부분 해를 이용한 IRIS 실시간 태스크용 온-라인 스케줄링 알고리즘의 성능향상

(Performance Enhancement of On-Line Scheduling Algorithm for IRIS Real-Time Tasks using Partial Solution)

심 재 홍 † 최 경 희 †† 정 기 현 †††
(Jae-Hong Shim) (Kyung-Hee Choi) (Gi-Hyun Jung)

요 약 본 논문에서는 가치함수를 가지면서 동적으로 도착하는 IRIS(Increasing Reward with Increasing Service) 실시간 태스크들의 총 가치를 최대화하기 위한 온-라인 스케줄링 알고리즘을 제안한다. 본 논문은 스케줄링 알고리즘의 성능향상에 역점을 두고 있으며, 이는 다음 두 가지 아이디어를 기반으로 한다. 첫째, 총가치를 최대화하는 문제는 가치함수들의 최대 도함수 값들 중 최소 값을 찾는 문제를 해결함으로써 풀 수 있다는 것이다. 둘째, 새로운 태스크가 도착하기 전까지 이 전에 스케줄된 태스크들 중 소수만이 실제 실행되고, 나머지는 새로 도착한 태스크와 함께 다시 스케줄링 된다는 사실을 발견하고, 매 스케줄링 시 모든 태스크들을 스케줄링하는 것이 아니라, 일부 태스크들만 스케줄링하자는 것이다. 제안 알고리즘의 성능은 다양한 경우에 대한 모의실험으로 검증되었다. 실험 결과 제안 알고리즘의 계산 복잡도는 최악의 경우 기존 알고리즘과 동일한 $O(N^2)$이지만, 평균적으로 이 보다 낮은 $O(N)$에 가까운 것으로 확인되었다

키워드 : 실시간 시스템, 온-라인 스케줄링, 총가치의 최대화

Abstract In this paper, we propose an on-line scheduling algorithm with the goal of maximizing the total reward of IRIS(Increasing Reward with Increasing Service) real-time tasks that have reward functions and arrive dynamically into the system. We focus on enhancing the performance of scheduling algorithm, which is based on the following two main ideas. First, we show that the problem to maximize the total reward of dynamic tasks can also be solved by the problem to find minimum of maximum derivatives of reward functions. Secondly, we observed that only a few of scheduled tasks are serviced until a new task arrives, and the rest tasks are rescheduled with the new task. Based on our observation, the proposed algorithm doesn't schedules all tasks in the system at every scheduling point, but a part of tasks. The performance of the proposed algorithm is verified through the simulations for various cases. The simulation result showed that the computational complexity of proposed algorithm is $O(N^2)$ in the worst case which is equal to those of the previous algorithms, but close to $O(N)$ on the average.

Key word : real-time system, on-line scheduling, maximizing total reward

1. 서 론

실시간 태스크 스케줄링은 주어진 모든 태스크들의 시간제약을 만족해야 한다. 경성(hard) 실시간 시스템에서 스케줄러는 사전에 정의된 태스크들의 만기(dead line), 실행시간, 도착시간, 주기 등의 시간제약을 모두 만족시켜야 한다[1]. 만약 만기 전까지 태스크들의 실행을 완료하지 못할 경우, 그때까지 생성된 중간 실행 결과는 아무런 의미가 없는 것으로 간주하고 버려진다.

그러나 사전에 태스크들의 실행시간을 고정시킬 수 없거나 또는 중간 실행 결과 역시 의미를 가지는 응용

· 본 논문은 2002년도 조선대학교 학술연구비 및 국가지정연구실 사업의 지원에 의해 연구되었음.
† 정 회 원 : 조선대학교 인터넷소프트웨어공학부 교수
jhshim@chosun.ac.kr
†† 종신회원 : 아주대학교 정보통신전문대학원 교수
khchoi@madang.ajou.ac.kr
††† 비 회 원 : 아주대학교 전자공학부 교수
khchung@madang.ajou.ac.kr
논문접수 : 2002년 8월 30일
심사완료 : 2002년 11월 18일

들이 있다. 예를 들어, 비디오 이미지를 압축하거나 해제하는 응용[2], 멀티미디어 응용[3], 정보 수집 및 추출[4], 데이타베이스 질의 처리[5], 자동 항법 계획[6], 실시간 휴리스틱(heuristic) 탐색[7], 의료 결정 시스템[8], 전통적인 반복적인 수치 알고리즘 계산 등과 같은 응용들을 들 수 있다. 이러한 응용에서는 해당 태스크에게 프로세서 시간을 많이 할애할수록 그 만큼 태스크의 실행 결과의 품질은 높아지게 된다. 즉, 태스크를 만기 전까지 완료하지 못하고, 부분적으로 실행한 중간 결과 역시 나름대로의 의미(가치)를 가진다. 부분적으로 실행됨으로써 생성된 중간 결과는 최소한 우리에게 그 태스크가 무엇을 하려고 했는지에 대한 대략적인 정보를 제공하거나 또는 수용 가능한 범위내의 근사치 결과를 제공한다.

이러한 응용들에게 소속된 태스크들은 그들의 실행 결과로 생성되는 가치(reward)를 가지게 되는데, 이는 만기 전까지 각 태스크에게 주어진 서비스 시간(실제 실행시간) 양의 함수 값으로 표현된다. 이 함수를 해당 태스크의 가치함수(reward function)라 하며, 점진적 증가 볼록(strictly increasing concave) 함수 특성을 가진다. 즉, 각 태스크에게 주어지는 서비스 시간이 증가 할수록 해당 태스크가 얻는 가치 또한 증가한다. 이러한 태스크를 IRIS(Increasing Reward with Increasing Service) 태스크라 한다[9, 10]. IRIS 태스크 모델은 시스템에 부하가 적을 경우 각 태스크에게 충분한 프로세서 시간을 할애하여 정확한 출력 결과(고 품질)를 생성하지만, 시스템의 부하가 높을 경우 각 태스크에게 상대적으로 적은 프로세서 시간을 할당하여 허용 가능한 범위 내의 정확도를 가진 중간 실행 결과를 생성하도록 한다. IRIS 태스크 모델에서는 기존의 부정확 계산 모델(imprecise computation model)[11]과는 달리 하나의 태스크가 다시 필수 서브태스크(mandatory subtask)와 선택 서브태스크(optional subtask)로 분리되지 않으며, 태스크의 실행시간 또한 사전에 결정되는 것이 아니라, 스케줄러에 의해 동적으로 결정된다. 스케줄러는 스케줄링 목적(총가치의 최대화)에 따라 모든 태스크들의 만기를 넘기지 않는 범위 내에서 각 태스크에게 할당할 서비스 시간을 결정해야 한다.

IRIS 모델과 관련한 기존 연구는 주로 모든 태스크들의 총가치(모든 태스크들의 가치의 합: total reward)를 최대화 시키는 스케줄링 알고리즘을 찾는데 주력했다. 참고 문헌[10, 12]에서 저자들은 새로운 태스크가 도착하지 않는다는 가정 하에 현재 시스템에 존재하는 IRIS 태스크들의 총가치를 최대화하는 정적 문제에 대한 최적(optimal) 해법을 구하는 알고리즘을 제안했다. 이 알고리즘의 계산 복잡도는 O(N2)이며, 여기서 N은 시스템 내의 태스크의 개수이다. 이를 기반으로 온-라인으로 스케줄링하는 세가지 스케줄링 정책을 제안하였고, 또한 IRIS 스케줄링 정책들이 생성할 수 있는 총가치에 대한 극한치를 구하는 해석학적 모델도 제시했다. 이 알고리즘과 동일한 계산 복잡도와 성능을 가진 또 다른 알고리즘이[13]에서 제시되기도 했다. 그러나 이들 두 알고리즘들[10, 13]은 새로운 태스크가 임의의 시간에 동적으로 도착할 때마다 기존의 스케줄링 결과(각 태스크에게 할당할 서비스 시간)를 무시하고, 재 스케줄링을 수행하여 모든 태스크들의 서비스 시간을 다시 결정해야 하는 스케줄링 부하를 가진다. 최근에 주기적 실시간 태스크들에 대한 최적 가치-기반 스케줄링(reward-based scheduling) 기법[14]이 제안되었는데, 여기서 각 태스크들은 부정확 계산 모델과 같이 필수적/선택적 서브태스크로 구성되지만, 선택적 서브태스크들은 선형 에러(linear error) 함수를 가지는 것이 아니라, IRIS 모델과 같은 점진적 증가 볼록형 가치 함수들을 가진다. 각 태스크의 선택적 서브태스크의 실행시간이 매 주기마다 항상 동일하게 주어진다는 조건 하에 최적의 스케줄이 항상 존재한다는 것을 증명했으며, 선택적 서브태스크의 실행시간을 효율적으로 결정하는 방법을 제시하였다. 그러나 가치기반 스케줄링 모델은 전체적으로 부정확 계산 모델을 따르고 있다.

IRIS 태스크들의 총가치를 최대화 시키기 위한 기존 온-라인 최적 스케줄링 알고리즘들[10, 12, 13]은 많은 태스크들을 가진 실제 시스템에 적용하기에는 여전히 높은 계산 복잡도를 가진다. 따라서 본 논문에서는 보다 낮은 평균 계산 복잡도를 가진 휴리스틱 온-라인 스케줄링 알고리즘을 제안한다. 제안 알고리즘은 다음 두 가지 아이디어를 기반으로 한다. 첫째, 동적 태스크들의 총가치를 최대화하는 문제는 가치함수들의 최대 도함수 값들 중 최소 값을 찾는 문제로 변환하여 해결할 수 있다는 것이다. 둘째는 새로운 태스크가 도착하기 전까지 이 전에 스케줄된 태스크들 중 소수만이 실제 실행되고, 나머지는 새로이 도착한 태스크와 함께 다시 스케줄링 된다는 사실을 발견하고, 매 스케줄링 시 모든 태스크들을 스케줄링할 것이 아니라, 실제 실행될 것으로 예측되는 일부 태스크들만 스케줄링하자는 것이다. 이러한 전략을 통해 제안 알고리즘은 기존 온-라인 최적 스케줄링 알고리즘들과 동일한 총가치를 생성하면서도 이들보다 더 낮은 평균 스케줄링 복잡도를 가질 수 있다.

본 논문의 구성은 다음과 같다. 2장에서 IRIS 태스크

모델과 연구 동기를 기술한다. 3장에서는 IRIS 태스크들을 위한 휴리스틱 온-라인 스케줄링 알고리즘을 제안한다. 4장에서는 제안 알고리즘의 성능을 측정하기 위해 실시된 다양한 모의실험 결과를 보이고, 이를 분석한다. 마지막 5장에서 본 논문의 결론과 향후 연구 방향을 제시한다.

2. 시스템 모델 및 연구 동기

단일 프로세서 상에서 실행되는 선점 가능한 태스크들이 있다고 가정하자. 각 태스크 t_i, i = 1, 2, .., N,는 도착시간 r_i, 만기 τ_i, 가치함수 $f_i(x)$를 가진다. 태스크는 임의의 시간에 시스템에 도착할 수 있고, 만기가 지난 후에는 시스템에서 삭제된다. 태스크들의 실행시간은 스케줄러에 의해 결정된다. 각 태스크의 가치함수 $f_i(x)$는 스케줄러에 의해 태스크 t_i에 할당된 프로세서 시간(서비스 시간 또는 실행시간)의 양(x)의 함수로 정의된다. 즉, 태스크 t_i가 만기 전까지 총 x시간 동안 서비스를 받았다면 $f_i(x)$의 가치를 가진다. 가치함수는 점진적으로 증가하는 볼록한 모양을 가지며, 음수가 아닌 실수(real number)상에서 연속적으로 가능하다. 가치함수 $f_i(x)$의 도함수를 $g_i(x)$로 표현하며, $g_i(x) \equiv df_i(x)/dx$라 정의한다. 도함수 $g_i(x)$는 가치함수 $f_i(x)$가 점진적 증가 볼록형이므로 점진적 감소 오목형(strictly decreasing convex) 함수이다. 즉, 태스크의 실행시간이 증가함에 따라 가치는 여전히 증가하지만, 가치 증가율은 점점 감소함을 의미한다. 도함수 $g_i(x)$의 역함수는 $g_i^{-1}(x)$로 표현하며, $g_i(0)$를 스케줄링 시점(τ_0)에서의 순간 가치 증가율이라 한다.

IRIS 태스크 스케줄링 알고리즘의 궁극적인 목표 중의 하나는 태스크들의 총가치를 최대화 ($max \sum_{i=1}^{N} f_i(x_i)$) 시키는 것이며, 이를 위해 각 태스크에게 할당할 서비스(실행) 시간(x_i)을 결정하고, 또한 그들의 실행 순서를 결정하는 것이다. 그러나 태스크들이 언제 도착하고 그들의 만기가 얼마인지 사전에 알려져 있지 않으므로, 진정한 최적(optimal)의 총가치를 생성하도록 스케줄링 한다는 것은 매우 어렵다 (NP-hard). 따라서 더 이상의 태스크들이 도착하지 않는다는 가정 하에서 현재 시스템에 도착해 있는 태스크들을 상대로 총가치가 최대화되도록 스케줄링하는 온-라인 최적 (on-line optimal) 스케줄링 알고리즘들이 제안되었다[10, 12, 13]. 온-라인 알고리즘들은 더 이상의 태스크들이 도착하지 않는다는 전제 하에 스케줄링하므로, 새로운 태스크들이 도착할 때마다 매번 총가치를 최대화하기 위해 시스템 내의 모든 태스크들에게 할당할 서비스 시간을 다시 결정(재스케줄링) 한다. 따라서 새로운 태스크의 도착은 이전 스케줄링 시점에서 생성된 스케줄링 결과를 쓸모없게 만들며, 이전 스케줄링시 스케줄 되었지만(서비스 시간이 주어졌지만) 새로운 태스크의 도착 전까지 실행되지 못한 태스크들은 결과적으로 불필요하게 스케줄링 되었다는 것을 알 수 있다. 이는 곧 스케줄링 오버헤드를 상대적으로 증가시켰음을 의미한다.

본 연구의 모의실험 결과를 통해 알 수 있듯이 대부분의 경우 이전 스케줄링 시점에서 스케줄된 태스크들이 모두 실행을 완료하기 전에 새로운 태스크들이 도착한다. 따라서 온-라인 스케줄링 알고리즘의 복잡도를 줄일 수 있는 가장 확실한 방법은 매 스케줄링 시점에서 시스템 내의 모든 태스크들을 스케줄링 할 것이 아니라, 총가치를 최대화하면서 새로운 태스크가 도착하기 전까지 실행될 태스크들만을 스케줄링하는 것이다. 그러나 미래에 도착할 태스크들의 도착시간이 미리 주어지지 않는 이상 스케줄될 태스크들의 수를 정확히 예측한다는 것은 불가능하다. 다음 절에서 이러한 전략을 가진 휴리스틱 온-라인 스케줄링 알고리즘을 제안하고, 스케줄될 태스크들의 적절한 수를 찾는 방안에 대해서도 함께 논의한다.

3. 온-라인 스케줄링 알고리즘

본 논문에서는 기존 연구[10, 12, 13]에서 널리 채택된 상하 계층으로 분리된 온-라인 스케줄링 알고리즘을 기본 모델로 삼는다. 상위 계층의 알고리즘은 태스크들의 만기를 넘기지 않는 범위 내에서 최적의 총가치를 생성할 수 있도록 태스크들에게 할당할 서비스 시간을 결정한다. 하위 계층의 알고리즘은 태스크들의 실행 순서를 결정하며, 각 태스크에게 상위 계층에서 결정된 서비스 시간만큼만 프로세서를 할당한다. 최적의 총가치를 생성하기 위해 하위 계층의 알고리즘으로 EDF(earliest deadline first)[15] 알고리즘을 채택한 경우가 가장 우수한 것으로 보고된바 있다[10]. 본 논문에서 제안될 온-라인 스케줄링 알고리즘 역시 하위 계층 알고리즘으로 EDF를 채택하고 있으며, 본 연구에서는 상위 계층 알고리즘 개발에 역점을 둔다.

3.1 스케줄링 문제

IRIS 태스크들을 위한 온-라인 최적 스케줄링 알고리즘은 더 이상의 새로운 태스크가 도착하지 않는다는 가정 하에 다음의 정적(static) 스케줄링 문제(P)에 대한 최적 해를 구한다.

(P) : 만기(τ_i)와 가치함수($f_i(x_i)$)가 주어진 N개의 태

스크들이 시스템 내에 존재한다고 가정하자. 이 때 태스크들의 총가치가 최대화($max \sum_{i}^{N} f_i(x_i)$) 되도록 각 태스크에게 할당할 서비스 시간들의 집합, 즉 최적 해 $X=\{x_1,x_2,x_3, ..., x_N\}$을 구하라.

총가치를 최대화하는 가장 간단한 방법은 가치 증가율($g_i()$: 가치 도함수 값)이 가장 큰 태스크에게 우선적으로 프로세서를 할당하는 것이다. 스케줄링 문제(P)의 임의의 최적 해 X가 구해지면, 가치 도함수 값$(g_i(x_i))$들의 집합 $D=\{d_1,d_2,d_3, ..., d_M\}$와 태스크들의 집합 $A=\{A_1,A_2,A_3, ..., A_M\}$를 구할 수 있다[10, 13]. 여기서, 모든 $i=1,2,3, ..., M(\leq N)$에 대해 $d_i>d_{i+1}$이고, $A_i=\{t_j \mid g_j(x_j)=d_i, x_j \in X\}$이며, 특히 $d_1=\max\{g_j(x_j), \forall x_j \in X\}$이다. 하나 이상의 태스크들이 동일한 도함수 값을 가질 수 있으므로, A_i는 동일한 도함수 값(d_i)을 갖는 태스크들의 집합이다. 따라서 M은 N보다 같거나 작다. 두 집합 D와 A가 구해지면 최적 해 X는 다시 부분집합들로 세분화할 수 있다. 즉, $X=\{X_1, X_2, X_3, ..., X_M\}$이다. 여기서, $X_i=\{x_j=g_j^{-1}(d_i), \forall t_j \in A_i\}$이다.

위의 세 집합 X, D, A의 관계를 통해 만약 집합 D와 A를 구할 수 있다면, 역으로 도함수의 역함수를 이용해 최적의 해 X를 구할 수 있음을 암시한다. 본 논문에서는 이 전략을 이용하여 정적 스케줄링 문제(P)의 최적의 해를 구하고자 한다.

[정의]

태스크 집합의 모든 태스크들이 만기 순서로 정렬(즉, $(\tau_i < (\tau_{i+1})$되어 있다고 가정하자. 총가치를 최대화 하기 위해 우리는 임의의 스케줄링 시점(τ_0)에서 각 태스크에게 만기를 넘기지 않는 범위 내에서 임의의 서비스 시간(y_i)을 할당할 수 있고, 이를 이용해 서비스 시간이 할당된 모든 태스크들 중 가장 큰 가치 도함수 값, 즉 $max_{i=1, ..., N}\{g_i(y_i) \mid \sum_{i=1}^{k} y_i \leq \tau_k - \tau_0, \forall k = 1,2, ..., N\}$을 구할 수 있다. 모든 태스크들에게 매번 다른 임의의 서비스 시간을 할당할 수 있고, 그때마다 가장 큰 가치 도함수 값을 구할 수 있다. 이러한 최대 도함수 값들 중 최소값을 $\phi(\tau_0)=min\{max_{i=1, ..., N}\{g_i(y_i)\}\}$라 표현한다.

다음 정리는 정적 스케줄링 문제(P)의 임의의 최적 해를 통해 구한 집합 D의 첫번째 원소 d_1은 최대 도함수 값들 중 최소값($\phi(\tau_0)$)과 동일하며, 이를 통해 역으로 부분집합 A_1과 X_1을 구할 수 있음을 보여준다.

[정리]

$\phi(\tau_0)$가 시점 τ_0에서 최대 도함수 값들 중 최소값이라 하고, 태스크 집합 $S=\{t_i \mid g_i(0) \geq \phi(\tau_0)\}=\{t_1, t_2, t_3, ..., t_L\}$ $(L \leq N)$라 하자. S내의 모든 태스크들은 만기 순서로 정렬되어 있다고 할 때, 다음 조건들이 성립한다.

(1) $\phi(\tau_0)=d_1$이고, A_1은 S의 부분집합이다.

(2) $\tau_i < \tau_j$인 S에 포함된 임의의 두 태스크 t_i와 t_j에 대해, 만약 $t_i \in A_1$이면, $t_j \in A_1$이다.

(3) S에 포함된 태스크들에 대해 $\sum_{i}^{k} y_i = \tau_k - \tau_0$인 그러한 $k(1 \leq k \leq L)$가 반드시 존재한다. 여기서, $y_i = g_i^{-1}(\phi(\tau_0))$이다.

(4) (3)에서 구해진 k개의 태스크들의 집합 $B_1=\{t_1,t_2, t_3, ..., t_k\}$은 A_1과 동일하다.

(5) (3)에서 구해진 k개의 해들의 집합 $Y_1=\{y_1,y_2, y_3,..., y_k\}$은 X_1과 동일하다.

[증명]

(1) 정의에 의해 $\phi(\tau_0)$는 최대 도함수 값들 중 최소값이므로, 항상 $\phi(\tau_0) \leq d_1$이다. 만약 $\phi(\tau_0) < d_1$이라 가정하자. d_1은 최대 도함수 값들 중 최소값이 아니므로, A_1의 임의의 태스크 t_i에게 서비스 시간을 좀 더 할당해 줄 수 있는 여유시간(ε)이 반드시 존재하게 된다. $\phi(\tau_0) < d_1$이므로 이 여유시간은 실제 $g_k(x_k) < d_1$인 임의의 태스크 t_k에게 할당되었다. 따라서 t_i에게는 $(x_i + \varepsilon)$으로 서비스 시간을 늘려 주고 t_k에게는 $(x_k - \varepsilon)$으로 줄임으로써, 총가치를 더 증가시킬 수 있다. 이는 d_1이 더 작아질 수 있다는 것이며, 최적 해를 통해 구해진 d1의 정의에 위배된다. 따라서 항상 $\phi(\tau_0) = d_1$이다. 또한 $d_1 = \phi(\tau_0)$이므로 $A_1 = \{t_j \mid g_j(x_j) = \phi(\tau_0), x_j \in X\}$이고, 가치 도함수 $g_j()$는 점진적으로 감소하는 오목형 함수이므로, A_1에 속한 각 태스크 t_j에 대해 $g_j(0) \geq \phi(\tau_0)$이다. 따라서 A_1은 S의 부분집합이다.

(2) $\tau_i < \tau_j$인 S에 포함된 임의의 두 태스크 t_i와 t_j에 대해, $t_j \in A_1$이지만 $t_i \notin A_1$라고 가정하자. 이 가정은 t_j에 할당된 서비스 시간 x_j가 $g_j^{-1}\phi(\tau_0)$보다 크다는 것을 의미한다. $g_j()$는 점진적으로 감소하는 함수이므로 시간 간격 $m = x_j - g_j^{-1}(\phi(\tau_0))$이 존재한다. $\tau_i < \tau_j$이므로 t_i에게 할당된 m의 일부 서비스 시간$(0 < \varepsilon \leq m)$은 t_j에게도 할당될 수 있다. 이 경우 $g_j(x_j + \varepsilon) < g_j(x_j)$, 즉 t_j는 $g_j(x_j + \varepsilon)$을 가질 수 있으므로 더 이상 A_1에 포함될 수 없다. 이는 $t_j \in A_1$라는 가정에 위배되며 따라서 조건 (2)는 성립된다.

(3) 집합 S의 모든 $k(=1,2,...,L)$에 대해 $\sum_{i=1}^{k} y_i < \tau_k - \tau_0$ (명백히 $\sum_{i=1}^{k} y_i > \tau_k - \tau_0$는 될 수 없음)라고 가정하고, $m = min\{(\tau_k - \tau_0) - \sum_{i=1}^{k} y_i\}$이라 하자. 이 경우, $\sum_{i=1}^{k}(y_1 + m/L) \le \sum_{i=1}^{k} y_i + m = \sum_{i=1}^{k} y_i + min\{(\tau_k - \tau_0) - \sum_{i=1}^{k} y_i\} \le \sum_{i=1}^{k} y_i + (\tau_k - \tau_0) - \sum_{i=1}^{k} y_i$ $(\tau_k - \tau_0)$이므로, 모든 태스크들의 서비스 시간 y_i를 m/L 만큼씩 증가시킬 수 있다. 이는 $(\phi(\tau_0)(=d)$를 $min\{g_i(y_i) - g_i(y_i + m/L), \forall k = 1, 2, ..., L\}$만큼 더 감소시킬 수 있다는 것을 의미한다. 이는 $\phi(\tau_0)$가 가치 도함수 값들 중 최소값이라는 정의에 위배된다. 따라서 조건(3)은 성립된다.

(4) 조건 (1)에 의해 A_1은 S의 부분집합이고, 조건 (2)와 (3)을 통해 S의 태스크들은 만기 순서로 A_1에 포함된다는 것을 알 수 있다. A_1에는 포함되지만 B_1에는 포함되지 않은 임의의 태스크 t_j가 존재한다고 가정하자. 이 경우 조건 (3)에 의해 $\tau_j > \tau_k$이고, $\sum_{i=1}^{k} y_i + y_j < \tau_j - \tau_0$이다. 즉, t_j에게 $y_j < (\tau_j - \tau_k)$인 y_j만 할당했다는 의미이다. 따라서 총가치를 최대화하기 위해 $g_m(x_m) < d_1$인 임의의 태스크 $t_m(t_m \in S,\ t_m \notin B_1)$에게 할당되었던 서비스 시간을 $\varepsilon\ (<(\tau_j - \tau_k - yj))$만큼 줄이고, t_j에게 $(y_j + \varepsilon$을 할당할 수 있다. 이 경우 $g_j(y_j + \varepsilon) < d_1$되고, 따라서 t_j는 A_1에 포함될 수 없다. 역으로, B1에는 포함되지만 A_1에는 포함되지 않은 임의의 태스크 t_j가 존재한다고 하자. 만약 B_1에서 t_j가 빠진다면, 나머지 태스크들에게 t_j에게 할당된 시간 x_j를 적절히 분배 시켜줄 수 있다. 따라서 $\phi(\tau_0)$는 더 작아 질 수 있고, 이는 최대 도함수 값들 중 최소값이라는 $\phi(\tau_0)$의 정의에 위배된다. 이상의 두 가정이 모두 거짓이므로 조건 (3)은 성립된다.

(5) $Y1 = \{y_j = g_j^{-1}(\phi(\tau_0)), \forall t_j \in (B_1\}$이고 $X_1 = \{x_j = g_j^{-1}(d_1),\ \forall t_j \in (A_1\}$이다. 조건 (1)~(4)를 통해 $\phi(\tau_0) = d_1$이고 $B_1 = A_1$이므로, $Y_1 = X_1$이다.

집합 B_1에 속한 k개의 태스크들을 시스템 내에 존재하는 전체 태스크들의 집합에서 삭제하고, 나머지 태스크들을 대상으로 또 다시 최대 도함수 값들 중 최소값 $(\phi_2(\tau_0))$을 구하면, 이는 최적 해를 통해 구한 d_2와 동일하게 된다. 따라서 $\phi_2(\tau_0)$를 통해 또 다른 k_2개의 태스크 집합 B_2와 서비스 시간들의 집합 Y_2를 구할 수 있으며, 이는 각각 최적 해를 통해 구한 A_2 및 X_2와 동일하게 된다. 이러한 과정을 반복하여 최적 해 $X = \{x_1, x_2, ..., x_N\}$와 동일한 서비스 시간들의 집합 $Y = \{y_1, y_2, ..., y_N\}$를 구할 수 있다. 이 정리를 통해 우리는 최대 도함수 값들 중 최소값 $(\phi(\tau_0))$를 구함으로써, 스케줄링 문제(P)의 해를 구할 수 있음을 알 수 있다. 제안할 스케줄링 알고리즘은 이 전략을 기반으로 한다.

온-라인 스케줄링 알고리즘의 복잡도를 줄이는 한가지 방법은 동일한 최대 총가치를 생성하면서 새로운 태스크가 도착하기 전까지 실행될 적절한 수의 태스크들만을 스케줄링하는 것이다. 위의 정리에서 우리는 최대 도함수 값들 중 최소값을 통해 최적 해와 동일한 k개의 부분 태스크 집합과 그들에게 할당할 서비스 시간을 구할 수 있었다. 따라서 본 연구에서는 매 스케줄링 시점에서 이들 k개의 태스크들만을 스케줄링하는 전략을 채택한다. 이 전략은 만약 새로운 태스크가 도착하기 전에 이전에 스케줄된 k개의 태스크들을 모두 서비스했다면, 또 다른 k개의 태스크들을 선택하고 이들의 서비스 시간을 결정하기 위해 재스케줄링을 실시해야 하는 오버헤드를 가진다. 그러나 4절의 모의실험 결과를 통해 우리는 단지 k개의 태스크들만 스케줄링해도 재스케줄링의 횟수는 총 태스크 도착 횟수의 16%이하라는 사실을 확인 할 수 있다. 다음 절에서 위 정리를 기반으로 하는 IRIS 태스크들을 위한 온-라인 최적 알고리즘을 제안한다.

3.2 제안 알고리즘

그림 1의 제안 알고리즘은 매 스케줄링 시점에서 실행된다. 여기서 스케줄링 시점이란 새로운 태스크가 시스템에 도착한 시점 또는 새로운 태스크가 도착하기 전에 이전 스케줄링 시점에서 스케줄된 태스크들의 실행을 모두 완료한 시점을 의미한다. 알고리즘에서 첨자 i, k등은 해당 태스크 리스트 내에서 태스크들의 만기 순서를 의미한다.

단계 (1)에서 $\phi(\tau_0)$를 찾는다. $\phi(\tau_0)$를 찾기 위해 계산 복잡도가 낮은 이분법(bisection method)[13]을 적용하였다. $\phi(\tau_0)$를 결정하면 단계 (2)에서 현 스케줄링 시점(τ_0)에서의 순간 가치 증가율 $g_i(0)$가 $\phi(\tau_0)$보다 같거나 큰 태스크들의 리스트 S를 구할 수 있다. 단계 (3)에서는 리스트 S의 모든 태스크들에 대해 할당 가능한 서비스 시간(y_i)을 계산한다. 이때 y_i는 $\phi(\tau_0)$를 매개변수로 사용하는 태스크 t_i의 도함수 $g_i()$의 역함수를 이용하여 구한다. 단계 (4)에서는 리스트 S 내의 모든 태

스크들의 서비스 시간 y_i에 대해 조건 $\sum_{i=1}^{k} y_i = \tau_k - \tau_0$을 만족하는 $k(1 \le k \le N)$개의 태스크들을 선택한다. 또한 선택된 마지막 태스크의 만기 k를 다음 스케줄링 시점으로 설정한다. 만약 새로운 태스크가 다음 스케줄링 시점 k전에 도착하면 도착한 그 시점이 다음 스케줄링 시점이 된다. 제안 알고리즘은 단계 (4)에서 선택된 단지 k개의 태스크에 대해서만 서비스 시간(y_i)을 최종적 확정 짓는다. 서비스 시간이 확정된 k개의 스케줄된 태스크들은 하위 계층인 단계 (5)의 EDF 알고리즘에 의해 그들의 실행순서가 결정되고 실제 프로세서가 할당된다. 이때, 태스크들은 만기 전까지 단계 (4)에서 확정된 서비스 시간 이상을 수행할 수 없다.

let T be a list of tasks in the system and sorted in increasing order of their deadlines
let τ_0 be the current time
(1) get $\phi(\tau_0)$, where $\phi(\tau_0)$ is the min. max. reward derivatives for the tasks in T
(2) get task list $S=\{t_i \mid g_i((0) \ge \phi(\tau_0), t_i \in T\}$
(3) get service time $y_i = g_i^{-1}(\phi(\tau_0))$ for all $t_i \in S$
(4) select k's tasks such that $\sum_{i=1}^{k} y_i = \tau_k - \tau_0$, for $t_i \in S$
set the next scheduling point to τ_k
(5) schedule k's tasks by the EDF:
execute each scheduled task for y_i time units;

그림 1 제안 알고리즘

만약 다음 스케줄링 시점 (k까지 스케줄된 k개의 태스크들이 모두 실행을 완료할 동안 새로운 태스크가 도착하지 않을 경우, 제안 알고리즘은 재스케줄링을 실시하여 또 다른 k(이전 스케줄링 시점에서의 k와는 다름) 개의 태스크들에 대해 서비스 시간을 결정한다. 시스템 내의 모든 태스크들의 실행을 완료할 때까지 새로운 태스크가 전혀 도착하지 않는 극단적인 경우에, 제안 알고리즘은 반복하여 재 스케줄링을 실시하게 되며, 궁극적으로 시스템 내의 모든 태스크들에 대해 그들의 서비스 시간(y_i)을 결정하게 될 것이다.

태스크 리스트 T와 S에 존재하는 태스크들이 기존 연구에서처럼 그들의 만기 순서로 적절히 관리된다고 가정할 경우, 매 스케줄링 시점에서 제안 알고리즘의 계산 복잡도는 $O(N)$이다. 즉, 단계 (1)의 이분법에 의해 요구되는 계산 복잡도가 $O(N)$[13]이고, 단계 (2)와 (3)은 N번의 반복 실행(iteration)을 요구하고, 단계 (4)와 (5)는 기껏해야 k번의 반복 실행을 요구하기 때문이다. 그러나 만약 매 스케줄링 시점에서 단지 하나의 태스크만 스케줄되고 시스템 내의 모든 태스크가 수행될 때까지 새로운 태스크가 도착하지 않는 극단적인 경우에, 제안 알고리즘의 수행 총 회수는 시스템 내의 태스크들의 총 개수와 같다. 따라서 제안 알고리즘의 최악의 경우 계산 복잡도는 $O(N^2)$이다. 그러나 스케줄된 k개의 태스크들의 실행을 완료하기 전에 항상 새로운 태스크가 도착한다면, 이는 가장 이상적인 경우(best case)이며, 이 경우의 계산 복잡도는 $O(N)$이 된다. 그러나 다음 절에서 제시될 모의실험 결과를 통해 실행시의 평균 계산 복잡도는 $O(N^2)$보다는 $O(N)$에 더 가깝다는 것을 확인할 수 있다. 이는 제안 알고리즘이 최악의 경우 기존 알고리즘과 동일한 계산 복잡도를 가지지만, 평균적으로 더 낮은 계산 복잡도를 가진다는 것을 의미한다.

4. 성능 평가

본 절에서는 다양한 모의실험 결과를 보이고 이를 분석한다. 모의실험은 동적으로 도착하는 태스크들의 특성을 분석하고, 제안 알고리즘의 평균 계산 복잡도를 확인하는데 목적이 있다. 이를 위해 각 실험에서 세 가지 비율 U/N, U/Y, $(S-T)/T$를 측정하였다. 여기서 U는 이전 스케줄링 시점에서 스케줄된 태스크들 중 다음 스케줄링 시점까지 실제로 실행된 태스크들의 평균 개수, N은 매 스케줄링 시점에서 시스템 내에 존재하는 태스크들의 평균 개수, Y는 매 스케줄링 시점에서 제안 알고리즘에 의해 스케줄된 태스크들의 평균 개수, S는 제안 알고리즘이 수행된 총 스케줄링 횟수, T는 시스템에 도착한 태스크들의 총 개수이다.

비율 U/N은 이전 스케줄링 시점과 다음 스케줄링 시점사이에서 시스템 내에 대기중인 태스크들(N) 중 실제 서비스되는 태스크들(U)의 수가 얼마나 되는지를 나타낸다. 즉, 이 비율이 낮을수록 실제 서비스되는 태스크들의 수가 그 만큼 적다는 것을 의미한다. 제안 알고리즘의 기본 개념은 시스템 내의 모든 태스크들을 스케줄링할 것이 아니라, 일부만 스케줄링하자는 것이다. 이의 타당성을 비율 U/N을 통해 확인하고자 한다. 비율 U/Y은 매 스케줄링 시점에서 제안 알고리즘에 의해 스케줄된 태스크들(Y) 중 다음 스케줄링 시점까지 몇 개 정도의 태스크(U)가 실제로 실행되었는지를 보여준다. 이 비율을 통해 제안 알고리즘이 새로운 태스크가 도착하기 전까지 수행될 태스크 개수를 얼마나 효율적으로 결정했는가를 확인할 수 있다. 제안 알고리즘의 계산 복잡도를 측정하기 위한 단위로 비율 $(S-T)/T$를 사용한다. 제안 알고리즘은 새로운 태스크가 도착하거나 또는 이전에 스케줄된 태스크들이 새로운 태스크가 도착하기 전에 모두 실행되었을 경우에 재스케줄링 한다. 비율

$(S-T)/T$는 태스크 도착 때를 제외한 추가로 실행된 제안 알고리즘의 실행 횟수($S-T$)와 새로운 태스크가 도착했을 때만 수행된 실행 횟수(T)와의 비율이다. (이후 비율 $(S-T)/T$는 ST/T로 표기함). 매 스케줄링 시점에서 제안 알고리즘의 계산 복잡도가 $O(N)$이므로, ST/T가 0이면 제안 알고리즘의 평균 계산 복잡도는 $O(N)$이 되고, ST/T가 1이면 $O(N^2)$가 된다.

모의실험에 사용될 태스크 집합은 $M/M/\infty$(큐잉 모델을 기반으로 생성되었다[8,10,13]. 단일 프로세서상에서 실행되는 태스크 집합 내의 태스크들의 도착 율은 평균 λ를 가진 포아송(Poisson) 분포를 따르고, 태스크들이 시스템 내에 도착 직후부터 만기까지 머무를 수 있는 여유시간(laxities), $p_i(=\tau_i - r_i)$,은 평균 $1/\mu$를 가진 지수(exponential) 분포를 따른다고 가정한다. 따라서 시스템 내에 상존하는 평균 태스크의 개수는 $\rho=\lambda/\mu$이다. 각 태스크 t_i의 가치함수는 기존의 여러 연구[10, 13, 14]에서와 동일한 $f_i(x) = 1 - e^{-w_i x}$형태의 지수 함수이다. (여기서, 가치함수 가중치 w_i는 $(0, w_u)$의 범위 내의 균등(uniform) 분포를 따르는 임의의 실수이며, w_u는 사전에 정의된 w_i의 상위 경계 값이다.) 동일한 조건에서 w_u가 큰 태스크 집합일수록 더 많은 총가치를 생성한다. 서로 다른 λ, ρ, w_u 변수 값을 가지는 태스크 집합에 대해 모의실험을 실시하였으며, 각 태스크 집합은 25,000개의 태스크들로 구성된다.

그림 2, 3, 4는 태스크 도착율 λ가 각각 0.01, 1, 100이고, 지수 가치 함수를 가지는 태스크 집합들에 대해 실시한 모의실험 결과인 U/N 비율(히스토그램)과 U/Y 비율(커버 곡선)을 보인 것이다. 다양한 시스템의 부하에 대한 제안 알고리즘의 스케줄링 결과를 확인하기 위해 ρ를 광범위하게 변경하였고, 또한 생성된 총가치가 많을 때와 적을 때의 상황도 확인하기 위해 대표적인 세 개의 w_u(1, 8, 64)에 대해 실험했다. 그림에서 λ가 0.1과 10일 때의 실험결과는 보이지 않았지만, 이들은 각각 λ가 0.01과 1, 그리고 1과 100일 때의 중간 값을 보였다.

비율 U/N은 전체적으로 w_u와 λ에 상관없이 시스템에 존재하는 태스크 개수 ρ가 증가함에 따라 감소한다. ρ가 매우 작을 경우($\rho = 1.25$), 시스템에 존재하는 태스크들의 약 90%가 다음 스케줄링 전까지 실제로 실행된다. 즉, 약 10% 태스크들만 프로세서로부터 서비스를 받지 못한다. ρ가 증가함에 따라 U/N은 빠른 속도로 감

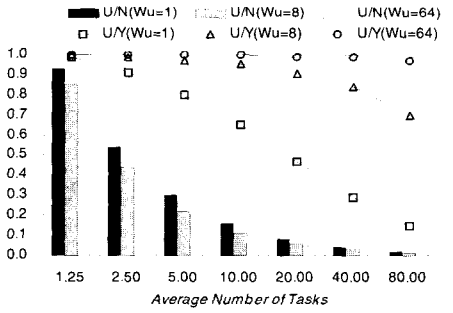

그림 2 비율U/N과 비율U/Y (λ = 0.01)

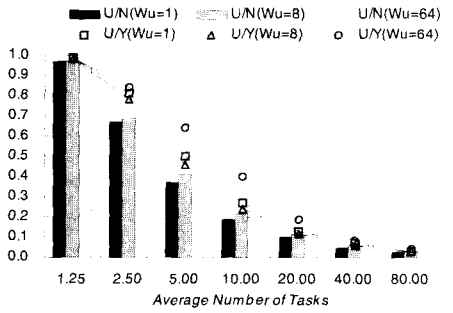

그림 3 비율U/N과 비율U/Y (λ = 1)

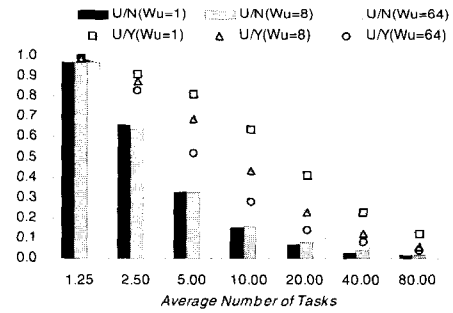

그림 4 비율U/N과 비율U/Y (λ = 100)

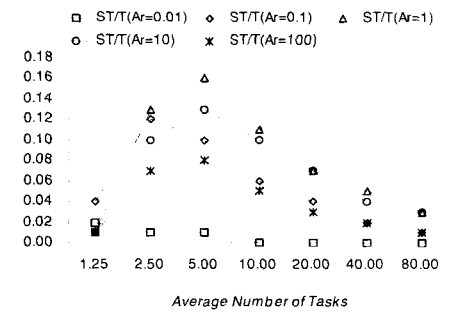

그림 5 비율 ST/T (w_u = 8)

소하며, ρ가 10일 경우 약 20%의 태스크들만 실제로 서비스를 받고, ρ가 10보다 더 커질 경우 매 스케줄링 시점사이에 실제로 서비스되는 태스크들의 수는 10% 미만이다. 이는 시스템 내의 태스크 수가 증가할지라도 스케줄링 시점사이에서 실제로 서비스되는 태스크의 수는 거의 증가하지 않는다는 것을 의미한다. 이러한 실험 결과는 매 스케줄링 시점에서 시스템에 존재하는 모든 태스크들을 스케줄링할 필요가 없다는 것을 시사한다. 이는 곧 기존 알고리즘과 동일한 총가치를 생성하면서도 시스템 내의 모든 태스크들을 스케줄링할 것이 아니라, 일부 태스크들만 스케줄링하자는 제안 알고리즘의 기본 전략이 타당성이 있음을 보여 준다.

모의실험 결과를 통해 확인할 수 있는 또 다른 중요한 사항은 제안 알고리즘이 새로운 태스크가 도착하기 전까지 수행될 태스크 개수를 얼마나 효율적으로 결정했는가 하는 것이다. 이는 비율인 U/Y를 통해 확인할 수 있다. 그림 2, 3, 4에서 ρ가 매우 작은 경우, U/Y은 λ에 상관없이 거의 1에 가깝다. 이는 시스템 내의 태스크 개수가 매우 적을 경우 λ나 w_u에 상관없이 스케줄된 태스크들이 다음 스케줄링 시점 전에 거의 대부분 실행을 완료했다는 것을 의미한다. 그러나 ρ가 증가함에 따라 스케줄된 태스크들의 수는 실제 서비스되는 태스크들의 수보다 훨씬 더 커진다는 것을 알 수 있다. (즉, U/Y는 점점 감소함) 이는 곧 스케줄되는 태스크들의 수는 시스템 내의 태스크 수가 증가함에 따라 함께 증가하지만, 실제 서비스되는 태스크들의 수는 스케줄되는 태스크들의 수만큼 증가하지 않거나 또는 특정 ρ에서부터 더 이상 증가하지 않는다는 것을 의미한다. 이 실험 결과를 통해 단지 k개의 태스크들만 스케줄링하는 제안 알고리즘은 시스템 내의 태스크 개수가 적을 경우 비교적 적절한 수의 태스크들을 스케줄링하지만, 시스템 내의 태스크 개수가 많을 경우 여전히 실제 서비스될 태스크들보다 더 많은 태스크들을 스케줄링 한다는 것을 알 수 있다.

제안 알고리즘은 이전에 스케줄된 태스크들이 새로운 태스크가 도착하기 전에 모두 실행되었을 경우에 재스케줄링(이를 추가 재스케줄링이라 함)을 실시한다. 만약 매 스케줄링 시점에서 스케줄된 태스크들의 개수가 적절하지 않을 경우, 제안 알고리즘은 자주 재스케줄링 실시하게 되며, 이는 제안 알고리즘의 스케줄링 복잡도를 증가시키게 되는 요인이 된다. 그림 5는 추가 재스케줄링 횟수와 새로운 태스크가 도착했을 때만 수행된 실행 횟수와의 비율인 ST/T를 보인 것이다. 그림에서 w_u는 8로 고정되었으며, Ar은 태스크 도착율 ρ를 의미한다. 그림에서 ρ가 5보다 클 경우, ST/T는 ρ가 증가함에 따라 점점 감소한다. 이는 스케줄되는 태스크들의 수는 ρ가 증가함에 따라 함께 증가하지만, 실제로 서비스되는 태스크들의 수는 그 만큼 증가하지 않기 때문이다. 이는 곧 시스템 내의 태스크 수가 많은 경우 제안 알고리즘의 추가 재스케줄링의 회수는 매우 작다는 것을 의미한다. 그림에서 ρ가 5일 때보다 1.25와 2.50일 때 더 작은 ST/T 비율을 보이는 것은 시스템 내의 태스크 개수가 적을 경우 스케줄된 태스크들이 모두 실행된 후 더 이상 스케줄할 태스크가 없는 경우가 종종 발생하기 때문이다. 앞서 그림 2, 3, 4에서 ρ가 매우 작은 경우, 비율 U/Y은 λ에 상관없이 거의 1에 가깝다는 것을 확인했다. 이는 실제 실행될 태스크 개수를 정확하게 판단하여 스케줄링 했다고 할 수 있지만, 재스케줄링 횟수도 그 만큼 증가하게 되는 요인이 될 수 있다. 그러나 그림 5의 ST/T 비율(ρ가 1.25와 2.50일 때)을 통해 재스케줄링 횟수는 증가하지 않음을 알 수 있다.

그림 5에서 ST/T 비율은 λ가 1일 때 가장 높다. ST/T에 영향을 미치는 정확한 요소는 스케줄되는 태스크의 개수보다는 스케줄된 태스크들에게 할당된 서비스 시간의 합($\sum_{i=1}^{k} y_i$)과 태스크간의 평균 도착간격($1/\lambda$)의 비율($\lambda \sum_{i=1}^{k} y_i$)에 의해 결정된다. 즉, 이 비율이 적을수록 스케줄된 태스크들에게 할당된 서비스 시간의 합이 태스크 도착 간격에 비해 상대적으로 적으므로 그 만큼 새로운 태스크가 도착하기 전에 재스케줄링할 확률이 높아지게 된다. 따라서 λ가 1이고 w_u가 8일 때 상대적으로 가장 작은 비율을 가지는 것으로 분석된다. 그림에는 보이지 않았지만, w_u가 1 또는 64일 때도 유사한 ST/T 비율을 보인다. 다만, w_u에 따라 가장 높은 ST/T 비율을 보이는 λ만 다를 뿐, 가장 높은 ST/T 비율 값은 거의 동일했다.

그림 5와 다양한 w_u에 대해 실시한 모의실험 결과, 가장 높은 ST/T 비율은 어떤 경우이든 0.16을 초과하지 않는다. 이는 제안 알고리즘의 추가 재스케줄링 횟수는 평균적으로 시스템에 도착하는 태스크들의 수의 16% 이하라는 것을 의미한다. 제안 알고리즘은 매 스케줄링 시점에서 $O(N)$ 계산 복잡도를 가지고 스케줄 가능한 태스크들의 서비스 시간을 결정한다. 그러나 추가 재스케줄링이 발생할 경우 계산 빈도는 증가하게 된다. 실험 결과를 통해 제안 알고리즘은 시스템의 부하(ρ), 태스크 도착율(λ), 가치함수 가중치 상위 경계(w_u)에 상관없이 평균적으로 1.16배 계산 빈도가 증가됨을 알 수

있다. 이는 제안 알고리즘의 실행시의 평균 복잡도는 $O(N)$에 더 가깝다는 것을 의미한다.

5. 결 론

본 논문에서는 동적으로 도착하는 실시간 태스크들의 총 가치를 최대화하기 위한 온-라인 스케줄링 알고리즘을 제안하였다. 제안 알고리즘은 동적 태스크들의 총가치를 최대화하는 문제를 동일한 총가치를 생성하면서 가치함수들의 최대 도함수 값들 중 최소 값을 찾는 문제로 변경하여 해결할 수 있음을 보였다. 또한 제안 알고리즘은 시스템 내의 모든 태스크들을 스케줄링하지 않고, 다음 스케줄링 시점까지 실제 서비스될 태스크들의 개수를 결정하여 그 개수만큼의 태스크들을 스케줄링하는 방안을 제시하였다. 이러한 두 가지 기본 전략은 제안 알고리즘의 계산 복잡도를 줄이는 효과를 가져온다. 이론적으로 제안 알고리즘의 계산 복잡도는 $O(N)$(최선의 경우)과 $O(N_2)$(최악의 경우)이다. 그러나 모의실험 결과 평균적으로 $O(N_2)$보다는 훨씬 적고 $O(N)$보다는 조금 높은 것으로 확인되었다. 기존 온-라인 최적 알고리즘과 비교해, 제안 알고리즘은 동일한 최적의 총가치를 생성하면서 상대적으로 낮은 평균 계산 복잡도를 가지며, 특히 많은 태스크가 상존하는 시스템에 적합하다고 할 수 있다.

본 연구팀은 현재 Wavelet 방식으로 압축된 영상을 복원하여 화면에 보여주는 응용을 위한 스케줄링 알고리즘으로 본 연구에서 제안된 알고리즘을 적용하여 실험하고 있다. 이 실험을 통해 가치함수를 구하는 구체적인 방법과 제안 알고리즘에 의해 복원된 프레임 화질의 실질적인 성능향상 효과 등에 대해 기존 알고리즘들과 비교 연구하고 있다. 또한 본 연구팀은 시스템 내의 모든 태스크들을 대상으로 스케줄링하는 것이 아니라, 가치 증가율이 높은 몇 개의 태스크를 선택하여 이들을 대상으로 스케줄링하는 방법 등을 연구할 계획이다.

참 고 문 헌

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

[2] E. Chang and A. Zakhor, "Scalable Video Coding Using 3-D Subband Velocity Coding and Multi-Rate Quantization," *Proc. IEEE Int'l Conf. Acoustic, Speech, and Signal Processing*, Minneapolis, July 1993.

[3] G. Jung, K. Yim, J. Jung, J. Shin, K. Choi, D. Kim, and J. Shim, "An Imprecise DCT Computation Model for Real-Time Applications," *Multimedia Technology and Applications*, edited by V. Chow, pp.153-161, Springer, Dec. 1996.

[4] J. Grass and S. Zilberstein, "A Value-Driven System for Autonomous Information Gathering," *J. Intelligent Information Systems*, Vol. 14, pp. 5-27, March 2000.

[5] S.V. Vrbsky and J.W.S. Liu, "APPROXIMATE - A Query Processor that Produces Monotonically Improving Approximate Answers," *IEEE Trans. Knowledge and Data Eng.*, Vol. 5, No.6, pp. 1056-1068, Dec. 1993.

[6] B. Hayes-Roth, "Architectural Foundations for Real-Time Performance in Intelligent Agents," *J. Real-Time Systems*, Vol. 2, No. 1, pp. 99-125, 1990.

[7] R.E. Korf, "Real-Time Heuristic Search," *Artificial Intelligence*, Vol. 42, No. 2, pp. 189-212, 1990.

[8] E.J. Horvitz, "Reasoning under Varying and Uncertain Resource Constraints," Proc. *7th Nat'l Conf. Artificial Intelligence (AAAI-88)*, pp. 111-116, St. Paul, Minn., Aug. 1988.

[9] M. Boddy and T. Dean, "Deliberation Scheduling for Problem Solving in Time-Constrained Environments," *Artificial Intelligence*, Vol. 67, No. 2, pp. 245-285, June 1994.

[10] 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 Trans. Computers*, Vol. 45, No. 7, 802-813, July 1996.

[11] J.W.S. Liu, K.J. Lin, W.K. Shih, A.C.S. Yu, J.Y. Chung, and W. Zhao, "Algorithms for Scheduling Imprecise Computations," *IEEE Computer*, Vol. 24, No. 5, pp. 58-68, May 1991.

[12] J.K. Dey, J.F. Kurose, D. Towsley, C.M. Krishna, and M. Girkar, "Efficient On-Line Processor Scheduling for a Class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks," *Proc. ACM Sigmetrics Conf. Measurement and Modeling of Computer Systems*, pp. 217-228, Santa Clara, Calif., May 1993.

[13] K. Choi and G. Jung, "Comment on On-line Scheduling Policies for a Class of IRIS Real-Time Tasks," *IEEE Trans. Computers*, Vol. 50, No. 5, pp. 526-528, May 2001.

[14] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Optimal Reward-based Scheduling for Periodic Real-Time Tasks," *IEEE Trans. Computers*, Vol. 50, No. 2, pp. 111-130, Feb. 2001.

[15] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM*, Vol. 20, No. 1, pp. 46-61, Jan. 1973.

심 재 홍

1987년 서울대학교 전산과학과 학사. 1989년 아주대학교 컴퓨터공학과 석사. 2001년 아주대학교 컴퓨터공학과 박사. 1989년~1994년 서울시스템(주) 공학연구소. 2001년~2001년 9월 아주대학교 정보통신전문대학원 BK21 전임연구원. 2001년 10월~현재 조선대학교 인터넷소프트웨어공학부 전임강사. 관심분야는 운영 체제, 분산시스템, 실시간 및 멀티미디어시스템

최 경 희

1976년 서울대학교 수학교육과 학사. 1979년 프랑스 그랑데꼴 Enseeiht대학 석사. 1982년 프랑스 Paul Sabatier대학 정보공학부 박사. 1982년~현재 아주대학교 정보통신전문대학원 교수. 관심분야는 운영 체제, 분산시스템, 실시간 및 멀티미디어시스템 등

정 기 현

1984년 서강대학교 전자공학과 학사. 1988년 미국 Ilinois주립대 EECS 석사. 1990년 미국 Purdue대학 전기전자공학부 박사. 1991~1992년 현대반도체 연구소. 1993년~현재 아주대학교 전자공학부 교수. 관심분야는 컴퓨터구조, VLSI 설계, 멀티미디어 및 실시간 시스템 등