확률적인 시간 제약 조건을 갖는 실시간 시스템을 위한 과실행 제어 및 합성 기법

(An Overrun Control Method and its Synthesis Method for Real-Time Systems with Probabilistic Timing Constraints)

김 강 희 † 황 호 영 ††
(Kanghee Kim) (Hoyoung Hwang)

요 약 멀티미디어와 같은 연성 실시간 응용들은 서비스 품질을 위해 각 태스크들이 마감시간 전에 실행을 마칠 수 있다는 확률적 보장을 요구한다. 시스템 설계자는 태스크들의 확률적 시간 제약 조건을 만족시키는 범위내에서 각 태스크에게 최악 처리기 요구량보다는 적은 처리기 시간을 할당함으로써 시스템 이용률 향상을 추구할 수 있다. 그러나 각 태스크에게 주어진 처리기 할당량이 최악 요구량보다 적은 경우에는 필연적으로 할당량을 초과하는 요구량을 가진 태스크 작업들, 즉 과실행 작업들이 발생하게 되므로 과실행 작업에 대한 제어 기법이 필요하다. 본 논문에서는 연성 실시간 시스템에서의 확률적인 과실행 제어 기법 및 합성 기법을 제안한다. 제안하는 기법은 우선순위 스케줄링의 기반 위에서 각 태스크가 주어진 확률적 시간 제약 조건을 만족할 수 있도록 과실행을 억제함과 동시에, 태스크 실행시간의 가변성으로 인한 여분의 처리기 시간을 일부 과실행 태스크에 유연하게 할당함으로써 시스템 이용률을 향상시킬 수 있다. 본 논문에서는 시스템 모델의 제시와 실험 결과의 분석을 통하여 제안하는 과실행 제어 기법이 마감시간 위반율 측면에서 기존의 기법들보다 우수함을 보이며, 또한 임의의 확률적 시간 제약 조건이 주어질 때 이를 만족시키는 과실행 제어 기법의 매개 변수들을 합성할 수 있음을 보인다.

키워드 : 실시간 시스템, 과실행 제어, 확률적 보장

Abstract Soft real-time applications such as multimedia feature highly variable processor requirements and probabilistic guarantees on deadline misses, meaning that each task in the application meets its deadline with a given probability. Thus, for such soft real-time applications, a system designer may want to improve the system utilization by allocating to each task a processor time less than its worst-case requirement, as long as the imposed probabilistic timing constraint is met. In this case, however, we have to address how to schedule jobs of a task that require more than (or, overrun) the allocated processor time to the task. In this paper, to address the overrun problem, we propose an overrun control method, which probabilistically controls the execution of overrunning jobs. The proposed overrun control method probabilistically allows overrunning jobs to complete for better system utilization, and also probabilistically prevents the overrunning jobs from completing so that the required probabilistic timing constraint for each task can be met. In the paper, we show that the proposed method outperforms previous methods proposed in the literature in terms of the overall deadline miss ratio, and that it is possible to synthesize the scheduling parameters of our method so that all tasks can meet the given probabilistic timing constraints.

Key words : Real-time systems, Overrun control, Probabilistic guarantee

· 이 논문은 2003년 두뇌한국21사업과 국가지정연구실사업에 의해서 지원되었습니다. 그리고 이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다.
† 비 회 원 : 삼성전자 연구원
khkim@archi.snu.ac.kr
†† 종신회원 : 안양대학교 디지털미디어학부 교수
hyhwang@aycc.anyang.ac.kr
논문접수 : 2003년 7월 21일
심사완료 : 2005년 2월 15일

1. 서 론

멀티미디어와 같은 연성 실시간 응용들은 많은 경우에 가변적인 처리기 요구량(variable processor requirement)과 확률적 시간 제약 조건(probabilistic timing constraint)을 갖는 태스크들로 구성되어 있다[1,2]. 확률적 시간 제약 조건이란 서비스 품질 보장을 위해 그

응용들을 구성하는 각 태스크들이 일정한 확률로 마감시간 전에 실행을 완료할 수 있어야 한다는 것이다. 확률적 시간 제약 조건을 가진 태스크들로 구성된 시스템에서는 이용률(utilization) 향상을 위해 각 태스크에게 최악 처리기 요구량보다는 적은 처리기 시간을 확률적으로 할당하는 것이 가능하다.

그러나 각 태스크에게 주어지는 처리기 할당량이 최악 요구량보다 적은 경우에는, 필연적으로 할당량을 초과하는 요구량을 가진 작업들을 어떻게 서비스할 것인가라는 문제가 발생한다. 본 논문에서 작업이란 주기적으로 도착하는 태스크의 인스턴스로 정의하며, 어떤 태스크가 매 주기마다 할당된 처리기 시간을 초과하는 경우를 과실행(overrun)이라고 정의한다. 과실행 작업들은 일시적일지라도 시스템 이용률이 100%을 초과하게 만들어 과부하를 발생시키고, 과실행 작업들 뒤에서 실행하는 많은 작업들로 하여금 마감시간을 놓치게 한다. 따라서, 시스템 이용률을 높일 목적으로 태스크들의 처리기 할당량을 최악 처리기 요구량보다 줄이려는 경우에는 각 태스크가 요구되는 마감시간 위반확률(deadline miss probability)을 만족할 수 있도록 과실행 작업들을 제어하는 방법을 마련해야 한다.

최근까지 과실행 제어 문제를 해결하기 위한 여러 연구들이 수행되어 왔으며[3-6], 이들 연구의 대부분은 과실행 제어 문제를 각 태스크 혹은 작업마다 고립된(isolated) 실행 환경을 제공함으로써 해결하려고 하였다. 여기서 고립된 실행 환경이란 각각의 작업에 대한 처리기 시간의 할당을 명백하게 사전에 예약함으로써 하나의 태스크에 대한 실행 결과나 제어가 다른 작업들의 실행에 영향을 미치지 않도록 한 환경을 말한다. 즉, 하나의 작업이 과실행을 하거나 또는 주어진 시간보다 일찍 작업을 끝내는 것이 다른 작업에 영향을 미치지 않으므로 각각의 작업은 사전에 할당된 처리기 시간내에서 독립적으로 혼자 실행되는 것과 같은 고립된 실행 환경을 가지게 된다. 고립된 실행 환경은 하나의 실행 환경에 속하는 태스크 혹은 작업의 과실행이 다른 실행 환경하에 있는 태스크들 혹은 작업들에게 영향을 주는 것을 방지하기 때문에 개별 태스크의 성능을 보장한다는 점에서 이점이 있다. 그러나 엄격한 고립성의 추구는 두 가지의 문제를 발생시킨다. 첫째, 과실행 작업에 대한 스케쥴링의 차별성이 발생할 수 있다. 일반적으로 과실행하지 않는 작업들에 대해서는 주어진 처리기 할당량을 준수하였으므로 무난하게 실행이 완료되는 것을 보장하지만, 과실행 작업들은 다른 작업들의 실행 결과에 따른 여분의 처리기 시간에 대한 확인과 관련없이 미리 주어진 실행 환경에서의 처리기 시간의 할당을 초과하였으므로 지연(pending), 혹은 폐기(dropping)되게 된다. 이는 과실행 작업의 발생이 시스템 이용률의 증가를 위해 각 태스크의 처리기 할당량을 최악 요구량보다 줄인 결과 비롯한 것이라는 점을 고려할 때 바람직하지 않다. 둘째, 엄격한 고립성은 전체적인 처리기 시간의 효율적인 이용을 저해할 수 있다. 하나의 실행 환경에서 과실행하지 않는 작업들이 주어진 처리기 할당량보다 일찍 작업을 마쳤을 경우에 여분의 처리기 시간이 발생하며 이는 다른 실행 환경에서 동작하는 작업의 수행에 활용될 수 있는 여분의 자원이 될 수 있다. 하지만 고립된 실행환경은 하나의 실행 환경에서 발생한 여분의 처리기 시간을 다른 실행 환경에 속한 작업들이 활용하는 것을 어렵게 만든다.

본 논문은 기존의 과실행 제어 기법들에서 많이 사용되고 있는 고립된 실행 환경이 가지는 문제점들을 극복할 수 있는 방안으로 *완화된 고립성*[7]을 제공하여 과실행 작업을 제어하는 기법을 제안한다. 제안하는 과실행 제어 기법은 우선순위 스케줄링 기반 위에서 각 태스크에게 처리기 할당량을 최악 요구량보다는 적게 할당하되, 과실행 작업들이 차별받지 않고 실행을 마칠 수 있는 기회를 확률적으로 제공한다. 이 제어 기법에서는 완화된 고립성의 구현을 위하여 과실행 작업의 발생시에 해당 작업에 대한 과실행을 허용할 것인가 또는 해당 작업을 폐기할 것인가를 확률적으로 결정한다. 이러한 결정 과정을 확률적 폐기(probabilistic or randomized dropping)라 하며, 확률적 폐기 메커니즘은 각 태스크에게 절대적인 의미가 아닌 확률적 의미에서 고립된 실행 환경을 제공함으로써 처리기 이용률을 높임과 동시에 각 태스크의 확률적 시간 제약 조건을 만족시킨다. 본 논문에서는 새로운 과실행 제어 기법의 모델을 제시하고 그 성능 실험 결과를 비교함으로써 제안하는 과실행 제어 기법이 기존 기법들에 비해서 전체적인 마감시간 위반율(overall deadline miss ratio) 측면에서 우수하고, 개별 태스크 혹은 작업 단위로 마감시간 위반 확률의 분석이 가능하므로 개별 멀티미디어 응용의 QoS 측면에서 보다 진전된 제어가 가능하며, 확률적 시간 제약 조건들이 주어질 때 그것을 만족시키도록 제어를 위한 매개 변수들이 합성 가능하다는 것을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 기술하고, 3장에서는 제안하는 과실행 제어 기법을 설명한다. 4장에서는 확률적 분석 기법을 이용하여 개별 태스크들의 마감시간 위반 확률의 계산 방법을 설명한다. 5장에서는 실험 결과를 통하여 과실행 제어 기법의 성능을 평가하고, 6장에서는 과실행 제어 기법의 매개변수들을 합성하는 방법을 설명하며, 7장에서 논문의 결론을 제시한다.

2. 관련 연구

과실행 제어 기법들은 고립성이 제공되는 수준에 따라 세 가지 그룹으로 나눌 수 있다. 첫 번째 그룹은 작업 수준과 태스크 수준에서 동시에 고립성을 제공하는 기법들로서 변형 태스크 기법(Transform-Task Method: TTM)[5]과 이 기법을 확장한 과실행 서버 기법(Overrun Server Method: OSM)[6]이 이 그룹에 속한다. 이들 기법의 경우 고립성이란 비과실행 작업들을 과실행 작업들로부터 보호하기 위한 것으로, 비과실행 작업들의 경우 마감시간 안에 실행을 마친다는 것을 100% 보장한다. 이것은 과실행 작업들을 무조건 과실행 서버라고 부르는 별도의 서버로 강제로 이주시켜 낮은 우선순위로 스케줄하기 때문이다. 따라서 과실행 작업들은 동일한 태스크에 속한 다른 비과실행 작업들과 다른 태스크들에게 전혀 영향을 미치지 않는다. 즉 작업 수준과 태스크 수준에서 동시에 고립성을 제공한다. 그러나 과실행 서버 기법의 단점은 비과실행 작업들의 경우에도 마감시간 위반을 부분적으로 허용할 수 있다는 사실을 이용하지 않고 무조건 과실행 작업들을 차별한다는 것이다.

두 번째 그룹은 작업 수준의 고립성은 포기하고 태스크 수준에서만 고립성을 제공하는 기법들로서 예약 기반 시스템(Reservation-Based System: RBS)과 통계적 비율 단조 스케줄링(Statistical Rate Monotonic Scheduling: SRM) 기법이 이 그룹에 속한다[3,4]. 이들 기법은 각 태스크마다 일정 비율의 처리기 용량을 예약해 주어 태스크들에게 자신만의 가상처리기를 제공하고, 각 태스크가 과실행 작업들을 해당 처리기 용량 안에서 해결하도록 한다. 이것은 해당 용량 안에서 과실행 작업들과 비과실행 작업들 사이에 처리기 시간의 공유가 이루어지도록 하기 위함이다. 그러나 과실행 문제를 해결하는 방식은 두 기법이 각기 다르다. 예약 기반 시스템은 과실행 작업이 실행을 마칠 때까지 필요한 처리기 시간을 동일한 태스크의 다른 작업들로부터 훔치는 것을 무조건적으로 허용한다[3]. 반면에 통계적 비율 단조 스케줄링 기법은 각 태스크마다 n개의 주기들의 총 길이와 같다고 정의되는 수퍼주기(super period)를 단위로 처리기 시간을 할당하고, 그 수퍼주기에 도착한 n개의 작업들이 과실행 여부와 상관없이 할당된 처리기 시간을 공유하게 한다[4]. 그러나 이들 기법의 단점은 과실행 작업들에게 실행 완료 기회를 제공하는 방식이 비과실행 작업들을 지나치게 위태롭게 하고, 그 결과 전체 시스템 성능을 심각하게 저하시킬 수 있다는 것이다. 시스템 이용률이 높으면 예약 기반 시스템은 많은 작업들로 하여금 마감시간을 놓치게 할 수 있고, 통계적 비율 단조 스케줄링 기법은 수퍼주기를 단위로 할당한 처리기 시간이 수퍼주기 초반에 지나치게 많이 소비된 경우 남은 처리기 시간이 부족하다는 이유로 수퍼주기 후반에 도착한 작업들을 무조건 폐기시킬 수 있다.

마지막 그룹은 태스크 수준의 고립성마저 포기하는 기법으로서 고립 서버 기법(Isolation Server Method)이다[6]. 고립 서버 기법은 여러 태스크들을 묶어 하나의 서버에 할당하고, 할당된 태스크들의 모든 작업들을 해당 서버에서 스케줄한다. 따라서 고립 서버들 간에는 고립성이 보장된다고 할지라도, 한 서버 안에서 실행되는 태스크들 간에는 고립성이 보장되지 않는다. 고립 서버 기법의 장점은 과실행 작업들과 비과실행 작업들 사이에 처리기 시간의 보상이 최대한으로 이루어질 가능성이 있다는 것이다. 즉, 비과실행 작업으로 인한 여분의 처리기 시간을 과실행 작업의 실행에 사용할 수 있게 된다. 그러나, 고립 서버 기법은 두 번째 그룹에서 소개된 기법들과 마찬가지로 시스템 이용률이 높으면 과실행 작업들로 인해 같은 서버에서 실행하는 다른 많은 작업들이 마감시간을 위반하게 되어 심각한 성능 저하를 초래할 수 있다.

최근에 과실행 문제를 다루는 새로운 접근법으로서 피드백 제어(feedback control) 기법이 연구되고 있다[8,9]. 이 기법은 전체 시스템의 마감시간 위반율을 관찰하면서 동적으로 처리기 용량 할당을 변경함으로써 마감시간 위반율을 일정 수준 이하로 유지하는 것을 추구하는 방식이다. 그러나, 피드백 제어는 개별 태스크들의 마감시간 위반확률을 예측할 수 없고 따라서 확률적 시간 제약 조건에 관한 어떤 보장도 제공할 수 없다는 단점이 있다.

3. 과실행 제어 기법

본 논문에서 제안하는 과실행 제어 기법은 확률적 폐기 메커니즘을 통해서 과실행 작업들에게 확률적인 방법으로 실행 완료 기회를 제공한다. 즉, 어떤 작업이 과실행하면 주어진 폐기 확률(dropping probability)에 따라서 계속 실행시킬지, 아니면 폐기할지를 결정한다. 이것은 궁극적으로 각 태스크에게 완화된 고립성을 제공함으로써 과실행 작업들을 차별적으로 스케줄하지 않음과 동시에 시스템 이용률이 높은 경우에 발생할 다수의 시간 제약 조건 위반을 방지하기 위함이다. 제안하는 과실행 제어 기법은 폐기 확률을 적절히 선택함으로써 과실행 작업들이 다른 태스크에게 주는 영향을 제한할 수 있으며, 그 결과 각 태스크의 확률적 시간 제약 조건을 만족시킬 수 있다.

3.1 시스템 모델

본 논문에서 제안하는 시스템 모델의 정의를 위해 표

1에서 설명하고 있는 기호를 사용하였다. 시스템 모델에서 가정하는 태스크 집합은 고정 주기 T_i와 가변적인 실행 시간 C_i을 가진 연성 실시간 태스크들로 구성된다. 각 태스크 τ_i는 일련의 작업들로 구성되며, j번째 작업을 $J_{i,j}$로 표기한다. 각 태스크의 실행 시간 분포는 자동 궤적 분석(automatic trace analysis)[10]과 같은 프로파일 기법을 통해서 주어진다고 가정한다. 한 태스크에 속한 모든 작업들은 주어진 실행 시간 분포를 따르며, 각 작업의 실행 시간은 같은 태스크에 속한 다른 작업들과 다른 태스크들의 작업들로부터 확률적으로 독립되어 있다고 가정한다. 각 태스크의 상대적 마감시간 D_i는 그 주기 T_i와 무관하게 결정되는 임의의 상수로서 T_i보다 작거나 같거나 클 수 있다.

본 논문은 스케줄링 알고리즘으로서 EDF(Earliest Deadline First)[11]를 가정한다. 그러나 제안하는 과실행 제어 기법은 EDF에 한정되지 않고 다른 우선순위 스케줄링 알고리즘에도 적용 가능하다. 그리고 제안하는 과실행 제어 기법은 확률적 폐기 메커니즘을 통해서 태스크들에게 완화된 고립성을 제공하는 것을 추구하므로 엄격한 고립성을 제공하는 별도의 예약 메커니즘을 필요로 하지 않는다.

표 1 시스템 모델에 사용된 기호들

기호	설 명
τ_i	i번째 태스크
T_i	태스크 τ_i의 주기
D_i	태스크 τ_i의 상대적 마감시간
C_i	태스트 τ_i의 가변적인 실행 시간
$J_{i,j}$	태스크 τ_i의 j번째 작업
$r_{i,j}$	작업 $J_{i,j}$의 도착 시간. 즉, $r_{i,j}=r_{i,1}+(j-1)\times T_i$ $(j\geq 1)$
$d_{i,j}$	작업 $J_{i,j}$의 절대적 마감 시간. 즉, $d_{i,j}=r_{i,j}+D_i$ $(j\geq 1)$
DPR_i	태스크 τ_i에 적용할 폐기 확률
$DP_{i,k}$	태스크 τ_i에 폐기 확률 DPR_i를 적용할 k번째 시점 (폐기 확률 적용은 최대 K회 할 수 있음. 즉 $k=1,2,\ldots,K$)
$e_{i,k}^{grt}$	태스크 τ_i가 폐기 시점 $DP_{i,k-1}(k>1)$의 폐기 테스트에서 생존한 경우 시스템이 τ_i에게 실행을 보장하는 처리기 시간. 단 $e_{i,1}^{grt}$는 폐기 테스트 없이 태스크 τ_i에게 항상 주어짐[1].

3.2 확률적 폐기 메커니즘

확률적 폐기 메커니즘은 그림 1로 설명할 수 있다. 일단, 각 태스크 τ_i에게 다수의 폐기 시점, $DP_{i,k}$(dropping points)와 매 폐기 시점에 적용할 폐기 확률 DPR_i(dropping probability)가 주어진다. 폐기 시점, $DP_{i,k}(k=1,2,\ldots,K)$는 각 태스크 τ_i에 대해서 미리 설정된 어떤 임계값 $e_{i,k}^{grt}(k=1,2,\ldots,K)$에 태스크 τ_i의 실행이 도달하는 순간으로 정의한다. 첫 번째 임계값 $e_{i,1}^{grt}$은 매 주기마다 태스크 τ_i의 작업들이 소비할 수 있다고 보장된(guaranteed) 최소의 처리기 시간이다. 그러나 이것은 τ_i의 작업들에게 $e_{i,1}^{grt}$만큼의 처리기 시간이 항상 마감시간 전에 제공된다는 것을 의미하지 않고, 단지 그 작업들이 마감시간을 위반하든지 그렇지 않든지 간에 그 만큼의 처리기 시간을 소비할 수 있도록 보장한다는 뜻이다. 반면에 다른 임계값 $e_{i,2}^{grt}$, $e_{i,3}^{grt}$, ... 들은 폐기 테스트를 통과하는 경우에만 보장되는 처리기 시간들이다. 폐기 테스트는 태스크 τ_i의 작업 $J_{i,j}$가 폐기 시점들에 도달할 때마다 실시된다. 매 폐기 시점 $DP_{i,k}$에서 $J_{i,j}$는 폐기 확률 DPR_i에 의해 폐기 테스트를 받는다. 즉, 각 작업 $J_{i,j}$에 대해 매 폐기 시점마다 0부터 1사이의 난수를 발생시켜, 발생된 난수가 DPR_i보다 작으면 해당 작업을 폐기하고, 그렇지 않으면 계속 실행시킨다. 따라서, k번째 폐기 시점을 통과한 작업 $J_{i,j}$는 $(k+1)$번째 폐기 시점에 도달하기 전에 $e_{i,k+1}^{grt}$ 만큼의 처리기 시간을 추가로 소비할 수 있는 기회를 갖게 된다. 말하자면, 과실행 작업은 모든 폐기 테스트에서 살아남아야만 실행을 완료할 수 있다.

확률적 폐기 메커니즘의 장점은 과실행 작업이 폐기 테스트에서 살아남은 경우 마감시간 안에 실행을 마칠 가능성이 높다는 것이다. 이것은 과실행 작업이 스케줄러가 부여한 원래의 우선순위를 유지하면서 스케줄되기 때문이다. 그리고, 과실행을 허용한다고 할지라도, 과실행 작업 뒤에서 실행될 작업들은 마감시간을 위반하지 않고 성공적으로 실행을 마칠 가능성이 있다. 그 이유는 시스템 내에서 여분의 처리기 시간이 발견되는 경우가 있기 때문이다. 여분의 처리기 시간은 다음과 같은 경우에 발생한다. 첫째, 평균 시스템 이용률이 낮은 경우 시스템 내에는 어느 태스크에 의해서도 사용되지 않는 여분의 처리기 시간이 존재할 가능성이 있다. 둘째, 과실행 작업 앞이나 뒤에서 실행하는 작업들이 소실행하여 처리기 시간을 남길 수가 있다. 그러나 여분의 처리기

1) 폐기 시점 $DP_{i,k}$는 작업 $J_{i,j}$가 스케줄러에 의해 실행이 시작된 순간부터 제공받은 처리기 시간들을 모두 합한 값. 즉 $e_{i,1}^{grt}+e_{i,2}^{grt}+\ldots+e_{i,k}^{grt}$와 항상 같지는 않다. 그 이유는 작업 $J_{i,j}$가 우선순위가 높은 다른 태스크 또는 작업에 의해서 선점될 수 있고, 따라서 선점된 시간만큼 처리기 시간의 소비가 지연될 수 있기 때문이다. 이 경우 폐기 시점 $DP_{i,k}$는 우선순위가 높은 태스크 또는 작업들로 인한 지연 시간만큼 연기된다.

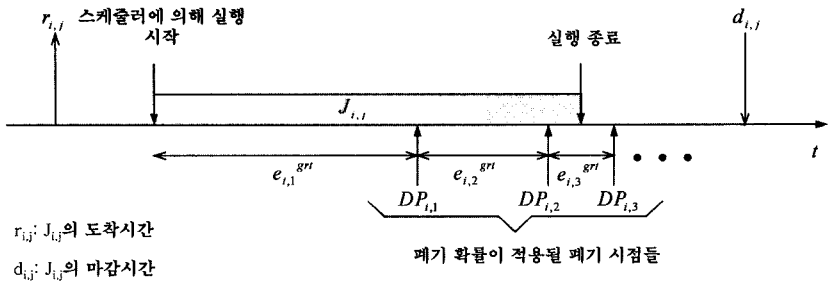


그림 1 확률적 폐기 메커니즘

시간에 대한 이와 같은 기대들이 어긋나는 경우, 과실행 작업은 뒤에서 실행되는 작업들을 지나치게 지연시켜 마감시간 위반을 초래할 수 있고 그 결과 각 태스크의 확률적 시간 제약 조건을 궁극적으로 위반하게 할 수 있다. 따라서 태스크들의 실행 시간 분포의 특성에 따라 적절한 폐기 시점과 폐기 확률을 결정해서 과실행을 억제할 것이 요구된다.

제안하는 확률적 폐기 메커니즘을 운영체계 수준에서 구현하는 경우 실행 오버헤드는 크지 않다. 각 태스크의 폐기 시점과 폐기 확률은 논문 뒤에서 설명할 합성 기법에 의해 미리 주어질 것이므로, 각 태스크마다 폐기 시점을 추적하기 위한 하나의 타이머와, 태스크들의 타이머가 완료할 때 폐기 결정을 내리기 위한 난수 발생 함수만이 필요하다. Park-Miller 유사 난수 발생 함수(pseudo-random number generator)[12,13]은 대략 10개 정도의 RISC(Reduced Instruction Set Computer) 명령어들로 구현할 수 있다[14].

4. 확률적 분석 방법

제안하는 과실행 제어 기법을 EDF 스케줄링과 결합하는 경우, 개별 태스크의 마감시간 위반확률을 분석적 방법을 이용하여 계산할 수 있다[7,15]. 확률적 폐기 효과를 각 태스크의 실행 시간 분포 수준에서 모델링하고, 주기적 실시간 시스템에 대한 확률적 분석 방법[15]의 입력으로 삼음으로써, 임의의 실행 시간 분포들이 주어질 때 우선순위 스케줄링을 사용하는 시스템에서 개별 태스크들의 정확한 응답 시간 분포를 구하고 마감시간 위반확률을 계산할 수 있다. 본 절에서는 확률적 분석 방법에 어떻게 확률적 폐기 효과를 반영할 수 있는지를 간략히 설명한다.

4.1 확률적 폐기 효과 모델링

확률적 폐기 효과는 실행 시간 분포의 변형(execution time distribution transformation)과 응답 시간 분포의 조건화(response time distribution conditioning)를 통해서 모델링할 수 있다. 전자는 각 태스크에 확률적 폐기를 적용한 결과 변형되는 실행 시간 분포를 얻는 과정이고, 후자는 변형된 실행 시간 분포로부터 응답 시간 분포를 유도해 마감시간 위반확률을 계산하는 과정이다.

실행 시간 분포의 변형: 확률적 폐기 효과는 해당 태스크의 실행 시간에만 영향을 미치므로 각 태스크의 실행 시간 분포를 변형함으로써 모델링할 수 있다. 예를 들어, 어떤 태스크 τ_i가 [0,100] 구간 사이에서 균일 분포(uniform distribution)의 실행 시간을 갖고 50과 75라는 시간에 각각 0.5의 폐기 확률로 폐기 시점들을 할당받았다고 가정하자. 그러면, 실제 시스템에서 태스크 τ_i에 속하는, 실행 시간이 50이 넘는 모든 작업들 중에서 평균적으로 절반은 첫번째 폐기 시점에서 버려질 것이다. 이것은 그 뒤에서 실행될 다른 작업들의 관점에서는 태스크 τ_i의 폐기된 작업들이 단지 50이라는 실행 시간을 갖는 작업들로 보인다는 뜻이다. 마찬가지로, 첫번째 폐기 시점에서 살아남은 작업들 중에 실행 시간이 75가 넘는 모든 태스크 τ_i의 작업들 중에서 평균적으로 절반은 두번째 폐기 시점에서 버려질 것이다. 이렇게 버려진 작업들도 그 뒤에서 실행될 다른 작업들의 관점에서는 단지 75라는 실행 시간을 갖는 작업으로 보일 것이다. 따라서, 태스크 τ_i의 실행 시간 분포는 그림 2와 같이 변형된다.

그림 2의 변형과정을 설명하면 다음과 같다. 가장 왼쪽에 있는 그래프에서 오른쪽 상단에 있는 사각형은 첫번째 폐기 시점에서 버려질 작업들을 나타내는데, 이것은 가운데 그래프에서 첫번째 폐기 시점에 0.25라는 확률을 가진 막대 그래프로 표현된다. 마찬가지로, 두번째 폐기 시점에서 버려질 작업들은 0.0625라는 확률을 가진 막대 그래프로 표현된다. 이 변형 과정을 임의의 실행 시간 분포에 대하여 일반화하면 다음과 같이 표현할 수 있다.

태스크 τ_i의 실행 시간 분포가 $f_{C_i}(c)$라는 임의의 확률 질량 함수로 주어진다고 가정하자. 그리고 k개의

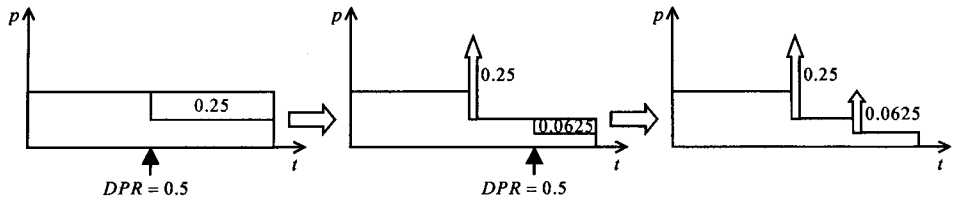


그림 2 실행 시간 분포의 변형

폐기 시점들 $DP_{i,1}$, $DP_{i,2}$, ... , $DP_{i,k}$의 폐기 효과가 이미 반영된 실행 시간 분포를 $f_{C_i}^{(k)}(c)$ 라고 하자. 그러면, 폐기 시점 $DP_{i,k+1}$의 폐기 효과를 반영하는 실행 시간 분포 $f_{C_i}^{(k+1)}(c)$ 는 $f_{C_i}^{(k)}(c)$ 를 이용하여 다음과 같이 표현 가능하다.

$$f_{C_i}^{(k+1)}(c) = f_{C_i}^{(k)}(c),$$

$$(0 < c < e_{i,k+1}^{grt} \text{ 일 때})$$

$$f_{C_i}^{(k+1)}(c) = f_{C_i}^{(k)}(c) + DPR_i \times \int_{e_{i,k+1}^{grt}}^{\infty} f_{C_i}^{(k)}(x)\,dx,$$

$$(c = e_{i,k+1}^{grt} \text{ 일 때})$$

$$f_{C_i}^{(k+1)}(c) = (1 - DPR_i) \times f_{C_i}^{(k)}(c),$$

$$(c > e_{i,k+1}^{grt} \text{ 일 때})$$

위 수식에서 $DPR_i \times \int_{e_{i,k+1}^{grt}}^{\infty} f_{C_i}^{(k)}(x)\,dx$는 폐기 시점 $DP_{i,k+1}$에서의 확률적 폐기 적용 결과 폐기될 작업들, 즉 그 실행 시간이 $c = e_{i,k+1}^{grt}$로 줄어들 작업들의 확률적 지분을 표현한다.

따라서, 모든 폐기 시점들 $DP_{i,1}$, $DP_{i,2}$, ... , $DP_{i,K}$에 대해서 위 수식들이 표현하는 실행 시간 분포 변형을 순차적으로 반복하면, 모든 폐기 효과를 반영하는 최종적인 실행 시간 분포를 얻을 수 있다. 최종적인 변형 실행 시간 분포는 확률적 분석 방법[15]의 입력으로 사용한다.

응답시간 분포의 조건화: 그러나 위에서 얻어진 변형된 실행 시간 분포를 확률적 분석에 그냥 이용할 경우 폐기된 작업들도 정상적으로 실행되는 작업들로 간주된다는 문제를 낳는다. 이 문제는 응답 시간 분포의 조건화를 통해서 해결할 수가 있는데, 이를 이해하기 위해서는 응답 시간 분포를 어떻게 계산하는지 이해할 필요가 있다. 하이퍼주기(각 태스크 주기들의 최소공배수에 해당하는 주기)에 속한 어떤 작업 $J_{i,j}$의 응답 시간 $R_{i,j}$는 다음과 같이 결정된다.

$$R_{i,j} = B_{i,j} + C_{i,j} + I$$

$B_{i,j}$는 작업 $J_{i,j}$보다 우선순위가 높거나 같은 선행 작업들이 남긴 적체작업량(backlog)이고, $C_{i,j}$는 작업 $J_{i,j}$의 실행 시간이며, I는 작업 $J_{i,j}$보다 늦게 도착하지만 우선순위가 높은 작업들이 $J_{i,j}$를 선점(preemption)하여 결과적으로 $J_{i,j}$가 지연되는 시간이다.

여기서 변형된 실행 시간 분포는 적체작업량 $B_{i,j}$와 선점 지연시간 I를 계산하는 데 사용된다. 왜냐하면 작업 $J_{i,j}$ 관점에서는 적체작업량 $B_{i,j}$와 선점 지연시간 I에 영향을 주는 작업들이 폐기된 것들인지 그렇지 않은 것들인지 자신의 응답 시간 계산에 상관이 없기 때문이다. 그러나 여기서 문제가 되는 것은 변형된 실행 시간 분포를 작업 $J_{i,j}$의 실행 시간 $C_{i,j}$의 분포로 삼는 것이다. 왜냐하면 이 때 확률적으로 폐기될 수 있는 작업 $J_{i,j}$의 인스턴스들이 정상적인 작업들로 간주되기 때문이다. 따라서, 이 문제를 해결하기 위해 응답 시간 분포의 조건화를 다음과 같이 수행한다. 첫째, 태스크 τ_i의 변형된 실행 시간 분포 중에서 폐기 효과를 반영하는 막대 그래프들을 제거한 조건부 실행 시간 분포를 얻는다. 이 분포는 그림2의 가장 오른쪽 그래프에서 계단 모양을 형성하는 부분이다. 둘째, 이 조건부 실행 시간 분포를 작업 $J_{i,j}$의 실행 시간, 즉 $C_{i,j}$의 분포로 삼아 응답 시간 분포를 계산한다. 그러면 결과적으로 작업 $J_{i,j}$가 폐기되지 않는다는 조건의 응답 시간 분포를 얻을 수 있다. 세째, 이렇게 얻어진 태스크 τ_i의 모든 작업들 $J_{i,j}$의 응답 시간 분포로부터 태스크 τ_i의 응답 시간 분포를 계산하고 마감시간 위반확률을 얻는다. 이 위반확률은 태스크 τ_i가 폐기되지 않는다는 조건 하에서 계산된 조건부 확률이다. 그러면 태스크 τ_i가 폐기된 경우들을 마감시간 위반으로 간주하면서, 태스크 τ_i의 정확한 마감시간 위반확률 DMSP(Deadline MisS Probability)을 다음과 같이 계산할 수가 있다.

$$DMSP_i = DMSP_i' \times (1 - DROP_i) + DROP_i$$

여기서 $DMSP_i'$는 조건부 마감시간 위반확률이고, $DROP_i$는 태스크 τ_i가 폐기될 확률의 총합이다. 즉 $DROP_i = DPR_i \times \sum_{k=1}^{K} \int_{e_{i,k}^{grt}}^{\infty} f_{C_i}^{(k-1)}(x)\,dx$ 이다. 그림 2의 예에서 $DROP_i$는 0.3125 = 0.25 + 0.0625 이다.

본 절에서 설명한 분석 방법을 통해서 계산한 마감시간 위반확률은 주어진 태스크 집합이 충분히 긴 시간 동안 시스템에서 수행한 경우 각 태스크의 마감시간 위반율이 수렴하는 값으로서, 각 태스크가 마감시간 내에 그 확률로 실행을 종료할 것을 보장한다.

5. 성능 평가

본 절에서는 제안하는 과실행 제어 기법의 성능을 문헌에서 제안된 다른 기법들과 비교 평가한다. 그리고 4절에서 설명한 분석 방법을 통하여 각 태스크에게 마감시간에 관한 확률적 보장을 제공할 수 있음을 보인다.

5.1 과실행 서버 기법과 예약 기반 시스템과의 비교

성능 비교를 목적으로, 제안하는 확률적 폐기(Randomized Dropping: RD) 메커니즘을 과실행 서버 기법(Overrun Server Method: OSM)[6]과 예약 기반 시스템(Reservation Based System: RBS)[3]과 비교하였다. OSM를 구현하기 위해서 각 태스크에 할당할 과실행 서버는 Total Bandwidth Server[16]를 이용하여 구현하였고, RBS를 구현하기 위해서 각 태스크에 할당할 가상처리기는 Constant Bandwidth Server[3]를 이용하여 구현하였다. 공정한 실험을 위해 모든 기법들을 작업들이 해당 태스크의 평균 실행 시간 $\overline{C_i}$를 초과할 때 과실행 제어를 수행하도록 설정하였다. 말하자면, 어떤 작업이 해당 태스크의 평균 실행 시간 $\overline{C_i}$를 초과하면, OSM의 경우 그 작업을 해당 과실행 서버로 강제로 옮겨 스케줄하고, RBS의 경우 그 작업이 동일한 태스크에 속한 다음 작업의 처리기 시간을 훔치는 것을 허용하면서 스케줄한다. RD의 경우에는 $\overline{C_i}=e_{i,1}^{avg}$로 간주하고 과실행 작업에 대해 폐기 테스트를 실시한다. 그러나 추가의 폐기 시점들은 없으며 적용할 폐기 확률은 모든 태스크에 대해서 동일하게 설정하였다.

한편 실험에 사용한 태스크 집합들을 관하여 설명하면 다음과 같다. 모든 과실행 제어 기법들은 시스템 이용률에 따라 성능에 큰 차이를 보이므로, 평균 시스템 이용률 $\overline{U}$가 0.75, 0.90, 0.95, 0.99인 태스크 집합들 각기 100개를 무작위 생성하여 실험에 이용하였다. 생성한 100개의 태스크 집합에 대해 최소한 1000개의 작업들이 도착하는 시간동안 각각 모의실험(simulation)를 수행하였고, 그 결과 얻은 전체 마감시간 준수율 DMTR (Deadline Meet Ratio)의 평균값을 성능 비교의 척도로 삼았다. 각 태스크 집합은 5개의 태스크들로 구성하였고, 태스크들의 주기는 [100, 1000] 구간의 균일 분포(uniform distribution)에서 임의의 값을 무작위 추출함으로써 결정하였다. 각 태스크의 실행 시간 분포는 5개의 태스크들이 주어진 평균 시스템 이용률 값을 결과적으로 만들어낼 수 있는 균일 분포로 결정하였다. 이 경우 실행 시간 분포의 평균값만을 정하면 되는데, 평균값은 각 태스크의 이용률($\overline{U_i}=\overline{C_i}/T_i$) 값을 [0, 1] 구간의 균일 분포에서 추출함으로써 결정하였다. 그리고 각 태스크의 상대적 마감시간 D_i는 주기 T_i와 같게 설정하였다. 그림 3은 RD, OSM, RBS에 대해 얻은 100개 태스크 집합의 전체 마감시간 준수율의 평균값을 보여준다. 이 그림에서 RD(0.0), RD(0.1), RD(0.2), RD(0.4)는 각각 폐기 확률을 0, 0.1, 0.2, 0.4로 설정한 경우들을 나타낸다.

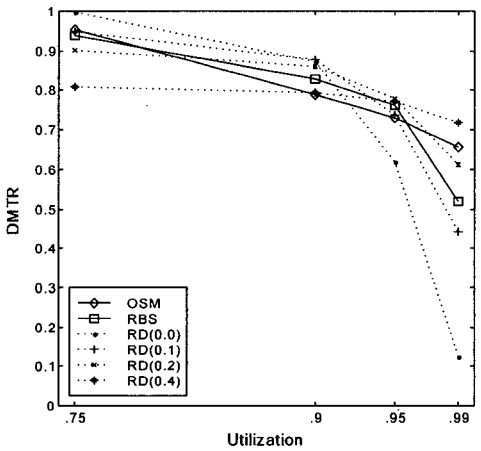


그림 3 RD, OSM, RBS의 비교

그림 3으로부터 작업 폐기의 필요성은 평균 시스템 이용률 $\overline{U}$가 증가함에 따라 함께 커진다는 사실을 알 수가 있다. 특히 $\overline{U}$=99%일 때 RD의 성능은 OSM보다 우수하고, RBS에 비해서는 훨씬 더 우수하다(RD의 성능은 $\overline{U}$=90%일 때 RD(0.1)에서, $\overline{U}$=95%일 때 RD(0.2)에서, $\overline{U}$=99%일 때 RD(0.4)에서 가장 좋다. 이것은 폐기 확률을 적절히 설정하여 얻은 완화된 고립성이 전반적인 시스템 성능을 향상시킴을 보여주는 것이다. OSM의 경우, 마감시간 준수율은 시스템 이용률이 증가함에 따라 완만하게 감소하고, $\overline{U}$=99%일 때는 65.6%의 준수율을 보인다. 이것은 시스템에 도착하는 전체 작업 중 약 50%에 해당하는 비과실행 작업들은 마감시간을 모두 준수한 반면, 나머지 50%에 해당하는 과실행 작업들 중 극히 일부만이(약 16%에 해당) 마감시간을 준수한 결과이다. 말하자면, OSM은 과실행 작업들을 차별적으로 스케줄하고 있다. 한편, OSM에 비교하면 RBS는 $\overline{U}$=90%, $\overline{U}$=95%일 때 성능이 좋다. 특히, $\overline{U}$=95%일 때에는 RBS와 RD(0.2) 사이에는 마감시간

준수율의 차이가 거의 없다는 사실은 주목할 만하다. 이것은 RBS가 작업 수준의 고립성만을 포기하였으면서도, 한 태스크 안에서 과실행 작업들과 비과실행 작업들 사이에 처리기 시간의 공유가 잘 이루어지고 있음을 뜻한다. 그러나 $\overline{U}$=99%일 때 RBS 역시 성능 저하가 뚜렷하다. 이 때 RBS의 마감시간 준수율(51.8%)은 OSM의 마감시간 준수율(65.6%)보다 낮고, RD(0.4)의 마감시간 준수율(72%)보다는 훨씬 낮다. 이것은 시스템 이용률이 높을 때 과실행을 무조건 허용하면 성능이 저하된다는 뜻이다. 요약하면, 위 실험은 RD가 적절한 확률적 제어를 통해서 과실행 작업들을 우선순위에 있어서 차별하지 않으면서도 시스템 이용률이 높을 때 나타나는 성능 저하를 막을 수 있음을 보여준다.

5.2 확률적 분석을 통한 성능 보장

본 절에서는 표 2에 제시된 태스크 집합들 $TS1$과 $TS2$에 대해 얻어진 모의실험 결과와 4절에서 설명한 분석 방법을 이용하여 얻은 분석 결과를 비교한다.

표 2 실험에 사용된 태스크 집합들

태스크 집합		T_i	D_i	$\overline{e_i}$	$\overline{U_i}$
$TS1$ ($\overline{U}=0.97$)	τ_1	60	60	22	0.367
	τ_2	100	100	32	0.32
	τ_3	150	150	42	0.28
$TS2$ ($\overline{U}=0.94$)	τ_1	60	60	17	0.283
	τ_2	100	100	32	0.32
	τ_3	150	150	51	0.34

그림 4는 폐기 시점의 수를 1로 설정하고 분석한 $TS1$과 $TS2$의 마감시간 준수율을 보여준다. 이 그림으로부터 5000번의 하이퍼 주기에 해당하는 시간 동안 위 태스크 집합들을 모의실험한 결과 얻은 마감시간 준수율이 4절에서 설명한 분석 방법에 의해 계산한 마감시간 준수확률에 거의 일치함을 알 수가 있다. $TS1$의 경우 폐기 확률이 0일 때 약간의 차이가 관찰되지만, 이것은 분석 방법에 의해 계산된 태스크들의 응답 시간 분포가 무한히 길기 때문에 디지털 컴퓨터에서 불가피하게 꼬리 일부를 절단한 결과 발생한 오차이다. 마찬가지 이유로, $TS2$의 경우에도 폐기 확률이 0.3보다 작을 때 작은 차이가 관찰된다. 그림 5는 태스크 집합 $TS1$에 속한 각 태스크에 대해서도 계산된 마감시간 위반확률이 모의실험을 통해 얻은 마감시간 위반율에 일치함을 보이고 있다.

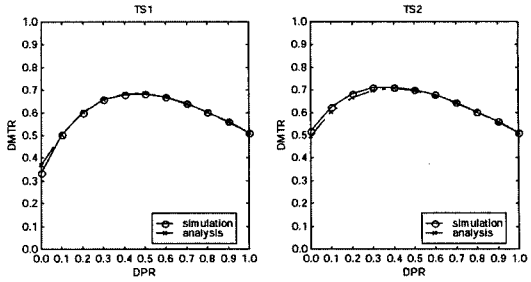


그림 4 태스크 집합 $TS1$과 $TS2$의 전체 마감시간 준수율과 준수확률

6. 확률적 과실행 제어 기법의 합성

5절의 실험결과에서 보듯이 적절한 폐기 확률의 설정은 확률적 폐기 메커니즘에 매우 중요하다. 본 절에서는 확률적 폐기 메커니즘의 폐기 확률과 폐기 시점을 각 태스크에게 요구되는 마감시간 위반확률을 만족시킬 수 있도록 합성하는 방법을 설명한다. 일단 설명의 편의를 위하여, 폐기 시점들이 주어지고 폐기 확률만을 합성하는 경우를 설명한다. 이 경우 폐기 확률의 합성 문제는 각 태스크 τ_i에게 요구되는 목표 마감시간 위반확률 $DMSP_i^*$보다 작거나 같은 마감시간 위반확률 $DMSP_i \leq DMSP_i^* (i=1,2,\ldots,N)$을 갖게 하는 폐기 확률들의 벡터 $\boldsymbol{DPR}=(DPR_1, DPR_2, \ldots, DPR_N)$를 찾아내는 문제

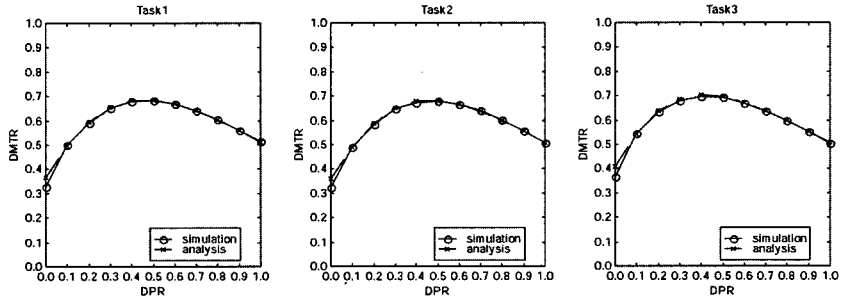


그림 5 태스크 집합 $TS1$에 속한 각 태스크의 마감시간 준수율과 준수확률

가 된다. 이 문제는 목표 달성법(Goal Attainment Method)[17]라고 불리는 비선형 최적화 문제로 모델링하여 풀 수 있다. 한편, 거꾸로 폐기 확률이 주어지고 폐기 시점들을 합성하는 경우에도 이 모델링은 똑같이 적용 가능하다. 본 절에서는 목표달성법 모델링을 설명하고, 하나의 합성 예를 제시한다.

6.1 목표달성법(Goal Attainment Method)를 이용한 모델링

5.2절에서 보였듯이 각 태스크의 마감시간 위반확률 $DMSP_i$는 4절에서 설명한 분석 방법에 의해서 정확히 계산할 수 있다. 이 사실로부터, 주어진 태스크 집합에 대하여 임의의 폐기 확률 벡터 $\boldsymbol{DPR}$이 주어질 때 각 태스크 τ_i의 마감시간 위반확률을 결정하는 어떤 함수 F_i가 존재함을 알 수가 있다. 즉, 4절에서 설명한 분석 방법은 폐기 확률 벡터 $\boldsymbol{DPR}$로부터 해당 함수 F_i, 즉 $DMSP_i = F_i(\boldsymbol{DPR})$의 값을 모든 $i(=1,2,\ldots,N)$에 대해서 동시에 계산하는 과정이다. 폐기 확률 벡터 $\boldsymbol{DPR}$은 집합 $\{(DPR_1, DPR_2, \ldots, DPR_N) \mid 0 \le DPR_i \le 1$ for $i=1,2,\ldots,N\}$에 속하는 어떤 벡터이다. 여기서 함수 F_i를 수학적으로 표현하는 것은 가능하지 않지만, 다음 성질이 성립함을 증명할 수가 있다.

정리 1. 어떤 태스크 집합에 대한 폐기 확률 벡터 $\boldsymbol{DPR}$이 주어져 있다고 가정하자. 이 때 다른 태스크들 τ_j의 폐기 확률들은 고정시킨 채로, 한 태스크 τ_i의 폐기 확률 DPR_i를 변화시키면 다음과 같은 결과를 얻는다.

(a) 태스크 τ_i의 폐기 확률 DPR_i를 감소(증가)시키면, 다른 태스크 τ_j의 마감시간 위반확률 $DMSP_j$는 결코 향상(저하)되지 않는다. 바꾸어 말하면, 저하되거나(향상되거나) 아니면 영향을 받지 않는다.

(b) 태스크 τ_i의 폐기 확률 DPR_i를 감소시키면, 자신의 마감시간 위반확률 $DMSP_i$는 어떤 최소값이 발견될 때까지는 단조적으로 감소한다. 이 때 그 최소값이 0 아닌 폐기 확률 DPR_i'에 의해서 얻어진 경우, 폐기 확률 DPR_i를 DPR_i'보다 더 작게 계속 감소시켜가면, 마감시간 위반확률 $DMSP_i$은 다시 증가한다.

증명. 정리 1(a)의 증명을 위해 EDF로 스케줄링되는 임의의 태스크 집합을 고려하자. 여기서 두 가지 경우를 나누어 생각한다. 첫번째 경우는 그 하이퍼 주기에 속한 모든 작업들의 우선순위가 어떤 작업도 다른 작업들에 의해 전혀 선점되지 않도록 할당된 경우이다. 두번째 경우는 그 우선순위들이 선점 가능성을 허용하도록 할당된 경우이다. 첫번째 경우, [15]에 의하면 하이퍼 주기 안에서 모든 작업들이 어떤 순서에 따라 순차적으로 실행된다. 여기서 태스크 τ_i의 폐기 확률의 감소는 τ_i에 속한 모든 작업들 $J_{i,j}$의 폐기 확률들의 감소를 의미한다. 따라서 $J_{i,j}$ 뒤에서 실행되는 다른 태스크 τ_j의 작업들은 $J_{i,j}$로 인해 지연 시간이 증가할 수 있고, 그 결과 응답 시간이 나빠질 수 있다. 즉 마감시간 위반확률이 높아질 수가 있다.

두번째 경우, [15]에 의하면 하이퍼 주기 안에 지상 작업(ground job)이라고 불리는 몇몇 작업들은 어떤 순서에 따라 순차적으로 실행된다. 반면에 나머지 비지상 작업(non-ground job)들은 지상 작업들을 선점하거나, 지상 작업들 사이에서 실행된다. 그리고 [15]에 의하면 한 태스크 τ_i에 속한 작업들 $J_{i,j}$은 모두 지상 작업과 비지상 작업 중 어느 하나로 분류되고 그 분류 결과는 어떠한 상황에서도 바뀌지 않는다. 여기서 태스크 τ_i의 폐기 확률의 감소는 τ_i에 속한 모든 작업들 $J_{i,j}$의 폐기 확률들의 감소를 의미한다. 따라서 지상 작업으로 분류된 태스크 τ_i의 작업들 $J_{i,j}$는 그 뒤에서 실행될 다른 태스크들의 지상 작업들과 비지상 작업들의 지연 시간을 증가시켜 다른 태스크들의 마감시간 위반확률을 높일 수가 있다. 또한 비지상 작업으로 분류된 태스크 τ_i의 작업들 $J_{i,j}$ 역시 다른 태스크들의 지상 작업들을 선점할 때 상대적으로 긴 지연 시간을 초래하여 다른 태스크들의 마감시간 위반확률을 높일 수가 있다. 그러므로 정리 1(a)는 임의의 태스크 집합에 대해서 성립한다.

정리 1(b)의 증명을 위해서는 태스크 τ_i의 폐기 확률들을 계속 낮추어갈 때, 태스크 τ_i의 마감시간 위반확률 또한 높아질 수 있다는 것을 보이면 된다. 이는 위에서 설명된 정리 1(a)의 증명 과정으로부터 자명하다. 즉, 폐기 확률이 낮아진 태스크 τ_i의 작업들 $J_{i,j}$ 뒤에서 실행될 작업들 중에는 다른 태스크들에 속한 작업들뿐만 아니라, 동일한 태스크 τ_i에 속한 작업들 $J_{i,j}$도 있을 수가 있다. 말하자면 선행하는 τ_i의 작업들은 그 뒤를 따르는 τ_i의 작업들의 성능을 저하시킬 수가 있다. 따라서 폐기 확률을 낮추어가면, 태스크 τ_i의 작업들은 다른 태스크의 작업들이 사용하지 않는 시스템의 여분 처리기 시간으로부터 이득을 얻는 한 그 마감시간 위반확률이 향상되다가, 결국에는 다시 나빠지기 시작한다. 그러므로 정리 1(b)는 성립한다. □

정리 1(a)의 의미는 목적 함수라고 부르는 각 함수 F_i들이 서로 경쟁 관계에 있다는 것이고, 정리 1(b)의 의미는 한 태스크 τ_i의 성능을 향상시키기 위해 τ_i의 과실행을 점점 더 많이 허용해 가면, 다른 태스크들의 성능 희생을 통해 어느 정도까지 성능 향상을 얻을 수 있지만, 결국에는 그 허용이 지나칠 때 자신의 성능도

다시 저하된다는 것이다. 이 경우 목표 확률들을 만족시키는 ***DPR*** 해는 하나 이상이 존재할 수 있다. 왜냐하면, 폐기 확률들은 0부터 1사이의 실수 범위에서 정의되므로, 어떤 ***DPR*** 해가 하나 존재한다면, 그 해를 구성하는 폐기 확률들 중 하나를 약간 변경해서 여전히 $DMSP_i \le DMSP_i^*$ 라는 조건들을 만족하는 다른 해를 구할 가능성이 있기 때문이다.

따라서 폐기 확률의 합성 문제는 최적화 이론 분야에서 소위 다중목적 최적화 문제(multi-objective optimization problem)로 분류되는 문제이다. 다중목적 최적화 문제에서는 여러 가지 해들 중에서도 비열등해(non-inferior solution)라고 불리는 특정한 해집합을 찾는 것이 중요하다[17,18]. 비열등해란 본 합성 문제의 경우 그 해를 구성하는 폐기 확률들 중 하나를 변경하여 한 목적 함수를 향상시키려고 할 때 반드시 다른 함수의 희생을 요구한다는 특성을 가진 해로서 정의된다. 비열등해는 전체 시스템 성능 측면에서 더 이상 모든 목적 함수들을 동시에 향상시킬 수 없는 상태에 있다는 점에서의 최적해이며, 정리 1(a)는 비열등해의 존재를 설명하는 것이다.

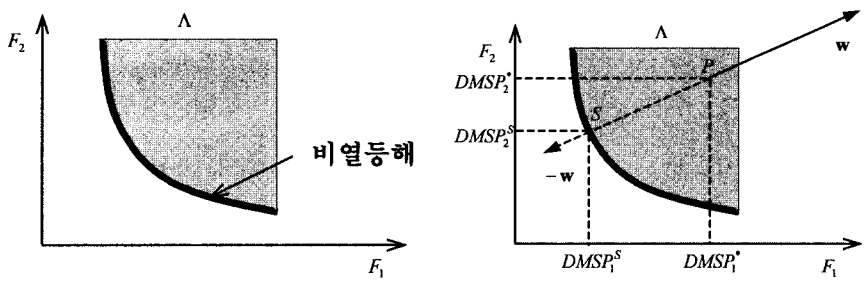
그러므로 본 절에서 제안하는 합성 방법도 비열등해를 찾는 것을 목표로 한다. 이를 위해 다양한 비열등해 탐색 알고리즘들 중에서 각 목적함수에 가중치(weight)를 두어서 비열등해를 찾는 목표달성법(Goal Attainment Method)[17]을 사용한다. 목표달성법은 목표 확률들을 만족시키는 비열등해들 중에서도 중요도가 상대적으로 높은 태스크의 성능을 최대화하는 해를 찾아준다는 점에서 유용하다. 각 태스크의 중요도를 시스템 설계자가 정하는 가중치 계수들의 벡터 $\boldsymbol{w}=\{w_1, w_2, \ldots, w_N\}$로 표현하면, 목표달성법은 목표 확률들 $\boldsymbol{DMSP}^*=\{DMSP_1^*, DMSP_2^*, \ldots, DMSP_N^*\}$을 모두 만족하면서 가중치 계수 벡터에 따라 목적 함수들 $\boldsymbol{F}(\boldsymbol{DPR})=(F_1(\boldsymbol{DPR}), F_2(\boldsymbol{DPR}), \ldots, F_N(\boldsymbol{DPR}))$를 최소화하는 비열등해를 찾는다. 예를 들어, 두 개의 태스크들로 구성된 어떤 태스크 집합에 대해 가중치 계수 벡터가 $\boldsymbol{w}=\{1,1\}$로 주어지고, 목표 확률 벡터가 $\boldsymbol{DMSP}^*=\{20\%, 20\%\}$로 주어지면, 목표달성법은 마감시간 위반확률들을 최소화하여, 예를 들어 $\boldsymbol{DMSP}=\{8\%, 8\%\}$로 얻어지는 어떤 ***DPR*** 해를 찾아준다. 결과적으로 가중치 계수 벡터는 과실행을 허용함으로써 얻을 수 있는 전체 성능 향상을 각 태스크들에게 그 가중치에 따라 분배하는 효과를 갖는다. 다음 수식은 ***DPR*** 합성 문제를 목표달성법으로 모델링한 것이다(R=실수집합, $\Omega=\{(DPR_1, DPR_2, \ldots, DPR_N) \mid 0 \le DPR_i \le 1 \text{ for } i=1,2,\ldots,N\}$).

$$\text{minimize } \gamma \ (\gamma \in R, \boldsymbol{DPR} \in \Omega)$$
$$\text{subject to } F_i(\boldsymbol{DPR}) - DMSP_i^* \le w_i \gamma \text{ for}$$
$$i=1,2,\ldots,N$$

목표달성법으로 모델링된 문제는 Sequential Quadratic Programming[18,19]이라는 비선형 최적화 기법을 통해서 해를 구할 수 있다.

그러나, 임의의 목표 확률들에 대해서 항상 그 목표 확률들을 달성하는 비열등해가 존재하는 것은 아니다. 예를 들어, 본 논문에서 가정하는 시스템은 그 최대 이용률이 1보다 큰 경우 목표 확률들이 $\boldsymbol{DMSP}^*=\{0\%, 0\%, \ldots, 0\%\}$로 주어지면, 이것을 만족시킬 있는 ***DPR*** 해는 존재하지 않는다. 그러나 조건을 만족하는 비열등해를 찾을 수 없는 경우에도, 목표달성법은 가중치에 따라 성능 이득을 분배한 합성 결과를 제시하며, 그 결과는 시스템 설계자가 현실적으로 합성 가능한 목표 확률들을 다시 제출하도록 하는 데에 도움을 줄 수 있다.

그림 6은 태스크들의 수가 2인 경우 목표달성법이 가중치에 따라 비열등해를 찾는 과정을 보여준다(Λ: 목적함수들의 공간). 그림 6(b)에서 P는 2개의 목표 확률들 $DMSP_i^*$로 결정되는 지점이고, 이 지점에서부터 가중치



(a) 비열등해의 집합 (b) 비열등해의 탐색

그림 6 비열등해와 목표달성법(2차원의 경우)

계수 벡터 $\boldsymbol{w}$에 따라 비열등해 $S=(DMSP_1^S, DMSP_2^S)$의 탐색이 γ가 최소화되어감에 따라 그 벡터의 음의 방향으로 진행된다.

목표달성법을 이용한 폐기 확률 합성 문제의 모델링은 폐기 시점 합성 문제에도 동일하게 적용된다. 이 경우 정리1에서 폐기 확률의 감소는 폐기 확률을 고정한 채 폐기 시점을 시간적으로 뒤로 미루는 경우에 해당하고, 폐기 확률의 증가는 앞으로 당기는 경우에 해당한다. 따라서 정리 1에서 언급된 목적 함수들의 성질은 동일하게 성립하며, 주어진 목표 확률들을 만족시키는 폐기 시점들을 찾는 문제도 목표달성법에 의해서 풀 수가 있다. 폐기 시점의 수가 1개 이상인 경우는 폐기 시점 벡터 $\boldsymbol{DP}$의 차원이 늘어나는 것에 불과하므로 목표달성법을 이용한 모델링은 여전히 성립한다.

6.2 합성 실험

본 절에서는 표 3에서 제시된 태스크 집합에 대해서 목표달성법을 이용하여 주어진 목표 확률들을 만족하는 $\boldsymbol{DPR}$을 합성한 결과를 제시한다.

표 3 합성 실험에 사용될 태스크 집합

태스크 집합		T_i	D_i	$\overline{e_i}$	$\overline{U_i}$
$TS3$ ($\overline{U}=0.83$)	τ_1	60	60	18	0.30
	τ_2	100	100	28	0.28
	τ_3	150	150	38	0.253

표 3에서 제시된 태스크 집합에 대해 4가지 서로 다른 목표 확률들 집합에 대해 합성을 수행하였다: $\boldsymbol{GS_A}$=(20%,20%,20%), $\boldsymbol{GS_B}$=(10%,20%,30%), $\boldsymbol{GS_C}$=(30%, 10%,20%), $\boldsymbol{GS_D}$=(20%,30%,10%). 각 목표 확률 집합에 대해 가중치 계수 벡터는 목표 확률 벡터와 동일하게 설정하였다: $\boldsymbol{w_A}=\boldsymbol{GS_A}$, $\boldsymbol{w_B}=\boldsymbol{GS_B}$, $\boldsymbol{w_C}=\boldsymbol{GS_C}$, $\boldsymbol{w_D}=\boldsymbol{GS_D}$. 이것은, 예를 들어 $\boldsymbol{GS_B}$의 경우, 과실행을 허용함으로써 얻어질 성능 향상을 태스크 τ_1, τ_2, τ_3에게 1:2:3의 비율로 분배하겠다는 뜻이다. 목표달성법의 구현을 위해서 MATLAB 소프트웨어 패키지[18]를 사용하였다. 표 4는 합성 결과를 보여준다.

표 4에서 세번째와 네번째 열은 목표달성법에 의해서 얻어진 $\boldsymbol{DPR}$ 해와 그 때 계산된 마감시간 위반확률들이다. 여기서 주어진 가중치 계수 벡터에 따라 모든 목표 확률 집합에 대해 합성이 성공적으로 이루어졌음을 알 수가 있다. 표 3의 마지막 열은 합성 과정에서 목적함수들 $\boldsymbol{F}(\boldsymbol{DPR})=(F_1(\boldsymbol{DPR}), F_2(\boldsymbol{DPR}), F_3(\boldsymbol{DPR}))$의 계산이 몇 회 이루어졌는지를 나타내고 있다. 말하자면 4절에서 기술된 확률적 분석 방법이 적용된 횟수이다.

본 논문에서 소개한 분석 방법의 계산 복잡도는 하나의 하이퍼주기에 속한 작업들의 수를 n, 태스크들의 실행 시간 분포들의 길이 중 최대 길이를 m이라 할 때 $O(n^3\times m^3)$이다[20]. 따라서 합성 방법의 계산 복잡도는 최적해를 찾을 때까지 분석을 반복한 횟수를 l이라 할 때 $O(n^3\times m^3\times l)$이 된다. 그러나 합성 방법의 계산 복잡도는 하이퍼주기를 줄여 작업들의 수 n을 줄이거나, 실행 시간 분포의 시간 단위(granularity)를 크게 설정하여 m을 줄이는 방법 등을 통해서 상당히 낮출 수 있다.

7. 결 론

본 논문은 연성 실시간 시스템을 위해서 확률적 폐기 방법을 이용한 과실행 제어 기법을 제안하였다. 이 기법은 태스크 수준의 실행 환경의 고립성을 확률적으로 완화함으로써 과실행하는 작업만을 일방적으로 차별하지 않으면서 성능 향상을 추구한다. 또한, 제안된 과실행 제어 기법이 EDF 스케줄링에 적용될 때 분석적 방법을 이용하여 각 태스크의 마감시간 위반확률을 계산함으로써 각 태스크에게 마감시간에 관한 확률적 보장을 제공할 수 있다.

제안한 기법의 성능 검증을 위하여, 과실행 서버 기법(OSM) 및 예약 기반 시스템(RBS) 등 기존의 기법들과의 비교 실험을 수행하였다. 실험 결과 특히 시스템 이용률이 높을 때 제안한 과실행 제어 기법이 그 기법들보다 전체 시스템의 성능을 향상시킨다는 것을 확인하였으며, 4장에서 설명한 분석 방법을 통해서 예측한 각 태스크의 성능이 실험에서 얻어진 것과 일치한다는 것을 보였다. 또한, 각 태스크에게 요구되는 마감시간 위반확률을 만족할 수 있도록 폐기 확률들과 폐기 시점들을 결정할 수 있는 합성 방법을 제시하였다. 이 합성 방법은 시스템 이용률이 다른 태스크 집합들에 대해 각각 가장 좋은 성능을 얻을 수 있는 폐기 확률과 폐기 시점

표 4 합성 결과

목표 집합	$\boldsymbol{DPR}$ 벡터	얻어진 $\boldsymbol{DMSP}$ 벡터	비율	합성횟수
$\boldsymbol{GS_A}$	(0.0072, 0.0000, 0.0411)	(7.9%, 7.9%, 7.9%)	(1:1:1)	16
$\boldsymbol{GS_B}$	(0.0000, 0.1089, 0.2414)	(5.2%, 10.3%, 15.5%)	(1:2:3)	16
$\boldsymbol{GS_C}$	(0.2309, 0.0000, 0.1369)	(15.6%, 5.2%, 10.4%)	(3:1:2)	16
$\boldsymbol{GS_D}$	(0.0737, 0.1689, 0.0000)	(9.2%, 13.8%, 4.6%)	(2:3:1)	16

들을 결정하는 데 이용될 수 있다.

참 고 문 헌

[1] M. Krunz, R. Sass, and H. Hughes, "Statistical Characteristics and Multiplexing of MPEG Streams," in *Proceedings of the IEEE INFOCOM 1995 Conference*, pp. 455-462, Apr. 1995.

[2] M. Krunz and S. K. Tripathi, "On the Characterization of VBR MPEG Streams," in *Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 192-202, 1997.

[3] L. Abeni and G. Buttazzo. "Integrating Multimedia Applications in Hard Real-Time Systems," In *Proceedings of the 19th Real-Time Systems Symposium*, pp. 3-13, Dec. 1998.

[4] 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.

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

[6] M. K. Gardner and J. W.S. Liu. "Performance of Algorithms for Scheduling Real-Time Systems with Overrun and Overload," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 287-296, Jun. 1999.

[7] K. Kim, L. Lo Bello, S. L. Min, and O. Mirabella. "On Relaxing Task Isolation in Overrun Handling to Provide Probabilistic Guarantees to Soft Real-Time Tasks with Varying Execution Times," In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, pp. 193-202, Jun. 2002.

[8] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. "Design and Evaluation of a Feedback Control EDF Scheduling Algorithm," in *Proceedings of the 20th Real-Time Systems Symposium*, pp. 56-67, Dec. 1999.

[9] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Marley. "Performance Specification and Metrics for Adaptive Real-time Systems," in *Proceedings of the 21st Real-Time Systems Symposium*, pp. 13-24, Dec. 2000.

[10] 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.

[11] 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.

[12] D. G. Carta. "Two Fast Implementations of the 'Minimal Standard' Random Number Generator," *Communications of the ACM*, Vol. 33, No. 1, pp. 87-88, Jan. 1990.

[13] S. K. Park and K. W. Miller. "Random Number Generators: Good Ones Are Hard to Find," *Communications of the ACM*, Vol 21, No. 10, Oct. 1988.

[14] C. A. Waldspurger and W. E. Weihl. "Lottery Scheduling: Flexible Proportional-Share Resource Management," In *Proceedings of the USENIX 1994 Operating Systems Design and Implementation Symposium*, pp. 1-11, 1994.

[15] J. L. Diaz, D. F. Garcia, K. Kim, C.-G. Lee, L. Lo Bello, J. M. Lopez, 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.

[16] M. Spuri, G. Buttazzo, and F. Sensini. "Robust Aperiodic Scheduling Under Dynamic Priority Systems," In *Proceedings of the 17th Real-Time Systems Symposium*, pp. 210-219, Dec. 1996.

[17] F.W. Gembicki, *"Vector Optimization for Control with Performance and Parameter Sensitivity Indices,"* Ph.D. Dissertation, Case Western Reserve University, Cleveland, Ohio, 1974.

[18] *"MATLAB Optimization Toolbox User's Guide,"* The Mathworks, Inc.

[19] D. G. Luenberger, *"Linear and Nonlinear Programming (Second Edition),"* Addison Wesley, 1984.

[20] K. Kim, J. L. Diaz, L. Lo Bello, J. M. Lopez, C.-G. Lee, D. F. Garcia, S. L. Min, and O. Mirabella, "An Exact Stochastic Analysis of Priority-Driven Periodic Real-Time Systems and Its Approximations," submitted to *the IEEE Transactions on Computers*, 2003.

김 강 희

1996년 서울대학교 컴퓨터공학과 학사. 1998년 서울대학교 컴퓨터공학과 석사. 2004년 서울대학교 전기컴퓨터공학부 박사. 2004년~현재 삼성전자 책임연구원. 관심분야는 실시간 시스템, 내장형 시스템, 운영 체계임

황 호 영

1993년 서울대학교 컴퓨터공학과 학사. 1995년 서울대학교 컴퓨터공학과 석사. 2003년 서울대학교 전기컴퓨터공학부 박사. 2003년~현재 안양대학교 디지털미디어학부 조교수. 관심분야는 컴퓨터 네트워크 및 무선 통신임