우선순위 스케줄링을 사용하는 실시간 시스템을 위한 정확한 확률적 분석 방법

(An Exact Stochastic Analysis Method for Priority-driven Real-time Systems)

김 강 희 †

(Kanghee Kim)

요 약 최근 멀티미디어, 신호처리와 같은 실시간 응용들에 대해서 그 응용에 속한 태스크들이 일정한 확률로 마감시간을 만족하는 것을 보장하는, 소위 확률적 보장을 제공하는 것이 점점 더 중요해지고 있다. 확률적 보장을 제공하기 위해서는 주어진 시스템에서 각 태스크의 마감시간 위반확률(deadline miss probability)을 예측할 수 있는 분석 방법이 필요하다. 본 논문에서는 Rate Monotonic 또는 Earliest Deadline First와 같은 우선순위 스케줄링을 사용하는 시스템에서 각 태스크의 마감시간 위반확률을 정확하게 계산하는 분석 방법을 제안한다. 제안하는 분석 방법은 임의의 실행 시간 분포를 갖는 태스크들에 대해서 각각의 응답 시간 분포를 정확하게 계산함으로써 개별 태스크들의 마감시간 위반확률을 결정한다. 본 논문에서는 실험을 통해서 제안하는 분석 방법의 정확성을 검증하였고, 기존의 다른 분석 방법보다 우수함을 보였다.

키워드 : 실시간 시스템, 확률적 분석, 응답 시간 분석

Abstract Recently, for real-time applications such as multimedia and signal processing, it becomes increasingly important to provide a probabilistic guarantee that each task in the application meets its deadline with a given probability. To provide the probabilistic guarantee, an analysis method is needed that can accurately predict the deadline miss probability for each task in a given system. This paper proposes a stochastic analysis method for real-time systems that use priority-driven scheduling, such as Rate Monotonic and Earliest Deadline First, in order to accurately compute the deadline miss probability of each task in the system. The proposed method accurately computes the response time distributions for tasks with arbitrary execution time distributions, and thus makes it possible to determine the deadline miss probability of individual tasks. In the paper, through experiments, we show that the proposed method is highly accurate and outperforms exisiting methods proposed in the literature.

Key words : Real-time systems, Probabilistic analysis, Response time analysis

1. 서 론

최근까지 경성 실시간 시스템(hard real-time system)에서는 주어진 태스크 집합의 스케줄가능성(schedulability)을 분석하기 위해 각 태스크가 시스템에 주기적으로 도착한다는 주기적 태스크 모델(periodic task model)[1]을 사용해왔다. 이 주기적 태스크 모델을 기반으로 개발된 스케줄가능성 분석방법들은 모두 각 태스크의 모든 작업들(jobs)이 최악 실행 시간(worst case execution time)을 요구한다고 가정하고 시스템 내에 있는 모든 작업들이 주어진 마감시간 안에 성공적으로 실행을 마치는지를 검사한다[1-3]. 스케줄가능성 분석에서 이와 같이 각 태스크의 모든 작업들이 최악 실행 시간을 요구한다고 가정하는 이유는, 경성 실시간 시스템의 경우 각각의 작업들이 어떠한 실행 시간 값을 갖든지 간에 해당 마감시간 안에 모두 성공적으로 실행을 완료한다는 결정적 보장이 요구되기 때문이다.

그러나, 멀티미디어 응용이나 신호 처리 응용들로 구성되는 연성 실시간 시스템(soft real-time system)에

· 이 논문은 2003년 두뇌한국21사업과 국가지정연구실사업에 의해서 지원되었습니다. 그리고 이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다.

† 학생회원 : 서울대학교 전기컴퓨터공학부

khkim@archi.snu.ac.kr

논문접수 : 2003년 8월 4일

심사완료 : 2003년 12월 4일

서는 마감시간에 관한 결정적 보장이 요구되지 않는다. 단지 각 태스크의 마감시간 위반률(deadline miss ratio)이 시스템 설계자가 요구하는 임계값보다 작다는 확률적 보장만 있으면 충분하다. 따라서, 연성 실시간 시스템 환경에서는 각 태스크의 모든 작업들이 최악 실행 시간을 요구한다는 가정을 완화해서 확률적인 스케줄가능성 검사가 가능하도록 새로운 분석 방법을 개발할 필요가 있다.

최근에 태스크에 속한 작업들이 가변 실행 시간을 갖는다는 가정을 기반으로 확률적인 스케줄가능성 검사를 행하는 방법들이 폭넓게 개발되고 있다. 이 방법들은 분석 과정에서 사용된 접근방식에 따라 크게 두 가지 그룹으로 나뉜다. 첫 번째 그룹은 Rate Monotonic[1]이나 Earliest Deadline First[1]와 같은 기존의 우선순위 스케줄링을 사용하는 시스템을 가능한 한 있는 그대로 분석하여 확률적 보장을 제공하는 것을 목표로 한다[4-9]. 반면에, 두 번째 그룹은 우선순위 스케줄링을 변경하거나 새로운 스케줄링 기법을 채택하여 예약 기반 스케줄링 모델을 구현하고, 이 모델을 분석함으로써 확률적 보장을 제공하는 것을 목표로 한다[10,11].

첫 번째 그룹의 한 가지 예는 Rate Monotonic이나 Deadline Monotonic[12]과 같은 고정 우선순위 스케줄링을 사용하는 시스템을 위해 제안된 확률적 시간 요구량 분석(Probabilistic Time Demand Analysis)[4]과 추계적 시간 요구량 분석(Stochastic Time Demand Analysis)[5,6]이다. 이들 두 가지 분석은 모두 태스크들이 임의의 실행 시간 분포를 갖는다는 가정을 도입하여, 전자는 상대적 마감시간(relative deadline)이 그 주기보다 작거나 같은 태스크들에 대해서, 후자는 상대적 마감시간이 그 주기보다 큰 태스크들을 포함하여 각 태스크의 마감시간 위반확률(deadline miss probability)을 분석한다. 이들 방법들은 기존의 결정적 시간 요구량 분석 방법[2,3]과 마찬가지로 시스템 내에서 관찰 가능한 태스크들의 모든 상호작용을 고려하는 것 대신에, 스케줄링의 "치명적 위기(critical instant)"로 불리는 실행 시나리오들만을 고려함으로써 마감시간 위반확률의 상한(upper bound)을 계산한다. 그러나, 이 접근법은 결과적으로 얻어진 마감시간 위반확률의 상한이 실제값으로부터 지나치게 멀어지고, 그 차이가 시스템 내에 태스크 수가 증가하거나 시스템 이용률이 커질 때 더 벌어지는 문제를 초래한다. 그리고 이 분석 방법들은 최대 시스템 이용률이 1보다 큰 시스템의 경우 계산된 마감시간 위반확률이 실제값의 상한이라는 증명을 제시하지 못한다.

첫 번째 그룹의 다른 예는 Earliest Deadline First와 같은 동적 우선순위 스케줄링을 사용하는 시스템에 적용할 수 있는 실시간 큐잉 이론(Real-Time Queueing Theory)[7,8]이다. 이 이론은 전통적인 큐잉 이론을 실시간 시스템에 확장한 것으로 Earliest Deadline First 스케줄링 알고리즘에 제한되지 않으며, 실시간 큐잉 네트워크에도 적용가능하다는 점에서 유연성이 높다. 그러나, 이 이론은 평균 시스템 이용률이 1에 가깝다고 하는 고밀도 트래픽(heavy traffic) 조건이 성립하는 시스템을 전제로 개발된 이론이기 때문에 그 조건이 성립하지 않는 시스템에는 적용이 불가능하며, 모든 태스크들의 도착간 시간들(interarrival times)과 실행 시간들이 각각 하나의 특정 분포를 따라야 한다는 제약이 있다.

첫 번째 그룹의 마지막 예는 [9]에서 제안된 분석 방법이다. 이 방법은 고정 우선순위 스케줄링 또는 동적 우선순위 스케줄링을 사용하는 시스템에 모두 적용가능하며 정확한 마감시간 위반확률을 계산한다. 그러나, 이 방법은 상대적 마감시간이 그 주기보다 작거나 같은 태스크들을 전제로 마감시간을 위반한 작업들을 모두 폐기하는 것을 가정한다. 이 분석 방법이 작업 폐기 정책을 가정하는 이유는 작업 폐기를 통해서 분석 과정에서 고려해야 하는 시스템 시간의 범위를 태스크들의 주기 값들의 최소공배수로 정의되는 하나의 하이퍼주기(hyper-period) 안으로 제한할 수 있기 때문이다. 그러나, 이 분석 방법은 작업 폐기를 가정하지 않는 일반적인 연성 실시간 시스템에 적용할 수가 없다.

한편, 두 번째 그룹에 속하는 분석 방법들은 모두 각 태스크가 마치 하나의 독립된 가상 프로세서에서 실행되는 것처럼 보이게 하는 예약 기반 스케줄링 모델을 전제로 한다. Abeni와 Buttazzo가 [10]에서 제안한 방법과 통계적 비율 단조 스케줄링(Statistical Rate Monotonic Scheduling)[11]이 이 그룹에 속한다. Abeni와 Buttazzo는 각 태스크에 대해 일정량의 예약된 프로세서 시간이 매 주기마다 제공된다고 가정하고, 통계적 비율 단조 스케줄링에서는 매 수퍼주기(superperiod: 태스크 τ_i의 수퍼주기는 우선순위가 낮은 다음 태스크, 즉 τ_{i+1}의 주기와 같으며 τ_i의 주기의 정수배이다)마다 제공된다고 가정한다. 이러한 예약 기반 스케줄링 모델은 결과적으로 각 태스크의 마감시간 위반확률을 해당 가상 프로세서 위에서 다른 태스크들과는 독립적으로 분석하는 것을 가능하게 한다는 점에서 분석에 유리하다. 그러나, 이들 분석 방법은 우선순위 스케줄링의 기본적인 규칙들을 변경하여 예약 메커니즘을 구현하거나, 전혀 새로운 예약 기반 스케줄링 알고리즘을 가정하기 때문에, 일반적인 우선순위 스케줄링을 사용하는 시스템에는 적용할 수 없다.

본 논문에서는 우선순위 스케줄링을 사용하는 모든 시스템에 적용가능하면서, 다음과 같은 점들에서 과거

분석 방법들과 구별되는 확률적 분석 방법을 제안한다.

- 제안하는 분석 방법은 분석 과정에 어떠한 제한적 가정도 도입하지 않는다. 즉 스케줄링의 치명적 위기(critical instant)와 같은 특정한 실행 시나리오나 고밀도 트래픽(heavy traffic)과 같은 특정한 시스템 조건을 전제로 하지 않는다. 이것은 제안하는 분석 방법이 확률적 또는 추계적 시간 요구량 분석 방법과 달리 모든 가능한 실행 시나리오들을 고려하여 "정확한" 마감시간 위반확률을 계산하고, 실시간 큐잉 이론과 달리 "임의의" 시스템 이용률을 갖는 시스템에 모두 적용가능함을 의미한다. 또한 분석의 편의를 위해 마감시간을 위반한 작업들을 폐기하는 정책을 가정하지 않는다.
- 제안하는 분석 방법은 Rate Monotonic, Deadline Monotonic과 같은 고정 우선순위 스케줄링 알고리즘이나 Earliest Deadline First와 같은 동적 우선순위 스케줄링 알고리즘을 사용하는 모든 실시간 시스템에 적용가능하다. 분석의 단순화를 목적으로 우선순위 스케줄링의 기본적인 규칙을 변경하지 않고, 예약 기반 스케줄링 방식을 결합하지 않는다.

제안하는 분석 방법은 시스템 내에서 발생가능한 모든 실행 시나리오를 고려하기 위해 주어진 태스크 집합의 하이퍼주기를 단위로 분석한다. 그리고, 최대 시스템 이용률이 1보다 큰 경우에 하나의 하이퍼주기에 속한 작업들의 실행이 그 다음 하이퍼주기에 속한 작업들에게 영향을 줄 수 있다는 사실을 고려하기 위해, 제안하는 분석 방법은 무한히 반복되는 하이퍼주기들을 가정하고 주어진 시스템을 마르코프 확률과정(Markov process)으로 모델링하는 접근방법을 택한다. 이것은 시스템이 안정 상태(steady state)에 있을 때 해당 하이퍼주기 안에 있는 모든 작업들의 응답 시간의 극한 분포(stationary distribution)을 계산하기 위함이다. 극한 분포는 실제 시스템에서 해당 작업에 대해 측정된 응답 시간 프로파일(response time profile)이 궁극적으로 수렴하는 분포로서, 그것으로부터 정확한 마감시간 위반확률의 계산이 가능하다. 참고로, 이 분석 방법은 태스크들이 모두 마감시간을 만족시키는 경우에 대해서는 마감시간 위반확률을 항상 0%로 제출하기 때문에, 경성 실시간 시스템에도 적용가능하다(말하자면 이 분석 방법은 기존의 결정적 스케줄가능성 분석 방법들과 항상 동일한 분석 결과를 제출한다).

본 논문은 다음과 같이 구성된다. 2절에서는 시스템 모델을 기술하고, 3절과 4절에서 본 논문에서 제안된 확률적 분석 방법을 상세히 설명한다. 5절에서는 실험을 통해서 제안하는 분석 방법의 정확성을 문헌에서 제안된 다른 분석 방법과 비교평가한다. 마지막으로 6절에서는 본 논문을 요약하고 결론을 맺는다.

2. 시스템 모델

본 논문에서 가정하는 시스템은 n개의 독립된 주기적 태스크들로 구성된다. 각 태스크 $\tau_i(1 \le i \le n)$는 주기 T_i, 위상 ϕ_i, 실행 시간 C_i, 그리고 상대적 마감시간 D_i 등 4가지 요소로 모델링된다. 태스크의 실행 시간 C_i는 주어진 확률 질량 함수(probability mass function)을 따르는 이산 확률 변수라고 가정하고, 그 확률 질량 함수는 $f_{C_i}(c)$로 표기한다(확률 질량 함수라는 용어는 확률 분포와 혼용해서 사용한다). 각 태스크의 실행 시간 분포는 자동 궤적 분석(automatic tracing analysis)[13]과 같은 측정 기반 분석 방법에 의해서 얻는다고 가정한다. 각 태스크 τ_i의 위상 ϕ_i는 주기 T_i보다 작다고 가정하고, 상대적 마감시간 D_i는 임의의 값으로 주어진다고 가정한다. 즉 D_i는 주기 T_i보다 작거나 같거나 크다.

각 태스크 집합에 대해 시스템 이용률(system utilization)은 그 집합을 구성하는 모든 태스크들의 이용률(utilization)들의 합으로 정의한다. 태스크 실행 시간은 가변적이므로, 다음과 같이 최소 시스템 이용률 $U^{\min}$, 평균 시스템 이용률 $\overline{U}$, 최대 시스템 이용률 $U^{\max}$을 정의한다:

$$U^{\min} = \sum_{i=1}^{n} C_i^{\min} / T_i,$$

$$\overline{U} = \sum_{i=1}^{n} \overline{C_i} / T_i,$$

$$U^{\max} = \sum_{i=1}^{n} C_i^{\max} / T_i$$

그리고 각 태스크 집합의 하이퍼주기(hyperperiod)를 그 집합을 구성하는 모든 태스크들의 주기값들의 최소 공배수로 정의한다.

각 태스크는 해당 주기에 따라 도착 시간이 결정되어 있는 무한한 수의 작업들(jobs)로 구성된다. 태스크 τ_i의 j번째 작업은 $J_{i,j}$로 표시하며, 그 도착 시간 $\lambda_{i,j}$은 $\phi_i + (j-1)T_i$와 같다. 각 작업 $J_{i,j}$는 태스크 τ_i의 확률 질량 함수 $f_{C_i}(c)$를 따르는 실행 시간을 가지며, 이 실행 시간은 동일한 태스크에 속한 다른 작업들 또는 다른 태스크들로부터 독립적으로 결정된다고 가정한다. 본 논문에서는 그 작업이 속한 태스크가 설명상 중요하지 않을 때 작업을 $J_{i,j}$ 대신 J_j로 표기하고, k번째 하이퍼주기에 속한 j번째 작업을 $J_j^{(k)}$로 표기한다.

본 논문에서 가정하는 스케줄링 모델은 Rate Monotonic 혹은 Deadline Monotonic과 같은 고정 우선순위

스케줄링과 Earliest Deadline First와 같은 동적 우선순위 스케줄링을 모두 포괄하는 일반적인 우선순위 스케줄링 모델이다. 단, 각 작업에 할당된 우선순위는 일단 할당된 뒤에는 바뀌지 않는다는 제약만이 존재한다(이 모델은 "작업 수준 고정 우선순위 모델"[14]로 알려져 있다). 할당된 우선순위에 따라서, 모든 작업들은 가장 우선순위가 높은 작업이 항상 먼저 실행되도록 스케줄된다. 만약, 어떤 시점에 우선순위가 같은 작업이 두 개 이상 존재하면, 먼저 도착한 작업이 먼저 스케줄된다. 작업 J_j의 우선순위는 p_j라는 우선순위"값"에 의해서 표현되는데, 우선순위값이 낮은 작업이 높은 우선순위를 갖는다.

각 작업 J_j의 응답 시간은 R_j로 표기되며, 그 확률 질량 함수는 $f_{R_j}(r)=P[R_j=r]$로 표기된다. 각 태스크의 응답 시간 분포는 그 태스크에 속한 작업들의 응답 시간 분포들, 즉 확률 질량 함수들을 평균함으로써 계산한다. 그리고 각 태스크의 마감시간 위반확률 DMP_i (deadline miss probability)는 그 태스크의 응답 시간 분포로부터 다음과 같이 계산한다.

$$DMP_i = P[R_i > D_i] = 1 - P[R_i \le D_i] \quad (1)$$

표 1은 본 논문에서 사용되는 기호들을 모두 보여준다.

표 1 논문에서 사용된 기호들

기호	설명
τ_i	i 번째 태스크
T_i	τ_i 의 주기
ϕ_i	τ_i 의 위상
C_i	τ_i 의 실행 시간(이산 확률 변수)
$f_{C_i}(c)$	τ_i 의 실행 시간 분포(C_i 의 확률 질량 함수, 즉 $f_{C_i}(c)=P[C_i=c]$)
DMP_i	τ_i의 마감시간 위반확률
J_j	주어진 하이퍼주기에서의 j 번째 작업
λ_j	J_j 의 도착 시간
R_j	J_j 의 응답 시간(이산 확률 변수)
$J_j^{(k)}$	k번째 하이퍼주기에서의 j 번째 작업
$R_j^{(k)}$	$J_j^{(k)}$ 의 응답 시간(이산 확률 변수)
$f_{R_j^{(k)}}(r)$	$J_j^{(k)}$ 의 응답 시간 분포($R_j^{(k)}$의 확률 질량 함수, 즉 $f_{R_j^{(k)}}(r)=P[R_j^{(k)}=r]$)
$W_{p_j}(t)$	시간 t에서 관찰된, J_j 보다 우선순위가 높거나 같은 작업들이 남긴 적체량
I_{p_j}	J_j 보다 늦게 도착한, J_j 보다 우선순위가 높은 작업들이 발생시키는 간섭량

3. 확률적 분석 방법

제안하는 분석 방법의 목표는 시스템이 안정 상태에 있을 때 얻어지는 각 작업들의 극한 응답 시간 분포를 정확하게 계산하는 것이다. 작업 J_j의 극한 응답 시간 분포는 다음과 같이 정의된다.

$$\lim_{k\to\infty} f_{R_j^{(k)}}(r) = f_{R_j^{(\infty)}}(r)$$

여기서 $f_{R_j^{(k)}}(r)$는 작업 $J_j^{(k)}$의 응답 시간 분포를 의미한다. 본 절에서는 임의의 하이퍼주기 k에 대해 각 작업들의 응답 시간 분포를 어떻게 계산하는지 설명하고, 다음 절에서는 하이퍼주기 k가 ∞로 접근할 때 얻어지는 극한 분포를 계산하는 방법을 설명한다. 논의를 진전시키기 위해, 일단 작업 J_j의 응답 시간 R_j가 어떻게 결정되는지를 이해하자.

작업 J_j의 응답 시간은 두 가지에 의해서 결정된다. 하나는 작업 J_j가 시스템에 도착한 순간 발견되는, J_j의 실행을 지연시키는 작업들이다. 이 작업들이 남기는 작업량을 "적체(backlog)"라고 부른다. 다른 하나는 작업 J_j가 시스템에 도착한 이후에 도착하여 J_j를 선점(preemption)할 가능성이 있는 작업들이다. 이 작업들은 "간섭(interference)"을 초래한다고 말한다. 적체 작업들과 간섭 작업들은 모두 작업 J_j보다 우선순위가 높다는 공통점이 있으므로, 본 논문에서는 작업 J_j에 대한 적체와 간섭을 각기 "p_j-적체"와 "p_j-간섭"이라고 표기한다. 따라서, 작업 J_j의 응답 시간 R_j는 다음 수식으로 표현될 수 있다.

$$R_j = W_{p_j}(\lambda_j) + C_j + I_{p_j} \quad (2)$$

여기서 $W_{p_j}(\lambda_j)$는 작업 J_j의 도착 시간 λ_j에 발견되는 p_j-적체량을 의미하고, C_j는 J_j의 실행 시간, I_{p_j}는 도착 시간 λ_j 이후에 발생할 p_j-간섭량을 의미한다. 그림1은 작업 J_j의 응답 시간을 결정하는 과정에 p_j-적체량과 p_j-간섭량이 어떻게 영향을 미치는지를 보여준다.

따라서, 본 논문에서 제안하는 분석 방법은 각 작업 J_j의 응답 시간 분포를 두 단계를 거쳐서 계산한다. 첫

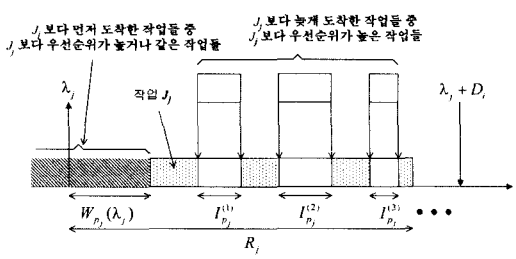


그림 1 한 작업 J_j의 응답 시간에 영향을 미치는 요인들

번째 단계는 적체 분석 단계로서 하이퍼주기에 속한 모든 작업들의 극한 p_j-적체 분포 $f_{W_{p_j}(\lambda_j)}(w)$를 계산하는 과정이고, 두 번째 단계는 간섭 분석 단계로서 앞 단계에서 얻어진 극한 p_j-적체 분포들 $f_{W_{p_j}(\lambda_j)}(w)$에 해당 작업 J_j의 실행 시간 분포 $f_{C_j}(c)$와 해당 p_j-간섭 효과 I_{p_j}를 반영함으로써 최종적으로 극한 응답 시간 분포 $f_{R_j}(r)$을 얻는 과정이다.

3.1 적체 분석 알고리즘

적체 분석은 하이퍼주기에 속한 각 작업 J_j에 대해서 J_j보다 우선순위가 높은 선행 작업들, 즉 J_j의 우선순위 값 p_j보다 작거나 같은 우선순위값을 가진 일련의 선행 작업들 $J_1, \ldots, J_{j-1}$을 전제로 한다. 그리고 그 선행 작업열을 구성하는 첫 번째 작업 J_1에 대해서는 그 도착 시간 λ_1에 관찰된 극한 p_j-적체 분포 $f_{W_{p_j}(\lambda_1)}(w)$가 주어져있다고 가정한다. 작업 J_1에 대해서 극한 p_j-적체 분포가 어떻게 계산되는지는 4절에서 설명한다. 그러면, J_j의 극한 응답 시간 분포 계산에 필요한 극한 p_j-적체 분포, 즉 J_j의 도착시간 λ_j에 관찰된 p_j-적체 분포는 아래에서 설명하는 알고리즘에 의해 계산 가능하다. 편의상, 기호 $W_{p_j}(\lambda_j)$에서 우선순위 수준을 표시하는 p_j를 생략하고 $W(\lambda_j)$로만 표기하자.

일단 모든 작업들의 실행 시간들이 확률 변수가 아닌 어떤 고정된 값들로 주어진 경우를 생각하자. 이러한 시나리오에서는, 주어진 작업열 $J_1, \ldots, J_j$에 대해 각 작업 $J_k(1 \le k < j)$의 도착 시간에 관찰된 적체량 $W(\lambda_k)$는 다음과 같이 표현된다.

$$W(\lambda_{k+1}) = \max\left(W(\lambda_k) + C_k - (\lambda_{k+1} - \lambda_k), 0\right) \quad (3)$$

따라서 일단 첫 번째 작업 J_1에 대한 적체량 $W(\lambda_1)$만 주어지면, 식 (3)을 주어진 작업열에 반복적으로 적용함으로써 각 작업들의 적체량 $W_{\lambda_2}, W_{\lambda_3}, \ldots, W_{\lambda_j}$이 모두 계산 가능하다.

그러면 식 (3)에 나타난 $W(\lambda_k)$와 C_k를 확률 변수로 취급하면서 적체 분석 알고리즘을 설명할 수 있다. 이 적체 분석 알고리즘은 그 확률 변수들이 따르는 확률 분포들에 가해지는 다음 세 가지 연산 절차로 요약된다.

1) 수식 "$W(\lambda_k) + C_k$"는 두 확률 변수 $W(\lambda_k)$와 C_k가 따르는 확률 분포들 간의 컨볼루션(convolution) 연산으로 변환된다. 예를 들어, 그림 2에서 "Convolve"라고 표시된 화살표가 그 컨볼루션 연산을 보여준다.

$$f_{W(\lambda_k)+C_k}(w') = f_{W(\lambda_k)}(w) \otimes f_{C_k}(c)$$

2) 수식 "$W(\lambda_k) + C_k - (\lambda_{k+1} - \lambda_k)$"는 위에서 얻어진 분포 $f_{W(\lambda_k)+C_k}(w)$를 $(\lambda_{k+1} - \lambda_k)$만큼 왼쪽으로 시프트(shift)시키는 연산으로 변환된다. 그림 2의 예에서 이 시프트 양은 6이다.

3) 수식 "$\max\left(W(\lambda_k) + C_k - (\lambda_{k+1} - \lambda_k), 0\right)$"는 위에서 얻어진 분포의 음수 범위의 확률값들을 모두 합해, 적체량 0일 확률값에 더하는 연산으로 변환된다. 그림 2의 예에서, 그 결과 얻어진 확률값은 $\frac{20}{54}$이다.

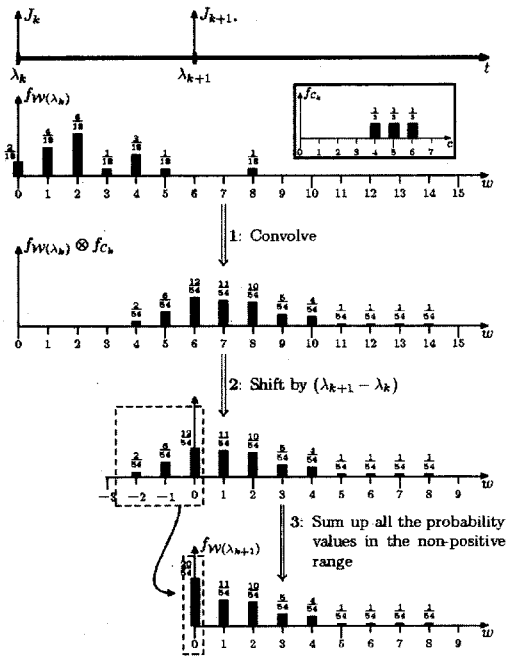


그림 2 적체 분석의 예

이 세 가지 단계들은 작업 J_k의 p_j-적체 분포 $f_{W_{p_j}(\lambda_k)}(w)$에서 작업 J_{k+1}의 p_j-적체 분포 $f_{W_{p_j}(\lambda_{k+1})}(w)$를 계산하는 과정을 정확히 표현한다. 따라서, 극한 p_j-적체 분포가 주어진다고 가정하는 작업 J_1부터 시작해서 위 세 가지 단계를 주어진 작업열에 반복적으로 적용해가면, 작업 J_j의 극한 p_j-적체 분포를 얻게 된다.

3.2 간섭 분석 알고리즘

일단 적체 분석 알고리즘에 의해 작업 J_1의 극한 적체 분포로부터 작업 J_j의 극한 적체 분포가 계산되면, J_j의 극한 응답 시간 분포는 쉽게 계산된다. 작업 J_j의 극한 응답 시간 분포는 극한 p_j-적체 분포 $f_{W_{p_j}(\lambda_j)}(w)$와 그 작업의 실행 시간 분포 $f_{C_j}(c)$를 컨볼루션함으로써 계산된다. 이렇게 얻어진 극한 분포는 그 작업 J_j가 다른 태스크들에 의해서 선점되지 않는 태스크로부터 나온 경우 그 작업의 최종 응답 시간 분포가 된다. 그러

나, 작업 J_j가 선점될 수 있는 태스크로부터 나온 작업이고, J_j를 뒤따르는 우선순위가 높은 작업들이 존재하는 경우에는, 그 우선순위가 높은 작업들이 야기하는 p_j-간섭 효과 I_{p_j}를 반영하는 새로운 극한 응답 시간 분포 $f_{R_j}(r)$를 얻어야 한다.

간섭 분석을 수행하기 위해서 일단 작업 J_j를 뒤따르는 작업들 중에서 우선순위가 높은 작업들을 $J_{j+1}, J_{j+2}, \ldots, J_{j+k}, \ldots$로 놓자. 그리고 설명의 편의를 위해서 작업 J_{j+k}의 절대적인 도착 시간을 표현하는 기호 λ_{j+k}를, 작업 J_j의 도착 시간을 기준으로 하는 상대적인 도착 시간으로 이해하자. 즉, $\lambda_{j+k} \leftarrow (\lambda_{j+k} - \lambda_j)$. 이 작업들은 작업 J_j보다 늦게 도착한 작업들의 우선순위 값을 J_j의 우선순위값과 비교함으로써 쉽게 결정된다.

그러면 적체 분석 알고리즘을 설명했던 경우와 마찬가지로, 관련된 모든 작업들의 실행 시간들이 확률 변수가 아닌 어떤 고정된 값들로 주어진 경우를 생각하자. 이 경우 작업 J_j의 응답시간 R_j는 다음 절차에 의해서 계산된다.

$$\begin{aligned} & R_j = W_{p_j}(\lambda_j) + C_j; \qquad k = 1 \\ & \text{while } R_j > \lambda_{j+k} \\ & R_j = R_j + C_{j+k}; \qquad k = k+1 \end{aligned} \tag{4}$$

위 while 루프에서 전체 루프의 반복 횟수를 나타내는 변수 k는, 계산 중인 응답 시간 R_j가 어떤 작업 J_{j+k+1}의 도착 시간에 도달하지 못하는 경우가 확인될 때 최종적으로 결정되는 값이다. 이 때 최종 응답 시간 R_j는 $W_{p_j}(\lambda_j) + C_j + \sum_{i=1}^{k} C_{j+i}$이다.

그러면 식 (4)에 나타난 R_j, C_j, $W_{p_j}(\lambda_j)$를 확률 변수로 취급하면서 간섭 분석 알고리즘을 설명할 수 있다. 이 간섭 분석 알고리즘은 그 확률 변수들이 따르는 확률 분포들에 가해지는 다음과 같은 연산 절차로 요약된다.

1) 수식 "$R_j = W_{p_j}(\lambda_j) + C_j$"는 $f_{R_j}(r) = f_{W_{p_j}(\lambda_j)}(w) \otimes f_{C_j}(c)$로 변환된다. 이 때 얻어진 응답 시간 분포 $f_{R_j}(r)$은 $(0, \lambda_{j+1}]$ 구간에서 유효하다. 예를 들어, 그림 3에서 첫 번째 컨볼루션 $\otimes f_{C_j}$이 해당 연산을 보여주고 있다.
2) $P[R_j > \lambda_{j+k}] > 0$이 성립할 때, 수식 "$R_j = R_j + C_{j+k}$"는 위에서 얻어진 응답 시간 분포 $f_{R_j}(r)$의 (λ_{j+k}, ∞) 구간에서 정의된 부분 분포와 실행 시간 분포 $f_{C_{j+k}}(c)$ 사이의 컨볼루션으로 변환된다. 이 때 얻어진 분포는 $(\lambda_{j+k}, \lambda_{j+k+1}]$ 구간에서 유효하다. 그리고 $P[R_j > \lambda_{j+k}] = 0$인 경우가 처음으로 발견될 때, while 루프는 종료된다. 그림 3의 예에서, 이 과정은 연속된 컨볼루션 $\otimes f_{C_{j+1}}$, $\otimes f_{C_{j+2}}$, ... 등으로 설명되고 있다 (이 예에서는 J_j보다 우선순위가 높은 작업들로서 세 개의 작업 J_{j+1}, J_{j+2}, J_{j+3}이 존재한다고 가정하였고 이 작업들이 모두 동일한 실행 시간 분포를 갖는다고 가정하였다).

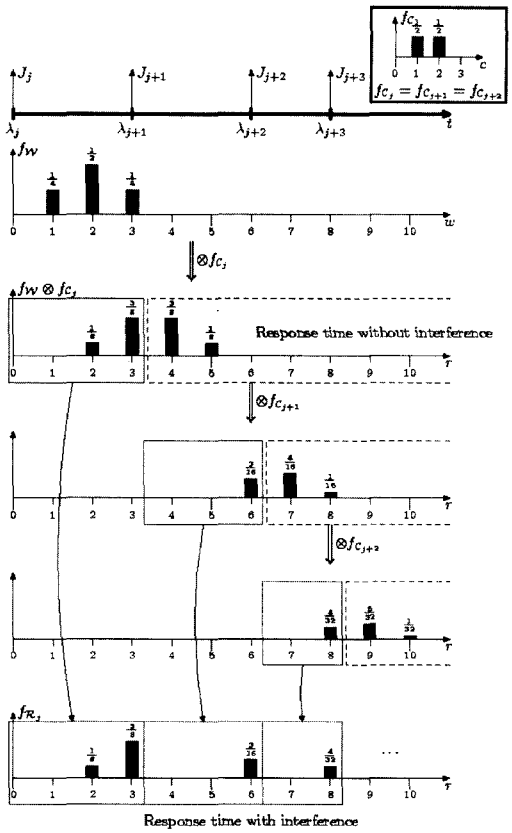


그림 3 간섭 분석의 예

참고로, 간섭 분석 과정에서 작업 J_j보다 우선순위가 높은 작업들의 수는 무한히 많을 수 있다는 사실에 주의하기 바란다. 그러나, 마감시간 위반확률 $P[R_j > D_j]$은 $[0, D_j]$에서 정의되는 "부분" 응답 시간 분포만으로도 계산이 가능하기 때문에(즉, $P[R_j > D_j] = 1 - P[R_j \le D_j]$), 간섭 분석 알고리즘에서 실제적으로 고려해야 하는 우선순위가 높은 작업들의 집합은 $(\lambda_j, \lambda_j + D_j)$ 구간에 도착한 작업들로 제한된다. 그러므로, 간섭 분석 알고리즘의 while 루프는 λ_{j+k}가 D_j보다 클 때 종료가능하다. 그림 3의 예에서, 만일 J_j의 상대적 마감시간 D_j가 7이라면, 마감시간 위반확률은 $P[R_j > D_j] = 1 - 11/16 = 5/16$이다.

3.3 적체 의존성 관계

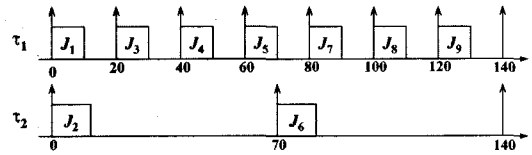
앞에서 적체 분석 알고리즘을 기술할 때 다음과 같은

가정이 있었다: 적체 분석 대상이 되는 각 작업 J_i에 대해, J_i보다 우선순위가 크거나 같은 선행 작업들의 열이 주어지고, 그 선행 작업열의 첫 번째 작업에 대해서는 극한 적체 분포가 또한 주어진다. 본 소절에서는 임의의 하이퍼주기에 속한 모든 작업들에 대해서 그 선행 작업열들을 어떻게 유도하는지 설명한다. 이 유도 과정은 "적체 의존성 나무(backlog dependency tree)"라는 자료 구조를 결과적으로 만들어내는데, 이 나무는 뿌리 노드에 해당하는 단 하나의 작업에 대한 극한 적체 분포만 주어지면, 나머지 모든 작업들의 극한 p_i-적체 분포들을 한꺼번에 계산할 수 있다는 특징이 있다. 이 때, 앞에서 기술한 적체 분석 알고리즘은 그 나무를 구성하는 모든 노드들을 순회하면서 적용된다. 적체 의존성 나무는 하이퍼주기에 속한 각각의 작업들에 대해서 요구되는 극한 적체 분포의 계산을, 궁극적으로 단 하나의 작업에 대한 극한 적체 분포의 계산으로 단순화하기 때문에 큰 장점이 된다. 뿌리 노드에 해당하는 작업에 대한 극한 적체 분포를 어떻게 계산하는지는 4절에서 다룬다.

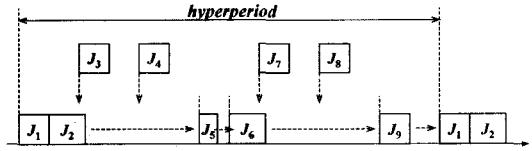
작업들 간의 적체 의존성 관계를 파악하기 위해서, 일단 하나의 하이퍼주기에 속한 모든 작업들을 "지상 작업(ground job)"과 "비지상 작업(non-ground job)"으로 분류한다. 지상 작업은 어떤 선행 작업들보다도 우선순위가 낮은 작업으로 정의한다. 즉, 작업 J_i는 도착시간 λ_k가 λ_i보다 앞서는 모든 작업들 J_k에 대해 $p_i \geq p_k$이면 지상 작업이다. 반면에 비지상 작업은 지상 작업이 아닌 작업으로 정의한다. 지상 작업은 정의상 그 p_i-적체량이 그 도착 시간에 관찰된 시스템 내 총 적체량 항상 같다는 성질이 있다(어떤 시점에서 관찰한 시스템 내 총 적체량을 "시스템 적체량"이라고 부르고, 그 시간이 t일 때 $W(t)$로 표시한다. 따라서, 지상 작업 J_i은 항상 $W_{p_i}(\lambda_i) = W(\lambda_i)$이다). 그리고나서, 각 작업들 사이에 존재하는 적체 의존성 관계를 "기저 작업(base job)"이라는 개념을 기반으로 찾아낸다. 기저 작업은 각 작업 J_i에 대해서, J_i에 선행하고 그 우선순위가 J_i보다 높거나 같은 마지막 지상 작업으로 정의한다. 각 작업 J_i에 대해서 그 기저 작업이 의미하는 바는, 작업 J_i의 p_i-적체 분포가 그 기저 작업의 적체 분포로부터 곧장 계산가능하다는 것이다. 결과적으로 여기서 유도하는 적체 의존성 나무는, 모든 작업들의 극한 p_i-적체 분포를 그 기저 작업들의 극한 시스템 적체 분포로부터 계산해내기 위한 것이다. 이 때, 극한 시스템 적체 분포는 그 적체 의존성 나무의 뿌리 노드에 해당하는 단 하나의 지상 작업에 대해서만 계산되면 충분하다.

그림 4(a)에서 제시된 태스크 집합을 예로 들어보자. 이 태스크 집합은 상대적 마감시간이 각기 그 주기와 같은 두 개의 태스크들로 구성된다. 두 태스크들의 위상은 모두 0이다. 이 태스크들이 EDF(Earliest Deadline First)에 의해서 스케줄된다고 가정한다.

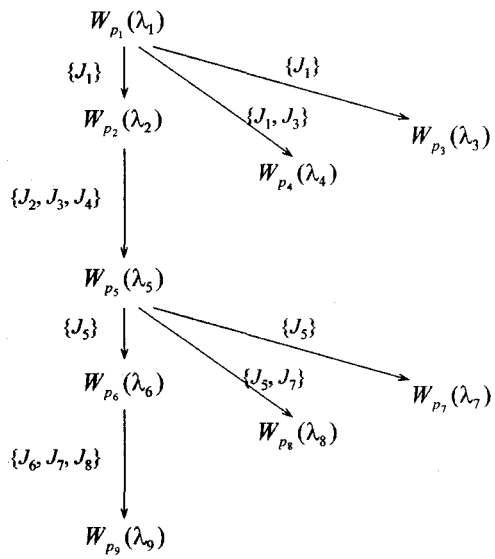
이 예의 경우, 위에서 설명한 작업 분류에 따르면, 5개의 지상 작업들 J_1, J_2, J_5, J_6, J_9와 4개의 비지상 작업들 J_3, J_4, J_7, J_8이 존재한다(그림 4(b) 참조). 즉, 각 작업들에 대해 어떠한 실행 시간 값을 가정한다고 하더라도, 그 다섯 개의 지상 작업들에 대해서는 항상 $W_{p_i}(\lambda_i) = W(\lambda_i)$가 성립한다(작업 J_1과 J_2는 도착시간 λ_1과 λ_2이 같지만, 적체의 정의상 $W(\lambda_2)$는 J_1의 실행 시간을 포함하는 것으로, $W(\lambda_1)$는 포함하지 않는 것으



(a) 태스크 집합



(b) 지상 작업들과 비지상 작업들



(c) 적체 의존성 나무

그림 4 적체 의존성 나무의 유도 예

로 간주되기 때문에, J_1과 J_2에 대해서도 $W_{p_i}(\lambda_j) = W(\lambda_j)$는 성립한다). 반면에, 비지상 작업들에 대해서는 $W_{p_i}(\lambda_j) \neq W(\lambda_j)$이다. 예를 들어, 작업 J_4의 경우, 작업 J_2가 J_4의 도착 시간까지 실행하고 있으면, 시스템 적체량 $W(\lambda_4)$는 J_2의 남은 실행 시간을 포함하지만 작업 J_4의 관점에서의 p_4-적체량 $W_{p_4}(\lambda_4)$은 포함하지 않으므로, $W_{p_4}(\lambda_4) \neq W(\lambda_4)$이다.

그러면 분류된 작업들을 기반으로 모든 작업들 J_i에 대해서 기저 작업을 찾아보자. 그림4의 예에서 비지상 작업들의 기저 작업들을 먼저 찾아보면, J_3과 J_4의 경우 모두 J_1이고, J_7과 J_8의 경우 모두 J_5이다. 한편, 지상 작업들의 기저 작업들은 항상 바로 직전에 선행하는 지상 작업들이므로, J_2의 기저 작업은 J_1이고, J_5의 기저 작업은 J_2이고, ..., J_9의 기저 작업은 J_6이다. 이 예로부터, 각 작업 J_i에 대해서 그 p_i-적체량은 J_i의 기저 작업의 시스템 적체량에 의존적임이 확인된다. 비지상 작업 J_3의 경우, 그 p_3-적체량은 기저 작업 J_1의 적체량에 의존적이며, 따라서 J_1의 실행 시간만을 고려하면 J_1의 적체량으로부터 계산할 수 있다. 마찬가지로, 비지상 작업 J_4의 경우 그 p_4-적체량은 기저 작업 J_1의 적체량으로부터 계산 가능한데, 이 경우에는 λ_1과 λ_4 사이에 도착한, J_4보다 우선순위가 높은 작업들(즉 J_3)을 추가적으로 고려하기만 하면 된다.

이것은 결과적으로 각 작업들 사이에 존재하는 모든 적체 의존성 관계는 그림 4(c)에 보여진 것과 같이 하나의 나무로 표현될 수 있다는 것을 의미한다. 그림 4(c)에서 각 노드는 작업 J_i의 p_i-적체 $W_{p_i}(\lambda_i)$를 표시하고, 각 링크 $W_{p_k}(\lambda_k) \rightarrow W_{p_i}(\lambda_i)$는 $W_{p_k}(\lambda_k)$와 $W_{p_i}(\lambda_i)$ 사이의 의존성 관계를 표시한다. 그리고 그 링크에 붙여진 레이블은 $W_{p_k}(\lambda_k)$로부터 $W_{p_i}(\lambda_i)$를 계산해낼 때 추가로 고려해야 할 중간 작업들의 집합을 표시한다.

적체 의존성 나무의 중요성은 그것이 하나의 하이퍼주기에 속한 각 작업들의 p_i-적체를 계산할 때 요구되는 선행 작업열을 모두 포함하고 있다는 데 있다. 예를 들어, 그림4(c)의 예에서, $W_{p_1}(\lambda_1)$로부터 $W_{p_8}(\lambda_8)$까지의 경로를 고려해 보자. 이 경로상에 발견되는 모든 레이블들 $J_1, J_2, J_3, J_4, J_5, J_7$은 $W_{p_1}(\lambda_1)$로부터 $W_{p_8}(\lambda_8)$을 계산할 때 고려해야 할 J_8의 선행 작업열을 정확하게 결정한다. 즉, 작업열 $J_1, J_2, J_3, J_4, J_5, J_7$은 J_8에 선행하는 작업들 중에서 J_8보다 우선순위가 높거나 같은 모든 작업들을 담고 있다. 이 성질은 적체 의존성 나무의 각 노드에 대해서 항상 성립한다. 그러므로, 적체 의존성 나무의 뿌리에 해당하는 극한 시스템 적체 분포 $f_{W_{p_i}(\lambda_1)}(w)$가 주어지면, 나머지 작업들의 극한 p_i-적체 분포들은 적체 의존성 나무를 순회하면서 앞에서 설명된 적체 분석 알고리즘을 적용하여 모두 계산 가능하다.

3.4 적체 의존성 관계의 일반성

본 소절에서는 적체 의존성 관계가 작업 수준 고정 우선순위 스케줄링 모델[14]에 대해 일반적으로 성립한다는 것을 보인다. 일단, 지상 작업들의 존재성을 먼저 증명하고, 고정 우선순위 스케줄링과 동적 우선순위 스케줄링을 구별하면서, 기저 작업들의 존재성을 증명한다.

정리 1. 집합 S를 고정 주기 T_i와 위상 ϕ_i를 갖는 주기적인 태스크들 $\tau_1, \ldots, \tau_n$의 집합이라고 가정하자. 그리고, T_H를 집합 S의 하이퍼주기의 길이라 하자. 시간 t에서 그 첫 번째 하이퍼주기가 시작하는 하이퍼주기들의 열을 고려할 때, 임의의 $t(0 \leq t < T_H)$에 대해 하이퍼주기 $[t+kT_H, t+(k+1)T_H)$에 속한 모든 작업들의 상대적인 우선순위가 그 다음 하이퍼주기 $[t+(k+1)T_H, t+(k+2)T_H)$ $(k=0,1,\ldots)$에 속한 모든 작업들의 상대적인 우선순위와 일치하면, 다음과 같은 사실들이 성립한다.

(1) 임의의 하이퍼주기에는 적어도 하나의 지상 작업이 존재한다.

(2) 모든 하이퍼주기들에는 동일한 지상 작업들의 집합이 존재한다.

증명: 부록을 참조하기 바란다.

정리 1의 증명에 이용된 핵심적인 사실은 임의의 하이퍼주기 내에서 최대 우선순위값을 가진 작업이 어떤 선행 작업보다도 우선순위가 항상 낮다는 것이다. 이 사실로부터 임의의 하이퍼주기에서 모든 지상 작업들을 다음과 같이 찾아낼 수 있다. 우선, 임의의 하이퍼주기를 선택하고, 그 안에서 최대 우선순위값을 가진 어떤 작업 J_l를 하나 찾는다. 그러면, 이 작업이 곧 지상 작업이다. 그리고나서, 작업 J_l의 도착 시간 λ_l에서 시작하는 하이퍼주기, 즉 $[\lambda_l, \lambda_l + T_H)$를 새로 고려하고, 이 안에서 나머지 지상 작업들을 찾는다. 이 과정에서 그 하이퍼주기에 속한 각 작업 J_i에 대해 J_i이 선행 작업들 $J_j, \ldots, J_{l-1}$보다 우선순위값이 가장 크다는 것이 확인되면 지상 작업으로 결정한다.

다음 정리는 EDF와 같은 동적 우선순위 스케줄링을 사용하는 시스템에 대해서 기저 작업들의 존재성을 증명한다.

정리 2. 정리1에서 정의된 시스템에 대해서, 하이퍼주기 $n(\geq 2)$에 속한 임의의 작업 $J_j^{(n)}$의 우선순위값 $p_j^{(n)}$

이 어떤 양의 상수 Δ를 이용하여 항상 $p_j^{(n)} = p_j^{(n-1)} + \Delta$로 표현가능하면, 어떤 작업도 그 기저 작업을 선행 지상 작업들 중에서 항상 찾을 수 있다.

증명: 부록을 참조하기 바란다.

EDF의 경우, 상수 Δ는 T_H와 같은데, 그 이유는 각 작업에 할당된 우선순위값이 그 작업의 절대적 마감시간이기 때문이다.

참고로, 본 논문에서 제안된 분석 방법에 대해, 어떤 비지상 작업 J_j의 기저 작업이 동일한 하이퍼주기(n번째 하이퍼주기라고 하자)에서 발견되는지, 아니면 어떤 선행 하이퍼주기(k번째 하이퍼주기라고 하자. $k<n$)에서 발견되는지는 중요하지 않다. 만약 그 기저 작업 J_i가 어떤 선행 하이퍼주기에서 발견된다면, 이것은 단지 J_j의 p_j-적체 분포를 계산하는데 필요한 J_i부터 J_j까지의 작업열이 하이퍼주기 k부터 n까지 걸쳐있다는 사실을 의미할 뿐이다. 이러한 경우에도 하이퍼주기 k부터 뻗어나오는 적체 의존성 나무를 상정할 수 있고, 그 뿌리에 해당하는 극한 적체 분포의 계산이 항상 가능하기 때문에(4절에서 설명한다), 그러한 비지상 작업 J_j의 p_j-적체 분포의 계산은 J_i부터 J_j까지 유도된 작업열로부터 항상 가능하다.

다음 정리는 EDF에 대해서 기저 작업의 탐색 범위가 최대 얼마나 커질 수 있는지를 설명한다.

정리 3. EDF로 스케줄링되는 시스템에서, 태스크들의 상대적 마감시간들의 최대값을 $D^{\max}$라 하자. 즉, $D^{\max} = \max_{1 \le i \le n} D_i$이다. 그러면 어떤 작업 J_j의 기저 작업은 항상 $[\lambda_j - (D^{\max} + T_H), \lambda_j]$ 구간에서 발견된다. 바꾸어 말하면, 기저 작업의 탐색 범위는 $D^{\max} + T_H$보다 작거나 같다.

증명: 부록을 참조하기 바란다.

정리 3은 만일 $D^{\max} < T_H$인 경우를 고려한다면(그 반대의 경우는 현실적으로 드물기 때문에) 한 하이퍼주기에 속한 모든 작업들의 기저 작업들을 찾기 위해서는 기껏해야 하나의 선행 하이퍼주기만을 탐색하면 충분하다는 뜻이다.

반면에, RM(Rate Monotonic)이나 DM(Deadline Monotonic)과 같은 고정 우선순위 스케줄링을 사용하는 시스템에서는, 비지상 작업들의 기저 작업들을 지상 작업들 안에서 찾는 것이 불가능하다(정리 2는 고정 우선순위 시스템의 경우 $\Delta = 0$이므로 성립하지 않는다). 고정 우선순위 시스템에서는, 우선순위가 가장 낮은 태스크 τ_n으로부터 나온 모든 작업들이 지상 작업들로 분류되고, 나머지 태스크들로부터 나온 모든 작업들이 비지상 작업들로 분류된다. 이 경우, 어떠한 선행 지상 작업들도 임의의 비지상 작업보다 낮은 우선순위를 갖기 때문에, 모든 선행 하이퍼주기들을 탐색한다고 할지라도, 고려중인 비지상 작업보다 우선순위가 높은 지상 작업, 즉 기저 작업을 찾는 것이 불가능하다.

그러나, 이러한 사실은 본 논문에서 제안하는 분석 방법이 적용 불가능하다는 것을 의미하지 않는다. 각 고정 순위 수준을 따로따로 고려함으로써, 모든 작업들의 적체 분포들을 계산하는 것이 가능하다. 말하자면, 각 우선순위 수준 $i = 1, 2, \ldots, n$에 대해서 태스크 부분 집합 $\tau_1, \ldots, \tau_i$를 고려하고, 태스크 τ_i로부터 나온 모든 작업들의 적체 분포들을 계산하는 것은 가능하다. 이 경우 태스크 부분 집합 $\tau_1, \ldots, \tau_i$만으로 구성된 시스템을 가정하면, 태스크 τ_i로부터 나온 모든 작업들은 지상 작업들로 분류되고, 적어도 그 지상 작업들에 대해서는 기저 작업들을 찾는 것이 항상 가능하기 때문이다. 따라서 첫 번째 지상 작업에 대한 극한 적체 분포만 주어지면, 지상 작업들 간에 성립하는 적체 의존성 관계를 이용하여 나머지 모든 지상 작업들의 극한 응답 시간 분포들을 계산할 수 있다.

그러므로, 동적 우선순위 시스템과 고정 우선순위 시스템 간의 유일한 차이점은, 전자의 경우 모든 작업들의 적체 분포들이 단 하나의 적체 의존성 나무로부터 한꺼번에 계산될 수 있는 것에 반해, 후자의 경우 n개의 우선순위 수준에 대해서 결정되는 각각의 태스크 부분 집합들에 대한 반복 분석(iterative analysis)으로 모든 작업들의 적체 분포들이 계산된다는 점이다. 바꾸어 말하면, 고정 우선순위 시스템은 각기 지상 작업들로만 구성되는 n개의 적체 의존성 리스트를 통해서 해결 가능하다.

4. 극한 적체 분포 분석

본 절에서는 앞 절에서 설명된 적체 의존성 나무의 뿌리 노드(혹은 적체 의존성 리스트의 머리 노드)로 사용될 지상 작업의 극한 적체 분포를 분석하는 방법을 설명한다. 지상 작업의 적체 분포는 같은 시간에 관찰된 시스템 적체 분포와 동일하므로, 지상 작업의 극한 적체 분포 분석에서는 지상 작업 J_j 앞에서 시스템에 도착한 모든 무한한 수의 작업들 $\ldots, J_{j-3}, J_{j-2}, J_{j-1}$을 고려한다.

4.1절에서는 극한 시스템 적체 분포의 존재성을 설명하고, 4.2절과 4.3절에서는 그 극한 분포를 계산하는 정해법과 근사해법들을 기술한다. 마지막으로 4.4절에서는, 무한히 긴 극한 적체 분포를 정해법으로 계산하고 생성하는 경우, 한정된 길이의 분포만을 다룰 수 있는 디지털 컴퓨터에서 사용하기 위해, 어떻게 안전하게 극한 적

체 분포를 절단하여 사용할 것인지를 설명한다.

4.1 극한 적체 분포의 존재성

다음 정리는 평균 시스템 이용률 $\overline{U}$가 1보다 작으면 극한 분포가 존재한다는 일반적인 큐잉 시스템의 성질[15]을 본 논문에서 가정하는 시스템의 시스템 적체 분포에 대해 풀어 쓴 것이다.

정리 4. 지상 작업 J_j의 도착 시간에서 그 첫 번째 하이퍼주기가 시작하는 무한한 수의 하이퍼주기들의 열을 고려하자. 그리고, 지상 작업 $J_j^{(k)}$의 도착 시간, 즉 하이퍼주기 k의 시작 시점에 관찰된 시스템 적체량 B_k의 분포를 $f_{B_k}(w)$라고 하자. 이 때, 만일 평균 시스템 이용률 $\overline{U}$가 1보다 작으면, 다음과 같은 시스템 적체량 B_k의 극한 분포 $f_{B_\infty}(w)$가 존재한다.

$$\lim_{k\to\infty} f_{B_k}(w) = f_{B_\infty}(w)$$

최대 시스템 이용률 $U^{\max} \le 1$인 특수한 경우에 대해서는, 시스템 적체 분포 $f_{B_k}(w)$가 모든 k에 대해서 동일하다[3]. 즉, $f_{B_1}(w) = \ldots = f_{B_\infty}(w)$. 이 경우에는, 극한 시스템 적체 분포 $f_{B_\infty}(w)$는 지상 작업 J_j의 도착 시간 이전에 도착한 "한정된" 수의 선행 작업들만을 고려함으로써 계산된다. 말하자면, $[0, \lambda_j)$ 구간에 도착한 작업들을 대상으로 시간 0에서의 시스템 적체량을 0이라고 가정하면서 적체 분석 알고리즘을 적용하여 작업 J_j의 도착 시간에 관찰된 시스템 적체 분포를 계산하면 된다. 그러므로, $U^{\max} \le 1$인 경우에 대해서는 다음에서 설명할 극한 적체 분포 분석을 수행할 필요가 없다.

4.2 정해법(Exact solution method)

$U^{\max} > 1$인 일반적인 경우에 대해서 극한 시스템 적체 분포 $f_{B_\infty}(w)$를 정확하게 계산하기 위해 일단 확률 변수들의 열 $B_0, B_1, \ldots, B_k, \ldots$을 마르코프 확률과정(Markov process)으로 모델링한다. 이를 위해 확률 변수 B_k의 분포를 조건부 확률을 이용하여 B_{k-1}의 분포로 표현하자.

$$P[B_k = x] = \sum_y P[B_{k-1} = y] P[B_k = x | B_{k-1} = y] \quad (5)$$

식 (5)에서 조건부 확률 $P[B_k = x | B_{k-1} = y]$은 k에 무관하다는 것을 알 수 있다. 이는 매 하이퍼주기마다 실행 시간 분포가 변하지 않는 동일한 작업열이 도착한다는 사실로부터 추론 가능하다. 즉, $P[B_k = x | B_{k-1} = y] = P[B_1 = x | B_0 = y]$이다. 이 사실로부터 확률 변수 B_k의 분포는 오직 B_{k-1}의 분포에만 의존하고, $B_{k-2}, B_{k-3}, \ldots$의 분포들에는 의존하지 않는다는 것이 확인된다. 그러므로, 확률 변수들의 열 $B_0, B_1, \ldots, B_k, \ldots$은 마르코프 확률과정이다[15]. 그러면 식 (5)를 행렬 형태로 다음과 같이 고쳐 쓰자.

$$\boldsymbol{b_k} = \boldsymbol{P} \cdot \boldsymbol{b_{k-1}} \quad (6)$$

여기서 $\boldsymbol{b_k}$는 열벡터 $[P[B_k=0], P[B_k=1], \ldots]^T$, 즉 B_k의 확률 분포를 의미하고, $\boldsymbol{P}$는 다음과 같이 정의된 전이 확률 $\boldsymbol{P}(x, y)$들로 구성된 마르코프 행렬이다.

$$\boldsymbol{P}(x, y) = b_y(x) = P[B_k = x | B_{k-1} = y] = P[B_1 = x | B_0 = y]$$

따라서, 극한 시스템 적체 분포 $f_{B_\infty}(w)$의 정확한 해 $\boldsymbol{\pi} = [P[B_\infty = 0], P[B_\infty = 1], \ldots]^T$를 계산하는 문제는 평형 방정식 $\boldsymbol{\pi} = \boldsymbol{P} \cdot \boldsymbol{\pi}$를 푸는 문제로 귀결된다.

그러나 평형 방정식 $\boldsymbol{\pi} = \boldsymbol{P} \cdot \boldsymbol{\pi}$은 무한히 많은 수의 선형 방정식들로 구성되기 때문에, 이것을 직접 푸는 것은 불가능하다. 이론적으로 k가 ∞에 접근할 때, $U^{\max} > 1$이면 시스템 적체량 B_∞은 무한히 길어진다. 이것은 극한 분포 $\boldsymbol{\pi}$가 무한히 긴 길이를 가지고 있으며, 따라서 마르코프 행렬 또한 무한히 커지는 행렬임을 의미한다. 따라서, 이 문제를 풀기 위해서는 그 무한한 수의 선형 방정식들과 동치 관계에 있는 유한한 수의 선형 방정식들을 유도하여 문제를 풀어야만 한다. 이것은 아래에서 보여지는 마르코프 행렬 $\boldsymbol{P}$의 규칙성을 활용하면 가능하다.

$$\boldsymbol{P} = \begin{pmatrix}
b_0(0) & b_1(0) & b_2(0) & \cdots & b_r(0) & 0 & 0 & 0 \\
b_0(1) & b_1(1) & b_2(1) & \cdots & b_r(1) & b_r(0) & 0 & 0 \\
b_0(2) & b_1(2) & b_2(2) & \cdots & b_r(2) & b_r(1) & b_r(0) & 0 \\
\vdots & \vdots & \vdots & \cdots & \vdots & b_r(2) & b_r(1) & \ddots \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots & b_r(2) & \ddots \\
b_0(m_r) & b_1(m_r) & b_2(m_r) & \cdots & b_r(m_r) & \vdots & \vdots & \ddots \\
0 & 0 & 0 & \cdots & 0 & b_r(m_r) & \vdots & \ddots \\
0 & 0 & 0 & \cdots & 0 & 0 & b_r(m_r) & \ddots \\
0 & 0 & 0 & \cdots & 0 & 0 & 0 & \ddots \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}$$

마르코프 행렬 $\boldsymbol{P}$의 각 열 y는 하이퍼주기의 시작 시점의 시스템 적체량이 y일 때 그 하이퍼주기의 끝 시점에 관찰된 시스템 적체량의 분포를 의미한다. 이 시스템 적체량 분포는, 하이퍼주기의 시작 시점의 시스템 적체량을 y라고 가정하고, 3.1절에서 기술하였던 적체 분석 알고리즘을 하이퍼주기 안에 도착한 모든 작업들에게 적용함으로써 계산된다. 이 때 $r, r+1, r+2, \ldots$번째 열들에서는 위와 같은 규칙성이 발견되는데, 이것은 어떤 상수 r이 존재하여 시작 시점의 시스템 적체량이 r보다 같거나 큰 경우에 그 하이퍼주기 끝에서 관찰된 시스템 적체 분포들이, 비록 시스템 내에서 하나의 시간 단위(time unit)만큼 시프트되기는 하지만, 모두 동일하다는 것을 의미한다. 이 상수 r은 하나의 하이퍼주기 안에서 발생 가능한 모든 시스템 유휴 시간(idle time)들의 총합이다. 즉,

$$r = T_H(1-U^{\min}) + W^{\min} \quad (7)$$

여기서 $W^{\min}$은 하이퍼주기의 시작 시점의 시스템 적체량이 0이고, 모든 작업들이 최소 실행 시간을 가질 때 하이퍼주기 끝에서 관찰된 시스템 적체량이다(시스템에 도착하는 작업 부하가 하이퍼주기 끝에 집중되는 특수한 경우를 제외하면, $W^{\min}$의 값은 일반적으로 0이다).

따라서, 하이퍼주기의 시작 시점의 시스템 적체량이 r이라는 사실은 전체 하이퍼주기에 유휴 시간이 전혀 없다는 것을 의미하게 되고, 하이퍼주기 끝에서 관찰된 시스템 적체량 분포는 모든 작업들의 실행 시간 분포들을 컨볼루션한 결과가 된다(정확히 표현하자면, 그 컨볼루션의 결과 얻은 분포를 (T_H-r)만큼 왼쪽으로 시프트한 분포이다). 그러나 이러한 사실은 하이퍼주기의 시작 시점의 시스템 적체량이 r보다 큰 모든 경우들에 대해서도 동일하게 성립하기 때문에, 하이퍼주기 끝 시점에서 관찰된 시스템 적체 분포는 시작 시점의 시스템 적체량이 r인 경우에 얻은 시스템 적체 분포를 조금씩 오른쪽으로 시프트한 것과 결과적으로 같아지게 된다.

위에서 기술한 규칙성을 이용하면 다음과 같이 극한 분포 $\boldsymbol{\pi}$를 구하는 문제를 풀 수가 있다. 우선, 무한한 수의 선형 방정식들로 구성된 행렬식 $\boldsymbol{\pi} = \boldsymbol{P}\cdot\boldsymbol{\pi}$에서 행렬 $\boldsymbol{P}$의 0번째 행부터 m_r번째 행까지에 해당하는 (m_r+1)개의 선형 방정식들을 선택한다 (m_r은 마르코프 행렬 $\boldsymbol{P}$에서 r번째 열의 0이 아닌 마지막 원소의 인덱스이다). 이 (m_r+1)개의 선형 방정식들에 나타나는 미지수들의 수는 $(r+m_r+1)$개, 즉 $\pi_0, \pi_1, \ldots, \pi_{r+m_r}$이다. 그리고나서, $x\to\infty$이면 $\pi_x\to 0$이라는 사실을 이용해서 r개의 선형 방정식들을 추가로 유도한다. 이렇게 추가로 유도된 방정식들과 먼저 선택된 방정식들을 모두 합치면, $(r+m_r+1)$개의 미지수들을 갖는 $(r+m_r+1)$개의 선형 방정식들의 집합이 된다. 이 유도 과정은 다음과 같다. 우선 마르코프 행렬 $\boldsymbol{P}$의 $(m_r+1), (m_r+2), \ldots$ 행들로부터 다음 행렬식을 유도한다.

$$\boldsymbol{Q}_{x+1} = \boldsymbol{A}\cdot\boldsymbol{Q}_x \qquad x \geq m_r+1 \quad (8)$$

여기서 $\boldsymbol{Q}_x$와 $\boldsymbol{A}$는 다음과 같이 정의된다.

$$\boldsymbol{Q}_x = [\pi_{x-d}, \pi_{x-d+1}, \ldots, \pi_{x-1}, \pi_x, \pi_{x+1}, \ldots, \pi_{x-d+m_r-1}]^T,$$
$$(d = m_r + r)$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -\frac{b_r(m_r)}{b_r(0)} & -\frac{b_r(m_r-1)}{b_r(0)} & \cdots & -\frac{b_r(d+1)}{b_r(0)} & \frac{1-b_r(d)}{b_r(0)} & -\frac{b_r(d-1)}{b_r(0)} & \cdots & -\frac{b_r(1)}{b_r(0)} \end{bmatrix}$$

그러면, 행렬 $\boldsymbol{A}$를 대각화(diagonalization)하여 π_x의 일반형을 다음과 같이 유도할 수 있다.

$$\pi_x = \sum_{k=1}^{m_r} a_k \lambda_k^{x-m_r-1} \quad (9)$$

여기서 $\lambda_1, \lambda_2, \ldots, \lambda_{m_r}$은 행렬 대각화에 의해 얻어진 고유치(eigenvalue)들이고, 각 계수 a_k는 $\pi_{r+1}, \pi_{r+2}, \ldots, \pi_{r+m_r}$의 일차 조합으로 표현된 식이다. 이 π_x의 일반형에서 $|\lambda_k|\geq 1$인 고유치 λ_k에 대해서는 해당 계수 a_k를 모두 0으로 놓을 수 있다. 그 이유는 $x\to\infty$이면 $\pi_x\to 0$이라는 조건이 만족되기 위해서 반드시 $|\lambda_k|\geq 1$인 고유치들이 모두 π_x의 일반형에서 사라져야 하기 때문이다. 이렇게 해서 r개의 선형 방정식들이 추가로 유도된다.

그러므로, 최종적으로 얻어진 전체 $(r+m_r+1)$개의 선형 방정식들의 집합은 행렬식 $\boldsymbol{\pi} = \boldsymbol{P}\cdot\boldsymbol{\pi}$의 0번째 행부터 m_r번째 행까지에 해당하는 (m_r+1)개의 선형 방정식들과 $|\lambda_k|\geq 1$인 각 고유치 λ_k의 계수 a_k를 0으로 놓음으로써 얻어진 r개의 선형 방정식들로 구성된다. 이 선형 방정식 집합의 $(r+m_r+1)$개의 미지수들 $\pi_0, \pi_1, \ldots, \pi_{r+m_r}$은 수치해석 패키지를 이용해서 쉽게 계산된다. 이 미지수들이 결정되면, $|\lambda_k|<1$인 고유치들 λ_k의 계수들 a_k도 모두 함께 결정되므로 π_x의 일반형이 최종적으로 결정된다. 결과적으로, 이렇게 결정된 π_x의 일반형을 가지고 무한한 길이의 극한 분포를 생성하는 것이 가능하다. 지금까지 설명된 정해법에 대한 자세한 설명은 [16]을 참조하기 바란다.

4.3 근사해법(Approximated solution methods)

마르코프 행렬 절단법(Markov matrix truncation method): 극한 분포 π를 계산하는 근사해법으로서 한 가지 가능한 방법은 무한한 크기의 마르코프 행렬 $\boldsymbol{P}$를 유한한 크기의 정사각 행렬 $\boldsymbol{P}'$으로 절단하는 것이다. 즉, 평형 방정식 $\boldsymbol{\pi} = \boldsymbol{P}\cdot\boldsymbol{\pi}$를 $\boldsymbol{\pi}' = \boldsymbol{P}'\cdot\boldsymbol{\pi}'$로 변형하여 문제를 푼다. 여기서 $\boldsymbol{\pi}'$는 $\boldsymbol{\pi}' = [\pi_0', \pi_1', \ldots, \pi_p']$이고, $\boldsymbol{P}'$는 마르코프 행렬의 원소들 $P(x,y)$ $(0\leq x, y\leq p)$로 구성된 $(p+1)$ 크기의 정사각 행렬이다. 이러한 절단 방법에 의해서 변형된 문제 또한 수치해석 패키지를 이용하여 근사치 $\boldsymbol{\pi}'$를 구할 수 있는 하나의 고유벡터(eigenvector) 문제이다. 그러나, 여기서 얻어지는 고유벡터들이 모두 근사해 $\boldsymbol{\pi}'$가 되는 것은 아니고, 그 중에서 고유치가 1과 같거나 또는 가장 가까운 고유벡터가 근사해가 된다. 정확한 극한 분포 $\boldsymbol{\pi}$에 아주 가까운 근사해를 얻기 위해서는, 1에 수렴하는 고유치를 갖는 고유벡터가 얻어질 수 있도록 마르코프 행렬의 절단점 p

를 충분히 크게 잡아야 한다.

반복법(Iterative method): 마르코프 행렬의 유도가 필요 없는 다른 대안으로서, 충분히 많은 수의 하이퍼주기들의 열에 대해서 시스템 적체 분석을 반복하는 방법이 있다. 정리4에 의하면 시스템 적체 분포 $f_{B_k}(w)$는 $f_{B_\infty}(w)$에 수렴하는 것이 보장되므로, $f_{B_1}(w)$, $f_{B_2}(w)$, ...를 차례대로 수렴할 때까지 계산한다. 말하자면, 연속된 두 시스템 적체 분포들의 2차원 벡터놈(vector norm) $\|f_{B_k}(w)-f_{B_{k-1}}(w)\|$을 모니터하면서[1], 이 값이 충분히 작은 임계값 ε보다 작아질 때까지 시스템 적체 분포 $f_{B_k}(w)$ $(k=1,2,\ldots)$들의 계산을 계속해 나가는 것이다.

위 두 가지 근사해법들은 모두 근사해의 정확도에 영향을 주는 제어 매개변수(마르코프 행렬 절단법의 경우 절단 크기 p, 반복법의 경우 반복되는 하이퍼주기의 수 l)들이 등장한다. 따라서, 정확도가 높은 근사해를 얻기 위해서는, 이들 제어 매개변수의 값을 결정하는 것이 중요한 문제인데, 일반적으로 평균 시스템 이용률 $\overline{U}$가 1에 접근하면 큰 값의 매개변수들이 사용되어야 한다. 그러나, 매개변수를 지나치게 크게 잡으면 계산 시간이 크게 증가할 수 있으므로, 근사해의 정확도와 계산 시간의 이해득실 관계를 고려하여 매개변수값을 적절히 설정해야 한다. 매개변수값의 영향은 5절에서 분석한다.

4.4 극한 분포의 안전한 사용

앞에서 언급했듯이, $U^{\max}>1$인 일반적인 경우 극한 시스템 적체 분포의 길이는 무한히 길다. 따라서 정해법을 통해서 얻어진 극한 분포를 디지털 컴퓨터 안에서 적체 의존성 나무의 뿌리 노드로 사용하려면, 불가피하게 극한 분포를 어느 지점에서 절단하여 사용해야 한다. 이 소절에서는 그렇게 극한 분포를 절단해서 사용해도 마감시간 위반확률의 측면에서는 안전하다는 것을 설명한다. 말하자면, 절단된 극한 분포의 사용은 절단되지 않은 극한 분포을 사용한 경우 얻어질 각 태스크의 정확한 마감시간 위반확률보다는 항상 큰 값으로 귀결된다는 뜻이다.

정해법으로 얻은 극한 분포를 어느 지점 M에서 절단하여 얻어진 분포를 $f_{B_\infty}'(w)$으로 놓자. 그러면 절단 분포 $f_{B_\infty}'(w)$는 다음과 같이 표현된다.

$$f_{B_\infty}'(w)=\begin{cases} f_{B_\infty}(w) & w\le M \\ 0 & w>M \end{cases}$$

이 절단 분포는 그 전체 확률합이 1보다 작기 때문에 완전한 확률 분포는 아니다. 바꾸어 말하면, 절단 분포는 $\sum_{w>M} f_{B_\infty}(w)$만큼의 확률이 결손된 상태이다. 그러나, 다음과 같은 의미에서 절단 분포 $f_{B_\infty}'(w)$는 완전한 극한 분포 $f_{B_\infty}(w)$보다 더 비관적이다.

$$\sum_{w=0}^{t} f_{B_\infty}'(w) \le \sum_{w=0}^{t} f_{B_\infty}(w) \quad \text{for any } t$$

이것은 절단 분포 $f_{B_\infty}'(w)$을 적체 의존성 나무의 뿌리로 사용한 경우가, 완전한 극한 분포 $f_{B_\infty}(w)$를 사용한 경우보다 각 태스크에 대해 상대적으로 더 큰 마감시간 위반확률을 결과적으로 가져온다는 것을 의미한다.

5. 실험 결과

본 절에서는 제안된 분석 방법을 통해 얻은 실험 결과를 제시한다. 첫째, 제안된 분석 방법의 우수성을 보이기 위해, [6]에서 제시된 실험예에 대하여 추계적 시간 요구량 분석 방법(Stochastic Time Demand Analysis)[5,6]이 제시한 실험 결과와 비교한다. 둘째, 극한 시스템 적체 분포를 계산하는 정해법과 근사해법을, 시스템 이용률을 변화시켜가면서 분석 정확도와 계산 시간 측면에서 비교평가한다. 실험에 사용한 분석기는 인텔(Intel)사에서 제공하는 선형 대수 패키지 Math Kernel Library 5.2 [17]를 사용하여 리눅스 환경에서 C 언어로 구현하였으며, 본 논문에서 제시하는 모든 분석 결과는 펜티엄 IV 2.0GHz 처리기와 256MB 메모리를 가진 PC 환경에서 얻었다. 특히 분석 시간의 측정은 리눅스 시스템 호출 times()를 이용하였다.

5.1 추계적 시간 요구량 분석 방법(STDA)와의 비교

제안된 분석 방법과 STDA를 비교하기 위하여, [6]에서 사용되었던 태스크 집합들을 동일하게 사용하였다(표 2의 좌측 부분 참조). 그 세 개의 태스크 집합들은 모두 RM에 의해서 스케줄되고, 그 집합들에 속하는 모든 태스크들이 동일한 주기, 위상, 마감시간들을 갖는다. 오직 실행 시간 분포만이 다른데, 태스크 τ_i의 실행 시간은 $C_i^{\min}$에서 $C_i^{\max}$ $(=2\times\overline{C_i}-C_i^{\min})$까지의 구간에 걸쳐 있는 균일 분포(uniform distribution)을 갖는다. 표1의 우측 부분에 각각의 태스크 집합에 대해서 [6]에서 제시한 모의 실험 결과와 STDA의 분석 결과를 그대로 옮겨 실었다. 그리고, 동일한 태스크 집합들에 대해서 본 논문에서 제안한 분석 방법을 이용해 분석한 결과를 추가하였다.

표 2로부터 본 논문에서 제안하는 분석 방법에 의해 계산된 마감시간 위반확률과 STDA에 의해 계산된 마감시간 위반확률 사이에 큰 차이를 확인할 수 있다. 그리고 제안하는 분석 방법의 분석 결과는 모의 실험 결과와 완전히 일치한다는 점에 주목하기 바란다. STDA

1) 2차원 벡터놈은 $\|\boldsymbol{x}-\boldsymbol{y}\|=\sqrt{\sum_i (x_i-y_i)^2}$ 으로 정의된다.

표 2 [6]에서 사용된 태스크 집합들에 대해서 얻어진 실험 결과 (마감시간 위반확률)

태스크 집합		T_i	D_i	C_i^{min}	$\overline{C_i}$	C_i^{max}	$\overline{U}$	U^{max}	모의 실험	STDA	제안하는 분석방법
$S1$	τ_1	300	300	72	100	128	.708	.997	**.000** ± .000	**.000**	**.000**
	τ_2	400	400	72	150	228			**.047** ± .001	**.141**	**.047**
$S2$	τ_1	300	300	50	100	150	.708	1.125	**.000** ± .000	**.000**	**.000**
	τ_2	400	400	50	150	250			**.074** ± .002	**.489**	**.074**
$S3$	τ_1	300	300	1	100	199	.708	1.411	**.000** ± .000	**.000**	**.000**
	τ_2	400	400	1	150	299			**.192** ± .001	**.608**	**.192**

에 의해 얻어진 결과가 실제값과 큰 차이를 보이는 이유는 스케줄링의 치명적 위기(critical instant)라 불리는 실행 시나리오만을 분석 과정에서 고려하기 때문이다. 이 가정의 부정적 영향은 최대 시스템 이용률 U^{max}가 증가함에 따라 계산된 마감시간 위반확률이 실제값에서 너무 멀어진다는 점에서 확인 가능하다. 태스크 집합 $S2$에 속한 태스크 τ_2의 경우, STDA에 의해 얻어진 마감시간 위반확률(48.9%)은 본 논문에서 제안하는 분석 방법에 의해 얻어진 값(7.4%)보다 6배 이상 크다.

5.2 정해법과 근사해법들의 비교

정해법과 근사해법들을 비교할 목적으로 표 3에 제시된 태스크 집합들에 대해서 실험 결과를 얻었다.

모든 태스크 집합들은 세 개의 태스크들로 구성되었고, 그 태스크들은 동일한 주기, 위상, 마감시간들을 갖는다. 따라서 모든 태스크 집합들은 주어진 스케줄링 알고리즘에 대해서 동일한 적체 의존성 나무를 만들어낸다. 그 태스크 집합들 간의 유일한 차이는 실행 시간 분포들이다. 태스크 집합들 A, B, C의 경우, 각 태스크의 최소 실행 시간 C_i^{min}과 최대 실행 시간 C_i^{max}은 변하지 않는다. 이것은, 동일한 길이의 극한 시스템 적체 분포가 적체 의존성 나무의 뿌리로 사용된다고 가정할 때, 모든 작업들의 적체 분석과 간섭 분석에 소요되는 시간이 항상 일정하다는 뜻이고, 따라서 극한 시스템 적체 분포에 대한 평균 시스템 이용률 $\overline{U}$의 영향을 순수하게 평가할 수 있다는 뜻이다. 반면에, 태스크 집합들 $C, C1, C2$의 경우, 각 태스크의 평균 실행 시간은 고정되어 있으나, 전체 실행 시간 분포들은 좌우로 벌려지고(stretch) 있다. 이것은 평균 시스템 이용률을 고정시킨 상태에서 극한 시스템 적체 분포에 대한 최대 시스템 이용률 U^{max}의 영향을 평가할 수 있다는 것을 의미한다.

표 4는 표 3에서 제시된 태스크 집합들이 EDF로 스케줄된다고 가정하면서 얻은 실험 결과를 보여준다. 이

표 3 실험에 사용된 태스크 집합들

태스크 집합		T_i	D_i	실행 시간			시스템 이용률		
				C_i^{min}	$\overline{C_i}$	C_i^{max}	U^{min}	$\overline{U}$	U^{max}
A	τ_1	20	20	4	6	10	.58	.82	1.27
	τ_2	60	60	12	16	22			
	τ_3	90	90	16	23	36			
B	τ_1	20	20	4	6	10	.58	.87	1.27
	τ_2	60	60	12	17	22			
	τ_3	90	90	16	26	36			
C	τ_1	20	20	4	7	10	.58	.92	1.27
	τ_2	60	60	12	17	22			
	τ_3	90	90	16	26	36			
$C1$	τ_1	20	20	3	7	11	.46	.92	1.38
	τ_2	60	60	10	17	24			
	τ_3	90	90	13	26	39			
$C2$	τ_1	20	20	2	7	12	.34	.92	1.50
	τ_2	60	60	8	17	26			
	τ_3	90	90	10	26	42			

표는 각 계산법(정해법, 마르코프 행렬 절단법, 반복법)에 의해 얻어진 극한 시스템 적체 분포를 사용할 때 얻어진 마감시간 위반확률들과 각 계산법이 해당 극한 시스템 적체 분포를 계산하는데 소요한 시간들을 보여준다. 이 계산 시간들은 적체 의존성 나무를 유도하는데 걸린 시간(거의 0ms에 가깝다)과 모든 작업들에 대해 적체 분석과 간섭 분석을 수행하는데 걸린 시간(10ms 미만이다)을 포함하지 않은 값들이다. 그리고 참고 목적으로 각 태스크 집합에 대해 얻어진 모의 실험 결과를

표 4 분석 정확도와 분석 시간 측면에서의 정해법과 근사해법들의 비교

태스크 집합		모의실험	분석 결과 (마감시간 위반확률)							분석 시간 (초)						
			정해법	절단법			반복법			정해법	절단법			반복법		
				δ_1	δ_2	δ_3	δ_1	δ_2	δ_3		δ_1	δ_2	δ_3	δ_1	δ_2	δ_3
A	τ_1	.0001±.0000	.0001	.0001	.0001	.0001	.0001	.0001	.0001	.13	.00 (p=5)	.00 (p=25)	.00 (p=35)	.00 (I=2)	.00 (I=2)	.00 (I=4)
	τ_2	.0000±.0000	.0000	.0000	.0000	.0000	.0000	.0000	.0000							
	τ_3	.0000±.0000	.0000	.0000	.0000	.0000	.0000	.0000	.0000							
B	τ_1	.0013±.0002	.0013	.0013	.0013	.0013	.0013	.0013	.0013	.13	.00 (p=15)	.01 (p=35)	.02 (p=50)	.00 (I=2)	.00 (I=5)	.01 (I=8)
	τ_2	.0005±.0002	.0005	.0005	.0005	.0005	.0005	.0005	.0005							
	τ_3	.0000±.0001	.0000	.0000	.0000	.0000	.0000	.0000	.0000							
C	τ_1	.0223±.0013	.0224	.0224	.0224	.0224	.0224	.0224	.0224	.15	.01 (p=40)	.05 (p=85)	.12 (p=130)	.01 (I=6)	.02 (I=17)	.07 (I=29)
	τ_2	.0168±.0014	.0169	.0168	.0169	.0169	.0169	.0169	.0169							
	τ_3	.0081±.0011	.0081	.0080	.0081	.0081	.0080	.0081	.0081							
$C1$	τ_1	.0626±.0031	.0630	.0626	.0627	.0627	.0627	.0627	.0627	.31	.04 (p=75)	.19 (p=155)	.52 (p=235)	.02 (I=11)	.14 (I=30)	.50 (I=50)
	τ_2	.0604±.0038	.0610	.0606	.0607	.0607	.0607	.0607	.0607							
	τ_3	.0461±.0032	.0466	.0463	.0463	.0463	.0463	.0463	.0463							
$C2$	τ_1	.1248±.0058	결과 없음	.1249	.1250	.1250	.1249	.1250	.1250	결과 없음	.14 (p=120)	.65 (p=245)	1.84 (p=360)	.05 (I=16)	.64 (I=45)	1.93 (I=75)
	τ_2	.1293±.0064		.1295	.1296	.1296	.1295	.1296	.1296							
	τ_3	.1136±.0063		.1137	.1138	.1138	.1137	.1138	.1138							

또한 제시하였다. 이 모의 실험 결과는, 해당 태스크 집합을 5000번의 하이퍼주기동안 실행하는 것을 1회 모의 실행으로 정의할 때, 100회 모의 실행하여 얻은 마감시간 위반율들의 평균값과 표준편차이다. 그리고 근사해법들에 대해서는 분석 정확도를 명세하기 위하여 δ라는 값을 정확한 극한 시스템 적체 분포[2] $SSBD_{exact}$ (Stationary System Backlog Distribution)와 근사해법에 의해 계산한 극한 분포 $SSBD_{approx}$ 사이의 2차원 벡터놈 $\delta = \|SSBD_{\approx} - SSBD_{approx}\|$으로 정의하였다. 즉, 분석 정확도를 $\delta_1 = 10^{-4}$, $\delta_2 = 10^{-8}$, $\delta_3 = 10^{-12}$로 명세하면서, 이것을 만족시키는 $SSBD_{approx}$를 얻기 위해 근사해법들의 매개변수들(절단법의 경우 절단 크기 p와 반복법의 경우 반복되는 하이퍼주기들의 수 I)을 변화시켜가며 적절한 값을 결정하였다.

태스크 집합들 A, B, C에 대해서 얻어진 결과로부터, 평균 시스템 이용률 $\overline{U}$의 극한 시스템 적체 분포에 대한 영향을 알 수가 있다. $\overline{U}$가 증가함에 따라 정해법의 분석 시간은 거의 일정한 반면, 절단법과 반복법의 분석 시간은 상대적으로 빠르게 증가한다. 그 이유는 $\overline{U}$가 증가함에 따라 극한 시스템 적체 분포의 확률값들이 시간축의 양의 방향으로 널리 퍼지므로 근사해법들은 그 퍼진 확률값들을 충분히 포함하기 위해 큰 값의 제어변수 p와 I를 사용해야 하기 때문이다. 즉, 절단법은 절단 크기 p를, 반복법은 반복하는 하이퍼주기들의 수 I를 증가시켜야 한다. 정해법의 경우, 그러한 극한 시스템 적체 분포의 확장은 분석 시간에 거의 영향을 주지 않는데, 원래 정해법은 무한히 긴 극한 분포를 생성할 수 있는 일반형의 해를 유도해내기 때문이다.

위와 같은 관찰은 태스크 집합들 $C, C1, C2$에 대해서도 동일하게 적용된다. 최대 시스템 이용률 U^{max}가 증가함에 따라 극한 시스템 적체 분포는 시간축의 양의 방향으로 더욱 퍼져나가기 때문에, 근사해법들은 주어진 정확도를 달성하기 위해 제어변수 값들을 더욱 증가시켜야만 한다. 태스크 집합들 $C, C1$에 대해서 얻은 결과로부터, 정해법의 분석 시간 역시 증가한다는 것을 관찰할 수 있지만, 이것은 위에서 설명한 극한 시스템 적체 분포의 확장 때문이 아니고 각 태스크들의 실행 시간 분포가 확장됨에 따라 정해법의 계산과정에서 유도되는 행렬 $\boldsymbol{A}$(4.2절 참조)의 크기가 커지기 때문이다.

마지막으로, 본 연구진이 구현한 정해법은 태스크 집

2) 다음 페이지에서 설명하듯이, 본 연구진이 구현한 정해법은 사용된 선형 대수 패키지로부터 기인하는 오차가 있다. 따라서, $SSBD_{exact}$는 정해법으로부터 계산한 극한 분포를 사용하지 않고, 반복 방법을 무한히 적용하여 얻은 극한 분포를 사용하였다.

합 $C2$에 대해서 의미있는 분석 결과를 보여줄 수 없었다는 사실에 유의하기 바란다. 이것은, 정해법의 구현에 사용된 선형 대수 패키지가 분석 과정에서 얻어지는 악조건의(ill-conditioned) 선형 방정식들을 다루기에 불충분한 precision (64-bit floating point type)을 갖고 있었기 때문이다. 정해법은 그 계산 과정에서 행렬 A를 얻을 때 확률 $b_{i,k}(0)$이 실수 나눗셈의 제수로 사용하는데(4.2절 참조), 이 확률은 하이퍼주기에 속한 모든 작업들이 최소 실행 시간들을 가질 확률이므로 그 값이 본래적으로 매우 작고, 결과적으로 악조건의 선형 방정식들을 발생시키기 쉽다. 태스크 집합 $C2$의 경우 그 확률 $b_{i,k}(0)$는 5×10^{-17}이었다. 사실상 이것은 태스크 집합 $C1$에 대해서 얻은 정해법의 분석 결과가, 다른 근사해법들과 모의 실험 결과를 기준으로 판단할 때, 약간의 오차를 유발한 이유이기도 하다. 그러나, 이러한 오차 문제는 사용된 선형 대수 패키지의 precision 문제이므로, 보다 큰 precision을 가진 패키지를 사용함으로써 쉽게 극복 가능하다.

요약하면, 정해법은 시스템 이용률에 무관하게 정확한 해를 제공할 수 있으나, 디지털 컴퓨터에 구현하기에는 알고리즘이 견고하지 못한 면이 있다. 반면에, 근사해법들은 알고리즘의 견고성 문제는 제기되지 않으나, 그 계산 시간이 시스템 이용률에 비례해서 심각하게 커진다. 그러나, 평균 시스템 이용률 $\overline{U}$가 적당히 작은 경우(예를 들어 80% 이하), 근사해법들은 정해법보다 훨씬 적은 계산 시간으로 높은 정확도의 분석 결과를 제공할 수 있다는 점은 주목할 만하다.

6. 결 론

본 논문에서는 우선순위 스케줄링을 사용하는 주기적인 실시간 시스템을 대상으로 태스크들의 응답 시간 분포들을 정확하게 계산해낼 수 있는 확률적 분석 방법을 제안하였다. 제안한 분석 방법은 하이퍼주기에 속한 모든 작업들 사이에 적체 의존성 관계가 존재한다는 사실을 이용하여, RM이나 DM과 같은 고정 우선순위 스케줄링을 사용하는 시스템부터 EDF와 같은 동적 우선순위 스케줄링을 사용하는 시스템까지 모두 분석 가능하다. 제안된 분석 방법은 전체 시스템을 마르코프 확률과정으로 모델링하고, 유도된 마르코프 행렬을 이용하여 극한 시스템 적체 분포를 계산한다. 이렇게 계산된 극한 시스템 적체 분포는 적체 의존성 관계를 표현하는 자료구조의 입력이 되어, 하이퍼주기에 속한 모든 작업들의 해당 적체 분포를 계산하는데 이용된다. 이 접근법은 모든 작업들의 극한 적체 분포를 분석하는 문제를 크게 단순화시킨다. 그리고나서 각 작업에 대해 간섭 분석을 수행하여 모든 작업들의 극한 응답 시간 분포들을 얻었다. 실험 결과를 통해서, 본 논문에서 제안한 분석 방법은 추계적 시간 요구량 분석 방법에 비해 정확도가 아주 높다는 사실이 입증되었다. 그리고 시스템 이용률이 적당히 낮은 경우에는 극한 시스템 적체 분포를 정확하게 계산하는 정해법을 대신하여 마르코프 행렬 절단법이나 반복법을 사용하면 적은 계산 시간으로 정확도가 높은 결과를 얻을 수 있다는 사실도 보였다. 향후 연구로는 절단법과 반복법에 등장하는 제어변수의 값을 시스템 이용률에 따라 적절히 설정하는 방법이 연구되어야 할 것이다.

참 고 문 헌

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

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

[3] J. P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," in *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pp. 201-209, Dec. 1990.

[4] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J.-S. Liu, "Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times," in *Proceedings of the Real-Time Technology and Applications Symposium*, Chicago, Illinois, pp. 164-173, May 1995.

[5] M. K. Gardner and J. W. Liu, "Analyzing Stochastic Fixed-Priority Real-Time Systems," in *Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 44-58, Mar. 1999.

[6] M. K. Gardner, "Probabilistic Analysis and Scheduling of Critical Soft Real-Time Systems," Ph.D. dissertation, School of Computer Science, University of Illinois, Urbana-Champaign, 1999.

[7] J. P. Lehoczky, "Real-Time Queueing Theory," in *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pp. 186-195, Dec. 1996.

[8] J. P. Lehoczky, "Real-Time Queueing Network Theory," in *Proceedings of the 18th IEEE Real-Time Systems Symposium*, pp. 58-67, Dec. 1997.

[9] S. Manolache, P. Eles, and Z. Peng, "Memory and Time-Efficient Schedulability Analysis of Task Sets with Stochastic Execution Times," in *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, pp. 19-26, Jun. 2001.

[10] L. Abeni and G. Buttazzo, "Stochastic Analysis of

a Reservation Based System," in *Proceedings of the 9th International Workshop on Parallel and Distributed Real-Time Systems*, Apr. 2001.

[11] A. K. Atlas and A. Bestavros, "Statistical Rate Monotonic Scheduling," in *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pp. 123-132, Dec. 1998.

[12] J. 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.

[13] A. Terrasa and G. Bernat, "Extracting Temporal Properties from Real-Time Systems by Automatic Tracing Analysis,", in *Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications*, Feb. 2003.

[14] J. W. S. Liu, *Real-Time Systems*, Prentice Hall, 2000.

[15] L. Kleinrock, *Queueing Systems Volume I*: Theory, John Wiley and Sons, Inc., 1975.

[16] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. L. Bello, J. M. López, S. L. Min, and O. Mirabella, "Stochastic Analysis of Periodic Real-Time Systems," in *Proceedings of the 23rd Real-Time Systems Symposium*, pp. 289-300. Dec. 2002.

[17] Intel, "Intel Math Kernel Library: Reference Manual," 2001, http://developers.intel.com/software/products/mkl

부 록

정리 1의 증명: (a) 본 증명의 핵심은 임의의 하이퍼주기 내에서 최대 우선순위값 $p_{\max}$을 가진 작업을 찾으면, 이 작업이 곧 지상 작업이라는 사실이다. 최대 우선순위값을 가진 작업은 어떤 경우에도 적어도 하나 존재하므로, 적어도 하나의 지상 작업이 존재한다는 사실은 쉽게 증명된다. 따라서, 최대 우선순위값을 가진 작업이 지상 작업이라는 사실만 증명하면 된다. 이 증명을 위해서 하이퍼주기에 속한 모든 작업들이 서로 다른 우선순위값을 갖는다고 가정한다. 만일 동일한 우선순위값을 갖는 작업들이 존재하는 경우, 선입선출(First Come First Serve) 또는 사용자가 정의한 어떤 규칙에 따라 그 작업들에게 서로 다른 우선순위값을 항상 다시 지정할 수 있다. 그리고, 하이퍼주기 k, 즉 $[t+kT_H, t+(k+1)T_H)$ 구간에서 작업 $J_j^{(k)}$가 **최대** 우선순위값 $p_{\max}$을 갖는다고 가정하자. 그러면, 가정에 의해 작업 $J_j^{(k)}$는 $[t+kT_H, \lambda_j)$ 구간에 도착한 어떤 선행 작업들보다는 항상 큰 우선순위값(즉, 낮은 우선순위)을 갖는다는 것이 쉽게 증명된다. 그러면, 남은 문제는 작업 $J_j^{(k)}$가 $[\lambda_j - T_H, t+kT_H)$ 구간에서 도착한 어떤 작업보다도 우선순위값이 크다는 것을 증명하는 것이다. 이것은 시간 $\lambda_j - T_H$에 도착한 작업 J_j의 이전 인스턴스 $J_j^{(k-1)}$을 고려함으로써 쉽게 해결가능하다. $J_j^{(k-1)}$는 가정에 의해 마찬가지로 $[\lambda_j - T_H, t+kT_H)$ 구간에 도착한 어떤 작업보다도 큰 우선순위값을 가질 것이고, 이때 $J_j^{(k-1)}$의 우선순위값은 $J_j^{(k)}$의 우선순위값보다 작다는 사실로부터, $J_j^{(k)}$가 $[\lambda_j - T_H, t+kT_H)$ 구간에 도착한 어떤 작업보다도 우선순위값이 크다는 것이 증명된다. 그러므로, 작업 $J_j^{(k)}$가 하이퍼주기 $[\lambda_j - T_H, \lambda_j)$에 도착한 어떤 작업들보다도 높은 우선순위값을 갖는다는 것이 증명된다. 이 추론을 반복적으로 적용하면, $J_j^{(k)}$는 $[\lambda_j - 2T_H, \lambda_j)$, $[\lambda_j - 3T_H, \lambda_j)$ 등등에 도착한 어떤 작업들보다도 우선순위값이 크다는 사실을 증명할 수 있다. 그러므로, 작업 J_j는 지상 작업이며, 임의의 하이퍼주기에 대해 적어도 하나의 지상 작업이 존재하는다는 것이 증명된다.

(b) 위의 증명으로부터 자명하다.

정리 2의 증명: 일단 지상 작업의 기저 작업은 바로 이전 지상 작업이라는 사실로부터 모든 지상 작업들의 기저 작업들을 찾을 수 있다는 것은 자명하다. 따라서 본 증명에서는 비지상 작업들의 기저 작업들을 찾는 경우만 고려한다. 일단 어떤 비지상 작업 $J_j^{(n)}$에 대해 그 기저 작업을 동일한 하이퍼주기 n에서 찾지 못하는 경우를 가정하자. 그리고, $J_i^{(n)}$을 동일한 하이퍼주기에 속한 지상 작업으로서 $J_j^{(n)}$보다 우선순위값이 큰 작업이라고 놓자. 즉, 지상 작업 $J_i^{(n)}$는 비지상 작업 $J_j^{(n)}$의 기저 작업이 아니다. 그러면, 여기서 부등식 $n-k \geq (p_i^{(n)} - p_j^{(n)})/\Delta$를 만족시키는 어떤 k를 적절히 선택함으로써, 하이퍼주기 $k(<n)$에서 $p_i^{(k)} \leq p_j^{(n)}$를 만족하는 $J_i^{(n)}$의 이전 인스턴스 $J_i^{(k)}$를 찾을 수가 있다. 말하자면 $p_i^{(n)} = p_i^{(k)} + (n-k)\Delta$가 성립하기 때문에, 그러한 값 k는 $p_i^{(k)} \leq p_j^{(n)}$를 만족시킨다. 여기서, 정리1(b)에 의해서 $J_i^{(k)}$는 지상 작업이므로, 만일 적합한 다른 어떤 지상 작업이 발견되지 않는다면 $J_i^{(k)}$는 $J_j^{(n)}$의 기저 작업으로 선택가능하다. 그러므로, 임의의 비지상 작업 J_j에 대해서 항상 그 기저 작업을 선행 지상 작업들 안에서 찾을 수 있다는 것이 증명된다.

정리 3의 증명: 태스크 τ_i가 마감시간 $D_i = D^{\max}$를 갖는다고 가정하자. 그리고 작업 J_k는 태스크 τ_i에 속한다고 가정하자. 그러면 J_k의 우선순위값 $p_k = \lambda_k + D^{\max}$는 이전에 도착한 어떤 작업들의 우선순위값들보다도 크기 때문에, J_k는 지상 작업이다. 여기서 J_j를 하이퍼

주기 $[\lambda_k + D^{max}, \lambda_k + D^{max} + T_H]$의 시작 시점에 도착하는 어떤 비지상 작업으로 놓자. 그러면, 지상 작업 J_k는 J_j보다 우선순위값이 작기 때문에(즉 우선순위가 높기 때문에), 최악의 경우 J_j의 기저 작업이 될 수 있다. 여기서 만일 J_j가 하이퍼주기 끝 시점, 즉 $\lambda_k + D^{max} + T_H$에 도착한다고 가정하여도, 지상 작업 J_k는 최악의 경우 J_j의 기저 작업이 될 수 있다. 그러므로, 임의의 비지상 작업 J_k와 그 기저 작업 사이의 최대 거리는 $D^{max} + T_H$보다 작거나 같다.

김 강 희

1996년 서울대학교 컴퓨터공학과 학사
1998년 서울대학교 컴퓨터공학과 석사
1998년~현재 서울대학교 전기컴퓨터공학부 박사과정 재학중. 관심분야는 실시간 시스템, 내장형 시스템, 운영 체계임