 $(t-t')U_s$보다 크지 않도록 비주기 태스크에 마감시간을 부여한다. 그림 2의 (b)에서 보듯이 TBS가 구간 [15, 27]에서 비주기 태스크들의 요구시간 3을 처리했는데, 해당 구간에서의 비주기 태스크의 처리기 이용률이 1/4 임을 알 수 있다. 그러나 구간 [15, 16]은 비주기 태스크가 없었다면 유휴시간이 되는데 이는 시간 15이전에 비주기 태스크들이 비주기 태스크를 위한 이용률을 모두 이용하지 않아 주기 작업이 수행을 먼저 마쳤기 때문이다. 유휴시간동안 수행되는 비주기 태스크의 수행량은 앞으로 수행될 주기 작업의 스케줄 가능성에 어떠한 영향도 주지 않는다. ETBS에서는 이런 시간이 지연계수에 반영되어 (비주기 태스크가 구간 [15, 16]에서 수행되었음에도 시간 16에서 지연계수가 0 이다), 다음 비주기 태스크의 마감시간이 TBS 보다 짧아지게 된다.

3.2 스케줄 가능성

본 절에서는 처리기 이용률이 U_p인 주기 태스크 집합 τ에 대해 비주기 태스크를 위한 이용률 U_s가 $1-U_p$인 ETBS 알고리즘의 스케줄 가능성을 증명한다.

보조정리 1. ETBS 스케줄에서 비주기 태스크 J_k가 수행을 마치는 시간이 f_k이고, 구간 $[r_k, f_k]$에 주기 작업이 계속 있는 경우 $R(f_k)=-(d_k-f_k)$이다.

증명. 비주기 태스크 J_k의 마감시간 결정 시간이 r_k인데, 구간 $[r_k, f_k]$에 주기 작업이 계속 있다면 (주기 작업이 계속 수행하는 것이 아님을 주의하자) 주기 작업 및 비주기 태스크 J_k가 구간 $[r_k, f_k]$에서 계속 수행되는 경우이다. 시간 f_k에 비주기 태스크 J_k가 수행을 마쳤으므로 구간 $[r_k, f_k]$에서 J_k는 e_k를 수행하고, 주기 작업의 수행시간은 $f_k-r_k-e_k$이다. 지연계수 계산 알고리즘 (ii)에 의해 e_k만큼 지연계수는 감소되고, (iii)에 의해 $(f_k-r_k-e_k)\rho$ 만큼 지연계수는 증가된다. 따라서 $R(f_k)=R(r_k)+(f_k-r_k-e_k)\rho-e_k=R(r_k)+(f_k-r_k)\rho-(\rho+1)e_k$이다. e_k는 식 (1)에 의해 $(d_k-r_k+R(r_k)/\rho)U_s$ 인데, $\rho=U_s/U_p$이므로 $e_k=(d_k-r_k)U_s+R(r_k)U_p$이다. 따라서

$$\begin{aligned}R(f_k)&=R(r_k)+(f_k-r_k)\rho-(\rho+1)((d_k-r_k)U_s+R(r_k)U_p)\\&=R(r_k)+(f_k-r_k)\rho-(\frac{1}{U_p})((d_k-r_k)U_s+R(r_k)U_p)\\&\quad(U_p+U_s=1\ \text{이므로})\\&=(f_k-r_k)\rho-(d_k-r_k)\frac{U_s}{U_p}=(f_k-r_k)\rho-(d_k-r_k)\rho\\&=-(d_k-f_k)\rho\end{aligned}$$

이다. ■

보조정리 2. ETBS 스케줄에서 비주기 태스크 J_k가 수행을 마치는 시간을 f_k라 하면, $-(d_k-f_k)\rho \le R(f_k) \le 0$이다.

증명. 지연계수 계산 알고리즘 (iv)에 의해 $R(f_k)\le 0$이다.

구간 $[r_k, f_k]$ 에서 수행 가능한 주기 작업이 없는 경우, 수행 가능한 주기 작업이 계속 있는 경우, 수행 가능한 주기 작업이 일부 구간에서 없는 경우로 나눌 수 있다.

- 구간 $[r_k, f_k]$에 주기 작업이 없는 경우 : 수행가능한 주기 작업이 없는 경우 비주기 태스크가 수행을 하여도 지연계수 계산 알고리즘 (i)에 의해 $R(f_k)=0$이다.
- 구간 $[r_k, f_k]$에 주기 작업이 계속 있는 경우 : 보조정리 1에 의해 $R(f_k)=-(d_k-f_k)$이다.
- 구간 $[r_k, f_k]$에 주기 작업이 없는 구간이 있는 경우 : 주기 작업이 없는 구간은 비주기 태스크가 수행을 한다. 이때 비주기 태스크가 수행을 하더라도 주기 작업

이 없기 때문에 지연계수 계산 알고리즘 (ⅰ)에 의해 지연계수는 0이 된다. 따라서 f_k 바로 이전 스케줄링 시점에 주기 작업이 없었다면 지연계수는 0이 된다. f_k 바로 이전 스케줄링 시점에 주기 작업이 있었다고 하고, 시간 t를 f_k이전에 마지막으로 주기 작업이 없는 구간이 끝나는 시간이라고 하자($r_k < t < f_k$). $t = r_k$이면 구간 $[r_k, f_k]$에 주기 작업이 계속 있는 경우이다. 그러면 t 바로 이전에는 주기 작업이 없으므로 지연계수 계산 알고리즘 (ⅰ)에 의해 t에서의 지연계수는 0이고, J_k는 주기 작업이 없는 구간에서 수행되었으므로 t에서 J_k의 남은 수행시간은 e_k보다 작다. 즉, $R(t)=0$이고, t에서 J_k의 남은 수행시간을 e_k'라고 하면 $e_k' < e_k$이다. 구간 $[t, f_k]$에 주기 작업이 계속 있으므로 주기 작업 및 비주기 태스크가 구간 $[t, f_k]$에서 계속 수행된다. 구간 $[t, f_k]$에서 J_k는 e_k'를 수행하고, 따라서 주기 작업의 수행시간은 $f_k - t - e_k'$이다. 지연계수 계산 알고리즘 (ⅱ)에 의해 e_k' 만큼 지연계수는 감소되고, (ⅲ)에 의해 $(f_k - t - e_k')\rho$ 만큼 지연계수는 증가된다. 보조정리 1과 같은 방법으로 정리하면,

$$\begin{aligned} R(f_k) &= R(t) + (f_k - t - e_k')\rho - e_k' \\ &= (f_k - t)\rho - (\rho+1)e_k' \\ &> (f_k - t)\rho - (\rho+1)e_k \\ &> (f_k - t)\rho - (\rho+1)((d_k - r_k)U_s + R(r_k)U_p) \\ &> (f_k - t)\rho - R(r_k) - (d_k - r_k)\rho \end{aligned}$$

이다. $(f_k - t)\rho > 0$이고, $R(r_k) \le 0$이므로

$R(f_k) > -(d_k - r_k)\rho$이다.

따라서 $-(d_k - f_k)\rho \le R(f_k) \le 0$이다. ■

보조정리 3. ETBS 스케줄에서 시간 t_1 바로 이전에 수행한 비주기 태스크가 없거나 수행한 비주기 태스크들은 모두 마감시간이 t_1보다 작거나 같다고 하자. 또한 구간 $[t_1, t_2]$에는 수행가능한 주기 작업이 계속 존재한다고 하자. 마감시간 결정 시간이 t_1보다 나중이거나 같고 마감시간이 t_2보다 앞서거나 같은 비주기 태스크들의 구간 $[t_1, t_2]$에서의 최대 수행시간을 C_{ap}라고 하면, $C_{ap} \le (t_2 - t_1)U_s$이다.

증명. 본 정리의 정의와 식 (1)에 의해

$$C_{ap} = \sum_{t_1 \le r_k, d_k \le t_2} e_k = \sum_{t_1 \le r_k, d_k \le t_2} ((d_k - r_k) + \frac{R(r_k)}{\rho})U_s$$

이다. 편의상 $t_1 \le r_k$와 $d_k \le t_2$을 만족하는 J_k들을 $J_1, J_2, \cdots, J_n$이라 하자. C_{ap}가 최대가 되기 위해서는 J_1의 마감시간은 t_1에 결정되고 J_n의 마감시간은 t_2이며 ($r_1 = t_1$, $d_n = t_2$) 비주기 태스크들은 구간 $[t_1, t_2]$에 연속적으로 있으면 된다. 즉, J_k의 마감시간 결정 시간 r_k는 J_{k-1}가 수행을 마치는 시간이다(단 $2 \le k \le n$).

지연계수 계산 알고리즘 (ⅰ)과 (ⅳ)에 의해 시간 t_1에서는 $R(t_1) = 0$이다. 또한 보조정리 1에 의해 $R(r_k) = -(d_{k-1} - r_k)\rho$이다. 따라서

$$\begin{aligned} C_{ap} &= \sum_{t_1 \le r_k, d_k \le t_2} ((d_k - r_k) + \frac{R(r_k)}{\rho})U_s \\ &\le ((d_1 - t_1) + (d_2 - r_2) - (d_1 - r_2) + (d_3 - r_3) - (d_2 - r_3) \\ &\quad + \cdots + (t_2 - r_n) - (d_{n-1} - r_n))U_s \\ &\le (t_2 - t_1)U_s. \end{aligned}$$

■

정리 1. 처리기 이용률이 U_p인 스케줄 가능한 주기 태스크 집합 τ와 비주기 태스크들은 U_s가 $1 - U_p$인 ETBS 알고리즘으로 스케줄 가능하다.

증명. $U_p + U_s = 1$인데 시간 t에서의 작업들의 요구 수행시간(demand execution time)이 처리기의 처리 가능한 시간을 넘는다고 하자. 구간 $[t', t]$는 도착시간이 t'보다 나중이거나 같고 마감시간이 t보다 앞서거나 같은 주기 작업과, 마감시간 결정 시간이 t'보다 나중이거나 같고 마감시간이 t보다 앞서거나 같은 비주기 태스크가 수행하는 가장 긴 구간이라고 하자. C_p와 C_{ap}는 각각 주기 작업들과 비주기 태스크들의 구간 $[t', t]$에서의 요구 수행시간이라고 하자. 시간 t에서 어떤 작업이 마감시간을 지키지 못했으므로 $C_p + C_{ap} > t - t'$ 이다.

요구 수행시간을 최대로 만들기 위해 수행 가능한 비주기 태스크들은 구간 내에 계속 있다고 하자. 구간 $[t', t]$의 정의에 의해 이 구간 내에는 수행 가능한 주기 작업이 없는 경우, 주기 작업이 계속 있는 경우, 수행 가능한 주기 작업이 일부 구간에서 없는 경우로 나눌 수 있다. 편의상 $t' \le r_k$, $d_k \le t$ 만족하는 J_k들을 $J_1, J_2, \cdots, J_n$이라 하자.

- 구간 $[t', t]$에 주기 작업이 없는 경우 : t'의 정의와 지연계수 계산 알고리즘 (ⅰ)과 (ⅳ)에 의해 $R(t') = 0$이다. 구간 $[t', t]$에 주기 작업이 없으므로 지연계수 계산 알고리즘 (ⅰ)에 의해 구간 $[t', t]$에서 비주기 태스크들의 마감시간이 결정될 때의 지연계수들은 모두 0이다. 마감시간이 결정되지 않은 비주기 태스크는 마감시간이 결정된 비주기 태스크가 끝날 때까지 수행 가능하지 않다. 또한 $U_s > 0$이므로 비주기 태스크의 마감시간은 자신의 수행시간보다 짧을 수 없다. 따라서 시간 r_n까지 $J_1, J_2, \cdots, J_{n-1}$은 모두 마감시간을 지키며 이 작업들의 요구 시간은 $r_n - t'$이다. $R(r_n) = 0$이므로, J_n의 최대 요구 수행시간은 식 (1)에 의해 $(t - r_n)U_s$이고, 구간 $[t', t]$에 주기 작업은 없으므로

$C_p + C_{ap} \le r_n - t' + (t - r_n)U_s$이다. 마감시간을 지키지 못하는 조건 $C_p + C_{ap} > t - t'$은 $C_p + C_{ap} > r_n - t' + t - r_n$과 같이 나타낼 수 있다. $r_n - t' + (t - r_n)U_s > r_n - t' + t - r_n$ 을 만족하기 위해서는 $U_s > 1$이어야 한다. 이는 $U_p + U_s = 1$인 전제 조건에 위배된다.

- 구간 $[t', t]$에 주기 작업이 계속 있는 경우 : C_p는 다음과 같다.

$$C_p = \sum_{i=1}^{n} \left\lfloor \frac{t-t'}{P_i} \right\rfloor C_i \le \sum_{i=1}^{n} \frac{t-t'}{P_i} C_i = (t-t')U_p \ .$$

t'의 정의에 의해 t' 바로 이전에는 비주기 태스크가 없는 경우이다. 따라서 보조정리 3에 의해 $C_{ap} \le (t-t')U_s$이다. $C_p + C_{ap} \le (t-t')(U_p + U_s)$인데, 전제 조건은 $C_p + C_{ap} > t - t'$이므로 $U_p + U_s > 1$이어야 한다. 이는 $U_p + U_s = 1$인 전제 조건에 위배된다.

- 구간 $[t', t]$에 주기 작업이 없는 구간이 있는 경우 : 주기 작업이 없는 구간의 총 시간을 I라고 하자. 요구 시간이 최대가 되기 위해서, 주기 작업이 없는 구간에서 비주기 태스크가 I시간 모두를 요구한다고 하자. 주기 작업이 있는 구간의 주기 작업 및 비주기 태스크의 최대 요구 시간은 위의 주기 작업이 계속 있는 경우의 증명에 의해 $(t-t'-I)(U_p+U_s)$ 보다 작거나 같다. 따라서 $C_p + C_{ap} \le (t-t'-I)(U_p+U_s)+I$이다. 마감시간을 지키지 못하는 전제 조건 $C_p + C_{ap} > t - t'$은 $C_p + C_{ap} > t - t' + I - I$과 같이 나타낼 수 있다. $(t-t'-I)(U_p+U_s)+I > t-t'+I-I$를 만족하기 위해서는 $U_p + U_s > 1$이어야 한다. 이는 $U_p + U_s = 1$인 전제 조건에 위배된다.

따라서 주어진 정리는 성립한다. ■

4. 성능평가

본 장에서는 ETBS와 TBS에서의 비주기 태스크 응답시간을 증명과 실험을 통해 비교한다. 증명은 ETBS에서 임의의 비주기 태스크[1]에 부여한 마감시간이 TBS에서 부여한 마감시간보다 늦지 않음을 보인다. EDF 스케줄링의 경우 기존의 비주기 태스크의 마감시간 보다(주기 태스크들의 마감시간을 보장하면서) 앞선 마감시간을 부여해 스케줄을 하면 기존의 마감시간으로 스케줄 했을 때보다 응답시간이 늦어지지 않는다.

TBS 알고리즘에서 비주기 태스크를 위한 이용률을 U_s라 할 때, $(U_p + U_s \le 1)$, 시간 a_k에 도착한 비주기 태스크 J_k의 마감시간 d_k는 다음 수식에 의해 부여된다.

$$d_k = \max(a_k, d_{k-1}) + \frac{e_k}{U_s} \qquad (2)$$

TBS 알고리즘에서 비주기 태스크 J_k의 마감시간이 이전 비주기 태스크 J_{k-1}의 마감시간에 영향을 받는 것에 주의하자.

4.1 이론적인 평가

보조정리 4. ETBS 스케줄에서 비주기 태스크 J_k가 수행을 마치는 시간을 f_k라 하고, t는 $f_k \le t \le d_k$을 만족한다고 하자. 구간 $[f_k, t]$에서 비주기 태스크가 수행되지 않았다면 $-(d_k - t)\rho \le R(t) \le 0$이다.

증명. 보조 정리 2에 의해 시간 f_k에서의 최소 지연계수는 $-(d_k - f_k)\rho$이다. 주기 작업이 $[f_k, t]$에서 계속 수행을 하면 $(t-f_k)\rho$ 만큼 지연계수가 증가 한다. 그러면, $R(t) = -(d_k - f_k)\rho + (t - f_k)\rho = -(d_k - t)\rho$이다. 시간 f_k에서의 지연계수가 $-(d_k - f_k)\rho$ 보다 크고 주기 작업이 t까지 계속 수행 하더라도 지연계수 계산 알고리즘 (iv)에 의해 지연계수는 0보다 클 수 없다.

구간 $[f_k, t]$내에서 유휴 시간이 있었다면, 유휴 시간이 끝날 때의 지연계수는 지연계수 계산 알고리즘 (i)에 의해 0이 되고 이후 시간 t까지 지연계수는 변경되지 않는다. 따라서 $-(d_k - t)\rho \le R(t) \le 0$이다. ■

정리 2. TBS와 ETBS 스케줄에서의 비주기 태스크 J_k의 마감시간을 각각 d_k^{TBS}와 d_k^{ETBS}라 하자. 주어진 주기 태스크 집합과 비주기 태스크 집합에 대한 유효한 TBS와 ETBS 스케줄에서 비주기 태스크들의 수행 순서가 같을 때, 모든 J_k에 대해 $d_k^{ETBS} \le d_k^{TBS}$ 이다.

증명. 귀납법을 사용하여 증명한다. 두 스케줄에서 J_1의 도착시간은 같고, ETBS 스케줄에서 $R(r_1) = 0$이므로 식 (1)과 (2)에 의해 $d_1^{TBS} = d_1^{ETBS}$이다. 귀납 가정 (inductive hypothesis) 은 J_{k-1}에 대해 $d_{k-1}^{ETBS} \le d_{k-1}^{TBS}$ 이라고 하자 $(k \ge 2)$.

J_k 에 대해서 $d_k^{ETBS} \le d_k^{TBS}$ 이 성립함을 보이기 위해, ETBS 스케줄에서 J_k의 마감시간 결정 시간 r_k가 J_{k-1}의 마감시간 d_{k-1}^{ETBS} 보다 작거나 같은 경우와 큰 경우에 대해 각각 고려한다.

- $r_k \le d_{k-1}^{ETBS}$ 인 경우 : $r_k \le d_{k-1}^{ETBS}$이므로 J_k의 도착시간 a_k는 $a_k \le d_{k-1}^{ETBS}$ 을 만족한다. 귀납 가정 $d_{k-1}^{ETBS} \le d_{k-1}^{TBS}$ 에 의해, $a_k \le d_{k-1}^{TBS}$ 이므로 TBS 스케줄에서의 J_k의 마감시간은 $d_k^{TBS} = d_{k-1}^{TBS} + \frac{e_k}{U_s}$ 이다. ETBS 스케줄에서 J_{k-1}가 수행을 마치는 시간을

1) 비주기 태스크의 처리 순서는 두 알고리즘에서 같다. 동시에 도착한 비주기 태스크들도 두 알고리즘에서 같은 순서로 처리함을 가정한다.

f_{k-1}라 하자. ETBS 스케줄에서 $f_{k-1} \le r_k \le d_{k-1}^{ETBS}$ 이고, J_k의 마감시간은 $d_k^{ETBS} = r_k + \frac{e_k}{U_s} - \frac{R(r_k)}{\rho}$ 이다. 보조 정리 4에 의해 시간 r_k에서의 최소 지연계수는 $-(d_{k-1}^{ETBS} - r_k)\rho$ 이다. 따라서,

$$\begin{aligned} d_k^{TBS} - d_k^{ETBS} &= d_{k-1}^{TBS} + \frac{e_k}{U_s} - r_k - \frac{e_k}{U_s} + \frac{R(r_k)}{\rho} \\ &\ge d_{k-1}^{TBS} - r_k - \frac{(d_{k-1}^{ETBS} - r_k)\rho}{\rho} \\ &\ge d_{k-1}^{TBS} - d_{k-1}^{ETBS} \end{aligned}$$

이다. $d_{k-1}^{ETBS} \le d_{k-1}^{TBS}$ 이므로 $d_k^{TBS} - d_k^{ETBS} \ge 0$ 이다.

- $r_k > d_{k-1}^{ETBS}$ 인 경우 : 보조 정리 4에 의해 ETBS 스케줄에서 d_{k-1}^{ETBS} 에서의 지연계수는 0이 되고, r_k까지 비주기 태스크가 없으므로 $R(r_k) = 0$이다. 또한 유효한 ETBS 스케줄이므로 ETBS 스케줄에서 J_{k-1}은 마감시간 d_{k-1}^{ETBS} 전에 끝났고, $r_k > d_{k-1}^{ETBS}$ 이므로 J_k는 도착시 마감시간이 결정되어 $r_k = a_k$이다. 따라서 $d_k^{ETBS} = a_k + \frac{e_k}{U_s}$ 이다. TBS 스케줄에서 J_k의 마감시간은 $\max(a_k, d_{k-1}^{TBS}) + \frac{e_k}{U_s}$ 인데, $a_k \le d_{k-1}^{TBS}$인 경우 $d_k^{TBS} = d_{k-1}^{TBS} + \frac{e_k}{U_s}$ 이므로 $d_k^{TBS} - d_k^{ETBS} \ge 0$이고, $a_k > d_{k-1}^{TBS}$인 경우 $d_k^{TBS} = a_k + \frac{e_k}{U_s}$ 이므로 $d_k^{TBS} - d_k^{ETBS} = 0$ 이다.

따라서 모든 경우에 $d_k^{ETBS} \le d_k^{TBS}$ 이다. ■

정리 2에서도 알 수 있듯이 k번째 비주기 태스크가 TBS 스케줄에서의 $k-1$번째 비주기 태스크의 마감시간 이후에 도착한다면 ETBS 스케줄에서 k번째 비주기 태스크의 마감시간은 TBS 스케줄에서의 k번째 비주기 태스크의 마감시간과 같다. 그러나 k번째 비주기 태스크

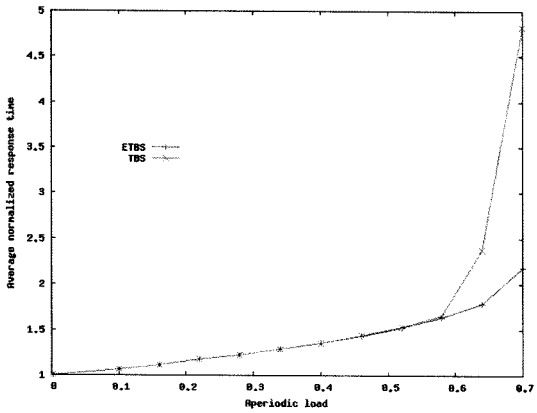

그림 3 $U_p = 0.3$일 때의 정규화 응답시간 비교

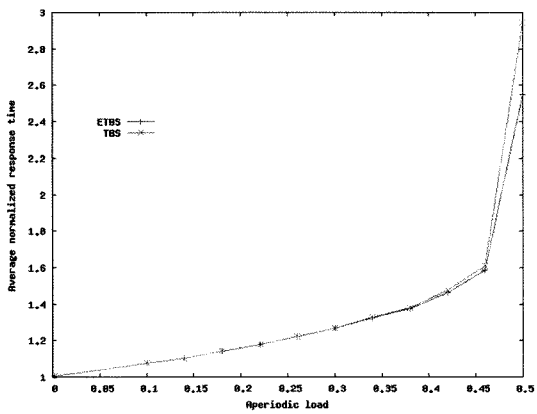

그림 4 $U_p = 0.5$일 때의 정규화 응답시간 비교

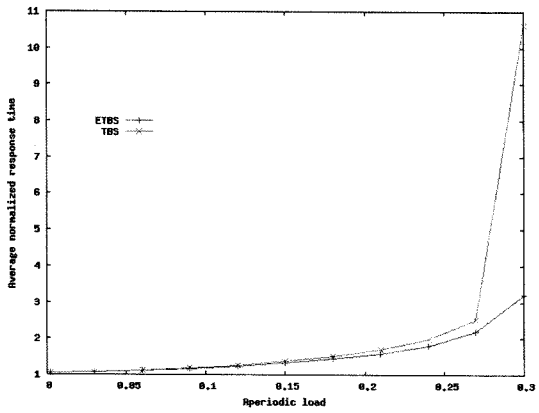

그림 5 $U_p = 0.7$일 때의 정규화 응답시간 비교

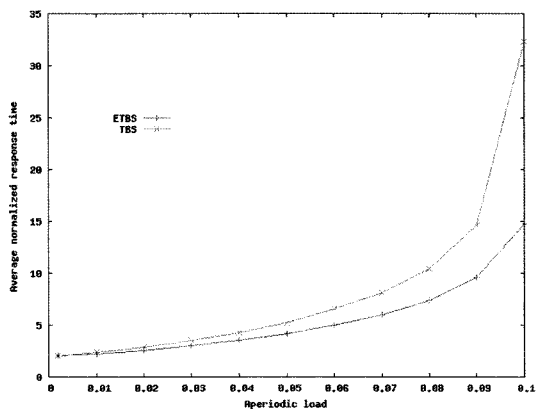

그림 6 $U_p = 0.9$일 때의 정규화 응답시간 비교

가 TBS 스케줄의 $k-1$번째 비주기 태스크의 마감시간 이전에 도착한다면, ETBS 스케줄과 TBS 스케줄에서의 $k-1$번째 비주기 태스크의 마감시간 차이만큼 ETBS 스케줄의 k번째 비주기 태스크의 마감시간은 짧아질 수 있다. 즉 비주기 태스크들의 부하가 많아 연속적으로 비주기 태스크가 도착하는 경우 ETBS 스케줄에서의 비주기 태스크가 TBS 스케줄에서의 비주기 태스크보다 빠른 응답시간을 보일 수 있다.

4.2 모의 실험

본 논문의 실험에서는 ETBS 알고리즘과 TBS 알고리즘의 비주기 태스크 응답시간 비교를 위해 1000개의 비주기 태스크 집합을 Buttazzo[13]의 실험과 같은 방법으로 무작위로 생성하였다. 단, 모의실험의 편의성을 위해 각 태스크의 주기 및 수행시간은 정수가 되도록 조정을 하였다. 주기 태스크 집합은 10개의 주기 태스크로 구성되며, 주기 태스크 집합의 처리기 이용률은 U_p이다. 이를 위해 먼저 주기 태스크 τ_i의 이용률 u_i는 (0,1] 범위에서 균일분포(uniform distribution)에 따라 생성된 값을 τ_i의 태스크 집합 이용률 U_p로 정규화 하여 정하였다. 그리고 태스크 τ_i의 주기 P_i는 [10,60] 범위에서 균일분포(uniform distribution)에 따라 생성한 후 수행시간 C_i ($C_i = u_i \cdot P_i$)을 정하였다. 각 비주기 태스크 집합은 10개의 비주기 태스크로 구성되며, 각 비주기 태스크의 수행시간은 [2,6] 범위에서 균일분포(uniform distribution)에 따라 생성하였다.

그림 3, 4, 5, 6은 각각 주기 작업의 이용률이 0.3, 0.5, 0.7, 0.9 인 경우에 비주기 태스크의 부하에 따른 ETBS 스케줄과 TBS 스케줄에서 비주기 태스크의 평균 정규화 응답시간(average normalized response time)을 나타낸 것이다. 비주기 태스크의 정규화 응답시간은 해당 태스크의 수행시간에 대해 응답 시간이 몇 배인지를 나타낸다. 즉 비주기 태스크의 정규화 응답시간이 3인 경우 해당 비주기 태스크의 수행시간의 3배가 실제 응답시간이 된다.

그림 3과 4에서도 알 수 있듯이 주기 작업 및 비주기 태스크의 부하가 모두 작은 경우 ETBS 스케줄에서의 응답시간과 TBS 스케줄에서의 응답시간은 차이가 매우 적다. 또한 주기 작업의 부하가 커지더라도 비주기 태스크의 부하가 작은 경우 두 알고리즘의 응답시간은 차이가 적다. 그러나 그림 3, 4, 5, 6에서 보이듯이 비주기 태스크들의 부하가 커짐에 따라 ETBS 스케줄에서의 응답시간이 TBS 스케줄에서의 응답시간 보다 많이 빨라진다.

그림 3, 5, 6에서 보이듯이 처리기 이용률이 100%에 근접하는 경우에는 TBS 스케줄에서의 응답시간이 ETBS 스케줄에서의 응답시간에 비하여 매우 느려진다. 이론적인 방법에서 보였듯이 ETBS 스케줄과 TBS 스케줄에서의 $k-1$번째 비주기 태스크의 마감시간 차이만큼 ETBS 스케줄에서의 k번째 비주기 태스크의 마감시간은 짧아질 수 있다. 즉 k번째 이후의 비주기 태스크의 마감시간은 ETBS 스케줄과 TBS 스케줄에서 계속 차이가 생길 수 있다. 비주기 태스크의 부하가 큰 경우 이 차이가 계속 유지 될 수 있음을 유의하자. 주기 태스크의 부하가 큰 경우에는 주기 작업의 간섭이 커져 $k-1$번째 비주기 태스크의 마감시간 차이가 다음 비주기 태스크의 마감시간에도 영향을 주는 경우가 많아진다. 따라서 이러한 경우 ETBS 스케줄에서의 마감시간이 TBS 스케줄에서의 마감시간보다 더 짧아지는 경우가 많아진다. 반면 주기 태스크의 부하가 작고 비주기 태스크의 부하도 작은 경우에는 $k-1$번째 비주기 태스크의 마감시간이 차이가 있더라도 다음 비주기 태스크가 늦게 도착해 (TBS의 $k-1$번째 마감시간 이후) 다음 비주기 태스크의 마감시간은 차이가 없는 경우가 많아진다. 따라서 이러한 경우 ETBS 스케줄과 TBS 스케줄에서의 응답시간은 큰 차이가 없다. 그러나 주기 태스크의 부하는 작지만 비주기 태스크의 부하가 큰 경우에는 상대적으로 비주기 태스크가 많아져 $k-1$번째 비주기 태스크의 마감시간 차이가 ETBS 스케줄에서 다음 비주기 태스크들의 마감시간에도 영향을 주는 경우가 많아져 ETBS 스케줄에서의 응답시간이 빨라지게 된다. 주기 태스크의 이용률이 0.5(그림 4)이고 비주기 태스크의 부하도 큰 경우에는 위 설명의 중간적인 위치로, 처리기 이용률이 100%에 근접하는 경우에도 두 알고리즘의 응답시간이 그림 3, 5, 6 만큼 큰 차이를 보이지는 않는다.

5. 결 론

본 논문에서는 단일처리기 시스템에서 처리기 이용률을 최대로 활용하면서 경성 실시간 태스크들의 마감시간을 보장하고 비주기 태스크들의 응답시간을 단축시켜주는 ETBS 알고리즘을 제안하였다. ETBS는 스케줄링 시점에만 부가적인 정보를 간단히 계산하여 비주기 태스크에 마감시간을 부여 할 수 있다. ETBS 스케줄에서 마감시간이 부여된 비주기 태스크는 주기 작업과 함께 EDF 스케줄링 알고리즘을 사용하여 스케줄 된다.

제안된 알고리즘 평가를 위해 계산 복잡도 $O(1)$이면서 비주기 태스크의 빠른 응답시간을 제공하는 TBS와 비교하였다. 비주기 태스크에 대하여 ETBS에서의 마감시간 계산은 TBS와 같은 계산 복잡도를 갖는다. ETBS 스케줄에서의 비주기 태스크의 응답시간이 TBS 스케줄에서의 비주기 태스크의 응답시간보다 늦지 않음을 이론적인 방법을 통해 보였다. 또한 모의실험을 통해

ETBS 알고리즘에서의 비주기 태스크들이 TBS 알고리즘에서의 비주기 태스크들보다 향상된 응답시간을 가짐을 보였다. 특히 비주기 태스크의 이용률이 큰 경우, 비주기 태스크들의 응답시간이 TBS보다 ETBS에서 많이 향상됨을 보였다.

참 고 문 헌

[1] Liu, C. L. and Layland, J. W, "Scheduling algorithms for multiprogramming in a hard-real-time environment," Journal of the ACM, Vol. 20, No. 1, pp. 46-61, 1973.

[2] J. K. Strosnider, J. P. Lehoczky, and L. Sha, "The deferrable server algorithm for enhancing aperiodic responsiveness in hard-real time environments," IEEE Transactions on Computers, Vol. 4, No. 1, pp. 73-91, 1995.

[3] J. P. Lehoczky, L. Sha and J. K. Strosnider, "Enhanced aperiodic responsiveness in hard-real time environments," In Proceedings of the IEEE Real-Time Systems Symposium, pp. 261-270, 1987.

[4] B. Sprunt, L. Sha and J. P. Lehoczky, "Aperiodic task scheduling for hard-real time systems," Journal of Real-Time Systems, Vol. 1, No. 1, pp. 27-60, July 1989.

[5] J. P. Lehoczky, and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems," In Proceedings of the IEEE Real-Time Systems Symposium, pp. 110-123, December 1992.

[6] M. Spuri and G. Buttazzo, "Efficient aperiodic service under earliest deadline scheduling," In Proc. of the IEEE Real-Time Systems Symposium, pp. 2-11, December 1994.

[7] M. Spuri and G. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," Journal of Real-Time Systems, Vol. 10, No. 2, pp. 179-210, 1996.

[8] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," In Proc. of the IEEE Real-Time Systems Symposium, December 1998.

[9] S. Baruah and G. Lipari, "A multiprocessor implementation of the total bandwidth server," In Proceedings of International Parallel and Distributed Processing Symposium, pp.26-30, April 2004.

[10] S. Baruah, J. Goosens, and G. Lipari, "Implementing constant-bandwidth servers upon multiprocessor platform," In Proceedings of the IEEE Real-Time Technology and Applications Symposium, pp. 154-163, September 2002.

[11] G. Buttazzo and E. Bini, "Optimal dimensioning of a constant bandwidth server," In Proc. of the IEEE Real-Time Systems Symposium, pp. 169-177, December 2006.

[12] M.Dertouzos, "Control robotics; the procedural control of physical processes," Information Processing, pp. 807-813, 1974.

[13] G. Buttazzo and F. Sensini, "Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments," IEEE Transactions on Computers, Vol. 48, No. 10, pp. 1035-1052, 1999.

김 희 헌

1999년 고려대학교 컴퓨터과학과 학사 2000년~현재 서울대학교 전기·컴퓨터공학부 석박사 통합과정. 관심분야는 운영체제, 분산 시스템, 실시간시스템, 내장형 시스템 등

박 학 봉

1998년 중국 연변대학교 법학과 학사 2002년 호서대학교 컴퓨터공학과 학사 2004년 호서대학교 컴퓨터공학과 공학석사. 2004년~현재 서울대학교 전기·컴퓨터공학부 박사과정 재학중. 관심분야는 운영체제, 시스템소프트웨어, 실시간 시스템, 임베디드 소프트웨어 등

박 문 주

1996년 서울대학교 조선해양공학과 학사 1998년 서울대학교 컴퓨터공학과 공학석사. 2002년 서울대학교 전기컴퓨터공학부 공학박사. 2002년~2006년 LG전자 단말연구소. 2006년~2007년 IBM Ubiquitous Computing Lab. 2007년~현재 인천대학교 컴퓨터공학과 전임강사. 관심분야는 운영체제, 시스템소프트웨어, 실시간시스템, 내장형시스템

박 민 규

1991년 서울대학교 컴퓨터공학과 학사 1993년 서울대학교 컴퓨터공학과 공학석사. 2005년 서울대학교 컴퓨터공학과 공학박사. 2004년~2006년 한국산업기술대학교 겸임교수. 2006년 9월~현재 건국대학교 컴퓨터응용과학부 재직. 관심분야는 운영체제, 시스템소프트웨어, 실시간시스템, 임베디드 소프트웨어 등

조 유 근

1971년 서울대학교 졸업 학사. 1978년 미국 미네소타 대학교 컴퓨터 과학 공학 박사. 1985년 미국 미네소타 대학교 방문 교수. 2001년 한국 정보과학회 회장 1979년~현재 서울대학교 컴퓨터공학부 교수. 관심분야는 운영체제, 알고리즘, 시스템 보안, 결함 허용 컴퓨팅

조 성 제

1989년 서울대학교 컴퓨터공학과 학사 1991년 서울대학교 컴퓨터공학과 공학석사. 1996년 서울대학교 컴퓨터공학과 공학박사. 2001년~2002년 미국 University of California, Irvine 객원연구원. 1997년 3월~현재 단국대학교 정보컴퓨터학부 부교수. 관심분야는 컴퓨터보안, 시스템소프트웨어, 실시간시스템, 임베디드 소프트웨어 등