실시간 시스템을 위한 고정 우선순위 선점가능 스케쥴링 알고리즘 및 스케쥴 가능성 분석 (Fixed-Priority Preemptive Scheduling Algorithms and Schedulability Analyses for Real-Time Systems)

서울대학교 컴퓨터 공학과

조 진 성

요 약

본 논문에서는 실시간 시스템에서의 고정 우선순위 선점가능 스케쥴링 알고리즘에 대해 소개한다. 최초의 고정 우선순위 스케쥴링 알고리즘인 Liu와 Layland의 비율 단조 알고리즘으로 부터 초기의 많은 가정을 완화시킨 현재의 연구 상황에 대해 살펴본다. 즉, 태스크의 종료시한이 주기와 무관한 임의의 종료시한을 가지는 태스크 집합, 태스크간의 동기화를 필요로하는 태스크 집합, 비주기 태스크를 포함하는 태스크 집합 등에 대한 스케쥴링 알고리즘 및 스케쥴 가능성 분석에 대해 살펴 본다.

1 서론

실시간 시스템이란 태스크의 수행 결과의 논리적 정확성 뿐만 아니라 태스크의 종료시한(deadline) 내 수행 여부가 시스템 전체의 올바른 동작의 판단 기준이 되는 시스템이다. 이러한 실시간 시스템 내에서의 실시간 태스크들은 도착 형태와 종료시한에 따라서 여러 가지로 분류된다. 태스크의 종료시한을 만족시키지 못할 경우 시스템 전체에 큰 재난을 가져오게 되는 경우 그 태스크는 경성(hard) 종료시한을 갖는다고 말하고 경성 실시간 태스크라고 불린다. 가능한 종료시한을 만족시키는 것이 바람직한 연성 실시간 태스크의 경우 연성(soft) 종료시한을 갖게 된다. 또한, 규칙적인 도착 형태를 갖는 태스크들을 주기(periodic) 태스크, 그렇지 못한 경우 비주기(aperiodic) 태스크라고 한다. 일반적으로 주기 태스크

는 경성 종료시한을 가지며, 비주기 태스크는 연성 종료시한을 가진다. 따라서 실시간 스케쥴링 알고리즘의 목표는 주기 태스크의 경우 모든 태스크의 종료시한 만족 여부를 보장하는 것이 되며 비주기 태스크의 경우 태스크의 응답시간(response time)을 최소화하는 것이 된다. 경성 종료시한을 갖는 비주기 태스크를 스포래딕(sporadic) 태스크라 하며 최저 도착 시간 간격(minimum interarrival time)이 태스크의 파라미터로서 추가된다. 최저 도착 시간 간격이 주어지지 않을 경우 경성 종료시한을 만족시키는 것을 보장할 수 없다[Che88, Spr89, Aud90, Gho94].

과거의 실시간 시스템은 그 크기가 작고 정적이며 매우 작은 수의 태스크만을 포함하였으므로 태스크의 수행을 오프라인(off-line)시에 분석할 수 있었다. 예를 들어 주기 태스크만을 포함하는 경우 첫번째 실행 시작 시간만 주어진다면 그 이후의 모든 실행은 정확하게 계산될 수 있다. 이러한 오프라인 스케쥴링을 순환적 실행체제(cyclic executive) 또는 표 참조 스케쥴링(static table-driven scheduling)이라 한다[Ram94]. 그러나 이 방법은 많은 문제점을 안고 있다. 태스크 집합에 대해 약간의 변경이라도 가해질 경우 시스템 전체의 스케쥴링을 다시해야 하며 시스템 내의 태스크가 많을 경우 분석이 어려워진다. 또한 비주기 태스크의 추가시 비주기 태스크의 응답시간이 예측 불가능해진다. 따라서 현재의 스케쥴링은 온라인(on-line)으로 이루어지며 다음과 같이 분류된다[Kle94].

- 선점가능 대 선점불가능 (Preemptive vs. Non-preemptive)
  선점가능 스케쥴링은 태스크들이 상대적을 작은 오버헤드에 선점될 수 있음을 가정한다. 선점가능 스케쥴링의 가장 큰 이유는 보다 높은 우선순위의 태스크들이 수행될 수 있도록 하기 위해서이다. 선점불가능 스케쥴링은 태스크들이 중단되지 못하므로 구현하기 쉬우나 높은 우선순위의 태스크들이 종료시한을 만족시키지 못하는 경우가 발생하기 쉽다. 따라서 현재의 실시간 스케쥴링은 선점가능을 대부분 허용한다. 그러나 동시 실행 제어(concurrency control)에서와 같이 선점불가능이 존재하는 경우가 있다.

- 정적 우선순위 대 동적 우선순위 (Static priority vs. Dynamic priority)
  정적 또는 고정(fixed) 우선순위 스케쥴링은 각 태스크에 우선순위를 미리 할당하고 태스크의 수행시 그 우선순위가 변경되지 않는다. 반대로 동적 우선순위 스케쥴링의 경우 태스크의 수행시 그 우선순위가 계속 변화된다. 정적 우선순위 스케쥴링 알고리즘의 대표적인 예로 비율 단조(rate monotonic) 알고리즘을 들 수 있으며 동적 우선순위 스케쥴링 알고리즘으로는 EDF(Earliest Deadline First), LSF(Least Slack First)등이 있다.

한편, 시스템의 시간 제약 조건을 만족시키기 위해 스케쥴러는 다음의 조건을 만족시키는 실시간 스케쥴링 알고리즘을 사용해야 한다[Spr89].

- 경성 실시간 태스크들이 시간 제약 조건을 만족하는지 여부를 보장해야 한다.
- 경성 실시간 태스크들의 스케쥴 가능한 사용률(schedulable utilization)을 높은 수준으로 유지해야 한다.
- 연성 실시간 태스크들에 대한 응답시간을 최소화 해야 한다.
- 일시적인 과부하(transient overload)에도 안정성을 보장해야 한다.

본 논문에서는 위의 조건들을 충분히 만족시키는 고정 우선순위 선점가능 스케쥴링 알고리즘(fixed priority preemptive scheduling algorithm)에 대해 살펴본다. 고정 우선순위 선점가능 스케쥴링은 1973년 Liu와 Layland에 의해 비율 단조 우선순위 할당(rate monotonic priority assignment) 알고리즘이 제안된 이래 주로 CMU와 York대학에서 많은 연구가 이루어졌으며 그 경과를 먼저 살펴본다.

Liu와 Layland는 여러가지 제약조건을 갖는 주기 태스크의 집합에 대해 태스크의 주기에 비례하여 우선순위를 할당하는 비율 단조 우선순위 할당 알고리즘을 제안하였다[Liu73]. 이때 태스크의 종료시한은 한 주기의 끝으로 설정된다. 또한 태스크 집합에 대한 사용률 경계치(utilization bound)를 제시함으로써 주기 태스크들에 대한 종료시한 만족 여부를 보장하였다. 그러나 이 분석은 모든 태스크들이 종료시한을 만족하기 위한 충분 조건일뿐 필요 조건은 되지 못한다. [Leh89]에서 이를 확장하여 스케쥴 가능성 분석을 위한 필요충분 조건을 제시한다. [Sha90b]에서는 PCP(Priority Ceiling Protocol)를 제시하여 태스크들간의 동기화 또는 공유 자원에 대한 접근을 허가하였다. 이 프로토콜은 교착상태(deadlock)와 연결 대기(chained blocking)를 없앰으로써 자신보다 낮은 우선순위를 갖는 태스크에 의해 대기(block)되는 횟수를 한번으로 제한한다. 이 프로토콜을 이용하여 대기 요소(blocking factor)를 스케쥴 가능성 분석에 포함시켜 독립적인 태스크 집합이라는 제약 조건을 제거시킬 수 있다. [Sha86]에서는 일시적인 과부하시에도 비율 단조 알고리즘이 안정적으로 동작할 수 있는 주기 변환법(period transformation method)이 제시되어 있다.

[Jos86]에서는 태스크의 최대 응답 시간(worst-case response time)을 구하기 위한 분석이 수행되었다. 이 분석은 태스크의 종료시한이 주기와 무관한 경우를 허용한다. 즉, 최대 응답 시간과 태스

크의 종료시한이 비교됨으로써 스케쥴 가능성을 검사한다. 태스크의 종료시한이 주기보다 작은 경우에는 종료시한에 비례하여 우선순위를 할당하는 종료시한 단조(deadline monotonic) 알고리즘이 사용된다[Leu82]. 태스크의 종료시한이 주기보다 큰 경우에 대한 연구는 [Leh90, Tin94]에서 수행되었다. [Leh90]에서는 비율 단조 알고리즘을 이용하여 사용률을 기준으로한 태스크 집합에 대한 스케쥴 가능성을 분석하였고, [Tin94]에서는 최대 응답시간을 토대로한 스케쥴 가능성 분석 및 최적의 우선순위 할당 정책을 제시하였다.

2 절에서는 고정 우선순위 선점가능 스케쥴링 알고리즘의 효시인 비율 단조 알고리즘에 대해 살펴본다. 그리고 가장 활발한 연구를 수행한 CMU와 York 대학의 연구내용을 각각 3 절과 4 절에서 살펴본 후, 5 절에서 결론을 맺는다.

2 비율 단조 알고리즘

주기 태스크들의 스케쥴링 문제는 1973년 Liu와 Layland에 의해 처음 시도되었다[Liu73]. Liu와 Layland는 고정 우선순위와 동적 우선순위 스케쥴링 알고리즘에 대해 모두 다루었지만, 본 논문에서는 고정 우선순위 알고리즘만을 다루기로 한다. Liu와 Layland의 비율 단조 알고리즘은 다음의 가정에서 출발한다.

- 태스크들은 모두 주기 태스크이며 주기의 시작에 바로 시작되며 주기의 끝이 종료시한이 된다. 또한 수행도중 자발적으로 중단되지 않는다.
- 태스크들은 선점가능하며, 문맥 교환, 스케쥴링 오버헤드등은 무시된다.
- 태스크들은 서로 독립적(independent)이다. 즉 태스크간의 동기화(task synchronization)등은 존재하지 않으며 따라서 최대 수행시간(worst-case execution time)은 미리 알 수 있다.

비록 위의 가정들은 매우 엄격하지만 Liu와 Layland는 매우 중요한 결과를 유도해 내었다. 따라서 그 결과는 실제의 응용에 사용될 수 있도록 확장시키는데 밑거름이 된다.

비율 단조 알고리즘은 n개의 주기 태스크($\tau_1, \tau_2, \cdots, \tau_n$)들의 집합을 대상으로 한다. 태스크 τ_i는 (C_i, T_i, D_i, I_i)로 이루어지며 여기에서 C_i는 τ_i의 최대 실행시간, T_i는 τ_i의 주기, D_i는 τ_i의 종료시한, I_i는 기준 시간에 대한 τ_i의 시작 시간을 나타낸다. 따라서 τ_i의 j번째 수행은 $I_i + (j-1)T_i$ 에 시

작되고 $I_i + (j-1)T_i + D_i$ 이전에 C_i만큼의 수행을 마쳐야 한다. Liu와 Layland는 $D_i = T_i$ 를 가정하였다. 이러한 태스크 집합이 스케쥴 가능(schedulable)하다는 것은 모든 태스크들의 실행에 대해 종료시한을 만족시킬 수 있는 스케쥴링 알고리즘이 존재하다는 뜻이다.

Liu와 Layland는 고정 우선순위 스케쥴링에 대해 세가지 중요한 결과를 증명하였다. 고정 우선순위 스케쥴링 알고리즘이 사용되고 태스크들은 우선순위별로 정렬되어 있을때, 즉 $i < j$ 이면 τ_i의 우선순위가 τ_j의 우선순위보다 높을때, 다음의 두가지 정리가 성립한다.

정리 1 τ_i의 최대 응답시간*(longest response time)*은 태스크의 첫번째 실행에 대해 $I_1 = I_2 = \cdots = I_n = 0$일 때이다. 이때를 긴급한 순간*(critical instant)*이라고 한다. [Liu73] □

긴급한 순간을 직관적으로 살펴보면, 태스크 집합내의 모든 태스크들이 동시에 시작되는 시점이다. 따라서 각 태스크는 최대 응답시간을 경험한다.

정리 2 긴급한 순간 이후 모든 태스크들의 첫번째 실행이 고정 우선순위 스케쥴링에 의해 종료시한을 만족하면 모든 태스크들은 항상 종료시한을 만족한다. 즉, 해당 태스크 집합은 스케쥴 가능하다. [Liu73] □

Liu와 Layland는 이로부터 최적(optimal)의 고정 우선순위 스케쥴링 알고리즘인 비율 단조 알고리즘을 제안한다. 비율 단조 알고리즘은 주기가 작은 태스크에 높은 우선순위를 할당한다. 따라서 τ_i가 τ_j보다 높은 우선순위를 갖는다면 $T_i < T_j$ 를 만족한다. 여기에서 최적이란, 주기 태스크 집합을 스케쥴링할 수 있는 다른 고정 우선순위 알고리즘이 존재한다면 비율 단조 스케쥴링 알고리즘으로도 스케쥴 가능하다는 뜻이다. 따라서 다음의 정리가 성립한다.

정리 3 비율 단조 스케쥴링 알고리즘은 $D_i = T_i$ 인 주기 태스크 집합을 스케쥴링함에 있어서 최적이다. [Liu73] □

여기에서 주목할만한 사실은 태스크의 중요도(importance, criticalness)는 고려되지 않았다는 사실이다.

또한, Liu와 Layland는 비율 단조 알고리즘을 사용할 경우에 대한 최악의 경계치를 유도하였다. 즉 n개의 태스크 집합에 대한 사용률 $(U = C_1/T_1 + \cdots + C_n/T_n)$이 어떤 경계치보다 작을 경우 비율 단조 알고리즘은 항상 종료시한을 만족함을 보장한다. 그 결과는 다음과 같다.

정리 4 $D_i = T_i,\ 1 \leq i \leq n$, 인 주기 태스크 집합, $\tau_1, \tau_2, \cdots, \tau_n$, 은 다음의 조건을 만족하면 비율 단조 스케쥴링 알고리즘에 의해 스케쥴 가능하다.

$$U_1 + U_2 + \cdots + U_n \ \leq \ U_n^* = n(2^{1/n} - 1), \quad n = 1, 2, \cdots$$

[Liu73] □

위의 결과를 무한개의 태스크를 가진 태스크 집합에 대해 살펴 볼 경우 그 경계치는 $U_\infty^* = ln2 = 0.693$ 이 된다. 다시 말하면 태스크들의 사용률이 0.693보다 작은 태스크 집합은 비율 단조 알고리즘에 의해 스케쥴 가능하다.

3 CMU의 연구 내용

본 절에서는 고정 우선순위 스케쥴링 알고리즘에 대한 연구를 가장 활발히 전개한 CMU의 연구내용을 소개한다. 3.1 절에서는 비율 단조 알고리즘을 확장시켜, 스케쥴 가능성을 위한 필요충분 조건을 유도하고 태스크의 종료시한이 주기와 일치하지 않을 경우에 대해 살펴본다. 3.2 절에서는 태스크간의 동기화 문제를 고려하여 태스크간의 독립성 가정을 제거시키는 방법을 소개하며, 3.3 절에서는 비주기 태스크들의 스케쥴링에 대해 논한다.

3.1 비율 단조 알고리즘의 확장

정리 4의 최악의 경계치 0.693은 충분히 큰 값이지만 충분 조건일 뿐, 필요 조건은 아니다. 즉, 태스크 집합의 사용률이 0.693보다 클지라도 비율 단조 알고리즘에 의해 스케쥴 가능한 태스크 집합이 존재할 수 있다. 우선, 태스크의 종료시한이 주기보다 작을 경우를 먼저 고려한다. 이는 실제 응용의 경우 흔히 볼 수 있는 제약조건이다. 이 문제는 1982년에 Leung과 Whitehead에 의해 처음으로 다루어졌으며 태스크의 우선순위를 종료시한에 비례하여 할당하는 종료시한 단조 스케쥴링 알고리즘을 제안하였다[Leu82]. 즉 $D_i < D_j$일 경우 τ_i는 τ_j보다 높은 우선순위를 갖는다. 그러나 스케쥴 가능성을 검사하기 위한 명시적인 조건은 유도되지 않았다.

정리 5 $D_i < T_i$ 인 태스크 집합에 대해 최적의 고정 우선순위 스케쥴링 알고리즘은 종료시한 단조 알고리즘이다. 각 태스크들이 긴급한 순간 이후 첫 실행이 종료시한을 만족하면 그 태스크 집합은 스케쥴 가능하다.

[Leu82] □

정리 4는 [Leh90]에 의해서도 개선되어 $D_i = \Delta T_i,\ 0 < \Delta \leq 1$, 일때 다음의 정리가 성립한다. 이 경우는 비율 단조 알고리즘과 종료시한 단조 알고리즘 모두 최적이다.

정리 6 $D_i = \Delta T_i$ 인 태스크 집합은 다음의 조건을 만족하면 스케쥴 가능하다.

$$U_1 + \cdots + U_n \ \leq \ U_n^*(\Delta) = \begin{cases} n((2\Delta)^{1/n} - 1) + (1 - \Delta) & ,\ 1/2 \leq \Delta \leq 1 \\ \Delta & ,\ 0 \leq \Delta \leq 1/2 \end{cases}$$

[Leh91] □

$D_i \leq T_i$ 인 태스크 집합의 고정 우선순위를 사용한 스케쥴 가능성에 대한 필요충분 조건은 [Leh89, Sha90a]에서 유도되었다. 긴급한 순간 이후에 $\sum_{j=1}^{i} C_j \lceil \frac{t}{T_j} \rceil = W_i(t)$는 태스크 τ_i와 자신보다 높은 우선순위를 갖는 태스크, 즉 $\tau_j,\ 1 \leq j \leq i$ 를 수행시키는데 필요한 시간이다. 정리 2에 의해 τ_i는 첫번째 종료시한을 만족하면 항상 종료시한을 만족할 것이다. 이 조건은 $W_i(t) \leq t,\ 0 \leq t \leq D_i$, 일때 발생되며 궁극적으로 $W_i(t)/t \leq 1$ 을 만족하는 $t,\ 0 \leq t \leq D_i$, 가 존재하면 태스크 τ_i는 스케쥴 가능하다. 위의 과정을 정리하면 다음과 같다.

정리 7 $D_i \leq T_i$ 인 태스크 집합에 대해 고정 우선순위 스케쥴링을 사용할 경우 태스크 τ_i의 스케쥴 가능성에 대한 필요충분 조건은 다음과 같다.

$$\min_{0 \leq t \leq D_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \ \leq \ 1$$

따라서 태스크 집합에 대한 스케쥴 가능성 검사는 다음 조건의 만족 여부로 이루어진다.

$$\max_{1 \leq i \leq n} \min_{0 \leq t \leq D_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \ \leq \ 1$$

[Leh91] □

위의 정리는 t가 연속 변수이므로 계산하기 매우 어렵다. 그러나 $T_1, T_2, \cdots, T_{i-1}, D_i$의 정수배에 대해서만 조사해도 같은 결과를 얻으므로 정리 7은 유한 시간 내에 계산될 수 있다[Leh89].

지금부터는 태스크의 종료시한이 주기보다 클 경우를 고려해 보자. 비율 단조 알고리즘 및 종료시한 단조 알고리즘은 이 경우 더이상 최적의 알고리즘이 아니다. [Leh90]에서는 우선순위 i에서의 사용

시간(level-i busy period)이라는 개념을 이용하여 이 문제를 해결한다. 우선순위 i에서의 사용 시간이란 우선순위 i 또는 그보다 높은 우선순위를 갖는 태스크들에 의해 점유되는 프로세서 시간 간격 [a,b]를 말한다. 따라서 종료시한이 주기보다 큰 경우 정리 1은 다음과 같이 일반화된다.

정리 8 τ_i의 최대 응답시간*(longest response time)*은 긴급한 순간 $I_1 = I_2 = \cdots = I_n = 0$ 에 시작하여 우선순위 i에서의 사용시간에 발생된다. [Leh91] □

정리 7도 일반화시키기 위하여 다음과 같은 다음과 같은 함수 $W_m(k,x)$를 정의한다.

$$W_m(k,x) = \min_{t \le x} \left(\left(\sum_{j=1}^{m-1} C_j \left\lceil \frac{t}{T_j} \right\rceil + kC_m \right) / t \right)$$

$W_m(k,x)$는 우선순위 m의 태스크가 k 번 수행될 때까지 우선순위 m에서의 사용 시간에 대한 x시간 내에서의 부하 함수, 즉 사용률이 된다. 예를 들어 $W_m(1,T_m) \le 1$이면 τ_m의 첫번째 수행에 대한 우선순위 m에서의 사용 시간이 주기보다 작다는 뜻이고,$W_m(1,D_m) \le 1$이면 τ_m의 첫번째 수행이 종료시한을 만족한다는 뜻이 된다. 따라서 태스크 τ_m에 대한 스케쥴 가능성 검사는 다음의 식으로 주어진다.

$$W_m(k,(k-1)T_m + D_m) \le 1$$

또한, [Leh90]에서는 정리 4와 정리 6을 일반화하여 태스크의 종료시한이 주기와 무관한 태스크 집합에 대한 사용률 경계치를 제시하였다. $D_i = \Delta T_i$라고 할때, Δ가 정수일때와 그렇지 않을 경우로 나누어 고려한다.

정리 9 $D_i = \Delta T_i,\ \Delta = 1,2,\cdots$ 인 태스크 집합은 다음의 조건을 만족하면 스케쥴 가능하다.

$$U_n^*(\Delta) = \begin{cases} n\left((\Delta+1)^{1/n} - 1\right) & ,\ \Delta = 1 \\ \Delta\left(n-1\right)\left(\left(\frac{\Delta+1}{\Delta}\right)^{1/(n-1)} - 1\right)\right) & ,\ \Delta = 2,3,\cdots \end{cases}$$

그리고

$$U_\infty^* = \Delta \log_e \left(\frac{\Delta+1}{\Delta}\right), \quad \Delta = 1,2,\cdots$$

이다. [Leh91] □

Δ가 정수가 아닐 경우는 사용률 경계치의 유도가 더욱 복잡해진다.

정리 10 $D_i = \Delta T_i,\ \Delta \in [k, k+1],\ k = 0, 1, \cdots$ 인 태스크 집합에 대한 최악의 사용률 경계치는 다음과 같다.

$$U^*_\infty = \begin{cases} (k+1)\log_e(\Delta/S(k+1)) - k\log_e((\Delta - S)/k) + (k+1)S - k \\ \qquad if \quad k \le \Delta \le k + 1 - 1/(k+2) \\ (k+1)\log_e((k+2)\Delta/(k+1)^2) + (k+1) - \Delta \\ \qquad if \quad k + 1 - 1/(k+2) \le \Delta \le k+1 \end{cases}$$

여기에서 S는 다음 방정식의 가장 작은 근이다.

$$S^2 - S[\Delta + (2k+1)/(k+1)] + \Delta = 0$$

[Leh91] □

3.2 태스크간의 동기화

지금까지는 태스크 집합내의 태스크들은 서로 무관하였다. 즉 세마포어등의 접근을 통한 동기화등은 존재하지 않는 것으로 가정하였다. 그러나 실제의 응용은 태스크들간의 동기화 및 공유 자원의 사용을 많이 필요로 한다. 세마포어 또는 모니터등의 태스크간의 동기화 기법을 사용할 경우 실시간 시스템에서는 우선순위 역전(priority inversion) 현상이 발생하게 된다. 그리고 우선순위 역전 기간이 제한되지 못하며 교착상태도 발생될 수 있다.

[Sha90b]에서는 위의 문제점을 해결하기 위해 PCP(Priority Ceiling Protocol)를 제안하였다. PCP의 동작은 [Sha90b]에 제시되어 있으므로 본 논문에서는 PCP를 스케쥴 가능성 측면에서만 살펴보기로 한다. PCP는 주목할만한 많은 특성이 있지만, 주요한 세가지를 살펴보자.

정리 11 *PCP*는 교착상태를 예방한다. [Sha90b] □

정리 12 *PCP*하에서 한 태스크는 하나의 임계영역에서 단 한번의 우선순위 역전 현상을 만난다.

[Sha90b] □

PCP와 비율 단조 알고리즘을 사용하여 스케쥴 가능성을 분석하기 위해 대기 요소(blocking factor), B_i를 정의한다. B_i는 태스크 τ_i가 경험하는 최대 대기 시간이다. 따라서 대기 요소를 이용함으로써 태스크들간의 동기화를 필요로하는 태스크 집합에 대한 스케쥴 가능성을 유도할 수 있다.

정리 13 $D_i = T_i$ 인 태스크 집합은 비율 단조 알고리즘과 PCP를 사용할 경우 다음의 조건을 만족하면 스케쥴 가능하다.

$$\max_{1 \le i \le n} \min_{0 \le t \le D_i} \left(\sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil + \frac{B_i}{t} \right) \le 1$$

[Sha90b] □

3.3 비주기 태스크 스케쥴링

지금까지는 주기 태스크에 대한 스케쥴링에 대해 살펴보았다. 본 절에서는 실시간 시스템 스케쥴링에서 발생하는 실질적인 문제, 즉 주기 태스크와 비주기 태스크의 혼합 스케쥴링에 대해 살펴본다. 과거의 방법은 다음의 세가지로 구분할 수 있다.

- 비주기 태스크가 주기 태스크를 선점하여 수행된다.
  이 방법은 주기 태스크의 경성 종료시한을 만족시키지 못할 우려가 있으므로 좋은 방법이 되지 못한다.

- 비주기 태스크들이 주기 태스크들의 배경작업(background job)으로 수행된다.
  이 방법은 비주기 태스크들의 응답시간을 예측할 수 없을 뿐더러 일반적으로 상당히 큰 응답시간을 보인다.

- 비주기 태스크만을 전담하여 처리하는 폴링 서버(polling server)를 둔다.
  이 방법은 주기 태스크의 종료시한을 보장하면서 비주기 태스크의 응답시간을 줄일 수 있다.

세번째 방법을 개선하여 비주기 태스크의 응답시간을 개선하는 알고리즘이 많이 제안되었다[Leh87, Spr88, Spr89].

[Leh87]에는 PE(Priority Exchange) 알고리즘과 DS(Deferrable Server) 알고리즘이 제시되었다. 이와 같이 비주기 태스크들을 처리하는 서버를 두는 방법을 대역폭 보존(bandwidth preserving) 기법이라 한다. 이 서버는 주기 태스크와 함께 스케쥴 가능성 분석이 수행되며 빠른 응답시간을 보장하기 위해 가장 높은 우선순위를 갖는다. 비주기 태스크들을 처리하는 서버는 자신의 수행 가능한 용량(capacity)을 가지며 그 한도내에서 비주기 태스크들을 처리한다. 서버의 용량은 서버의 주기마다 새로 채워진다. PE 알고리즘은 서버의 용량이 비교적 크지만 복잡한 단점을 가지고 있고, DS 알고리즘은 개념적으로 간단하여 구현하기 쉽지만 서버의 용량이 작다. [Spr89]에서 이 알고리즘들의 장단점을 보완하여 SS(Sporadic Server) 알고리즘을 제안한다. 간단히 차이점을 설명한다면 SS 알고리즘은 서버

의 용량을 비주기 태스크에 의해 사용된 만큼을 비주기 태스크 수행 후 서버의 주기 이후에 채운다. SS 알고리즘의 중요한 결과를 스케쥴 가능성 측면에서 살펴보면 다음과 같다.

- 서버는 스케쥴 가능성 분석에서 일반적인 주기 태스크와 동일하게 취급된다.
- 몇개의 서버가 각각 다른 우선순위를 갖고 생성될 수 있으므로 각기 다른 우선순위의 비주기 태스크를 처리할 수 있다.
- 스포래딕 태스크의 경우 한 스포래딕 태스크마다 하나의 서버를 할당함으로써 스케쥴 가능성을 보장할 수 있다.

정리 14 주기 태스크 τ_i를 포함한 태스크 집합이 스케쥴 가능할 경우 τ_i가 같은 주기와 수행시간을 갖는 서버로 대체되어도 스케쥴 가능하다. [Spr89] □

정리 15 *SS* 알고리즘은 적어도 폴링 서버 알고리즘과 같은 응답시간을 갖는다. [Leh91] □

4 York 대학의 연구 내용

본 절에서는 York 대학에서의 고정 우선순위 스케쥴링 알고리즘에 대해 살펴본다. 4.1 절에서는 태스크의 종료시한이 주기와 무관한 임의의 종료시한을 가질때의 스케쥴링 알고리즘에 대해 살펴본다. 여기에서의 태스크 집합은 주기 태스크와 스포래딕 태스크를 포함한다. 4.2 절에서는 고정 우선순위 선점가능 스케쥴링시 새로이 고려해야 할 사항들에 대해 살펴보고, 4.3 절에서는 태스크가 임의의 종료시한을 가질때 최적의 우선순위 할당 방법에 대해 설명한다.

4.1 비율 단조 알고리즘의 확장

York 대학에서의 연구는 1986년 Joseph과 Pandya의 최대 응답시간(worst-case response time) 분석에 대한 연구를 기반으로 한다. 태스크내의 모든 태스크가 동시에 시작되는 긴급한 순간을 기준으로 하여 각 태스크의 최대 응답시간, r_i는 다음과 같다[Jos86].

$$r_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i}{T_j} \right\rceil C_j$$

위의 식은 r_i가 양쪽식에 존재하므로 쉽게 계산할 수 없다. 따라서 다음의 식을 사용하며, 정리 16에서 항상 값을 구할 수 있음을 증명한다.

$$r_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i^n}{T_j} \right\rceil C_j$$

여기서 $r_i^0 = C_i$ 이다.

정리 16 우선순위 i와 그보다 높은 우선순위를 갖는 태스크들의 사용률이 *1* 보다 작을 경우 r_i^n은 유한 횟수*(finite iteration)*에 수렴한다. 즉, $r_i^{n+1} = r_i^n$ 인 n이 존재한다. [Jos86] □

태스크의 종료시한이 주기보다 같거나 작은 경우에는 $r_i \leq D_i$ 이면 태스크는 항상 종료시한을 만족한다. 그러나 종료시한이 주기와 무관한 임의의 종료시한을 갖는 태스크의 집합에 대해 살펴보기로 하자. $r_i > T_i$일 경우 $r_i - T_i$ 의 구간에 시작되는 태스크들에 대해서는 고려되지 않았으므로 r_i는 새로이 정의되어야 한다. 따라서 3.1절에서 언급한 우선순위 i에서의 사용시간 개념을 이용한다. 주기적인 태스크의 각 실행의 응답시간은 매회 다르므로 최대 응답시간을 고려하여 스케쥴 가능성을 분석하여야 한다. q번째 실행에서 우선순위 i에서의 사용시간은 다음과 같다[Tin94]. 여기에서 $q = 0$ 일때가 태스크의 첫 번째 실행이 되며 이 경우가 Joseph과 Pandya의 분석에 해당된다.

$$w_i(q) = (q+1)C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i(q)}{T_j} \right\rceil C_j$$

위의 식도 $w_i(q)$가 양쪽에 존재하므로 반복적인 방법에 의해 $w_i(q)$를 구한다. 여기에서 B_i는 3.2 절에서 언급된 PCP를 사용하여 발생되는 태스크의 최대 대기시간이다.

태스크 τ_i의 q번째 실행은 qT_i에 시작되므로 q번째 실행의 응답시간은 $(w_i(q) - qT_i)$이다. 따라서 τ_i에 대한 최대 응답시간은

$$r_i = \max_{q=0,1,2,\cdots} (w_i(q) - qT_i)$$

으로 주어진다. 여기에서도 q의 값의 경계가 주어져야 한다. q의 값은 $w_i(q) \leq (q+1)T_i$ 일때까지 증가되어 r_i가 구해지며 q 값의 경계치도 반드시 존재한다[Tin94]. 따라서 다음의 정리가 유도된다.

정리 17 임의의 종료시한을 갖으며 주기 태스크와 스포래딕 태스크로 구성된 태스크 집합은 $r_i \leq D_i,\ 1 \leq i \leq n$, 을 만족하면 스케쥴 가능하다. [Tin94] □

4.2 새로운 고려사항

대부분의 고정 우선순위 선점가능 디스패쳐(dispatcher)는 틱(tick) 스케쥴링을 통해 구현된다. 즉, 주기적인 클럭 인터럽트 핸들러에 의해서 이루어진다. 따라서 태스크의 시작 시간에 바로 스케쥴 되지 못하는 경우가 발생한다. 다시 말하면, 클럭 인터럽트 핸들러의 주기 사이에 태스크가 시작되는 경우이다. 이때 태스크가 시작된 시점부터 그 다음 인터럽트 핸들러가 태스크를 스케쥴 할 때까지의 시간은 4.1 절에서 고려되지 않았다. 이 시간을 시작 지터(release jitter)라고 하며 J_i로 나타내기로 한다[Aud93, Tin94]. 이 시작 지터를 고려한 우선순위 i에서의 사용시간 및 최대 응답시간은 다음과 같다[Tin94].

$$w_i(q) = (q+1)C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{J_i + w_i(q)}{T_j} \right\rceil C_j$$

$$r_i = \max_{q=0,1,2,\cdots} (w_i(q) + J_i - qT_i)$$

지금부터는 다른 형태의 태스크에 대한 경우를 살펴본다. 내부 주기(t_i)와 외부 주기(T_i)를 갖는 비주기적 주기 태스크(sporadically periodic task)는 외부 주기를 단위로 발생되며 외부주기 동안 최소 도착 시간 간격이 내부 주기인 스포래딕 태스크를 말한다. 이러한 태스크의 예로는 네트워크 패킷 드라이버를 들 수 있다. 네트워크상의 패킷은 한꺼번에(burst) 도착되는 경우가 일반적이므로 버스트 패킷이 도착되는 시간 간격을 외부 주기, 버스트 패킷 내에서 각각의 패킷을 처리하는 시간을 내부 주기로 설정할 수 있으며 기존의 태스크 모델로는 표현 불가능하다. [Aud93, Tin94]에서는 기존의 주기 태스크 및 스포래딕 태스크는 물론, 비주기적 주기 태스크의 스케쥴 가능성을 분석하였다.

정리 18 임의의 주기를 갖으며 주기 태스크, 스포래딕 태스크, 비주기적 주기 태스크로 구성된 태스크 집합은 $r_i \le D_i,\ 1 \le i \le n$ 를 만족하면 스케쥴 가능하다. 여기에서

$$\begin{aligned}
&r_i = \max_{q=0,1,2,\cdots}(w_i(q) + J_i - m_i t_i - M_i T_i) \\
&m_i = q - M_i n_i \\
&M_i = \left\lfloor \tfrac{q}{n_i} \right\rfloor \\
&w_i(q) = (M_i n_i + m_i + 1)C_i + B_i \\
&\qquad + \textstyle\sum_{\forall j \in hp(i)} \left(\min\left(n_j, \left\lceil \frac{J_j + w_i(q) - F_j T_j}{t_j} \right\rceil\right) + F_j n_j \right) C_j \\
&F_j = \left\lceil \tfrac{J_j + w_i(q)}{T_j} \right\rceil - 1
\end{aligned}$$

[Tin94] □

위의 정리에서 n_i는 외부 주기내에서의 수행 횟수를 나타내며 $n_i = 1, t_i = T_i$ 일 경우 정리 17과 같다.

4.3 최적의 우선순위 할당(Optimal Priority Assignment)

4.1 절과 4.2 절에서의 분석은 태스크들의 우선순위가 할당되어있는 상황에서 행해진 것이었다. 그런데, 임의의 종료시한을 갖는 태스크 집합에서 비율 단조 알고리즘 또는 종료시한 단조 알고리즘은 더이상 최적이지 못하다. [Aud91]에서는 n개의 태스크로 구성된 태스크 집합에 대해 최적의 우선순위 할당 알고리즘을 제안하였다. 알고리즘을 개략적으로 설명하면 다음과 같다. 태스크 집합으로부터 임의의 태스크를 선택하여 가장 낮은 우선순위, 즉 n을 할당하고 정리 18에 의해 스케쥴 가능성 검사를 수행한다. 그다음 또하나의 태스크를 선택하여 $n-1$의 우선순위를 할당하고 현재까지 스케쥴 가능한 태스크들(지금은 두개의 태스크)에 대해 스케쥴 가능성을 검사한다. 만약 스케쥴 불가능하면 해당 태스크 집합에 대한 최적의 우선순위 할당은 존재하지 않는다. 위의 작업을 모든 태스크에 대해 적용하면 최적의 우선순위 할당을 얻는다. 위의 최적의 우선순위 할당 알고리즘은 다음의 세가지 정리에 의해 최적이다.

정리 19 만일 태스크 τ_a가 가장 낮은 우선순위 n에 할당되고 스케쥴 불가능하다면 τ_a를 우선순위 n에 할당하고 스케쥴 가능한 우선순위 할당 정책은 존재하지 않는다. [Aud91] □

정리 20 만일 태스크 τ_a가 가장 낮은 우선순위 n에 할당되고 스케쥴 가능하다면 그 태스크 집합을 스케쥴 가능하게 하는 우선순위 할당 정책이 존재할 경우 τ_a는 가장 낮은 우선순위 n에 할당된다. [Aud91] □

정리 21 임의의 우선순위 할당 정책에 의해 우선순위 $i, i+1, \cdots, n$ 에 할당된 태스크들이 스케쥴 가능하다고 하자. 이때 전체 태스크 집합을 스케쥴 가능하게 하는 우선순위 할당 정책이 존재할 경우 그 태스크들은 마찬가지로 우선순위 $i, i+1, \cdots, n$ 에 할당된다. [Aud91] □

5 결론

지금까지 실시간 시스템에서의 고정 우선순위 선점가능 스케쥴링 알고리즘에 대해 살펴보았다.

CMU에서는 주기 태스크와 비주기 태스크를 구분하여 비주기 태스크는 별도의 서버를 이용한 방법을 취하였다. 주기 태스크는 비율 단조 알고리즘을 이용하여 스케쥴링하며 경성 종료시한 만족을 보장하기 위해 Liu와 Layland의 스케쥴 가능성에 대한 충분 조건을 확장시켜 필요충분 조건을 유도하였

다. 또한, 종료 시한이 주기와 무관한 임의의 종료시한을 가지는 태스크 집합에 대해 프로세서 사용률을 이용한 최대 경계치를 구하였고 태스크간의 동기화나 자원 공유를 허용한 분석 결과를 유도하였다. 비주기 태스크는 특정 서버를 두어 처리하고 서버는 주기 태스크와 함께 스케쥴 가능성 분석에 포함된다. 스포래딕 태스크의 경우 각각의 서버를 두어 경성 종료시한 만족을 보장받는다. CMU의 연구를 GRMS(Generalized Rate Monotonic Scheduling)라고 하며 [Sha94]에서 분산 시스템으로 확장시킨 내용이 소개되었다.

York 대학에서는 주기 태스크와 스포래딕 태스크를 포함하며 임의의 종료시한을 갖는 태스크 집합에 대해 스케쥴 가능성을 분석하였다. 태스크의 최대 응답시간을 이용하므로 CMU의 접근 방법보다 깨끗한 결과를 보인다. 또한, 태스크들의 우선순위 할당을 위해 최적의 우선순위 할당 알고리즘을 제안하였다. 태스크의 동기화는 PCP를 사용하며 시작 지터 및 비주기적 주기 태스크에 대한 분석을 수행하였다.

그외의 연구로서 태스크들의 여유시간(slack, laxity)을 이용한 스케쥴링 알고리즘이 있다[Leh92, Dav92]. CMU에서는 정적 분석을 통하여 태스크의 여유시간을 비주기 태스크에 할당함으로써 비주기 태스크의 응답시간을 향상시키는 알고리즘을 제안하였으며 이 알고리즘이 비주기 태스크의 응답시간을 최소화 시킨다는 측면에서 최적임을 증명하였다[Leh92]. York 대학에서는 위의 알고리즘을 개선하여 동적으로 여유시간을 계산한다. 따라서 스포래딕 태스크의 경성 종료시한 만족 여부를 동적으로 검사할 수 있다[Dav92]. 그러나 실행시의 계산량이 너무 크므로 근사(approxiamte) 기법을 사용하며 실험을 통해 그리 크지 않은 오버헤드하에 큰 성능향상을 나타냄을 보인다. 이 기법은 태스크의 최대 실행시간이 실제 실행시간과 큰 차이를 보이는 현재 상황에서는 더욱 효율적일 것으로 생각된다.

본 논문에서 대상으로 한 태스크 집합은 그 특성이 실행 도중 변하지 않는다는 가정을 하고 있다. 그러나 실시간 시스템에서 태스크 집합의 특성이 시간에 따라 변화하는 응용이 존재할 수 있다. 예를 들어 태스크가 추가 또는 삭제되는 경우, 태스크의 주기가 바뀌는 경우, 태스크의 실행 시간이 바뀌는 경우 등은 실제로 존재하는 응용이 많을 것으로 예상된다. 따라서 CMU와 York 대학에서 자신의 접근 방법에 따라 이 상황에 적합한 모드 변환 프로토콜(Mode Change Protocol)을 제안하였다[Sha89, Tin92].

마지막으로, 고정 우선순위 선점가능 스케쥴링 알고리즘은 실시간 태스크들을 스케쥴함에 있어 다음과 같은 바람직한 특성을 갖는다[Spr89]. 이 사항들은 비율 단조 알고리즘의 예를 들어 CMU에서 주

장하는 장점이지만 모든 고정 우선순위 선점가능 스케쥴링 알고리즘에 적용된다.

- 스케쥴 가능한 사용률이 높다.
- 일시적인 과부하에도 안정적인 결과를 나타낸다.
- 비주기 태스크를 주기 태스크의 처리에 영향을 주지 않고 처리할 수 있다.
- PCP를 사용하여 태스크간의 동기화 또는 자원의 공유가 가능하다.
- 스케쥴링 오버헤드가 매우 작다.

참고 문헌

[Liu73] 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. 46–61, Jan. 1973.

[Leu82] J. Y. T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks," *Performance Evaluation*, Vol. 2, No. 4, pp. 237–250, 1982.

[Jos86] M. Joseph and P. Pandya, "Finding response times in a real-time system," *BCS Computer Journal*, Vol. 29, No. 5, pp. 390–395, 1986.

[Sha86] L. Sha, J. P. Lehoczky, and R. Rajkumar, "Solutions for some practical problems in prioritized preemptive scheduling," *Proceedings of the 7th IEEE Real-Time Systems Symposium*, pp. 181–191, Dec. 1986.

[Leh87] J. P. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced aperiodic responsiveness in hard real-time environments," *Proceedings of the 8th IEEE Real-Time Systems Symposium*, pp. 261–270, Dec. 1987.

[Spr88] B. Sprunt, J. P. Lehoczky, and L. Sha, "Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm," *Proceedings of the 9th IEEE Real-Time Systems Symposium*, pp. 251–258, Dec. 1988.

[Che88] S. Cheng, J. A. Stankovic, and K. Ramamritham, "Scheduling algorithms for hard real-time systems: A brief survey," *Hard Real-Time Systems: Tutorial*, ed. J. A. Stankovic and K. Ramamritham, IEEE, 1988.

[Sha89] L. Sha, R. Rajkumar, J. P. Lehoczky, and K. Ramamritham, "Mode change protocols for priority-driven preemptive scheduling," *The Journal of Real-Time Systems,* 1, pp. 243–264, 1989.

[Leh89] J. P. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behaviour," *Proceedings of the 10th IEEE Real-Time Systems Symposium*, pp. 166–171, Dec. 1989.

[Spr89] B. Sprunt, L. Sha, and J. P. Lehoczky, "Aperiodic task scheduling for hard real-time systems," *The Journal of Real-Time Systems*, 1, pp. 27–60, 1989.

[Leh90] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pp. 201–209, Dec. 1990.

[Sha90a] L. Sha and J. Goodenough, "Real-time scheduling theory and ada," *IEEE Computer*, Vol. 23, No. 4, pp. 53–62, 1990.

[Sha90b] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on Computers*, Vol. 39, No. 9, pp. 1175–1185, 1990.

[Aud90] N. Audsley and A. Burns, "Real-time system scheduling," *Dept. Computer Science Report YCS 134*, University of York, Jan. 1990.

[Leh91] J. P. Lehoczky, L. Sha, J. K. Strosnider, and H. Tokuda, "Fixed priority-scheduling theory for hard real-time systems," A. M. van Tilborg and G. M. Koob, eds., *Foundations of Real-Time Computing: Scheduling and Resource Management*, Kluwer Academic Publishers, Boston, pp. 1–30, 1991.

[Aud91] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," *Dept. Computer Science Report YCS 164*, University of York, Dec. 1991.

[Leh92] J. P. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems," *Proceedings of the 13th IEEE Real-Time Systems Symposium*, pp. 110–123, Dec. 1992.

[Tin92] K. W. Tindell, A. Burns, and A. J. Wellings, "Mode changes in priority pre-emptively scheduled systems," *Proceedings of the 13th IEEE Real-Time Systems Symposium*, pp. 110–109, Dec. 1992.

[Dav92] R. I. Davis, K. W. Tindell, and A. Burns, "Scheduling slack time in fixed priority preemptive systems," *Proceedings of the 13th IEEE Real-Time Systems Symposium*, pp. 222–231, Dec. 1992.

[Aud93] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, Vol. 8, No. 5, pp. 284–292, Sep. 1993.

[Kle94] M. H. Klein, J. P. Lehoczky, and R. Rajkumar, "Rate-monotonic analysis for real-time industrial computing," *IEEE Computer*, Vol. 27, No. 1, pp. 24–33, Jan. 1994.

[Gho94] K. Ghosh, B. Mukherjee, and K. Schwan, "A survey of real-time operating systems – Draft," *Technical Report GIT-CC-93/18*, Georgia Institute of Technology, Feb. 1994.

[Ram94] K. Ramamritham and J. Stankovic, "Scheduling Algorithms and Operating Systems Support for Real-Time Systems," *Proceedings of the IEEE*, Vol. 82, No. 1, pp. 55–67, 1994.

[Sha94] L. Sha, R. Rajkumar, and S. Sathaye, "Generalized rate-monotonic-scheduling theory: A framework for developing real-time systems," *Proceedings of the IEEE*, Vol. 82, No. 1, pp. 68–82, 1994.

[Tin94] K. W. Tindell, A. Burns, and A. J. Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks," *The Journal of Real-Time Systems*, 6, pp. 131–151, 1994.