논문 2009-46CI-3-10

가변 실행시간의 실시간 태스크들에 대하여 공유대역폭을 활용한 응답시간의 개선

(Enhancement of Response Time of Real-Time Tasks with Variable Execution Times by Using Shared Bandwidth)

김 용 석*

(Yong-Seok Kim)

요 약

태스크의 실행시간은 다양한 입력 데이터에 따라 가변적일 수 있다. 최악의 실행시간을 만족하도록 높은 성능의 프로세서를 사용하면 하드웨어 비용이 증가하고 에너지 소비 가 늘어나게 된다. 따라서 적절히 낮은 성능의 프로세서를 적용하기 위해서는, 스케줄링에서는 프로세서의 용량을 최대한 활용하되 가끔씩 일부 태스크가 마감시한을 초과하더라도 다른 태스크에는 영향을 미치지 않도록 제한하는 것이 필요하다. 본 논문에서 제시하는 SBP (Shared Bandwidth Partitioning)는 프로세서의 공유 대역폭을 확보하여 태스크들이 나누어 사용할 수 있도록 하였다. 실행시간이 길어지는 태스크는 이 공유대역폭의 일부를 분할하여 사용하도록 한다. 시뮬레이션으로 평가한 결과, SBP는 기존의 알고리즘들에 비해서 개선된 결과를 얻을 수 있었다. 스케줄링 결과의 질에 해당하는 마감시한 초과 비율이 낮아지고 시스템의 오버헤드에 해당하는 문맥교환 횟수도 감소하는 것을 확인하였다.

Abstract

Execution times of tasks can be variable depend on input data. If we choose a high performance processor to satisfy the worst case execution times, the hard cost becomes high and the energy consumption also becomes large. To apply a lower performance processor, we have to utilize processor capacity maximally while overrunning tasks can not affect deadlines of other tasks. To be used for such systems, this paper presents SBP (Shared Bandwidth Partitioning) that a processor bandwidth is reserved and shared among all tasks. If a task needs more processor capacity, it can use a portion of the shared bandwidth. A simulation result shows that SBP provides better performance than previous algorithms. SBP reduces deadline miss ratio which is related to scheduling quality. And the number of context switches, which is related to system overhead, is also reduced.

Keywords : 스케줄링, 연성 실시간 태스크, 공유 대역폭, 슬랙 재활용

I. 서 론

내장형 시스템들은 일반적으로 마이크로프로세서를 기반으로 한 하드웨어에 실시간 운영체제를 적용하고, 목적에 따른 적절한 응용 태스크들을 실행하는 구조를 가지고 있다. 응용 태스크들은 해당 시스템의 고유 목적에 따라 작성된 태스크들로서 시스템이 종료될 때까지 주기적으로 일정한 작업을 반복하는 형태로 작성된다. 이러한 태스크들은 주기내의 적절한 시점까지 실행을 완료해야 하는 마감시한을 제한조건으로 만족해야 한다.

주기적인 실시간 태스크들을 스케줄링하기위한 알고리즘들은 다양하게 개발되어 왔다.[1~2] 이러한 스케줄링 알고리즘들은 대부분 각주기마다 실행에 필요한 시간이

* 정회원, 강원대학교 컴퓨터학부
(Kangwon National University)
※ 본 논문은 부분적으로 강원대학교 정보통신연구소의 지원을 받았음
접수일자 : 2009년1월7일, 수정완료일: 2009년4월28일

주어져 있는 것을 전제로 하고 있는데, 실제 실행되는 과정에서 태스크들은 다양한 입력 조건에 따라 실행시간이 가변적이다. 따라서 최악의 경우에도 마감시한을 만족하도록 하기 위해서는 최악의 실행시간을 전제로 하여 스케줄링을 실시하게 된다. 그 결과로 평균적인 실행시간보다 훨씬 고성능의 프로세서를 필요로 하게 된다.

최근에는 실행시간이 가변적인 시스템을 위하여, 대부분의 경우에는 마감시한을 만족하도록 하지만 일시적인 과부하 상태에서는 부분적으로 마감시한을 넘기는 것을 허용하는 방향의 연구들이 이루어져 왔다. 이렇게 함으로써 과잉 성능의 하드웨어가 필요하지 않게 된다. 이것은 하드웨어 비용을 절감하는 효과 외에도, 이동성이 필요한 시스템에서는 전력소모를 최소화하는 것도 매우 중요한 측면인데, 프로세서의 성능을 적절히 낮추어 줌으로써 전력소모를 낮출 수 있는 효과를 얻을 수 있게 된다. 일시적은 과부하를 전제로 스케줄링하는 데에는 두 가지 방향이 있는데, 하나는 각 태스크가 꼭 실행해야만 하는 부분과 그렇지 않은 부분으로 구성된 것을 전제로 하여 스케줄링을 하는 것이다.[3~4] 다른 하나는 태스크들의 실행시간이 가변적인 시스템에서, 평균적으로는 마감시한을 만족하지만 일시적인 과부하 상태에서는 마감시한을 초과하는 것이 허용되는 것을 전제로 하여 마감시한을 초과하는 정도를 최소화하는 것이다.[7~9] 이들은 태스크들이 할당된 실행시간 이내에 조기 완료되면 남는 시간 구간들을 적절히 재활용하여 여러 태스크들이 자신의 마감시한을 만족하도록 스케줄링하는 것을 시도하고 있다. 본 논문도 이러한 시스템을 기초로 하고 있다.

경성 실시간 태스크는 마감시한을 반드시 만족해야 하는 태스크들을 의미하고, 연성 태스크는 경우에 따라서는 마감시한을 초과할 수도 있는 태스크들을 의미한다. 본 논문에서는 경성 실시간 태스크와 연성 실시간 태스크들이 혼재되어 있는 시스템에서 경성 실시간 태스크들은 마감시한을 보장하면서, 연성 실시간 태스크들도 최대한 마감시한을 만족하도록 하는 스케줄링 정책을 제시한다. 태스크들이 공유하여 사용할 수 있는 프로세서 대역폭을 확보하여 실행에 필요한 시간이 모자라는 태스크들이 활용할 수 있도록 한다. 태스크들이 자신에게 할당된 실행시간보다 일찍 완료되는 경우에 남는 여분의 시간도 실행시간이 길어지는 태스크들에게 할당하여 전체적으로 마감시한을 초과하는 비율을 줄이도록 하였다. 또한 각 태스크들은 단순히 경성 실시간 태스크와 연성 실시간 태스크로 양분하지 않고, 마감시한을 초과해도 무방한 정도를 적절히 지정할 수 있도록 하였다. 이렇게 함으로써 각 태스크들에 대하여 경성실시간 태스크로부터 아주 느슨한 연성 실시간 태스크까지의 넓은 스펙트럼으로 설정할 수 있도록 하였다.

Ⅱ. 관련연구

하나의 태스크가 실행시간을 초과하여 실행하게 되면 다른 태스크들도 실행시간이 지연되어 마감시한을 넘기는 상황이 발생할 수 있으며, 이러한 현상이 태스크들 간에 도미노처럼 발생할 수 있다. 이러한 현상을 없애기 위해서는 실행시간을 초과한 태스크를 중지시키는 방법을 적용할 수도 있으나, 실행중인 태스크를 중지시킬 경우에는 이로 인해서 임계영역을 실행 중이거나 중간에 중지함으로써 재난을 초래할 수 있는 상황 등이 발생할 수 있는 경우에는 적절하지 못한 방법이다. 보다 현실적인 방법으로는 우선순위를 낮추어서 다른 태스크에 영향을 미치지 않는 한도 내에서 계속 실행하도록 한다.

각 태스크에는 프로세서의 대역폭을 일정량 할당하고 이 한도 내에서만 실행되도록 할 수 있다.[5] 따라서 하나의 태스크가 실행시간을 초과하더라도 자신의 실행만 늦어질 뿐이고 다른 태스크의 마감시한에 대한 영향은 발생하지 않는다. 이러한 방법으로 제시된 CBS[6] (Constant Bandwidth Server)는 EDF 정책을 기반으로 하면서, 초과 소요시간에 대하여 태스크 자신의 이후 주기의 인스턴스들의 사용가능 시간을 활용하여 실행하는 방식을 제시하였다. 이 방식의 문제점으로는 특정 태스크에 할당된 시간이 실제 실행과정의 평균적인 실행시간 보다 부족하면 이 태스크는 계속 느리게 실행하게 된다. 반대로 특정 태스크에 할당된 시간이 실제 실행에 필요한 시간보다 과도하면 프로세서가 아무것도 실행하지 않고 낭비하는 시간이 발생하여 전체 시스템의 성능을 저하시키는 원인이 된다.

이러한 문제점을 해결하기 위한 방안으로서 각 태스크 인스턴스들이 할당된 시간보다 일찍 완료되면 이 슬랙을 다른 태스크들을 실행하는데 활용할 수 있다. 이러한 관점에서 제시된 GRUB[7] (Greedy Reclamation of Unused Bandwidth)에서는 CBS와 마찬가지로 EDF 정책에 따라 태스크들을 스케줄링하지만, 태스크가 자

신에게 할당된 용량을 다 사용하지 않고 완료되면 남은 용량을 슬랙으로 등록한다. 조기에 완료된 시점부터 주기의 마감시한까지 구간은 이 슬랙으로 인한 효과만큼을 전체 활용률에서 감소시킨다. 각 태스크 인스턴스는 실제 실행된 시간에 전체 활용률을 나눈 양만큼 자신의 가상시간을 증가시켜나가는데, 이것은 실제 실행된 시간구간에 현 시점의 전체 태스크들의 활용률을 곱한 양만큼만 자신에게 할당된 용량에서 차감하는 효과를 얻는다. 슬랙으로 인해 전체 활용률이 감소하면, 실행되는 태스크는 자신에게 할당된 용량을 적게 소비한 것으로 처리되는 것이다. 이것은 곧 각 슬랙 구간에 대하여 그 활용률 만큼을 재활용하는 효과를 얻는다. GRUB는 이렇게 다른 태스크들의 슬랙 부분을 활용함으로써 CBS에 비해 응답시간 면에서 우수하고 또한 문맥교환 횟수도 줄일 수 있는 것으로 나타났다.

BASH[8] (BAndwidth SHaring)에서는 DPE[10] (Dynamic Priority Exchange) 서버에서 비주기적 태스크들을 실행하는 것과 유사하게 슬랙들을 활용한다. 할당된 실행시간보다 조기에 완료됨으로써 남는 시간은 그 인스턴스의 마감시한 이내에 남은 용량만큼을 다른 태스크가 사용할 수 있도록 슬랙으로 등록한다. CBS나 GRUB와 마찬가지로 실행가능 태스크 인스턴스 들중에서 EDF 정책에 따라 선택되며, 각 태스크 인스턴스는 실행될 때 자신의 마감시한보다 이른 슬랙들의 용량을 슬랙 끝 시간이 이른 순서대로 먼저 사용한 후에, 모자라는 양을 자신의 용량에서 차감한다. 시뮬레이션 결과에 의하면 BASH가 응답시간 면에서 GRUB보다 우수하게 나타난다. 그러나 문맥교환 회수에 있어서는 본 논문에서 시뮬레이션을 통해 확인한 결과 [7]에서 주장한 것처럼 GRUB가 더 우수함을 보여주고 있다. BASH에서는 태스크 자신의 마감시한보다 이른 슬랙들만 활용하도록 한데 반해서 ASR[9] (Aggressive Slack Reclaiming)에서는 슬랙들을 보다 적극적으로 활용한다. 즉, 자신의 마감시한보다 늦은 슬랙에 대해서도 실행중인 태스크 인스턴스의 마감시한을 기준으로 슬랙을 양분한 다음 앞부분도 추가로 활용한다. 이렇게 함으로써 BASH보다 응답시간 면에서 우수한 결과를 얻을 수 있게 되었다.

CBS, GRUB 및 BASH는 모두 스케줄링 과정에서는 용량이 남아있는 실행가능 태스크 인스턴스들 중에서 EDF 정책에 의해 실행할 인스턴스를 선택한다. 각 태스크 인스턴스들이 슬랙들과 자신의 할당용량을 다 활용하고도 작업을 완료하지 못하면 EDF 정책상의 우선순위를 낮추어주기 위해서 마감시한을 다음주기의 마감시한으로 연장하고 자신의 다음 주기의 인스턴스를 위한 용량을 추가로 활용한다. 이렇게 함으로써 실행시간이 주어진 용량을 초과하더라도, 이로 인해서 다른 태스크들이 마감시간을 만족하지 못하게 되는 도미노 현상을 막을 수 있게 된다.

본 논문에서는 태스크들의 응답시간을 개선하기 위해서 프로세서의 대역폭 일부를 공유 대역폭으로 남겨서 이것을 태스크들이 공동으로 나누어 사용하는 공유대역폭분할 (SBP: Shared Bandwidth Partitioning) 알고리즘을 제시하였다. 태스크들은 자신의 최악의 실행시간에 필요한 대역폭을 할당받는 것이 아니라 이보다 작은 적절한 대역폭만 할당받고, 실행 중에 실행시간이 길어지면 공유대역폭에서 일정부분을 할당하여 추가로 사용함으로써 마감시한을 만족하도록 한다. 태스크의 특성에 따라 마감시한을 반드시 준수해야 하는 경성 실시간 태스크에는 한 주기당 최악의 실행시간을 기준으로 프로세서 대역폭을 할당하고, 연성 실시간 태스크들에 대하여 실행시간을 초과해도 무방한 정도에 따라 적절한 대역폭을 할당한다. 이 과정에서 다른 태스크들이 조기에 완료되면서 남는 여분 시간의 슬랙도 최대한 활용한다.

III. 공유대역폭 분할 알고리즘

1. 활용가능 용량의 확보

본 논문에서 제시하는 SBP 알고리즘은 태스크들 간에 공유하여 사용할 수 있는 공유 대역폭을 확보하고, 각 태스크들에는 마감시한을 만족해야하는 정도에 따라 적절히 프로세서 대역폭을 할당한다. 공유 대역폭은 실제 실행하는 동안에 할당된 대역폭을 초과한 태스크들이 나누어 사용하게 된다. 태스크들이 할당된 시간보다 조기에 완료되면 ASR에서와 마찬가지로 슬랙으로 등록하여 다른 태스크들이 사용할 수 있게 하고, 그래도 용량이 모자라면 자신의 마감시한을 연장하여 자신의 대역폭에서 다음 주기의 인스턴스가 사용할 용량을 추가로 활용한다. 자신의 용량을 추가로 활용할 때에는 다음주기의 용량을 한꺼번에 추가하는 것이 아니라, EDF에 정책에 따라 다음에 선택될 태스크의 마감시한까지의 분량만을 먼저 추가함으로써 이 태스크가 다른 태스크들보다 먼저 조금 더 실행할 수 있도록 한다.

EDF 정책을 기반으로 스케줄링을 하면 하나의 태스크 인스턴스는 두 부분으로 나누어서 각각을 실행하는 것으

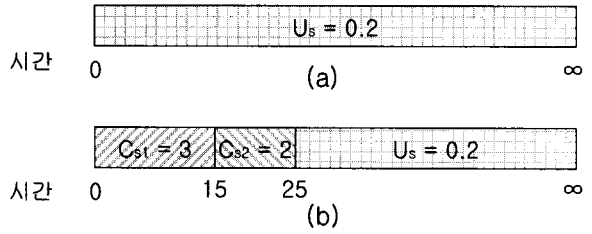

그림 1. 공유대역폭의 분할
Fig. 1. Partitioning of Shared Bandwidth.

로 처리해도 전체 실시간 태스크들에 대하여 마감시간을 만족하는 데에는 영향을 미치지 않는다.[1] 나누어진 세부 인스턴스의 할당 용량 비율은 실행구간 비율과 동일하게 한다. SBP 알고리즘은 태스크별로 꼭 필요한 만큼의 대역폭을 할당하고 남는 부분을 공유대역폭으로 남긴다. 공유대역폭의 활용률을 U_s라고하고 가상의 태스크 T_s가 이 대역폭을 사용한다고 가정하자. T_s의 주기는 무한대이다. 그림 1 (a)는 U_s = 0.2인 경우를 보여준다. 이것은 임의의 구간에 대해 프로세서의 대역폭 0.2 만큼을 여러 태스크들이 공유하여 사용할수 있음을 의미한다. 따라서 그림 1(b)와 같이 0부터 15까지의 구간이라면 15 x 0.2 = 3 단위시간만큼을 임의의 태스크가 사용할수 있고 15부터 25까지의 구간이라면 10 x 0.2 = 2 단위시간 만큼을 공유하여 활용할수 있는 것이다. 즉, 만약 T_1과 T_2가 각각 C_{s1} = 3 및 C_{s2} = 2만큼씩의 실행시간이 추가로 필요하다면, T_1은 C_{s1}/U_s, = 3/0.2 = 15만큼의 공유대역폭 구간을 활용하고 T_2는 C_{s2}/U_s, = 2/0.2 = 10만큼의 공유대역폭 구간을 활용할 수 있는 것이다.

태스크들이 실행 중에 용량이 부족하면 공유대역폭 U_s를 적절히 분할하여 추가로 사용한다. 스케줄링에 의해 선택된 태스크 T_i의 마감시한이 D_i 이고, 가상의 공유대역폭 태스크 T_s의 이미 사용된 마지막 인스턴스의 마감시한이 D_s 이었다고 가정하자. T_i가 D_s부터 D_i까지의 공유대역폭을 사용한다면, 이것은 T_s의 남은 부분에서 마감시한이 D_i이고 용량이

$$C_s = (D_i - D_s)\,U_s \tag{1}$$

인 인스턴스를 별도로 분할하고 이 용량을 T_i가 사용하는 것을 의미 한다. 그림 1에서 T_i 이전에 실행된 태스크들에 의해 공유대역폭의 용량을 3만큼 사용하였다면 T_s는 D_s=15까지의 대역폭을 사용한 상태로 된다. 이 상태에서 T_i 의 마감시한이 D_i=25라면 T_i는 자신의 용량 이외에 공유대역폭에서 2만큼의 용량을 추가로 활용할 수 있는 것이다.

태스크 T_i는 완료된 시점에 자신의 용량을 Q 만큼 남기게 되면 이것을 슬랙으로 등록한다. 용량 Q만큼에 대한 T_i의 시간 구간은 Q/U_i 이므로, 슬랙은 시작시간 begin, 끝 시간 end, 및 대역폭 U를 식(2)와 같이 계산하여 $< begin, end, U >$로 등록한다.

$$begin = D_i - \frac{Q}{U_i},\ end = D_i,\ U = U_i \tag{2}$$

즉, T_i의 현재 인스턴스를 begin을 기준으로 나누고 이 전부분은 태스크 자신의 실행에 사용되었고, 이후 부분은 다른 태스크가 활용할 수 있는 슬랙으로 남게 되는 것이다.

이렇게 남은 슬랙들은 공유대역폭을 나누어 사용하듯이 다른 태스크들이 활용할 수 있다. 태스크 T_i는 자신의 마감시한 D_i 이전 부분의 슬랙 용량을 활용할 수 있다. 만약 슬랙 구간의 끝 시간이 D_i 보다 이르면 슬랙 전부를 활용할 수 있고, 시작시간이 D_i 보다 늦은 슬랙은 활용할 수 없다. 시간 D_i 이내에 슬랙 $S_k = < begin, end, U >$의 활용가능 용량은

$$SC_k = \begin{cases} (end - begin)\,U & \text{if} \quad end \leq D_i \\ (D_i - begin)\,U & \text{if} \quad begin < D_i < end \\ 0 & \text{if} \quad begin \geq D_i \end{cases} \tag{3}$$

이다. 따라서 T_i가 자신의 남은 용량 RC_i를 포함하여 마감시한 D_i 이내에 사용할 수 있는 총 용량은

$$Q = RC_i + (D_i - D_s)\,U_s + \sum_k SC_k \tag{4}$$

이다.

2. 활용가능 용량의 사용

만약 T_i가 Q′ 만큼의 시간을 실행했다면, 활용가능 용량에서 Q′ 만큼을 차감한다. T_i를 실행할 때 최대 활용가능 용량은 식 (4)와 같이 결정되므로 Q′는 식 (4)의 총 용량보다 크지 않을 것이다. 먼저 자신의 마감시한보다 끝시간이 이른 슬랙들이 있다면 이들을 먼저 활용한다. 이것은 DPE나 BASH에서와 마찬가지로 슬랙의 끝 시간이 이른 순서대로 활용한다. 이것으로 용량이 모자라면 나머지 슬랙들을 마감시한 D_i를 기준으로 분할하여 전반부를 활용한다. 그래도 모자라는 부분은 자신의 용량을 활용하고, 마지막으로는 공유대역폭에서 D_i 이전 부분을 활용한다.

슬랙들을 활용할 때 차감할 분량이 슬랙의 크기보다

작으면 필요한 만큼만 분할하여 활용한다. 만약 슬랙 $< begin, end, U >$에서 일부분인 Q_{sc} 만큼만 차감해야 한다면 남는 슬랙의 시작 시간을 다음과 같이 증가시킨다.

$$begin \leftarrow begin + \frac{Q_{sc}}{U} \quad (5)$$

공유대역폭에서 용량 Q_s 만큼을 활용할 때도 이전에 이미 활용한 구간의 끝 시간 D_s를 다음과 같이 증가시킨다.

$$D_s \leftarrow D_s + \frac{Q_s}{U_s} \quad (6)$$

만약 T_i가 완료되고 자신에게 원래 할당된 용량에서 Q만큼이 남는다면 이 부분은 식(2)와 같이 계산된 슬랙으로 등록한다.

3. 용량의 추가 확보

태스크는 자신의 마감시한 이내에 해당하는 활용가능 용량을 전부 사용하고도 실행이 완료되지 않으면 마감시한을 늘려서 용량을 추가로 확보한다. 이렇게 되면 자신의 마감시한이 늦어지므로 자신의 우선순위가 낮아지게 된다. CBS에서는 태스크 별로 자신에게 할당용량 한도 내에서만 실행하도록 제한하고 실행시간이 부족할 경우 자신의 다음 한 주기를 위한 할당시간 만큼을 할당용량에 보충한다. 이와 동시에 마감시간을 한주기 뒤로 늘려줌으로써 EDF에 의한 스케줄링 과정에서 실행 우선순위가 낮아지고, 다른 태스크들에 영향을 주지 않으면서 완료할 수 있도록 한다. GRUB와 BASH에서도 이와 동일한 방식을 사용하고 있다.

그러나 자신의 우선순위를 다른 태스크들에 비해 높게 유지하기 위해서는 마감시한의 연장을 다음차례의 태스크의 마감시한 이내로 한정하면 되고, 그 결과로 이 태스크가 우선적으로 실행됨으로써 자신의 마감시한을 만족할 가능성이 높아지게 된다. 따라서 태스크 T_i의 마감시한은 식 (7)과 같이 연장한다. 이때 추가의 용량을 확보할 수 있기 위해서는 마감시한이 D_i보다 큰 태스크의 마감시한들 중에서 가장 이른 것으로 선택한다. 만약 이렇게 선택된 마감시한이 T_i의 다음 주기의 마감시한보다 더 늦다면 다음주기로 한정한다.

$$D_i \leftarrow \min(\min_{D_k > D_i}(D_k),\ D_i + P_i) \quad (7)$$

ASR에서도 이러한 방식으로 적용하고 있으며, 본 논문에서는 이렇게 마감시한을 연장되면 이에 맞춰서 활용가능 용량을 다시 계산하여 사용한다.

4. SBP 알고리즘

SBP의 전체 알고리즘은 다음과 같다. 각 태스크 T_i의 속성으로서 P_i (period)는 주기, U_i (utilization)는 할당된 프로세서 대역폭을 나타낸다. 만약 T_i의 매주기마다 C_i만큼의 용량을 할당한다면 $U_i = C_i/P_i$ 로 계산된다. D_i (effective deadline)는 EDF 정책을 적용할 때 사용되는 유효 마감시한, RD_i (real deadline)는 현재 인스턴스의 원래 마감시한, RC_i (remaining capacity)는 남은 용량을 나타낸다. 공유대역폭은 태스크들에 할당한 활용률을 뺀 나머지인 $U_s = 1 - \sum_i U_i$로 설정하고, D_s (deadline of shared bandwidth)는 공유대역폭의 이미 사용된 구간의 끝 시간을 나타낸다. 슬랙 S_k의 속성으로서 $BEGIN_k$ 는 슬랙 구간의 시작시간, END_k 은 슬랙 구간의 끝시간, U_k 는 슬랙의 대역폭을 나타낸다.

(1) T_i 의 j 번째 주기의 시작시점에는

$RD_i \leftarrow j \cdot P_i$;
if $(D_i < RD_i)$ // 이번 주기의 용량이 남았으면
  $RC_i \leftarrow (RD_i - D_i)U_i$; $D_i \leftarrow RD_i$
else
  $RC_i \leftarrow 0$;
insert T_i into ready task queue;
Reschedule;

(2) Reschedule 요청의 처리는

find T_i of the earliest effective deadline;
$Q \leftarrow RC_i + \max((D_i - D_s)U_s, 0) + \sum_k SC_k$
  where SC_k is as equation (3);
if $(Q > 0)$ {
  run T_i *within computation capacity of* Q;
} else {
  $D \leftarrow \min(\min_{D_k > D_i}(D_k),\ D_i + P_i)$;
  $RC_i \leftarrow (D - D_i)U_i$; $D_i \leftarrow D$;
  Reschedule;
}

(3) T_i 가 Q 시간만큼을 실행한 시점에 스케줄링 이벤트가 발생하면

// 슬랙들을 먼저 사용
$S_k \leftarrow$ *the slack of the earliest END*;

```
while (Q > 0 and S_k is not nul) {
  if (END_k ≦ D_i) {
    if (Q ≧ (END_k − BEGIN_k) U_k) {
      Q ← Q − (END_k − BEGIN_k) U_k;
      delete S_k;
    } else {
      BEGIN_k ← BEGIN_k + Q/U_k;   Q ← 0;
    }
  } else if (BEGIN_k < D_i) {
    if (Q ≧ (D_i − BEGIN_k) U_k) {
      Q ← Q − (D_i − BEGIN_k) U_k;
      BEGIN_k ← D_i;
    } else
      BEGIN_k ← BEGIN_k + Q/U_k;   Q ← 0;
  }
  S_k ← the slack of the next earliest END;
}
// 자신의 고유 대역폭 용량 사용
if (Q > 0 and RC_i > 0) {
  if (Q ≦ RC_i)
    RC_i ← RC_i − Q;  Q ← 0;
  else
    Q ← Q − RC_i;  RC_i ← 0;
}
// 공유대역폭 사용
if (Q > 0 and D_s < D_i) {
  if (Q ≧ (D_i − D_s) U_s)
    Q ← Q − (D_i − D_s) U_s;   D_s ← D_i;
  else
    D_s ← D_s + Q/U_s;   Q ← 0;
}
if (T_i is completed) {
  if (RC_i > 0)
    D_i ← D_i − RC_i/U_i,  RC_i ← 0
  if (D_i < RD_i) {
    add slack S_k such that
      BEGIN_k = D_i,   END_k = RD_i,   U_k = U_i;
    D_i ← RD_i;
  }
  delete T_i from ready task queue;
}
Reschedule;
```

Ⅳ. 성능 평가

본 논문에서 제시한 SBP는 기존의 알고리즘들에 비해서 태스크 인스턴스들의 마감시한 초과 비율을 개선하고, 시스템의 오버헤드에 영향을 미치는 문맥교환 회수 면에서도 개선된 결과를 보여주었다. 성능평가는 시뮬레이션을 통하여 기존의 GRUB, BASH, 및 ASR과 비교하였다. CBS는 [8]과 [9]에서 제시한 결과처럼 다른 것들에 비해 미흡한 결과를 보여주고 있으므로 본 논문의 성능비교에서는 제외하였다. 시뮬레이션에 적용한 값으로는 태스크들의 주기를 10부터 500까지의 범위에서 임의로 선택하고, 태스크 별 평균 활용률은 임의로 선택하되 전체 합이 1이하의 지정된 값이 되도록 하였다. 전체 시뮬레이션의 결과는 십만 시간 단위까지 실행하는 작업을 100번 반복하여 평균을 취한 것이다.

실행시간 변동 폭은 태스크 인스턴스들의 실제 실행시간이 한 주기당 미리 할당된 용량에 비하여 변화하는 정도를 나타낸다. 예를 들어서 변동폭이 30%이면 실제 실행시간이 할당된 용량 대비 70%에서 130% 까지 변화할 수 있음을 의미한다. 만약 주기가 50이고 제공된 활용률이 0.2라면, 변동폭이 30%일 경우 매 주기별 실제 실행시간은 적게는 50 x 0.2 x 70% = 7에서 많게는 50 x 0.2 x 130% = 13까지의 임의의 값을 가질 수 있음을 의미한다.

그림 2는 태스크들의 부하를 변화시켜가면서 마감시간을 초과한 인스턴스의 비율을 측정한 것으로서 SBP가 가장 우수한 결과를 보여주고 있다. GRUB, BASH, 및

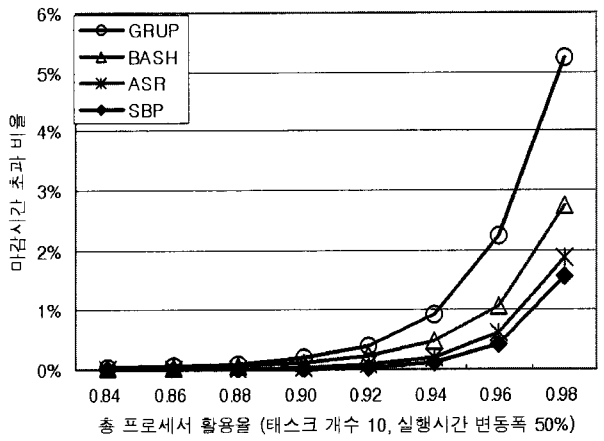

그림 2. 총 프로세서 활용률에 따른 태스크별 마감시간 초과비율

Fig. 2. Deadline Miss Ratio in Processor Total Utilization.

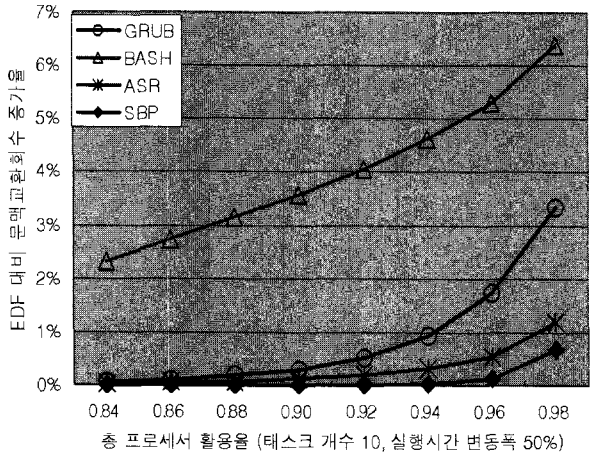

그림 3. 총 활용률에 따른 문맥교환회수 증가율
Fig. 3. Number of Context Switches in Processor Total Utilization.

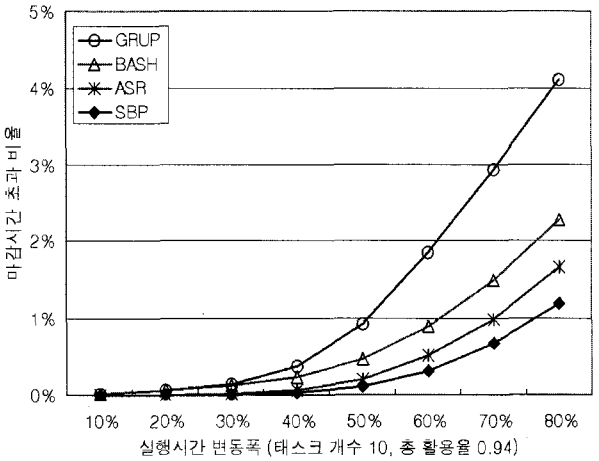

그림 4. 실행시간 변동폭에 따른 마감시간 초과비율
Fig. 4. Deadline Miss Ratio in Variance of Computation Time.

ASR은 공유대역폭 개념이 없으므로 이들과의 비교를 위해서 평균적인 총 프로세서 활용률을 적용하였다. 적용된 값은 0.84부터 0.98까지 0.02의 단위로 적용하였다. 실행시간 변동폭은 50%를 적용하였으므로 최악의 경우의 총 활용률은 1을 초과할 수도 있다. 평균적인 활용률이 0.84인 경우에도 최악의 경우의 총 활용률은 0.84 x 1.5 = 1.26까지 될 수가 있는 것이다. 따라서 GRUP, BASH, 및 ASR에는 태스크별로 할당한 활용률의 합이 1이 되도록 태스크별 할당 활용률을 조정하였다. SBP에는 태스크별로 평균적인 활용률 만큼만 할당하고 나머지는 전부 공유대역폭으로 할당하였다. 따라서 공유대역폭은 1 - 0.84 = 0.16부터 1 - 0.98 = 0.02까지 0.02의 단위로 적용한 것이 된다.

그림 3은 그림 2와 동일한 조건에서 태스크 간의 문맥교환 회수를 비교한 것이다. 문맥교환 회수는 시스템의 오버헤드와 직접적으로 관련되는 파라미터로서 많을수록 시스템 오버헤드도 늘어나는 것을 의미한다. 문맥교환 회수는 그냥 EDF를 적용할 때가 최소인 것으로 알려져 있다.[2] 그림 3에서는 최적인 EDF에 비해서 어느 정도 늘어나는지의 비율을 나타낸 것인데, SBP는 문맥교환 회수 면에서도 기존의 GRUB, BASH, 및 ASR에 비해서 우수한 결과를 보여주고 있다. 참고로, EDF만을 그대로 적용하면 문맥교환 회수 면에서는 우수하지만 하나의 태스크가 마감시한을 넘기는 경우 다른 태스크들도 밀려서 마감시한을 놓치게 되는 도미노 현상이 발생할 수 있다.

그림 4는 태스크들의 실행시간 변동폭을 변화시켜가면서 마감시간을 초과한 인스턴스의 비율을 비교한 것이다. 실행시간 변동폭이 커질수록 자연스럽게 마감시간

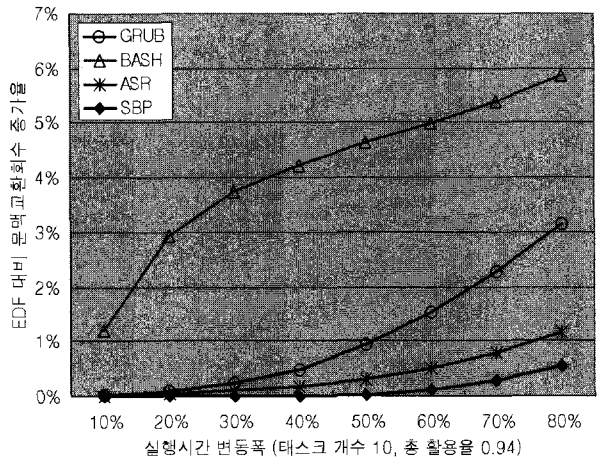

그림 5. 실행시간 변동폭에 따른 문맥교환회수 증가율
Fig. 5. Number of Context Switches in Variance of Computation Time.

초과 비율도 늘어나지만, 변동 폭에 상관없이 일관되게 SBP가 가장 우수한 결과를 보여주고 있다. 그림 5는 그림 4와 동일한 조건에서 문맥교환 회수를 비교한 것이다. 이 측면에서도 SBP가 일관되게 우수한 결과를 보여주고 있다.

그림 6과 7은 태스크들의 태스크 개수를 변화시켜가면서 마감시간을 초과한 인스턴스의 비율과 문맥교환 회수를 비교한 것이다. 태스크 개수가 늘어나면 이들 간에 남는 슬랙을 서로 공유하면서 마감시간 초과 비율이 감소하는 결과를 보인다. SBP는 태스크 개수에 상관없이 일관되게 가장 우수한 결과를 보여주고 있다.

SBP에서는 경성실시간 태스크와 연성 실시간 태스크들이 혼재해 있는 경우에도 태스크 별로 마감시한을 만

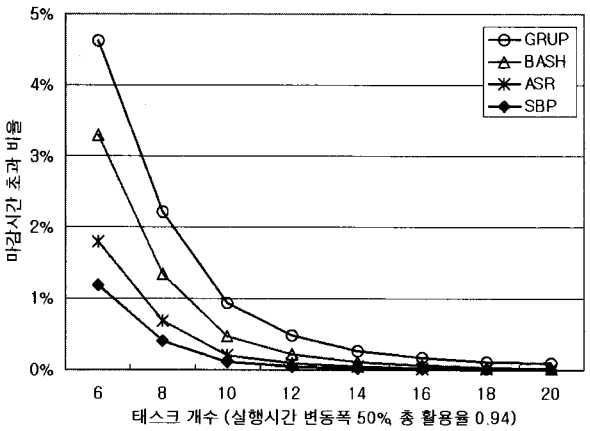

그림 6. 태스크 개수에 따른 마감시간 초과비율
Fig. 6. Deadline Miss Ratio in Number of Tasks.

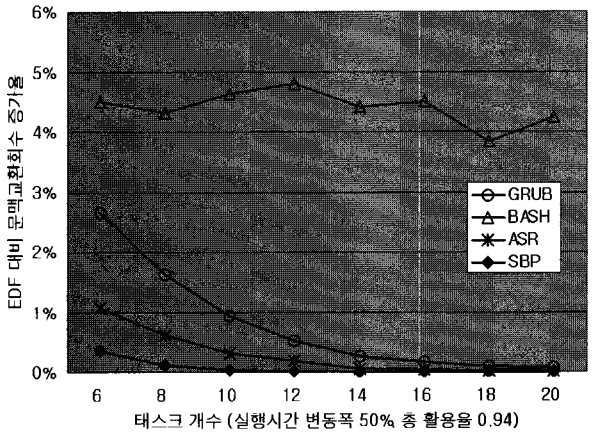

그림 7. 태스크 개수에 따른 문맥교환회수 증가율
Fig. 7. Number of Context Switches in Number of Tasks.

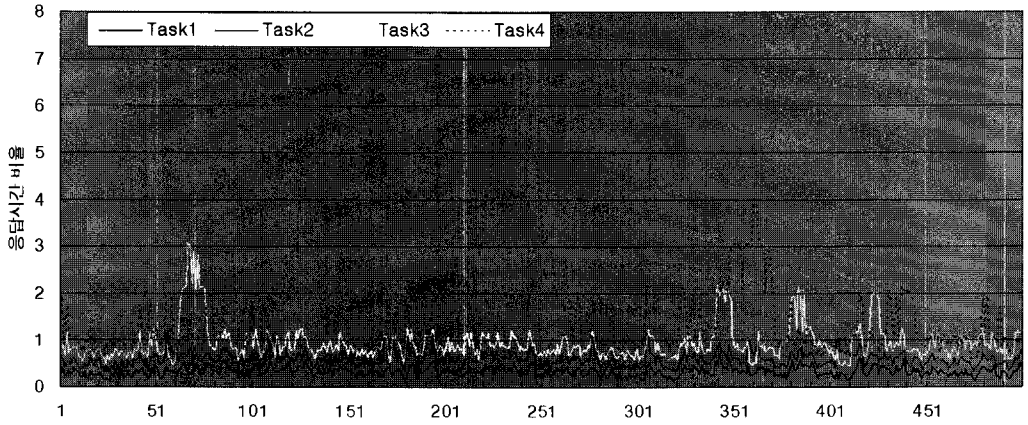

그림 8. 태스크별 경성 정도에 따른 응답시간 비교
Fig. 8. Comparison of Response Time of Tasks in Various Hardness.

족해야 하는 정도에 따라 할당시간에 차이를 둘 수 있다. 그림 8은 이러한 경우에 대한 시뮬레이션 결과로서 마감시한 대비 실제 완료된 시간의 비율을 보여주고 있다. Task1은 어떠한 경우에도 마감시한을 초과하면 안되는 경성 실시간 태스크이고 Task2, 3, 4로 갈수록 마감시한을 지켜야 하는 정도가 느슨해 진다. 즉, Task1은 최악의 실행시간에 맞춰서 프로세서 대역폭을 할당하고, 나머지는 느슨한 정도에 따라 대역폭을 적게 할당한다. 그림에서 보듯이 Task1은 항상 1 이하로서 마감시한을 만족하고 있으며, Task2, 3, 4로 갈수록 마감시한을 초과하는 빈도가 늘어나고 마감시한 대비 응답시간의 비율도 늘어나고 있는 것을 볼 수 있다.

IV. 결 론

실시간 시스템에서 응용 태스크들은 해당 시스템의 고유 목적에 따라 작성된 태스크들로서 시스템이 종료될 때까지 주기적으로 일정한 작업을 반복하는 형태로 작성된다. 이러한 태스크들은 주기 내에 실행을 완료해야 하는 조건을 만족해야 한다. 태스크의 실행 시간은 다양한 입력조건에 따라 가변적일 수 있다. 이때 최악의 경우에도 모두 마감시한을 만족해야 한다면, 각 태스크의 최악의 실행시간을 전제로 충분한 성능의 프로세서를 채용해야 할 것이다. 그러나 하드웨어 비용 절감과 에너지 소비 감소를 위해서는 보다 낮은 성능의 프로세서를 적용할 필요가 있고, 따라서 스케줄링에서는 프로세서의 용량을 최대한 활용하되 가끔씩 일부 태

스크가 마감시한을 초과해도 다른 태스크에는 영향을 미치지 않도록 해야 한다.

이러한 목적으로 본 논문에서 제시한 SBP는, 태스크들마다는 필요한 필수 대역폭만 할당하고 나머지를 공유대역폭으로 확보한 다음 태스크들이 이를 공유하여 활용할 수 있도록 함으로써, 기존의 CBS, GRUB, BASH, 및 ASR 등에 비해서 우수한 결과를 얻을 수 있었다. 태스크들의 실행시간이 가변적이므로, 할당된 시간보다 조기에 완료되면 남는 시간은 슬랙으로 등록하여 실행시간이 길어지는 다른 태스크들이 적극적으로 활용하도록 하였다. 시뮬레이션을 통해 성능을 비교 평가해본 결과, 스케줄링의 질에 해당하는 마감시한의 초과비율 면에서 기존의 알고리즘들에 비해서 일관되게 낮은 비율을 보여주었으며, 시스템의 오버헤드에 해당하는 문맥교환 회수 면에서도 우수한 것으로 나타났다.

경성 실시간 태스크와 연성 실시간 태스크들이 혼재되어 있는 시스템에서 경성 실시간 태스크들은 마감시한을 보장하면서, 연성 실시간 태스크들도 최대한 마감시한을 만족하도록 하는 스케줄링 정책도 SBP에는 쉽게 적용할 수 있다. 각 태스크들은 단순히 경성 실시간 태스크와 연성 실시간 태스크로 양분하지 않고, 마감시한을 초과해도 무방한 정도를 적절히 지정할 수 있다. 즉, 경성의 정도가 강할수록 최악의 실행시간에 가깝게 프로세서 대역폭을 할당하고 느슨할수록 적은 양의 대역폭을 할당함으로써, 일시적인 과부하가 발생하더라도 태스크의 특성에 따라 적절히 마감시한을 초과하는 비율을 조절할 수 있는 것이다. 공유대역폭은 태스크집합의 특성에 따라 0부터 1까지의 값으로 설정하게 되는데, 한 극단으로서 태스크 집합이 경성 실시간 태스크들만으로 이루어져 있을 경우에는 공유대역폭을 0으로 설정하면 된다.

참 고 문 헌

[1] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 2nd ed., Springer, 2005.

[2] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of ACM*, vol. 20, no. 1, pp. 40-61, 1973.

[3] J.W.S Liu, K. Lin, W. Shih, A. Yu, C. Chung, J. Yao, and M. Zhao, "Algorithms for Scheduling Imprecise Computations", *IEEE Computer*, 24(5) pp. 587-68, May 1991.

[4] Y. Chu and A. Burns, "Supporting Deliverable Real-Time AI Systems: A Fixed Priority Scheduling Approach", *19th Euromicro Conf. on Real-Time Systems*, pp. 259-268, July 2007.

[5] C. W. Mercer, S. Savage, and H. Tokuda, "Processor Capacity Reserves: Operating System Support for Multimedia Applications," *Proc. 1st Int. Conference on Multimedia Computing and Systems*, IEEE, 1994.

[6] L. Abeni and G. Buttazo, "Resource Reservations in Dynamic Real-Time Systems," *Real-Time Systems*, vol. 27, no. 2, pp. 123-167, Kluwer Academic Publishers, 2004.

[7] G. Lipari and S. Baruah, "Greedy Reclamation of Unused Bandwidth in Constant Bandwidth Servers," *Proc. of IEEE 12th Euromicro Conference on Real-Time Systems*, pp.234-241, June 2000.

[8] M. Caccamo, G. Buttazo, and D. Thomas, "Efficient Reclaiming in Reservation-Based Real-Time Systems with Variable Execution Times,", *IEEE Trans. on Computers*, vol. 54, no. 2, pp. 198-213, Feb. 2005.

[9] 김용석, "연성 실시간 태스크들의 스케줄링을 위한 적극적인 슬랙 재활용", *대한전자공학회 논문지*, 제43권 CI편 제2호, pp. 12-20, 2006년 3월

[10] M. Spurri and G. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems," *Journal of Real-Time Systems*, 10(2), 1996.

저 자 소 개

김 용 석(정회원)

1984년 서울대학교 해양학과 학사 졸업.

1986년 KAIST 전기및전자공학과 석사졸업.

1989년 KAIST 전기및전자공학과 박사 졸업.

1990년~1995년 한국생산기술연구원 및 전자부품연구원 선임연구원

1995년~현재 강원대학교 컴퓨터학부 교수

<주관심분야 : 운영체제, 임베디드시스템, 실시간 시스템>