□특집□

실시간 스케줄링

김 성 관† 하 란††

◈ 목 차 ◈

요 약

실시간 시스템은 범용 시스템과는 달리 계산 결과의 정확성뿐만 아니라 계산이 종료되는 시점에 의해 그 가치가 결정되는 시스템을 말한다. 따라서, 실시간 태스크는 시간적 제한 조건으로서 마감 시한(deadline)을 가지고 있으며, 실시간 스케줄링 방법은 범용 시스템에서 사용되는 스케줄링 방법과는 달리 태스크가 마감 시한 내에 종료될 수 있음을 보장해 주어야 한다. 또한, 실시간 스케줄링 방법은 새로운 태스크의 실행을 허가하기 전에 새로운 태스크 집합의 스케줄 가능성(schedulability)을 분석함으로써 시스템 전체의 안전성을 유지할 수 있어야 한다. 실시간 스케줄링 방법은 크게 시간 구동형(time driven) 방식과 우선 순위 기반의 이벤트 구동형(event driven) 방식으로 나누어지는데, 본 논문에서는 주로 우선 순위 기반의 스케줄링 방법에 대해서 살펴본다. 또한, 비주기적인 태스크를 우선 순위 기반 스케줄링 방법에 적용하기 위한 여러 가지 기법들에 대해서도 살펴본다.

† 정회원 : 서울대학교 컴퓨터공학과 박사과정
†† 정회원 : 홍익대학교 컴퓨터공학과 조교수

본 논문은 한국과학재단 핵심 전문 연구 (961-0908-049-2)의 부분적인 지원을 받았음.

1. 서 론

실시간 시스템은 범용 시스템과는 달리 계산 결과의 정확성뿐만 아니라 계산이 종료되는 시점에 의해 그 가치가 결정되는 시스템을 말한다. 따라서, 실시간 시스템에서 수행되는 태스크들은 시간적 제한 조건을 갖는다. 이러한 시간 제한 조건으로 가장 널리 사용되고 있는 것은 마감 시한(deadline)이다. 실시간 태스크는 마감 시한이 지켜지지 못할 경우 실행 결과에 대한 가치를 상실하게 되는데, 그 정도에 따라서 경성 실시간(hard real-time) 태스크와 연성 실시간(soft real-time) 태스크로 구분된다. 경성 실시간 태스크는 마감 시한이 지켜지지 못할 경우 실행 결과의 가치가 영 또는 음의 값을 갖게 된다. 반면 연성 실시간 태스크의 경우에는 그 가치가 마감 시한을 넘어서도 점진적으로 감소하는 특징이 있다. 실시간 시스템의 운영체제는 자신이 관리하는 경성 실시간 태스크가 마감 시한 내에 실행이 종료될 수 있도록 보장해 주어야 하며 연성 실시간 태스크에 대해서는 가능한 한 마감 시한 내에 종료될 수 있도록 최대한의 노력(best effort)을 해야 한다. 따라서, 일반적인 범용 운영체제에서 사용되던 태스크 스케줄링 정책은 실시간 스케줄링에 적합하지 않

다. 범용 시스템에서는 프로세서 이용률(utilization)을 높이는 방향으로 스케줄링을 수행할 뿐 개별 태스크의 마감 시한에는 상관하지 않기 때문이다. 또한, 범용 시스템에서는 새로운 태스크가 실행을 요구해 왔을 때 그 실행을 거부하지 않는데, 이 또한 실시간 시스템에서는 적합하지 않다. 새로운 태스크로 인해 시스템이 과부하 상태에 들 경우 그 새로운 태스크뿐만 아니라 기존의 태스크들 역시 마감 시한을 지키지 못할 수 있기 때문이다. 결과적으로 실시간 시스템에서는 범용 스케줄링 방법과는 다른 스케줄링 방법이 필요하며, 또한 새로운 태스크의 실행을 허가하는 것이 시스템 전체의 안전성을 손상시키지 않는지를 검사하기 위한 방법이 필요하다. 관련 문헌에서는 후자의 검사 방법을 보통 스케줄 가능성 분석(schedulability analysis) 방법이라고 한다. 그리고 주어진 스케줄링 방법으로 스케줄 가능한 모든 태스크 집합에 대해서 "예"라고 지시할 수 있고 그렇지 않은 태스크 집합에 대해서는 "아니오"라고 지시할 수 있는 스케줄 가능성 분석 방법이 있다면 그러한 방법을 정확한(exact) 스케줄 가능성 분석 방법이라 한다.

실시간 스케줄링 방법을 자세히 논의하기에 앞서 태스크 모델에 대해서 살펴 볼 필요가 있다. 가장 널리 사용되고 있는 실시간 태스크 모델은 주기적 태스크 시스템(Periodic Task System, PTS) 모델이다. 이 모델은 실시간 시스템의 태스크들이 동일한 작업을 반복적으로 수행한다는 관찰에 근거한다. PTS 모델에서는 태스크 집합 T 가 다음과 같이 기술된다.

$$T = \{\tau_i = (T_i, C_i, I_i, D_i) \mid 1 \le i \le N\}$$

여기서 매개변수 T_i, C_i, I_i, D_i 는 각각 태스크 τ_i 의 주기, 최악 실행시간(Worst Case Execution Time), 위상(phase), 마감 시한을 가리키는데, 위상 I_i 는 태스크 τ_i 의 첫 번째 job[1]의 릴리스 시점(release time)[2]이 기준 시점 $t=0$ 에 대해서 얼마만큼 떨어져 있는지를 의미한다. 실시간 시스템에는 위와 같이 기술될 수 있는 주기적인 태스크가 있는 반면 비주기적으로 도착하는 태스크도 있다. 비주기적인(aperiodic) 태스크는 마감 시한의 성격에 따라서 경성 비주기적 태스크와 연성 비주기적 태스크로 구분되는데 두 가지 모두 (A_i, C_i, D_i) 의 형태로 기술될 수 있다. 여기서 A_i 는 비주기적 태스크의 도착 시점(arrival time)을 말한다. 경성 비주기적 태스크가 연성 비주기적 태스크와 다른 점은 스케줄 가능성 분석을 통한 실행 허가 검사(acceptance test)를 통과하지 못할 경우 실행을 거부당할 수 있다는 점이다.

태스크 집합이 주어지고 태스크 모델의 각 매개변수가 결정되고 나면 어떤 스케줄링 방법을 선택할 것인가 하는 문제가 남는다. 여러 가지 실시간 스케줄링 방법들은 크게 시간 구동형(time driven)과 이벤트 구동형(event driven) 방식으로 나누어 볼 수 있는데, 이는 스케줄링 시점이 결정되는 방식에 따라 구분된 것이다. 시간 구동형의 경우에는 스케줄링 시점이 주기적인 클럭 인터럽트에 의해서 결정되거나, 또는 예약된 시점에 걸리는 비주기적인 클럭 인터럽트에 의해서 결정된다. 각각의 스케줄링 시점에서 스케줄러는 다음 시간 구간 동안 실행될 태스크를 결정하는데, 보통 정적으로 구해진 스케줄을 테이블에 저장해 두고 이를 참조하여 결정한다. 스케줄이 정적으로 결정되어 있기 때문에 이러한 스케줄링 방법을 오프라인(off-line) 스케줄링이라고도 한다. 시간 구동

1) 주기적으로 실행되는 각각의 태스크 인스턴스(instance)를 job이라 한다.

2) 릴리스 시점이란 job이 실행 가능하게 되는 시점을 말한다.

형 스케줄링 방식의 전형적인 예로 순환 실행 구조(cyclic executive) 방법을 들 수 있다[2]. 순환 실행 구조 스케줄링 방법에서는 정적으로 정해진 실행 순서에 따라서 태스크들을 차례로 실행한다. 이 때, 각 태스크의 주기 T_i 의 최소공배수를 M_C 라고 하면 태스크 스케줄은 M_C 를 주기로 하는 반복되는 형태로 나타난다. 따라서, 스케줄 테이블에는 M_C 만큼의 시간 구간에 대한 스케줄만 저장해 두면 된다. 시간 구동형 스케줄링 방식은 스케줄링 행태가 미리 결정되어 있기 때문에 시스템에 대한 예측 가능성이 매우 높다는 장점이 있다. 그러나, 같은 이유로 유연성이 부족하기 때문에 태스크 집합이 유동적인 시스템에는 적용하기 어렵다.

이벤트 구동형 스케줄링 방식에서는 스케줄링 시점이 각각의 이벤트가 발생하는 시점과 일치한다. 여기서 말하는 이벤트란 태스크가 실행되기 위해 릴리스되거나 또는 실행을 완료하는 두 가지 상황을 의미한다. 그리고, 각각의 이벤트가 발생한 시점에서 다음에 실행될 태스크를 결정하는데, 이 경우 우선 순위(priority)가 가장 높은 태스크가 선택된다. 이런 이유로 이벤트 구동형 스케줄링 방식은 우선 순위 기반 스케줄링 방식으로 더 많이 불린다. 우선 순위 기반 스케줄링 방식은 태스크의 우선 순위가 고정되어 있는 고정 우선 순위 기반 스케줄링 방식과 태스크 인스턴스마다 서로 다른 우선 순위가 동적으로 부여될 수 있는 동적 우선 순위 기반 스케줄링 방식으로 구분된다. 한편, 실시간 스케줄링 방식은 범용 스케줄링 방식에서와 마찬가지로 선점(preemption) 가능한 방식과 불가능한 방식으로 구분되는데, 대부분의 우선 순위 기반 스케줄링 방법은 선점형 방식을 따른다.

본 논문에서는 비록 결정적인(deterministic) 성질이 순환 실행 구조 스케줄링 방식보다는 떨어지지만 유연성 측면에서 장점이 있는 우선 순위 기반 실시간 스케줄링 방법에 초점을 맞추어 살펴본다. 또한, 비주기적인 태스크를 우선 순위 기반 스케줄링 방법에 의해 스케줄링 하는 경우에 대해서도 살펴본다. 본 논문에서는 논의의 단순화를 위해 단일 프로세서에 대한 실시간 스케줄링 문제만을 다루며, 태스크간에 프로세서 외에 공유된 자원은 없다고 가정함으로써 동기화(synchronization) 문제는 다루지 않는다.

본 논문은 다음과 같은 순서로 구성되어 있다. 우선 2장에서 고정 우선 순위 기반 스케줄링 방법에 대해서 살펴보고, 3장에서 동적 우선 순위 기반 스케줄링 방법에 대해서 살펴본다. 그리고, 4장에서 비주기적 태스크를 우선 순위 기반 스케줄링 방식에서 어떻게 다루고 있는지 고찰한다. 5장에서 그 외의 실시간 스케줄링 관련 연구를 언급한 다음 6장에서 본 논문을 끝맺는다.

2. 고정 우선 순위 기반 스케줄링

고정 우선 순위 기반 스케줄링으로서 가장 널리 사용되고 있는 방법은 RM(Rate Monotonic) 스케줄링 기법으로 Liu와 Layland에 의해서 처음으로 제안되었다[14]. 이들은 우선 태스크 모델로서 PTS 모델을 가정하고 있으며 상대적인 마감 시한[3]이 주기와 같음($D_i = T_i$)을 가정하였다. RM은 각 태스크 주기의 역수인 빈도율(rate)이 높을수록, 즉 주기가 짧을수록 더 높은 우선 순위를 부여하는 방식이다. 따라서, 각 태스크의 주기가 주어지면 태스크들간의 우선 순위는 정적으로 결정될 수 있으며 이 우선 순위는 시스템이 수행되는 동안 고정된다. 그리고, RM은 선점형이기 때

3) 상대적인 마감 시한은 (절대적인 마감 시한-릴리스 시점) 으로 구해진다.

문에 어떤 태스크가 실행 중일 때 더 높은 우선 순위를 가지는 태스크가 도착하면 바로 선점이 발생한다. Liu와 Layland는 RM이 고정 우선 순위 기반 선점형 스케줄링(fixed priority based preemptive scheduling) 방식 하에서는 최적의 스케줄링 방법임을 보였다. 이는 고정 우선 순위 기반 선점형 스케줄링 방식 하에서 타당한(feasible)[4] 스케줄이 존재하는 태스크 집합은 RM에 의해서 항상 스케줄 가능하며, 동시에 RM에 의해서 스케줄 가능하지 못한 태스크 집합은 어떠한 고정 우선 순위 기반 선점형 스케줄링 방법을 사용한다고 해도 타당한 스케줄을 찾을 수 없음을 의미한다.

Liu와 Layland는 RM 스케줄링 기법을 위하여 다음과 같은 스케줄 가능성 분석 조건을 제시하였는데, 전체 프로세서 이용률(total utilization) U 를 이용하고 있다.

$$U=\sum_{i=1}^{N} U_i=\sum_{i=1}^{N}\frac{C_i}{T_i}\le N(2^{1/N}-1) \quad (1)$$

이 식에서 $N(2^{1/N}-1)$ 값은 태스크의 개수 N 이 증가함에 따라 점점 감소하여 0.69에 수렴한다. 이는 전체 프로세서 이용률 U 의 값이 0.69보다 작은 태스크 집합은 RM 스케줄링 기법으로 항상 스케줄 가능함을 의미한다. 그러나, 식 (1)에 의한 스케줄 가능성 분석 방법은 정확한 것이 아니다. 즉, 식 (1)에 나타나 있는 조건은 태스크 집합이 스케줄 가능하기 위한 충분조건일 뿐, 필요조건은 아니다. 따라서, 이 조건식을 만족시키지 못하는 태스크 집합이라 할지라도 RM 스케줄링 기법에 의해서 타당한 스케줄이 생성될 수 있다. 비록 식 (1)에 의한 스케줄 가능성 분석 방법은 정확하지는 않지만 태스크의 개수에만 의존하고 있다는 단순함이 그 장점이 된다.

4) 모든 태스크의 마감 시한이 만족되는 스케줄을 타당한 스케줄이라고 한다.

Lehoczky 등은 RM 스케줄링 기법에 대한 식 (1)에 의한 스케줄 가능성 분석 방법의 부정확성을 보완하여 스케줄 가능성 검사를 위한 필요충분조건을 제시하였다[13].

$$\forall i=1,2\ldots,N,\ \min_{\{0<t\le T_i\}}\sum_{j=1}^{i}\frac{C_j}{t}\cdot\left\lceil\frac{t}{T_j}\right\rceil\le 1 \quad (2)$$

이 식에서 태스크 첨자 $1,2,\ldots,N$ 은 우선 순위 순서와 같은데, 태스크 τ_1 이 가장 높은 우선 순위를 갖는다. 이 식의 기본 개념은 최악의 경우에도 태스크 τ_i 가 자신의 주기 T_i 내에서 C_i 만큼의 실행시간을 확보할 수 있는지를 검사하는 것이다. 최악의 경우를 만들기 위해 임계 시점(critical instant)[14]에서의 태스크 실행을 가정하는데, 임계 시점에서는 모든 태스크가 $t=0$ 에서 동시에 릴리스된다. 식 (2)는 이러한 임계 시점에서 릴리스된 태스크 τ_i 가 자신보다 우선 순위가 높은 태스크 인스턴스의 실행시간을 모두 고려한 뒤에도 주기 T_i 내에서 자신의 실행시간 C_i 를 확보할 수 있는지 검사한다.

한편, Joseph 등[9]과 Audsley 등[1]은 또 다른 접근 방법을 사용하여 스케줄 가능성 분석을 위한 필요충분조건을 제시하였다. 이들의 기법에서는 태스크의 최악 반응 시간(worst case response time)을 계산하여 태스크의 스케줄 가능성을 분석한다.

$$r_i^{n+1}=C_i+\sum_{j=1}^{i-1}\left\lceil\frac{r_i^n}{T_j}\right\rceil\cdot C_j,\quad n=0,1,2,\ldots \quad (3)$$

이 식에서는 태스크 τ_i 의 최악 반응 시간 r_i 를 구하기 위하여 자신의 실행시간 C_i 와 함께 $[0,r_i]$ 구간 내에서 릴리스되는 자신보다 우선 순위가 높은 태스크 인스턴스들의 실행시간

C_j 의 합을 더한다. r_i 는 반복적인 계산을 통해서 구해지는데, 최초 $r_i^0 = C_i$ 로 주어지며 계산은 $r_i^{n+1} = r_i^n$ 으로 수렴될 때까지 반복된다. r_i 의 값이 발산하는 경우에는 프로세서 이용률이 1 보다 크다는 의미이므로 태스크 τ_i 는 스케줄될 수 없음을 알 수 있다. r_i 의 값이 수렴하는 경우에도 그 값이 태스크의 마감 시한보다 크다면 태스크 τ_i 는 역시 스케줄될 수 없다.

지금까지 살펴본 스케줄 가능성 분석 방법들은 각각의 태스크 τ_i 에 대해서 $D_i \leq T_i$ 를 가정한 것이다. Tindell 등은 이러한 가정을 제거하여 임의의 마감 시한을 갖는 태스크들에 대해서도 최악 반응 시간의 계산을 통하여 스케줄 가능성을 분석할 수 있는 필요충분조건을 제시하였다[20]. Tindell 등은 또한 릴리스 지터(release jitter)[5] 가 존재하는 경우를 고려하여 스케줄 가능성 분석식을 확장하였다[20].

한편, 상대적인 마감 시한 값을 기준으로 하는 정적 우선 순위 기반 스케줄링 방법으로서 DM (Deadline Monotonic) 스케줄링 기법이 있다[1]. 이 기법에서는 태스크 τ_i 의 상대적인 마감 시한 D_i 가 작을수록 높은 우선 순위를 부여한다. $D_i = T_i$ 인 경우에 DM 기법과 RM 기법은 완전히 동일하다. $D_i < T_i$ 인 경우에는 DM 기법이 최적의 고정 우선 순위 기반 선점형 스케줄링 방법이 된다.

3. 동적 우선 순위 기반 스케줄링

동적 우선 순위 기반 스케줄링 방법 중에서 가장 널리 알려져 있는 것은 EDF(Earliest Deadline First) 스케줄링 기법으로 Liu와 Layland에 의해서 처음으로 제시되었다[14]. EDF 기법에서는 새로운 job이 도착할 때마다 현재 실행 중인 job들과 새로운 job의 마감 시한을 서로 비교하여 새로운 job의 우선 순위를 결정한다. 이 때 마감 시한이 빠를수록 더 높은 우선 순위를 부여받게 된다. Liu와 Layland는 EDF 기법에 대해서도 스케줄 가능성 분석을 위한 조건을 제시하였는데, RM 기법에서처럼 전체 프로세서 이용률 U 를 이용하고 있다.

$$U = \sum_{i=1}^{N} U_i = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1 \qquad (4)$$

이 조건은 필요충분조건으로서 정확한 스케줄 가능성 분석 방법을 제공한다. 이는 또한 EDF 기법이 우선 순위 기반 선점형 스케줄링 방식 중에서는 최적의 스케줄링 방법임을 의미한다. 즉, EDF 기법에 의해서 스케줄될 수 없는 태스크 집합은 다른 어떤 우선 순위 기반 스케줄링 방법을 사용하더라도 스케줄될 수 없다.

한편, EDF 기법과 유사한 동적 우선 순위 기반 스케줄링 기법으로 MLF(Minimum Laxity First) 스케줄링 기법이 있다. 이 기법에서는 어떤 시점 t 에서 현재 실행 중인 각각의 job에 대해서 유휴시간(laxity, slack)을 다음과 같이 구한다.

$$\text{laxity} = D_{i,k} - t - (C_i - C_{consumed}) \qquad (5)$$

이 식에서 $C_{consumed}$ 는 시점 t 까지 태스크 τ_i 의 k 번째 job $\tau_{i,k}$ 가 소비한 시간이며, $D_{i,k}$ 는 $\tau_{i,k}$ 의 마감 시한이다. 식 (5)에 의해서 구해진 유휴시간이 적게 남아 있을수록 job은 더 높은 우선 순위를 부여받게 된다. MLF 기법도 EDF 기법과 같이 최적의 우선 순위 기반 선점형 스케줄링 방법이다. 그러나, $C_{consumed}$ 값

5) 어떤 job의 릴리스 지터는 (릴리스 시점 - 도착 시점) 으로 정의된다.

을 실행 중에 구하기가 어렵고 식 (5)의 계산 작업을 연속적인 t 에 대해서 수행하기 어렵다는 문제점이 있다.

EDF, MLF 기법과 같은 동적 우선 순위 기반 스케줄링 방법은 스케줄링 성능 면에서 정적 우선 순위 기반 스케줄링 방법보다 우수하다. 그러나, 시스템에 대한 예측성 측면에서는 정적 우선 순위 기반 스케줄링 방법보다 못하다. 예를 들어 RM 기법에 의해서 스케줄되는 시스템에서는 높은 우선 순위의 태스크가 언제 실행이 시작되어 언제 종료될 지에 대한 정확한 예측이 어느 정도 가능하다. 이는 태스크의 우선 순위가 정적으로 고정되어 있기 때문이다. 그러나, EDF 기법에 의하면 중요도가 높은 태스크라 할지라도 그 태스크의 어떤 job은 낮은 우선 순위를 동적으로 부여받을 수 있기 때문에 태스크의 실행 행태에 대한 예측을 정확하게 하기 어렵다.

4. 비주기적 실시간 태스크에 대한 스케줄링

주기적 센서 입력에 따른 제어 작업 등이 주로 주기적 태스크로 모델링 되는데 비해 사용자 입력에 대한 처리 작업이나 에러 복구 작업 등이 주로 비주기적 태스크로 모델링 된다. 비주기적 태스크는 (A_i, C_i, D_i) 의 세 가지 매개변수에 의해서 기술될 수 있는데, 이들은 각각 도착 시간, 최악 실행시간, 마감 시한을 의미한다[6]. 경성 비주기적 태스크는 마감 시한이 반드시 지켜져야 한다. 따라서, 경성 비주기적 태스크의 경우에는 실행하기에 앞서 스케줄 가능성 분석을 실시하여 실행 허가 여부를 결정한다. 반면, 연성 비주기적 태스크의 경우에는 마감 시한이 반드시 지켜져야 하는 것은 아니며, 스케줄러는 다른 주기적 태스크와 경성 비주기적 태스크가 마감 시한 내에 종료될 수 있는 범위 내에서 최대한의 프로세서 시간을 제공함으로써 연성 비주기적 태스크의 반응 시간을 최소화시킨다.

비주기적 태스크를 스케줄링 하기 위한 간단한 방법으로 폴링 서버 기법이 있다. 이 기법에서는 비주기적 태스크의 실행을 전담하기 위한 폴링 서버를 두고 스케줄러는 폴링 서버를 하나의 주기적 태스크처럼 스케줄링 한다. 따라서, 비주기적 태스크가 도착하면 폴링 서버가 실행되는 동안 서비스를 받을 수 있다. 그러나, 이 방법에 의하면 도착한 비주기적 태스크가 없을 경우 폴링 서버에게 할당된 시간이 낭비되는 결과가 초래된다. Lehoczky 등은 폴링 서버의 문제점을 극복하기 위하여 Priority Exchange Server와 Deferrable Server 알고리즘을 제안하였다[12]. 또한 Sprunt 등은 Deferrable Server 알고리즘과 유사하지만 이 알고리즘의 단점을 보완한 Sporadic Server 알고리즘을 제안하였다[17]. 이 알고리즘들은 모두 도착한 비주기적 태스크가 없는 경우에 서버의 대역폭(bandwidth)이 낭비되는 것을 방지해 주기 때문에 대역폭 보존 서버(bandwidth preserving server) 알고리즘으로 불린다. 그러나, 이러한 대역폭 보존 서버 알고리즘들은 주기적인 태스크가 실행을 일찍 종료하게 됨으로써 생기게 되는 추가적인 유휴 시간을 활용하지는 못하였다. 이를 개선하여 Extended Priority Exchange Server 알고리즘이 제시되었는데[16], 이 알고리즘의 기본 개념은 다음에 설명할 유휴 시간 획득 기법과 유사하다.

Lehoczky와 Thuel은 고정 우선 순위 스케줄링 기법 하에서 연성 비주기적 태스크 스케줄링을 위한 유휴 시간 획득(slack stealing) 기법을 제안하였고[11], 다시 이를 확장하여 경성 비주기적

6) 연성 비주기적 태스크의 경우에는 C_i 가 알려지지 않을 수도 있다.

태스크 스케줄링을 위한 기법을 제안하였다[19]. 유휴 시간 획득 기법의 기본 개념은 주기적인 태스크의 마감 시한이 만족될 수 있는 범위 안에서 최대의 유휴 시간을 확보하여 이를 비주기적인 태스크의 실행을 위해 사용한다는 데 있다. 시간 구간 $[s,t]$ 동안 비주기적인 태스크의 실행을 위해 사용할 수 있는 최대의 유휴 시간 $A^{\star}(s,t)$ 는 다음과 같이 구할 수 있다.

$$A^{\star}(s,t) = \min_{\{1\le i\le N\}} A_i(s,t)$$
$$= \min_{\{1\le i\le N\}} (A_{i,j} - A(s) - I_i(s)), \quad F_{i,j-1}\le t\le F_{i,j}, \quad j\ge 1 \tag{6}$$

$F_{i,j}$ 는 job $\tau_{i,j}$ 의 실행 종료 시점이며, $A_{i,j}$ 는 시간 구간 $[0,F_{i,j}]$ 에 대해 job $\tau_{i,j}$ 의 마감 시한이 지켜지는 범위 내에서 우선 순위가 i 또는 그보다 높은 비주기적 태스크의 처리를 위해 사용할 수 있는 최대 유휴 시간으로 다음 식을 만족하는 최대값이다.

$$\min_{\{0\le t\le D_{i,j}\}}\{(A_{i,j}+P_i(t))/t\} = 1$$

여기서 $P_i(t)$ 는 우선 순위가 i 또는 그보다 높은 주기적 태스크의 처리에 필요한 시간으로 $P_i(t) = jC_i + \sum_{k=1}^{i-1}(\lceil t/T_k\rceil \cdot C_k)$ 로 주어진다. 식 (6)에서 $A(s)$ 는 시간 구간 $[0,s]$ 동안 비주기적 태스크의 처리를 위해 이미 사용된 시간이며, $I_i(s)$ 는 시간 구간 $[0,s]$ 동안 우선 순위가 i 보다 낮은 주기적인 태스크의 처리를 위해 사용된 시간과 프로세서가 유휴(idle) 상태에 있었던 시간을 합한 것이다.

연성 비주기적 태스크는 $A^{\star}(s,t)$ 시간 동안 최우선 순위로 스케줄링 된다[11]. 이는 그렇게 해도 주기적 태스크가 마감 시한을 만족시키는 데 아무런 문제도 주지 않을 뿐만 아니라 연성 비주기적 태스크의 반응 시간을 단축시킬 수 있기 때문이다. 그리고, $A^{\star}(s,t)$ 시간이 다 사용되고 나면 유휴 시간이 다시 생길 때까지 기다리게 된다. 경성 비주기적 태스크를 스케줄링 하는 경우에는 우선 $A^{\star}(s,t)$ 로 주어지는 유휴 시간 안에 스케줄이 가능한지 분석한다. 스케줄이 가능하면 실행을 허가하는데 경성 비주기적 태스크의 경우 최우선 순위로 스케줄링 하여 반응 시간을 빠르게 할 필요가 없으므로 가능한 한 낮은 우선 순위를 부여하여 다른 주기적 태스크의 실행이 우선되도록 한다[19].

Lehoczky와 Thuel의 유휴 시간 획득 기법은 스케줄러의 계산 부담을 줄이고자 $A_{i,j}$ 값 등을 하이퍼피리어드[7]에 대해서 오프라인으로 계산하여 테이블에 저장해 둔 다음 실행시에 사용하기 때문에 정적 유휴 시간 획득 기법으로 볼 수 있다. 이에 Davis 등은 동적으로 적용할 수 있는 유휴 시간 획득 기법을 제시하였다[5].

한편, Chetto 등[4]과 Schwan 등[15]은 동적 우선 순위 기반 스케줄링 기법에 기초하여 비주기적 태스크를 스케줄링 하기 위한 방법을 연구하였다. 이들은 모두 EDF 기법에 기반하고 있는데, Chetto 등이 제안한 알고리즘에서는 주기적 태스크가 모두 임계 시점에서 실행되며 그들의 주기가 마감 시한과 동일함을 가정하고 있다. 또한, 비주기적 태스크에 대한 실행 요청이 그 전에 실행 중이던 비주기적 태스크들이 모두 실행을 종료한 시점에서만 이루어진다고 가정하였다[4]. Schwan 등은 이러한 제한 조건이 제거된 알고리즘을 제시하였는데, 태스크 도착시 이루어지는 스케줄 가능성 분석 방법의 복잡도는 주기적 태스크의 개수를 N 이라 할 때 Chetto 등의 알고리즘이 $O(N)$ 이었던 것에 비해 $O(N\log N)$ 이다.

7) hyperperiod. 주기적 태스크들의 주기의 최소 공배수가 되는 시간 구간.

5. 기타 연구

지금까지 살펴 본 스케줄링 방법들은 주로 PTS 모델에 근거한 것으로 주기적인 경성 실시간 태스크가 주 스케줄링 대상이 되며 여기에 비주기적인 경성 또는 연성 실시간 태스크를 추가적으로 고려하였다. 그러나 멀티미디어 처리와 관련된 태스크들은 PTS 모델을 적용하기에 부적합한 면이 있어서 RM 또는 EDF 등의 스케줄링 기법을 그대로 사용하기 어렵다. 구체적으로, 멀티미디어 처리와 관련된 태스크, 예를 들어 화상 회의 시스템을 구동하는 태스크의 경우 주어진 마감 시한을 지키는 것도 중요하지만, 지터를 최소화하는 것 역시 중요하기 때문이다. RM 또는 EDF의 경우 정상적인 스케줄링의 경우에도 지터가 최대 $2T_i - C_i$ 만큼 발생할 수 있어서 QoS(Quality of Service)를 중요시하는 멀티미디어 서비스에 부적합할 수 있다. 이러한 문제를 해결하기 위한 스케줄링 방법으로는 Han 등이 제안한 DCM (Distance Constraint Monotonic) 스케줄링 기법이 있다[6,7]. 이 기법에서는 PTS 모델을 적용하는 대신 job들간의 거리(distance)를 제한 조건으로 하는 DCTS(Distance Coustraint Task System) 모델을 적용하여 DCM 스케줄링을 수행하는데 핀휠(Pinwheel) 스케줄링 기법으로도 불린다. Han 등은 DCM 스케줄링 기법에 대한 스케줄 가능성 분석 방법을 제시하였는데, 그들이 제시한 충분 조건은 거리 제한 조건을 주기로 바꿀 경우 식 (1)에 나타나 있는 RM에 대한 Liu와 Layland의 충분 조건과 동일하다[6].

멀티미디어 서비스를 위한 또 다른 스케줄링 방법으로 fair queuing에 기반한 스케줄링 기법이 있다. 원래 fair queuing은 네트워크 스위치에서의 패킷 스케줄링을 위하여 제시된 것으로, 전체 서비스율을 공평하게 배분하여 각 세션이 요청한 서비스율을 보장하는 것을 주목적으로 한다. Stoica 등은 이러한 개념을 프로세서 스케줄링 문제에 적용하여 태스크가 요청한 서비스율을 보장할 수 있는 스케줄링 기법을 제시하였는데, 발생 가능한 지터의 크기가 프로세서 시간의 할당 단위(퀀텀, 슬롯) 하나로 한정될 수 있음을 보였다[18].

한편, 실제의 시스템 수행 환경을 고려하여 인터럽트 처리 시간과 스케줄러의 큐잉 지연 시간, 그리고 스케줄링 과정에서 발생하는 문맥 교환(context switch) 비용 등을 스케줄링 과정에 반영하기 위한 기법들이 제시되었는데[3,8,10], 이 중에서 Katcher 등은 프로세서 이용률을 이용한 분석을 바탕으로 기존의 조건식에 인터럽트 처리 비용, 태스크 큐 운용 비용, 문맥 교환 비용 등을 포함시켜 확장된 형태의 스케줄 가능성 조건식을 제시하였다[10]. 또한, Burns 등은 최악 반응 시간을 이용한 분석 방법에 기반하여 보다 정교하게 스케줄러의 동작을 모델링함으로써 실제 수행 환경에 더욱 적합한 스케줄 가능성 분석 기법을 제시하였다[3].

6. 결 론

본 논문에서는 우선 순위 기반의 선점형 실시간 스케줄링 방법을 위주로 살펴 보았다. 여기서 살펴 본 기법들은 주로 단일 프로세서 시스템을 가정한 것인데 단일 프로세서를 위한 실시간 스케줄링 분야에 대한 연구는 상당한 수준에 도달한 것으로 판단된다. 반면, 다중 프로세서 시스템이나 분산 시스템을 고려할 경우의 태스크 할당 문제, 자원에 공유에 따른 동기화 문제 등은 앞으로 더 많은 연구가 진행되어야 할 분야이다.

참고문헌

[1] N. C. Audsley, A. Burns, M. Richardson, and A. Wellings, "Hard Real-Time Scheduling: The

Deadline-Monotonic Approach", In Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software, pp.133-137, 1991.

[2] T. P. Baker and A. Shaw, "The Cyclic Executive Model and Ada", In Proceedings of the IEEE Real-Time Systems Symposium, pp.120-129, 1988.

[3] A. Burns, K. Tindell, and A. Wellings, "Effective Analysis for Engineering Real-Time Fixed Priority Schedulers", IEEE Transactions on Software Engineering, vol. 21, no. 5, pp.475-480, 1995.

[4] H. Chetto and M. Chetto. "Some Results of the Earliest Deadline Scheduling Algorithm", IEEE Transactions on Software Engineering, vol. 15, no. 10, pp.1261-1269, 1989.

[5] R. I. Davis, K. W. Tindell, and A. Burns, "Scheduling Slack Time in Fixed Priority Pre-emptive Systems", In Proceedings of the 14th Real-Time Systems Symposium, pp.222-231, 1993.

[6] C. C. Han, C. J. Hou, and K. J. Lin, "Distance-Constrained Scheduling and Its Applications to Real-Time Systems", IEEE Transactions on Computers, vol. 45, no. 7, pp.814-826, 1996.

[7] C. Hsueh, K. J. Lin, and N. Fan, "Distributed Pinwheel Scheduling with End-to-End Timing Constraints", In Proceedings of the 16th Real-Time Systems Symposium, pp.172-181, 1995.

[8] K. Jeffay and D. L. Stone, "Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems", In Proceedings of the 14th Real-Time Systems Symposium, pp.212-221, 1993.

[9] M. Joseph and P. Pandya. "Finding Response Times in a Real-Time System", The BCS Computer Journal, vol. 29, no. 5, pp.390-395, 1986.

[10] D. I. Katcher, H. Arakawa, and J. K. Strosnider, "Engineering and Analysis of Fixed Priority Schedulers", IEEE Transactions on Software Engineering, vol. 19, no. 9, pp.920-934, 1993.

[11] J. P. Lehoczky and S. R. Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems", In Proceedings of the 13th Real-Time Systems Symposium, pp.110-123, 1992.

[12] J. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", In Proceedings of the 8th Real-Time Systems Symposium, pp.261-270, 1987.

[13] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior", In Proceedings of the 10th Real-Time Systems Symposium, pp.166-171, 1989.

[14] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", Journal of the ACM, vol. 20, no. 1, pp.46-61, 1973.

[15] K. Schwan and H. Zhou, "Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Threads", IEEE Transactions on Software Engineering, vol. 18, no. 8, pp.736-747, 1992.

[16] B. Sprunt, J. Lehoczky, and L. Sha, "Explo-

iting Unused Periodic Time for Aperiodic Service Using the Extended Priority Exchange Algorithm", In Proceedings of the 9th Real-Time Systems Symposium, pp.251-258, 1988.

[17] B. Sprunt, L. Sha, and J. P. Lehoczky, "Aperiodic Task Scheduling for Hard Real-Time Systems", Real-Time Systems, vol. 1, no. 1, pp.27-60, 1989.

[18] 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", In Proceedings of the 17th Real-Time Systems Symposium, pp.288-299, 1996.

[19] S. R. Thuel and J. P. Lehoczky, "On-Line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority Systems", In Proceedings of the 14th gReal-Time Systems Symposium, pp.160-171, 1993.

[20] K. W. Tindell, A. Burns, and A. J. Wellings, "An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks", Real-Time Systems, vol. 6, no. 2, pp.133-151, 1994.

김 성 관

1991년 서울대학교 컴퓨터공학과 (학사)
1996년 서울대학교 컴퓨터공학과 (석사)
1996년-현재 서울대학교 컴퓨터공학과 박사과정
관심분야 : 스케줄링 이론, 실시간 시스템, 멀티미디어 시스템

하 란

1987년 서울대학교 컴퓨터공학과 (학사)
1989년 서울대학교 컴퓨터공학과 (석사)
1989년-1990년 한국통신 전임연구원
1995년 University of Illinois at Urbana-Champaign, Dept. of Computer Science 박사
1995년-현재 홍익대학교 컴퓨터공학과 조교수
관심분야 : 스케줄링 이론, 실시간 시스템, 분산시스템, 멀티미디어 시스템