단일 프로세서상에서 수행되는 실시간 타스크의 실행 가치 최대화를 위한 동적 스케줄링

김 인 수[†] · 이 윤 열[††] · 이 춘 희[†††] · 정 기 현[††††] · 최 경 희[†††††]

요 약

타스크의 총 가치가 최대가 되도록 스케줄하는 실시간 스케줄러는 새로운 타스크가 도착할 때마다 스케줄 가능한, 모든 타스크들에게 서비스 시간을 할당한다. 새로운 타스크가 도착하기 전에 스케줄 된 모든 타스크들이 완전히 실행된다면 최대 총 가치는 얻어질 것이나, 실제 상황에서는 스케줄된 모든 타스크들이 완전히 실행되기 이전에 새로운 타스크가 도착하는 것이 일반적이다. 본 논문에서는 실시간 타스크들에 대한 새로운 스케줄링 알고리즘을 제안한다. 기존의 알고리즘들은 스케줄 가능한 모든 타스크들에 대해 서비스 시간을 계산하지만, 제안된 알고리즘은 일부의, 보다 빠른 만기를 갖는, 타스크들에 대해서만 서비스 시간을 결정한다. 이렇게 부분적으로 서비스 시간을 할당함으로써 평균 스케줄링 복잡도는 크게 떨어진다. 최악의 경우라도 제안된 알고리즘의 복잡도는 기존의 알고리즘 중에서 보다 성능이 좋은 것으로 평가되는 알고리즘의 복잡도인 $O(N^2)$과 같다.

A Dynamic Scheduling Algorithm to Maximize the Total Value of Real-time Tasks running on a Single Processor

In-Soo Kim[†] · Yun-Yeol Lee[††] · Chun-Hee Lee[†††] · Gi-Hyun Jung[††††] · Kyung-Hee Choi[†††††]

ABSTRACT

In most of the existing real-time schedulers producing the total value as large as possible, the service times for all schedulable tasks are computed at each time a new task arrives. If all scheduled tasks would be executed completely before a new task arrives, the schedule may produce the greatest total value. But this is not always true in real situations. In many cases, (a) new tasks arrive(s) before all the scheduled tasks are executed completely. In this paper, we propose a unique scheduling algorithm for real-time tasks. The proposed algorithm determines the service times only for some tasks with earlier deadlines, while the existing algorithms determine the service times for all tasks. This partial computation decreases the average scheduling complexity dramatically, even though, in the worst case, the complexity of the proposed algorithm becomes $O(N^2)$, which is equal to that of a previous algorithm that has been known as a less complicated one.

† 정 회 원 : 한국전자통신연구원 연구원
†† 비 회 원 : 천안공업대학 전자계산학과 교수
††† 정 회 원 : 시스템공학연구 및 ETRI 책임연구원
†††† 정 회 원 : 아주대학교 전기전자공학부 교수
††††† 정 회 원 : 아주대학교 정보 및 컴퓨터공학부 교수

논문접수 : 1999년 3월 13일, 심사완료 : 1999년 4월 26일

1. 서 론

고전적인 하드 실시간 시스템(hard real-time system)에서 타스크들(tasks)에 대한 시간 제약 조건(timing requirements)은 반드시 지켜져야 한다. 타스크들은 정해진 만기(deadline) 이내에 정해진 시간(processing time)만큼 실행되어야 타스크의 결과는 유효한 것으로 간주된다. 만기를 만족하지 못하는 경우 timing fault가 발생하고, 그때까지 실행된 결과는, 만약 있다면, 무시된다.

이에 반하여 부정확한 실시간 계산 모델(imprecise real-time computation model)에서는 타스크를 필수 실행 부분(mandatory part)과 임의 실행 부분(optional part)으로 나눈다. 필수 실행 부분은 태스크의 실행 결과가 의미를 갖기 위해서는 반드시 실행되어야 하는 부분이며, 임의 실행 부분은 필수 실행 부분이 완료된 후, 시간적으로 여유가 있다면, 실행되어 필수 실행 부분에 의한 결과의 품질을 향상시킨다. 이러한 2개의 실시간 계산 모델은 타스크들의 처리 시간에 대한 제약 조건(constraints)에서 정도 차이가 있다.

이와는 달리 처리 시간에 제한이 가해지지 않는 실시간 타스크들이 있다. 이러한 타스크들의 실행 결과는 실행 시간이 길면 길수록 정답에 가까운 결과가 얻어진다. 이러한 타스크의 대표적인 예로서 원주율을 구하는 알고리즘이 있다. 이 알고리즘은 실행하는 시간이 길면 길수록 오차가 작은, 보다 정확한 원주율이 얻어진다.

타스크의 가치(value)란 타스크를 실행하면서 얻을 수 있는 결과의 유용성을 수치적으로 표현한 것으로 타스크의 실행에 따라 기대되는 보상(reward)이라고도 한다. 실행 시간에 따른 타스크의 가치, 또는 보상을 정의하는 함수를 가치 함수(value function), 또는 보상 함수(reward function)라고 한다.

이러한 타스크에 관한 연구는 최초에는 인공지능의 분야에서 독립적으로 연구되었으나, 최근 들어 실시간 시스템의 응용 영역이 신호 처리, 멀티미디어 통신 및 디스플레이(display), 시각화(visualization), DB 질의 처리, 그리고 인공지능 및 의사 결정 등의 분야로 확대되면서 더욱 연구가 활발히 진행되고 있다. 이러한 타스크는 문헌에 따라 여러 가지 이름으로 불리는데, IRIS(Increasing Reward with Increasing Service) 타스크[1], anytime 알고리즘[2,3], 실시간 approximate processing 알고리즘[4], segmented 알고리즘[5], flexible 타스크[6] 등이 그것이다.

타스크의 가치와 관련된 기준이 최적화되도록 온라인 실시간 타스크들을 스케줄링 하는 것은 어려운 문제이다. 고전적인 하드 실시간 시스템에서 스케줄러는 각 타스크가 정해진 만기 이내에 완료될 수 있도록 스케줄을 수립한다. 이때의 기준은 만기 이내 완료(success) 아니면 만기 이내 처리 불가(failure)의 2가지 이다. 이러한 경우 스케줄링 입장에서 관심 사항은 타스크들을 각각의 만기 이내에 처리할 수 있는 스케줄이 존재하는가 하는 스케줄 가능성(schedulability)으로 타스크들의 실행에 의한 중간 가치의 최적화는 아니다. 단일 프로세서 상에서 처리되는 온라인 하드 실시간 타스크에 대한 타당한 스케줄(feasible schedule)을 얻기 위해 EDF(Earliest Deadline First), 또는 MLF (Minimum Laxity First) 등이 널리 사용되는 최적화 알고리즘이다[7].

그러나 온라인 실시간 타스크들이 실행 시간에 따라 유효한, 서로 다른 가치를 만들어내는 경우, 상황은 달라진다. 이러한 경우 앞에서의 잘 알려진 알고리즘들도 새로운 기준을 만족할 수 없으며, 따라서 새로운 접근 방식이 요구된다[1,8]. 새로운 접근 방식은 모든 타스크들이 최적의 해답을 만들어내도록 적당한 실행 시간을 제공해야 하며 동시에 각각의 정해진 만기 이내에 실행이 완료되도록 스케줄을 수립해야 한다. 그러나 타스크들의 가치 함수는 일반 함수이고, 도착 시간, 만기 등과 같은 타스크 속성들을 스케줄링 이전에 알 수가 없으므로 가치를 최적화하는 스케줄을 수립하기란 거의 불가능하다. 따라서 현실적으로는 가치 함수, 도착 시간, 만기 등에 제약을 가한 휴리스틱 스케줄링 전략을 사용한다.

본 논문은 임의의 시간에 도착하는, 가치 함수를 갖는 실시간 타스크들의 실행 가치가 가능한 최대가 되도록 각 타스크에게 서비스 시간을 할당하는 스케줄링 알고리즘을 제안한다. 다음 장에서는 기존의 관련된 연구들에 대하여, 3장에서는 시스템 모델과 제안된 접근 방식의 배경을, 4장에서는 제안된 알고리즘을 기술한다. 제안된 알고리즘에 대한 예는 5장에서, 그리고 결론은 6장에 기술하였다.

2. 온라인 스케줄링 전략

첫번째 동적 스케줄링 전략으로서 greedy 전략을 생

각할 수 있다. 이 전략에 의하면 타스크들이 만들어내는 가치가 가능한 한 크게 되도록 타스크들에게 균등하게 서비스 시간이 할당되도록 스케줄을 수립한다. 이러한 전략은 이상적이기는 하나 구현하기는 거의 불가능하다. 이에 대한 차선으로서 round-robin 전략을 생각할 수 있으나, 이 전략은 잦은 context switching으로 인해 성능이 좋지 않다. [1]에서 이와는 다른 2단계 동적 스케줄링 알고리즘이 제안되었다. 이 알고리즘에서 상위 단계 알고리즘은 새로운 타스크가 도착할 때마다 실행되어 스케줄 가능한 타스크들의 가치 합이 최대가 되도록 타스크들의 서비스 시간을 결정한다. 하위 단계에서는 상위 단계 알고리즘에 의해 결정된 스케줄 정보(서비스 시간)에 따라 타스크의 실행 순서를 결정하고, 프로세서를 할당하여 실행한다. 이 단계에서는 EDF나 FIFO 알고리즘이 사용될 수 있는데, EDF가 보다 합리적으로 타스크들에게 프로세서를 할당한다. 그러나 [1]에서 제안된 상위 단계 알고리즘의 복잡도(computational complexity)는 현실에 적용할 만큼 낮지가 않다. 예를 들어, 가치 함수가 지수 함수일 경우 상위 단계 알고리즘의 복잡도는 알고리즘 실행 시점에 시스템에 들어있는 타스크의 수의 제곱에 비례한다.

상위 단계 알고리즘의 실행을 살펴보면 몇 가지 흥미로운 점이 있다 : 상위 단계 알고리즘은 타스크가 도착할 때마다 실행된다. 앞선 상위 단계 알고리즘의 실행에서 얻어진 스케줄은 재실행시 의미가 없으며, 사용되지 않는다. 시스템에 들어있는 모든 타스크에 대하여 서비스 시간을 할당한다. 상위 단계 알고리즘이 정한 서비스 시간은 하위 단계 스케줄링동안 스케줄 된 모든 타스크들에게 할당되지 않을 수 있다.

프로세서 활용도(utilization)가 높거나 평균 타스크 간 도착 시간(inter-arrival time)이 짧다면, 하위 단계 스케줄러는 새로운 타스크의 도착까지 또는 상위 단계 알고리즘의 재실행까지 항상 소수의 타스크(상위 단계에서 서비스 시간이 결정된 모든 타스크가 아니라)만을 스케줄할 것이다. 프로세서 활용도가 낮고 타스크 간 도착 시간이 길다면, 어떤 시점에서 시스템에 들어있는 타스크의 수는 거의 항상 1(또는 아주 소수 개)과 같을 것이므로, 각 타스크는 정해진 만기 이전에 완전히 실행될 수 있을 것이다. 이러한 경우 하위 단계 알고리즘은 다음 타스크 도착까지 적은 수의 타스크만을 대상으로 스케줄하게 된다.

이러한 관찰이 의미하는 것은 프로세서 활용도가 높건 낮건 관계없이, 하위 단계 알고리즘은, 일반적으로, 어떤 시점에서 시스템에 들어있는 전체 타스크가 아닌, 스케줄 가능한 타스크 중 아주 적은 수의 타스크에 대한 스케줄 정보만을 필요로 한다는 것이다. 이것은 가치를 최대화하기 위해서 상위 단계 스케줄러는 소수 개의 타스크에 대해서만 서비스 시간을 결정해도 충분하다는 것을 의미한다. 이러한 관찰에 의해 우리는 각 상위 단계 스케줄링의 실행에서 일부 타스크들에 대해서만 서비스 시간을 결정하는 스케줄링 방식을 제안한다.

3. 시스템 모델과 배경

단일 프로세서 상에서 실행되는 선점형(preemptive) 실시간 타스크 집합이 있다고 하자. 타스크 집합에 속한 각 타스크, $t_i(i=1,2,\cdots,N)$는 임의의 도착 시간 r_i, 만기 τ_i, 가치 함수 $f_i(\ )$을 갖는다. 여기서 $f_i(\ )$는 타스크 t_i에 할당된 서비스 시간 x_i에 대한 함수이다. 가치 함수를 갖는 타스크들을 스케줄하기 위하여 몇 가지 기준이 적용될 수 있는데, 예를 들면 가치 $f_i(x_i)$의 합을 최대로 하거나[1], 아니면 오차 값의 최대를 최소화하는 것[8] 등이다. 본 논문에서는 타스크들의 가치 $f_i(x_i)$의 합이 최대가 되도록 타스크들을 스케줄링한다. 여기서 $f_i(\ )$는 지수 함수로 가정하였으며, g_i는 타스크 t_i의 가치 함수의 도함수이다. 즉, $g_i = df_i/dt$.

제안된 알고리즘은 2부분으로 나누어진다 : 상위 단계 알고리즘과 하위 단계 알고리즘. 상위 단계 알고리즘은 태스크들의 가치 합이 최대가 되도록 스케줄 가능한 타스크들에게 실행 시간을 할당하고, 하위 단계 알고리즘을 이 정보에 따라 타스크들에게 프로세서를 할당한다.

상위 단계 알고리즘은 타스크가 도착할 때마다 실행되므로 실행 시점에서 보면 스케줄 대상이 되는 타스크들의 도착 시간을 현 시점으로 간주하여, 동일한 도착 시간을 갖는, 유한 개의 타스크로 이루어진 타스크 집합을 대상으로 스케줄링하므로 다음과 같은 정적 최적화 문제, P로 변형하여 모형화 할 수 있다:

(P) : 동일한 도착 시간을 갖는, 만기 τ_i와 가치 함수 f_i가 알려진, 선점형 실시간 타스크들의 집합이 주어졌을 때, 타스크들의 가치 합이 최대가 되도록 하는, 타스크 t_i의 서비스 시간 x_i를 결정함.

위의 문제는 모든 x_i가 구해졌을 때 해결되었다고 한다. 기존의 연구들은 가능한 한 빠르게 해답을 찾는

알고리즘에 대한 연구가 주를 이루었다. 앞의 관측(하위 단계 스케줄러는 상위 단계 알고리즘에 의해 얻어진 타스크들의 총 서비스 시간 x_i 중에서 극히 일부만을 사용하는 경우가 많다)에 따라 우리는 상기 문제 P에 대한 해답(모든 타스크들에 대한 서비스 시간, x_i)의 부분집합을 구하는 접근 방식을 개발하였다.

다음 상위 단계 알고리즘의 실행, 또는 다음 타스크의 도착까지 사용되는 x_i의 개수를 정확히 아는 것은 불가능하지만, 하위 단계 알고리즘이 x_i를 사용하는 순서(즉, 타스크의 실행 순서)를 정하는 것은 가능하다. [1]의 실험 결과에 따라 다른 것들보다 성능이 높다고 평가된 EDF가 하위 단계의 알고리즘으로 사용되는 경우, 보다 빠른 만기를 갖는 t_i의 $x_i(>0)$가 사용되지 않는다면, 그것보다 더 늦은 만기를 갖는 t_j의 x_j는 당연히 사용되지 않을 것이다. 따라서 앞의 관측에 따라 상위 단계 알고리즘에서 한번에 모든 타스크의 x_i를 계산하지 않고, 보다 빠른 만기를 갖는 타스크들의 x_i만을 부분적으로 구하는 것이 보다 좋은 방법일 것이다. 하위 단계 알고리즘이 상위 단계 알고리즘에 의해 결정된 모든 x_i를 사용하기 전에 새로운 타스크가 도착한다면, 상위 단계 알고리즘은 재실행된다. 하위 단계 알고리즘에 의해 사용되지 않을 x_i들을 계산하지 않음으로써 상당한 처리 시간이 절약될 수 있다. 구해진 모든 x_i가 완전히 사용된 이후에도 타스크가 도착하지 않으면, 상위 단계 알고리즘은 남아있는 타스크들을 대상으로 재실행되어 새로운 x_i 집합을 만들어낸다.

온라인 스케줄링을 위한 최선의 방법은 새로운 타스크가 도착하기 전에 완전히 사용될 수 있는 수의 x_i만을 구하는 것이다. 그러나 그렇게 하기 위해서는 타스크들에 대해 사전 지식이 필요하다. x_i가 너무 적게 구해지면, 상위 단계 알고리즘은 빈번히 실행될 것이다. 반대로 x_i를 너무 많이 구하면, CPU 시간이 낭비될 것이다. 그러면 x_i의 수는 어느 정도가 적당한가? 이에 대하여 다음 장에서는 적당한 실행 시간 리스트인, (P)의 해답 $\{x_i\}$의 부분집합 $\mathcal{Y}=\{y_i\}$을 구하는 알고리즘을 유도한다. 여기서 x_i는 (P)의 모든 타스크에 대해 얻어진 서비스 시간이며, y_i는 제안된 알고리즘에 의해 얻어진 서비스 시간으로 x_i의 부분집합이다.

4. 휴리스틱 온라인 알고리즘

(그림 1)에 제안된 알고리즘을 보였다.

Algorithm : *Heuristic algorithm producing total reward as much as possible. This algorithm is executed at every scheduling point.*

Input : *S = a list of tasks that are currently present in the system and sorted in increasing order of their deadlines.*
t^* = a new task (may be empty)

Function

if a new task t^* arrives, then insert t^* into S;
let τ_0 be the current time;
get $\phi(\tau_0)$, the minimum of maximum derivative for the tasks in S;
let $\aleph = \{t_i \mid g_i(0) \geq \phi(\tau_0)\}$, and $y_i = g_i^{-1}(\phi(\tau_0))$, for all $t_i \in \aleph$;
find the next scheduling point τ_k such that $\sum_{i=1}^{k} y_i = \tau_k - \tau_0$;
schedule tasks t_i such that $\tau_i \leq \tau_k$, one by one, by the EDF until the next scheduling point and remove all terminating tasks from S;

(그림 1) 제안된 알고리즘

이 알고리즘은 매 스케줄링 시점(scheduling point)에서 실행된다. 스케줄링 시점은 새로운 타스크가 도착하거나, 또는 하위 단계 스케줄러에 의해 새로운 y_i 집합이 요구되는 시점이다(앞 장에서 기술하였듯이 제안된 알고리즘은 문제 (P)의 해답, 모든 x_i를 결정하는 것이 아니라, x_i의 부분집합인 y_i를 구한다).

제안된 알고리즘은 우선 각 스케줄링 시점에서 현재의 시점 τ_0 기준으로 한 타스크의 가치 함수의 도함수 값 중 최대 값을 최소화 한 값(minmax value) $\phi(\tau_0)$을 구한다. $\phi(\tau_0)$ 값은 이분법(bisection method, 최대치는 최대 도함수 값, 최소치는 0으로 함)에 의해 구해지는데 일정 횟수 또는 일정 정도(precision)에 도달할 때까지 반복된다. 이것은 새로운 타스크가 도착하지 않는다고 가정하여 현재 시스템에 들어있는 타스크들 $(t_1, t_2, \cdots, t_N)$이 최소 y_i 만큼의 서비스 시간을 할당 받아 실행되어 각각의 만기까지 완료되도록 하는 스케줄(feasible schedule) 수립을 가능하게 하는 값이다. 그런 후 구해진 minmax 값보다 큰 도함수 값을 갖는 타

스크들을 선정하고(이것들이 서비스 시간을 할당 받을 타스크들임), 각각에 대해 minmax 값에 의한 서비스 시간(y_i, 각 타스크의 가치 함수에서 함수 값을 min-max로 취했을 때, 그에 따른 역함수 값)을 결정한다. 타스크의 만기 중 구해진 y_i 값들의 합에 최초로 일치하거나, 이하 값으로 가장 근접하는 타스크의 만기를 선택하여 다음 스케줄링 시점 τ_k을 정한다. 현 시점 τ_0와 τ_k와의 시간 간격만큼 선정된 타스크들 중에서 만기가 빠른 순서로 서비스 시간을 할당한다. 할당된 서비스 시간의 합은 τ_0와 τ_k와의 시간 간격과 같거나 작아야 한다. 여기서 구해진 서비스 시간이 제안 알고리즘에 의한 y_i 값들이다. 현 시점에서 만약 τ_k 이전에 새로운 타스크가 도착하면 그 시점이 실제적인 스케줄링 시점이 되고, 새로 도착한 타스크를 포함하여 다시 새로운 y_i 집합을 구하게 된다. 물론 스케줄링 시점보다 앞선 만기를 갖는 타스크들은 제외된다. 이러한 과정은 약간의 오버헤드만을 도입할 뿐이므로 전체적으로 실행 시간에 미치는 영향은 크지 않다. τ_k에 대해 다음과 같은 조건이 성립된다 :

$$\sum_{i=1}^{k} y_i = \tau_k - \tau_0$$

여기서 $\tau_i \le \tau_k$에 대하여 $y_i = g_i^{-1}(\phi(\tau_0))$

$g_i^{-1}(\ )$는 $g_i(\ )$의 역함수이며,

$g_i(0) < \phi(\tau_0)$이면, $y_i = g_i^{-1}(\phi(\tau_0)) = 0$

(사실 $g_i(\ )$는 감소함수이므로 $g_i(0) < \phi(\tau_0)$인 경우, $g_i^{-1}(\phi(\tau_0))$는 존재하지 않는다)

최대 도함수를 최소화 한 값 $\phi(\tau_0)$과 (P)에 대한 해답과의 관계, 제안된 알고리즘에 의해 얻어진 처리 시간 집합 $\{y_i\}$와 (P)의 해답으로서의 처리 시간 $\{x_i\}$과의 관계 등은 다음과 같이 설명될 수 있다. [1]에서 제안된 Dey의 알고리즘이 $\{x_i\}$를 구하는 방법 중의 하나이다. Dey의 알고리즘은 타스크들의 만기들로 이루어진 구간 $(\tau_{i-1}, \tau_i]$, i = 1, 2, ···, N들에 대해 늦은 구간부터 이른 구간으로 후방 반복법(backward iteration)에 의해 x_i를 구한다. 구간 $(\tau_{i-1}, \tau_i]$에서 Dey의 알고리즘은 그 구간에서 스케줄할 수 있는 타스크들$(t_i, t_{i+1}, t_{i+2}, \cdots, t_N)$ 중에서 가장 큰 도함수를 갖는 타스크들에게 구간 시간을 분배한다. 반복이 완료된 이후 모든 타스크에 대한 서비스 시간 x_i가 결정된다. 제안된 알고리즘에 의한 $\{y_i\}$가 Dey의 알고리즘에 의한 $\{x_i\}$의 부분집합임은 다음 정리에 의해 증명된다. 아래에서 d는 x_i에 대한 최대 도함수 $g_i(x_i)$이다.

(정리)

$\aleph = \{t_i \mid g_i(0) \ge \phi(\tau_0)\} = \{t_1, t_2, \cdots, t_L\}$의 원소인 타스크들이 만기에 의해 정렬되어있고, (즉, $\tau_i < \tau_{i+1}$, i=1, 2, ···, L-1), x_i를 시점 τ_0에서 (P)에 대한 서비스 시간이라고 하자. $\aleph_d = \{t_i \mid g_i(x_i) = d\}$라고 하면,

(1) $\phi(\tau_0)$는 d와 같고, $\aleph_d$는 $\aleph$의 부분집합이다.

(2) $\tau_i < \tau_j$인 타스크 t_i와 $\aleph$에 속한 t_j에 대해 t_j이 $\aleph_d$에 속하면 t_i도 $\aleph_d$에 속한다.

(3) $\sum_{i=1}^{k} y_i = \tau_k - \tau_0$을 만족하는 $1 \le k \le L$인 k가 있다면(i = 1, 2, ···, k에 대해 $y_i = g_i^{-1}(\phi(\tau_0))$), 그러한 k와 모든 $1 \le i \le k$인 i에 대해 $y_i = x_i$

(증명)

(1) d는 또한 minmax $g_i(\ )$이므로, 당연히 $\phi(\tau_0) = d$이며, $\aleph_d$는 $\aleph$의 부분집합이다.

(2) $\tau_i < \tau_j$이고 $t_i \notin \aleph_d$, $t_j \in \aleph_d$인 타스크 t_i와 t_j가 $\aleph$에 속한다고 하자. t_i가 $\aleph_d$에 속하지 않고, t_j가 $\aleph$에 속했다는 것은 t_i에 할당된 서비스 시간 y_i는 $g_i^{-1}(\phi(\tau_0))$보다 크다는 것을 의미한다. $g_i(\ )$는 감소 함수이므로, $m = y_i - g_i^{-1}(\phi(\tau_0))$와 같은 길이의 구간이 존재한다. t_i의 만기가 t_j보다 앞서므로, t_j 또한 그 구간에 스케줄 될 수 있다. 이것은 t_i와 t_j가 그 구간을 공유하도록 $g_j(\ )$를 감소시킬 수 있다는 것을 의미한다. 따라서 t_j는 $\aleph_d$에 속해서는 안된다. 이것은 가정 $t_j \in \aleph_d$에 모순된다.

(3) 모든 k = 1, 2, ···, L에 대해 $\sum_{i=1}^{k} y_i < \tau_k - \tau_0$라고 하자. $m = \text{minimum}\{(\tau_k - \tau_0) - \sum_{i=1}^{k} y_i \mid k = 1, 2, \cdots, L\}$라 하면, y_k를 모든 k = 1, 2, ···, L에 대해 m/L 만큼 증가시키는 것이 가능하다. 왜냐하면,

$$\sum_{i=1}^{k}(y_i + m/L) \le \sum_{i=1}^{k} y_i + m$$

$$= \sum_{i=1}^{k} y_i + \text{minimum}\{(\tau_k - \tau_0) - \sum_{i=1}^{k} y_i\}$$

$\leq \sum_{i=1}^{k} y_i + (\tau_k - \tau_0) - \sum_{i=1}^{k} y_i = (\tau_k - \tau_0)$

이기 때문이다. 이것은 $\phi(\tau_0)$를 감소시킬 수 있음을 의미하는데, $\phi(\tau_0)$가 minmax 도함수라는 가정에 모순된다. 그래서 다음과 같이 결론을 내릴 수 있다 : $\exists k$, s.t. $k \leq L$,

$$\sum_{i=1}^{k} y_i = \tau_k - \tau_0$$

모든 $i \leq k$에 대해 y_i를 증가시킬 수 없다는 사실은 다음을 의미한다 :

$$t_i \in \aleph_d, \quad y_i = x_i. \text{ Q.E.D.}$$

S내의 타스크들은 이미 각자의 만기에 따라 오름차순으로 정렬되어 있기 때문에, minmax 도함수 $\phi(\tau_0)$는 [9]에서 제시된 알고리즘을 적용하면 O(N) 시간 이내에 계산될 수 있다. [9]에서 제시된 알고리즘은 이분법에 의해 minmax를 구한다.

스케줄링 시점에서 y_i를 구하는 복잡도가 O(N)일지라도, 시스템 내에서 N개의 타스크를 완료하기 위한 최악의 복잡도는 여전히 $O(N^2)$이다. 극단적인 경우는 1) 각 스케줄링 시점에서 제안된 알고리즘이 오직 1개의 y_i만 계산하고, 그것이 완료될 때까지 새로운 타스크가 도착하지 않은 경우와 2) 이 프로시주어가 모든 타스크가 완료될 때까지 반복되는 경우 등 이다. 이러한 경우 알고리즘은 O(N)번 실행 되고, 전체적인 복잡도는, Dey에 의해 제안된 알고리즘의 복잡도와 같은, $O(N^2)$이다. 그러나 새로운 타스크들이 빈번히 도착하고, 그것들의 가치 함수가 급격히 변하지 않으면, 제안된 알고리즘은 여러 경우에 Dey의 알고리즘보다 효율이 좋으며, $O(N^2)$보다 훨씬 작은 평균 복잡도를 가진다.

5. 예

이번 장에서는 제안된 알고리즘의 동작을 설명하기 위해 <표 1>과 같은 간단한 예를 보인다. 타스크들의 가치 함수는 지수 함수로 가정한다. τ_0는 현 시점이다. time = 0인 시점에서 시스템에는 타스크 t_1만이 있다. 가치함수는 $f_1(x) = 1 - e^{-0.4x}$이며, 만기는 $\tau_1 = 10$이다. 현 시점에서 t_1이 유일한 타스크이므로 time = 0에서 $y_1 = 10$, 다음 스케줄링 시점은 10이 된다. 그리고 하위 단계 스케줄러에 의해 EDF 전략으로 타스크 t_1에게 프로세서가 할당된다.

〈표 1〉 타스크 집합

t_i	r_i	τ_i	$f_i(x)$
t_1	0	10	$1 - e^{-0.4x}$
t_2	1	4	$1 - e^{-0.2x}$
t_3	2	3	$1 - e^{-0.2x}$
t_4	4	7	$1 - e^{-0.4x}$

time = 1인 시점에서 가치 함수가 $f_2(x) = 1 - e^{-0.2x}$이고, $\tau_2 = 4$인, 새로운 타스크 t_2가 도착한다. t_1은 time = 0에서 time = 1까지 이미 1 time unit을 실행 때문에 가치 함수는 $f_1(x) = 1 - e^{-0.4(x+1)}$로 수정된다. 새로운 타스크가 도착했기 때문에 스케줄링 시점은 1로 되고, 제안 알고리즘은 재실행된다. 그리고 $\phi(2) = \text{minmax}\{g_1(y_1), g_2(y_2)\} = 0.109762$이 된다. $y_2 = g_2^{-1}(\phi(2)) = g_2^{-1}(0.109762) = 3$ $(= \tau_2 - \tau_0 = 3)$, $y_1 = g_1^{-1}(\phi(2)) = g_1^{-1}(0.109762) = 2.233$이고, $y_1 + y_2 = 5.233 < \tau_1 - \tau_0 = 9$이므로 다음 스케줄링 시점은 4가 되고, 그 시점까지 t_2만이 하위 단계 알고리즘에 의해 스케줄 된다. 그러나 t_1은 고려되지 않는다. 그 이유는 y_2는 총 가치를 최대화 시키지만, y_1은 그렇지 않기 때문이다. 전통적인 방법은 총 가치를 최대화 하는 y_2 뿐만 아니라 y_1까지도 구하지만, 우리가 제안한 알고리즘은 y_1을 계산하지 않는다.

time = 2인 시점에서 가치 함수가 $f_3(x) = 1 - e^{-0.2x}$이고, $\tau_3 = 3$인, 새로운 타스크 t_3가 도착한다. t_1은 time = 0에서 time = 1까지, 그리고 t_2는 time = 1에서 time = 2까지 각각 1 time unit을 실행했기 때문에 가치 함수는 $f_1(x) = 1 - e^{-0.4(x+1)}$, $f_1(x) = 1 - e^{-0.2(x+1)}$로 수정된다. 새로운 타스크가 도착했기 때문에 스케줄링 시점은 2가 되고, 제안된 알고리즘은 재실행된다. 그리고 $\phi(3) = \text{minmax}\{g_1(y_1), g_2(y_2), g_3(y_3)\} = 0.163746$가 된다. $y_3 = g_3^{-1}(\phi(3)) = g_3^{-1}(0.163746) = 1$, $y_2 = g_2^{-1}(\phi(3)) = g_2^{-1}(0.163746) = 0.000005$, $y_1 = g_1^{-1}(\phi(3)) = g_1^{-1}(0.163746) = 1.23$이고, $y_3 = 1 < \tau_3 - \tau_0 = 1$ 이므로 다음 스케줄링 시점은 3이 되고, 그 시점까지 t_3가 하위 단계 알고리즘에 의해 스케줄 된다. 그러나 t_1, t_2는 고려되지 않는다.

다음 스케줄링 시점인 time = 3에서 더 이상 새로운 타스크가 도착하지 않는다면, EDF 전략에 따라 t_3가 시스템을 떠나고, 제안된 알고리즘은 스케줄링 시점 3에서 타스크 t_1, t_2에 대하여 재실행되어 앞에서와 마찬가지의 과정을 통해 y_2 = 1을 할당한다. time = 4 시점에서 새로운 타스크 t_4가 도착되므로 상위 단계 알고리즘은 그 시점에서 t_1과 t_4를 대상으로 실행되어 time = 7까지 y_4 = 3을 할당한다. time = 7인 시점에서는 더 이상 도착하는 타스크가 없이 t_1만이 남으므로 그것의 만기까지 실행한다. <표 2>는 제안된 알고리즘의 실행 결과를 보이고 있는데, 각 스케줄링 시점 r_i = 0, 1, 2, 3, 4 그리고 7에서 타스크들이 할당받을 서비스 시간(y_i)과 그 시점에서의 가치 함수를 보이고 있다.

6. 결 론

본 논문에서는 실시간 타스크의 총 가치를 최대화하는 동적 스케줄링 알고리즘에 대하여 기술하였다. 제안 알고리즘은 매 스케줄링 시점에서 실행되어 다음 스케줄 할 시점을 결정하고, 그 시점과 같거나 작은 만기를 갖는 타스크들에게만 서비스 시간을 할당한다. 이러한 방법으로 제안 알고리즘은 시스템에 들어있는 타스크 중 일부에 대해서만 서비스 시간을 결정한다. 이와는 달리 Dey가 제안한 알고리즘은 현재 시스템에 들어있는 모든 타스크에게 서비스 시간을 할당한다. 제안된 알고리즘의 계산 복잡도는 매 실행마다 O(N)인 반면, Dey의 알고리즘은 $O(N^2)$이다[1].

제안된 알고리즘이 Dey의 것보다 더 빈번히 실행될 수도 있으나, 한번에 한 개의 타스크에 대한 서비스 시간이 구해지는 최악의 경우라도 minmax 값을 구하기 위해 수행되는 반복 횟수는 전체적인 실행 시간에 큰 영향을 미치지 않는다. 이러한 경우 복잡도는 $O(cN^2)$ 같다. 제안 알고리즘은 간단하고, 평균 복잡도가 기존 연구의 알고리즘들보다 낮아, 제안 알고리즘이 성능과 실제 활용성이 높을 것으로 생각한다.

참 고 문 헌

[1] J.K. Dey, J.F. Kurose, and D. Towsley, "On-line scheduling policies for a class of IRIS(Increasing Reward with Increasing Service) real-time tasks," IEEE Transactions on Computers, Vol.45, No.7, pp.802-813, July, 1996.

[2] M. Boddy and T. Dean, "Solving time-dependent planning problems," Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89), Detroit, Mi., IJCAI-89, pp.979-984, August 1989.

[3] E.K. Chong and W. Zhao, "Performance evaluation of scheduling algorithms for imprecise computer systems," Journals of Systems and Software Vol.15, No.3, pp.261-277, July 1991.

[4] K. Decker, V.R. Lesser, and R.C. Whitehair, "Extending a blackboard architecture for approximate processing," The Journal of Real-Time Systems, Vol.2, pp.47-79, 1990.

[5] J. Liu, "Timing constraints and algorithms," in Report on the Embedded AI Languages Workshop, University of Michigan, pp.9-11, November 1988.

[6] J. Liu, K. Nahrstedt, D. Hull, S. Chen, and B. Li, "EPIQ QoS characterization," Draft, July 1997.

[7] J.A. Stankovic, M. Spuri, M.D. Natale, and G. Buttazzo, "Implication of classical scheduling

〈표 2〉 제안된 알고리즘의 실행 결과

t_i	r_i	r_i	$r_0=0$		$r_0=1$		$r_0=2$		$r_0=3$		$r_0=4$		$r_0=7$	
			y_i	$f_i(x)$	y_i	$f_i(x)$	y_i	$f_i(x)$	y_i	$f_i(x)$	y_i	$f_i(x)$	y_i	$f_i(x)$
t_1	0	10	10	$1-e^{-x}$		$1-e^{-0.4(x-1)}$		$1-e^{-0.4(x-1)}$		$1-e^{-0.4(x-1)}$		$1-e^{-0.4(x-1)}$	3	$1-e^{-0.4(x+1)}$
t_2	1	4			1	$1-e^{-0.2x}$		$1-e^{-0.2(x+1)}$	1	$1-e^{-0.2x}$				
t_3	2	3					1	$1^{-e-0.2x}$						
t_4	4	7									3	$1-e^{-0.4x}$		

results for real-time systems," Computers, Vol. 28, No.6, pp.16-25, June 1995.

[8] W.K. Shih and J. Liu, "On-line scheduling of imprecise computations to minimize error, Proceedings of the 13th Real-Time Systems Symposium," Phoenix, Arizona, pp.280-289, December 1992.

[9] K. Choi, G. Jung, T. Kim, and S. Jung, "Real-time scheduling algorithm for minimizing maximum weighted error with O(NlogN + cN) complexity," Information Processing Letters, Vol.67, No.6, pp.311-315, 1998.

김 인 수

e-mail : insoo@etri.re.kr

1981년 경기대학교 관광경영학과 졸업(경영학사)

1983년 중앙대학교 대학원 컴퓨터공학과 졸업(이학석사)

1999년 아주대학교 대학원 컴퓨터공학과 박사

1983년~현재 한국전자통신연구원 연구원

관심분야 : 실시간 시스템, 분산 시스템, 멀티미디어 시스템

이 윤 열

e-mail : yylee@dragon.cntc.ac.kr

1968년~현재 천안공업대학 전자계산학과 교수

관심분야 : 운영체제, 분산 시스템, 실시간 시스템

이 춘 희

e-mail : chlee@seri.re.kr

1964년 성균관대학교 영어영문학과(학사)

1985년 연세대학교 대학원 전산학과(석사)

1990년 아주대학교 대학원 컴퓨터공학과 박사 과정

1968년~1998년 시스템공학연구 및 ETRI 책임연구원

관심분야 : 운영체제, 네트워크 컴퓨팅 시스템, 실시간 시스템

정 기 현

e-mail : khchung@madang.ajou.ac.kr

1990년 Purdue 대학(미) 전기공학과 졸업

1992년~현재 아주대학교 전기전자공학부 교수

관심분야 : 컴퓨터구조, VLSI 설계, 멀티미디어, 실시간 시스템

최 경 희

e-mail : khchoi@madang.ajou.ac.kr

1982년 Paul Sabatier 대학(프) 정보공학과 졸업

1982년~현재 아주대학교 정보 및 컴퓨터공학부 교수

관심분야 : 운영체제, 분산시스템, 멀티미디어, 실시간시스템