태스크 선택정책에 기반을 둔 IRIS 태스크 스케줄링 알고리즘

심 재 홍[†] · 최 경 희[††] · 정 기 현[†††]

요 약

본 논문에서는 IRIS(Increasing Reward with Increasing Service) 태스크들을 위한 기존 온-라인 최적 알고리즘에 근접한 총가치(total reward)를 생성하면서 보다 낮은 스케줄링 복잡도를 가진 휴리스틱(heuristic) 온-라인 스케줄링 알고리즘을 제안한다. 기존 알고리즘들은 총가치를 최대화하기 위해 시스템 내의 모든 태스크들을 스케줄링 대상으로 고려한다. 따라서 이들 알고리즘들은 많은 태스크들을 가진 실제 시스템에 적용하기에는 매우 높은 시간 복잡도를 가진다. 제안 알고리즘은 시스템 내의 모든 태스크들을 대상으로 스케줄링하는 것이 아니라, 상수 W 개의 태스크들을 선택하여 이들을 대상으로 스케줄링 한다. 제안 알고리즘은 W 개의 태스크를 어떤 기준에 의해 선택할 것인가를 규정하는 태스크 선택정책에 기반을 두고 있으며, 간단하면서도 직관적인 두 가지 선택정책과 이 둘을 통합한 보다 일반화된 선택정책을 제안한다. 스케줄링 대상을 축소함으로써 제안 알고리즘의 복잡도를 $O(Wn)$로 줄일 수 있었다. 다양한 성능실험 결과 알고리즘 평균 계산 빈도는 $O(W)$에 더 가깝다는 것을 확인할 수 있었다.

IRIS Task Scheduling Algorithm Based on Task Selection Policies

Jae-Hong Shim[†] · Kyung-Hee Choi[††] · Gi-Hyun Jung[†††]

ABSTRACT

We propose a heuristic on-line scheduling algorithm for the IRIS (Increasing Reward with Increasing Service) tasks, which has low computation complexity and produces total reward approximated to that of previous on-line optimal algorithms. The previous on-line optimal algorithms for IRIS tasks perform scheduling on all tasks in a system to maximize total reward. Therefore, the complexities of these algorithms are too high to apply them to practical systems handling many tasks. The proposed algorithm doesn't perform scheduling on *all* tasks in a system, but on (*constant*) W's tasks selected by a predefined task selection policy. The proposed algorithm is based on task selection policies that define how to select tasks to be scheduled. We suggest two simple and intuitive selection policies and a generalized selection policy that integrates previous two selection policies. By narrowing down scheduling scope to only W's selected tasks, the computation complexity of proposed algorithm can be reduced to $O(Wn)$. However, simulation results for various cases show that it is closed to $O(W)$ on the average.

키워드 : 실시간 시스템(Real-Time System), 온-라인 스케줄링(On-Line Scheduling), 총가치의 최대화(Maximizing Total Reward), 태스크 선택 정책(Task Selection Policies)

1. 서 론

실시간 시스템을 구성하는 태스크들은 항상 그들의 만기(deadline) 이전에 실행이 완료되어야 한다. 특히 경성(hard) 실시간 시스템에서는 태스크가 만기 내에 실행을 종료하지 못할 경우, 이를 시스템 실패로 간주하며, 그 동안 실행된 태스크의 실행 결과는 무시된다. 이러한 시스템에서는 만기까지 완료하지 못한 태스크의 중간 실행 결과는 아무런 의미가 없는 것으로 인식한다[1].

※ 본 논문은 2002년도 조선대학교 학술연구비의 지원을 받아 연구되었음.
† 정 회 원 : 조선대학교 인터넷소프트웨어공학부 교수
†† 정 회 원 : 아주대학교 정보통신전문대학원 교수
††† 정 회 원 : 아주대학교 전자공학부 교수
논문접수 : 2002년 8월 30일, 심사완료 : 2003년 7월 21일

IRIS(Increasing Reward with Increasing Service) 태스크 시스템[2, 3]은 모든 태스크들을 그들의 만기 전까지 완료되도록 스케줄링할 수 없을 경우, 각 태스크들에게 할당되는 프로세서 시간을 조금씩 줄여 스케줄링하며, 각 태스크의 중간 실행 결과 역시 어떤 의미(가치)를 가지는 실시간 시스템이다. IRIS 태스크 시스템은 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. 이러한 시스템의 예로 비디오 이미지를 압축하거나 해제하는 응용, 멀티미디어 응용, 정보 수집 및 추출, 데이터베이스 질의어 처리를 위한 시스

템 등을 들 수 있다[4-7].

IRIS 태스크는 그것의 실행 결과로 하나의 가치(reward)를 가지며, 이는 만기 전까지 각 태스크에게 주어진 서비스 시간(실제 실행시간) 양의 함수 값으로 표현된다. 이 함수를 가치함수(reward function)라 하며, 비감소 함수(non-decreasing function) 특성을 가진다. 즉, 각 태스크에게 주어지는 서비스 시간이 증가 할수록 해당 태스크가 얻는 가치 또한 증가한다. IRIS 태스크 시스템은 기존의 부정확 계산 모델(imprecise computation model)[8]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 목적에 따라 모든 태스크들의 만기를 넘기지 않는 범위 내에서 각 태스크에게 할당할 서비스 시간을 결정해야 한다.

IRIS 태스크 시스템의 가장 근본적인 스케줄링 목적 중의 하나는 태스크들의 총가치(모든 태스크들의 가치들의 합)를 최대화시키는 것이다. 또한 IRIS 태스크들을 위한 온-라인 스케줄링과 관련한 대부분의 기존 연구들은 태스크들의 총가치를 최대화하면서 알고리즘의 시간 복잡도를 줄이는데 총력을 기울였다. Dey와 그의 동료들은 시간 복잡도 $O(n^2)$을 가지는 온-라인 최적 알고리즘을 제안했다[3]. 여기서, 태스크들은 점진적으로 증가하는 볼록형(strictly increasing concave) 가치함수를 가지며, n은 시스템 내의 태스크의 개수이다. 이 알고리즘은 새로운 태스크가 임의의 시간에 동적으로 도착할 때마다 기존의 스케줄링 결과(각 태스크에게 할당된 서비스 시간)를 무시하고 재 스케줄링을 실시하여 각 태스크에게 할당할 서비스 시간을 다시 결정한다. 이 알고리즘과 동일한 시간 복잡도와 성능을 가진 또 다른 알고리즘이 Choi에 의해 제시되기도 했다[9].

또한 Dey가 제시한 알고리즘과 동일한 최악의 경우 시간 복잡도를 가지지만 평균적으로 $O(n)$에 가까운 시간 복잡도를 가지는 보다 향상된 온-라인 최적 알고리즘이 Jung에 의해 제안되었다[10]. 이 알고리즘은 새로운 태스크가 도착하기 전까지 이전 스케줄링 시점에서 스케줄된 태스크들 중 소수만이 실제 실행되고 나머지는 새로이 도착한 태스크와 함께 다시 스케줄링 된다는 사실에 주목하고, 매 스케줄링 시 Dey의 알고리즘과 같이 모든 태스크들의 서비스 시간을 결정하는 것이 아니라, 일부 태스크들에 대해서만 서비스 시간을 결정한다(이후, 스케줄러에 의해 서비스 시간이 결정된 태스크를 스케줄된 태스크라 한다.).

그러나 IRIS 태스크들의 총가치를 최대화시키기 위한 기존 온-라인 최적 스케줄링 알고리즘들은 많은 태스크들을 가진 실제 시스템에 적용하기에는 여전히 높은 시간 복잡도를 가진다. 이러한 높은 시간 복잡도는 최적의 스케줄링 결과를 생성하기 위해 시스템 내의 모든 태스크들을 스케줄링 대상으로 고려하기 때문이다. 즉, Dey와 Choi에 의해 제안된 알고리즘들은 시스템 내의 모든 태스크들에 대해 서비스 시간을 결정하고 Jung의 알고리즘은 비록 일부 태스크들에 대해서만 서비스 시간을 결정하지만, 총가치를 최대화시키기 위해 이들 알고리즘들은 시스템 내의 모든 태스크들의 만기와 순간 가치 증가율 등을 종합적으로 고려하기 때문이다. 이는 곧 스케줄링 복잡도가 여전히 시스템 내의 태스크의 개수에 의존한다는 것을 의미한다.

본 논문에서는 스케줄링 복잡도를 줄이면서 여전히 수용 가능한 총가치를 생성하는 휴리스틱 온-라인 스케줄링 알고리즘을 제안한다. 제안 알고리즘은 시스템 내의 태스크 개수와 무관하게 사전에 정의된 태스크 선택정책에 의해 단지 W(상수)개의 중요한 태스크들만을 선택하고, 선택된 W개의 태스크들을 대상으로 총가치를 최대화시키는 스케줄링 기법을 적용한다. 제안 알고리즘은 기존 알고리즘처럼 시스템 내의 모든 태스크들을 스케줄링 대상으로 삼는 것이 아니라, 선택된 W개의 태스크들로 스케줄링 범위를 축소시킨다. 이러한 전략은 기존 온-라인 최적 알고리즘[3, 10]보다 조금 감소되었지만 여전히 수용 가능한 총가치를 생성하면서 상대적으로 낮은 시간 복잡도를 가질 수 있게 한다. 제안 알고리즘은 W개의 태스크를 어떤 기준에 의해 선택할 것인가를 규정하는 태스크 선택정책에 기반을 두고 있으므로, 본 연구에서는 또한 간단하면서도 직관적인 두 가지 선택정책과 이 둘을 통합한 보다 일반화된 선택정책을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서 IRIS 태스크 모델과 그들이 수행되는 시스템 환경에 대해 기술한다. 3장에서는 태스크 선택정책을 기반으로 하는 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

단일 프로세서 상에서 실행되는 선점 가능한 태스크들의 집합 T가 있다고 가정하자. 태스크 집합 T를 구성하는 원소인 각 태스크 t_i, $i=1,2,\cdots,n$는 도착시간 r_i, 만기(deadline) τ_i, 가치함수(reward function) $f_i(x)$를 가진다. 태스크는 임의의 시간에 시스템에 도착할 수 있고 만기가 지난 후에는 시스템에서 삭제된다.

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 가치(reward)를 가진다. 가치함수는 비감소 볼록한(non-decreasing concave) 모양을 가진다. 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 비감소 볼록형이므로 비증가 오목형(non-increasing 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 현 스케줄링 시점에서의 순간 가치 증가율이라 한다.

스케줄러는 태스크들의 만기를 넘기지 않는 범위 내에서 최적의 총가치를 생성할 수 있도록 태스크들에게 할당할 실행시간을 결정하고 또한 이들의 실행 순서를 적절히 결정해야 한다. 그러나 태스크들이 언제 도착하고 그들의 만기가 얼마인지는 사전에 알려져 있지 않으므로 최적의 총가치를 생성하도록 스케줄링 한다는 것은 매우 어렵다(NP-hard). 따라서 스케줄러는 더 이상의 태스크들이 도착하지 않는다는 가정 하에서 현재 시스템에 도착해 있는 태스크들을 상대로 총가치가 최대화되도록 스케줄링하는 온-라인 스케줄링 알고리즘을 채택한다. 온-라인 스케줄링 알고리즘은 스케줄된 태스크들이 실행되고 있는 도중에 새로운 태스크들이 도착할 경우, 총가치를 최대화하기 위해 다시 스케줄링을 실시한다.

본 논문에서는 기존의 연구들에서 널리 채택된 상하 계층으로 분리된 온-라인 스케줄링 알고리즘을 기본 모델로 삼는다[3, 9, 10]. 이들 연구에서 상위 계층의 부-알고리즘(sub-algorithm)은 태스크들의 만기를 넘기지 않는 범위 내에서 최적의 총가치를 생성할 수 있도록 태스크들에게 할당할 서비스 시간을 결정한다. 반면, 하위 계층의 부-알고리즘은 태스크들의 실행 순서를 결정하며, 각 태스크에게 상위 계층에서 결정된 서비스 시간만큼만 프로세서를 할당한다. 최적의 총가치를 생성하기 위해 하위 계층의 부-알고리즘으로 EDF(Earliest Deadline First)[11] 스케줄링 알고리즘을 채택한 경우가 가장 우수한 것으로 보고된바 있다[3]. 본 논문에서 제안될 온-라인 스케줄링 알고리즘 역시 하위 계층 부-알고리즘으로 EDF를 채택하고 있으며, 본 연구에서는 상위 계층의 부-알고리즘 개발에 역점을 둔다.

3. 태스크 선택정책에 기반을 둔 스케줄링 알고리즘

본 절에서는 IRIS 태스크 모델을 따르는 태스크들의 스케줄링 범위 축소를 위해 W개의 태스크들을 어떤 기준에 의해 선택할 것인가를 규정하는 태스크 선택정책을 먼저 제시하고, 이를 기반으로 IRIS 태스크 스케줄링 알고리즘을 제안한다.

3.1 태스크 선택정책(task selection policy)

IRIS 시스템에서 태스크들은 새로운 태스크가 도착할 때마다 총가치를 최대화시키기 위해 재스케줄링 되어야 한다. 따라서 새로운 태스크의 도착은 이전 스케줄링 시점에서 생성된 스케줄링 결과를 쓸모없게 만든다. 즉, 이전 스케줄링 시 스케줄 되었지만 새로운 태스크의 도착 전까지 실행되지 못한 태스크들은 결과적으로 불필요하게 스케줄링 되었으며, 이는 곧 스케줄링 부하를 상대적으로 증가시켰음을 의미한다. 가장 이상적인 것은 총가치를 최대화하면서 새로운 태스크가 도착하기 전까지 실행될 태스크들만을 스케줄링하는 것이다. 그러나 이는 동적으로 도착하는 태스크들의 정확한 도착시간이 주어지지 않는 이상 현실적으로 불가능하다.

따라서 본 논문에서는 새로운 태스크가 도착하기 전까지 실행될 태스크들을 인위적으로 예측하여 이들을 대상으로 스케줄링하는 방안을 제시하고자 한다. 이는 이전 스케줄링 시점에 스케줄된 태스크들 중 소수의 태스크들만 새로운 태스크가 도착하기 전까지 실제 실행된다는 기존 연구 결과[10]를 바탕으로 하고 있다. 제안 알고리즘은 하나의 가상적인 스케줄링 창(*scheduling window*)을 가지며, 이는 단지 W개의 태스크들만을 가질 수 있다. 이때 상수 W를 창의 크기(*window size*)라 하며, 사전에 정의된 임의의 양의 정수 값을 가진다. 스케줄링 창에 포함된 태스크들은 아래에서 기술될 **태스크 선택정책**(*task selection policy*)에 의해 선택된다. 제안 알고리즘은 시스템 내의 모든 태스크들을 대상으로 스케줄링 하는 것이 아니라, 단지 선택된 W개의 태스크들만을 대상으로 스케줄링 한다.

본 논문의 주된 아이디어는 시스템 내의 모든 태스크들을 대상으로 스케줄링하지 말고, 상수 W개의 태스크를 선택하여 이들을 대상으로 스케줄링하여 W보다 같거나 작은 수의 태스크들에게 실제 서비스 시간을 할당하는 것이다. 따라서 제안 알고리즘의 성능(총가치와 스케줄링 복잡도)은 시스템 내의 n개의 태스크들 중 몇 개의 태스크를 선택할 것이가(창의 크기 W 결정)와 W개의 태스크를 어떤 기준에 의해 선별할 것인가를 규정하는 선택정책에 의해 영향을 받는다. 스케줄링 창 크기 W가 작으면 작을수록 스케줄링 대상이 줄어들기 때문에 알고리즘의 속도는 증가될 수 있지만, 총 가치는 더 줄어 들 수 있기 때문이다.

본 연구에서는 태스크 선택정책으로 먼저 가장 간단하면서도 직관적인 다음 두 가지 정책을 제안한다. 첫째는 스케줄링 시점의 순간 가치 증가율이 가장 높은 태스크 순으로 W개의 태스크를 선택하는 방안이다. 이 정책은 *HRR*(highest reward rate) 선택정책이라 불리며, 총가치를 최대화하려는 스케줄링 알고리즘의 기본 목표와 일치하는 가장 직관적인 선택 방법이다. 이는 순간 가치 증가율을 기준으로 태스크를 선택하기 때문에 다른 태스크들보다 월등히 높은 가치 증가율을 가진 몇 개 태스크들이 존재할 경우 서비스를 받지 못하는 태스크가 증가할 수 있다.

둘째는 만기가 가장 빠른 태스크 순으로 W개의 태스크를 선택하는 방안이다. 이는 ED(earliest deadline) 선택정책이라 하며, 가치 증가율을 고려하지 않기 때문에 모든 태스크들에게 균등한 서비스 기회를 제공할 수 있다.

위의 두 선택정책은 단지 태스크의 순간 가치 증가율 또는 태스크의 만기만을 고려하여 태스크를 선택하지만, 이들 두 요소를 동시에 고려한 **통합** 선택정책을 정의할 수 있다. 제안하는 세 번째 태스크 선택정책인 통합 선택정책은 각 태스크에 대해 다음의 선택 수식을 계산하여, 가장 작은 선

택 값(c_i)을 가지는 W개의 태스크를 선택한다.

$$c_i = \alpha(\tau_i - \tau_0)/(\tau_n - \tau_0) + (1-\alpha)(1 - g_i(0)/g_{\max}(0))$$

여기서, $(\tau_i - \tau_0)/(\tau_n - \tau_0)$는 현 스케줄링 시점(τ_0)을 기준으로 상대만기(relative deadline)가 가장 긴 태스크(τ_n)와 각 태스크간의 상대만기 비율이며, 만기가 빠른 태스크일수록 작은 비율 값을 가진다. $g_i(0)/g_{\max}(0)$는 가장 높은 순간 가치 증가율을 가진 태스크와 각 태스크간의 순간 가치 증가율의 비율이며, 순간 가치 증가율이 높은 태스크일수록 작은 $(1-g_i(0)/g_{\max}(0))$ 값을 가진다. α는 선택 가중치(selection weight)라 하며, 만기와 순간 가치 증가율의 반영 정도를 결정하는 사전에 정의된 상수이다.

통합 선택정책은 만기가 가장 빠른 태스크들과 순간 가치 증가율이 가장 큰 태스크들 중 선택 가중치에 따라 이들을 적절히 섞어 선택할 수 있도록 한다. α가 0인 통합 선택정책은 순간 가치 증가율이 높은 순서로 태스크를 선택하는 HRR과 동일한 정책이 되고, α가 1이면 만기가 가장 빠른 순서로 태스크를 선택하는 ED와 동일한 선택정책이 된다. 따라서 α가 0에 가까울수록 만기가 빠른 태스크들이 많이 선택되고, α가 1에 가까울수록 순간 가치 증가율이 높은 태스크들이 우선적으로 선택된다.

3.2 휴리스틱 온-라인 스케줄링 알고리즘

본 절에서는 앞서 제시된 태스크 선택정책에 의해 선택된 W개의 태스크들을 대상으로 스케줄링 복잡도를 줄이면서 여전히 수용 가능한 총가치를 생성하는 휴리스틱 온-라인 스케줄링 알고리즘을 제안한다. 제안 알고리즘은 다음과 같다.

T를 만기 순으로 정렬된 시스템 내의 모든 태스크들의 리스트라고 가정하고, T의 모든 태스크들 중에서 태스크 선택정책에 의해 선택된 W개의 태스크들의 리스트를 S(스케줄링 창)라 하자. 또한 τ_0는 현 스케줄링 시점을 의미하며, 첨자 i, k, m 등은 해당 태스크 리스트 내에서의 만기 순서를 의미한다.

① 스케줄링 창 S에 이분법(*bisection method*)[9]을 적용하여 $\phi(\tau_0)$를 구한다. 여기서, $\phi(\tau_0)$는 S에 속한 모든 태스크들의 만기를 고려한 최대 도함수 값들 중 최소값, 즉 $\min\{\max\{g_i(x_i) \mid \sum_{i=1}^{m} x_i \le \tau_m - \tau_0, \text{ for all } m=1,2,\cdots,W\}\}$ 이다.

② 태스크 리스트 $\aleph = \{t_i \mid g_i(0) \ge \phi(\tau_0), t_i \in S\}$를 구한다.

③ $\aleph$에 속한 모든 태스크에 대해 할당 가능한 서비스 시간 $y_i = g_i^{-1}(\phi(\tau_0))$를 계산한다.

④ 태스크 리스트 $\aleph$에 대해 조건 $\sum_{i=1}^{k} y_i = \tau_k - \tau_0$를 만족하는 k개의 태스크를 선택한다.

⑤ EDF 알고리즘을 적용하여 k개의 태스크들에게 프로세서를 할당한다. 이때, 스케줄된 태스크들은 기껏해야 y_i 시간 동안 실행될 수 있다.

제안 스케줄링 알고리즘은 두 단계의 상하 계층으로 분리된다. 상위 계층(단계 ①~단계 ④)에서는 선택된 W개의 태스크들을 대상으로 W 또는 이보다 적은 k개의 태스크를 실제 스케줄 한다. 상위 계층에 의해 스케줄된 태스크들은 하위 계층(단계 ⑤)에서 EDF 알고리즘에 의해 프로세서가 할당된다. 제안 알고리즘은 매 스케줄링 시점(scheduling point : τ_0)에서 실행된다. 여기서 스케줄링 시점이란 새로운 태스크가 시스템에 도착한 시점 또는 새로운 태스크가 도착하기 전에 이전 스케줄링 시점에서 스케줄된 $k(\le W)$개의 태스크들의 실행을 모두 완료한 시점이다. 단계 ①에서 스케줄링 창 내에 존재하는 W개의 태스크들 중 그들의 만기를 준수하면서 가장 큰 가치 증가율($g_i(x_i)$) 중 가장 작은 도함수 값 $\phi(\tau_0)$를 찾는다. $\phi(\tau_0)$를 찾기 위해[9]에서 사용된 이분법(bisection method)을 적용하였다. $\phi(\tau_0)$를 결정하면 단계 ②에서 현 스케줄링 시점(τ_0)에서의 순간 가치 증가율 $g_i(0)$가 $\phi(\tau_0)$보다 같거나 큰 태스크들의 리스트 $\aleph$를 구할 수 있다. 단계 ③에서는 리스트 $\aleph$의 모든 태스크들에 대해 그들의 가능한 서비스 시간 y_i를 결정한다. 이때 y_i는 $\phi(\tau_0)$를 매개변수로 사용하는 태스크 t_i의 도함수 $g_i()$의 역함수를 이용하여 구한다. 이렇게 구해진 $\aleph$내의 모든 태스크들의 서비스 시간 y_i에 대해 항상 조건 $\sum_{i=1}^{k} y_i = \tau_k - \tau_0$을 만족하는 $k(1 \le k \le W)$가 존재한다[10]. 단계 ④에서는 이러한 조건을 만족하는 k개의 태스크를 선택하여 이들을 하위 계층으로 넘긴다. 만약 새로운 태스크가 선택된 마지막 태스크의 만기인 τ_k전에 도착하면 도착한 그 시점이 다음 스케줄링 시점이 되지만, 그렇지 않으면(k가 다음 스케줄링 시점이 된다. 따라서 알고리즘은 단지 k개의 태스크에 대해서만 서비스 시간 y_i를 결정한다. 서비스 시간이 결정된 k개의 스케줄된 태스크들은 하위 계층인 단계 ⑤에서 EDF 알고리즘에 의해 실제 프로세서가 할당된다.

기존 연구에서처럼 각 태스크 리스트 T, S, $\aleph$에 존재하는 태스크들이 그들의 만기 또는 가치 증가율 순서로 적절히 관리된다고 가정할 경우[3, 9, 10], 단계 ①은 T에 존재하는 모든 태스크가 아닌 스케줄링 창 S에 존재하는 W개의 태스크들을 대상으로 $\phi(\tau_0)$를 찾는다. 그러므로 단계 ①에서 요구되는 시간 복잡도는 $O(W)$이다[9]. 단계 ②와 단계 ③은 W번의 반복 실행(iteration)을 요구하고, 단계 ④와 단계 ⑤는 기껏해야 k번의 반복 실행을 요한다. 여기서 k는 스케줄된 태스크의 개수이고 W보다 같거나 작다. 따라서 매 스케줄링 시점에서 제안 알고리즘은 $O(W)$의 계산 복잡도를 요한다. 그러나 제안 알고리즘은 매 스케줄링 시점에서 시스템 내의 모든 태스크들을 스케줄링하는 것이 아니

라, 단지 k개의 태스크만 스케줄링하므로, 만약 스케줄된 k개의 태스크 모두가 실행을 완료할 때까지 새로운 태스크가 도착하지 않을 경우 재스케줄링을 실시해야 한다. 따라서 최악의 경우 매 스케줄링 시점에서 단지 하나의 태스크만 스케줄된다고 가정한다면, 새로운 태스크가 도착하기 전까지 스케줄링의 총 회수는 시스템 내의 태스크의 총 개수와 같다. 따라서 제안 알고리즘의 최악의 경우 시간 복잡도는 $O(W^2)$이다. 그러나 4절에서 제시될 모의실험 결과에 의하면 실제 실행시의 평균 시간 복잡도는 $O(W)$보다 훨씬 적은 $O(W)$에 가깝다.

제안 알고리즘은 또한 스케줄링 창의 크기 W와 선택정책을 적절히 선택함으로써 기존의 다른 알고리즘과 동일하거나 유사한 스케줄링 결과를 생성할 수 있는 유연성도 제공한다. 만약 W가 1이고 선택정책이 ED라면, 항상 만기가 가장 빠른 태스크 하나만을 선택하여 스케줄링하는 것이므로 기존의 EDF[11]와 유사하다. 그러나 서비스 시간이 사전에 결정되지 않는다는 면에서 EDF와 차이가 있다. 만약 W가 무한대(즉, 시스템 내의 태스크 개수와 동일)라면 선택정책과 상관없이 시스템 내의 모든 태스크를 대상으로 총 가치를 최대화하도록 스케줄링하는 것이므로[10]에서 제안된 알고리즘과 동일하게 된다. 또한 W가 1이고 선택정책이 HRR이라면, 항상 스케줄링 시점에서의 가치 증가율($g_i(0)$)이 가장 높은 태스크 하나만을 스케줄링 한다는 면에서 과욕 정책(greedy policy)[3]과 유사하다. 그러나 과욕 정책은 동일한 시간 구간(time slot)을 가진 시분할 방식을 채택하였고, 라운드-로빈 방식으로 매 시간 구간에서 가치 증가율이 가장 높은 태스크를 선택하여 프로세서를 할당한다는 면에서 차이가 있다.

4. 성능 평가

본 절에서는 모의실험을 통해 제안 알고리즘과 기존 알고리즘의 성능(시간 복잡도와 총가치)을 비교 평가한다. 제안 알고리즘의 실행시의 평균 시간 복잡도와 태스크들의 총가치를 기존 알고리즘과 비교 분석을 위해 두 가지 평가 기준을 정의하였다. 첫째는 제안 알고리즘에 의해 생성되는 총가치(R)와 [3, 9, 10]에서 제시된 온-라인 최적 알고리즘의 총가치(O)와의 비율인 R/O이다([3, 9, 10]에서 제안된 온-라인 알고리즘들은 스케줄링 회수는 서로 다르지만 항상 동일한 최적의 총가치를 생성한다.). 둘째는 스케줄링 횟수의 비율인 $(S-T)/T$이다. 여기서 S는 제안 알고리즘의 총 스케줄링 횟수이고, T는 시스템에 도착한 태스크의 총 개수이다. [3]의 알고리즘은 새로운 태스크가 도착할 때만 재스케줄링하여 모든 태스크들에게 서비스 시간을 결정한다. 하지만 제안 알고리즘은 매번 임의의 $k(\leq W)$개의 태스크를 스케줄하고, 새로운 태스크가 도착하거나 또는 이전에 스케줄된 k개의 태스크가 새로운 태스크가 도착하기 전에 모두 실행되었을 경우에도 재스케줄링을 실시한다. 따라서 $(S-T)/T$는 새로운 태스크의 도착 때를 제외한 추가로 실행된 제안 알고리즘의 실행 횟수($S-T$)와 [3]의 알고리즘의 실행 횟수(T)의 비율이다(이후 비율 $(S-T)/T$는 ST/T로 표기함). 제안 알고리즘이 추구하는 성능 목표는 R/O이 1이고, ST/T는 0이다.

모의실험에 사용될 태스크 집합은 $M/M/\infty$ 큐잉 모델을 기반으로 생성되었다. 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[3, 9, 10, 12]에서와 동일한 $f_i(x)=1-e^{-w_i x}$ 형태의 지수 함수이다(여기서 w_i는 $(0, w_u)$의 범위 내의 균등(uniform) 분포를 따르는 임의의 실수이며, w_u는 사전에 정의된 w_i의 상위 경계 값이다). 서로 다른 λ, ρ, w_u 변수 값을 갖는 태스크 집합에 대해 모의실험을 실시하였으며, 각 태스크 집합은 25,000개의 태스크로 구성된다.

<표 1>은 서로 다른 가치함수 가중치(w_i)의 상위 경계 값(w_u)을 가진 태스크 집합에 대해 기존 최적 온-라인 알고리즘[3, 9, 10]이 생성하는 평균가치(RT)를 보여 주고 있다. 이 실험에서 시스템 내에 상존하는 평균 태스크 개수 ρ는 10이고, 평균 태스크 도착율 λ는 1이다. 평균가치는 스케줄링 결과 태스크들이 확보한 총가치를 태스크 개수로 나눈 값이며, 이는 항상 1보다 같거나 작다. 따라서 평균가치는 총가치와 동일한 의미를 가진다. 표에서 알 수 있듯이 w_u가 증가할수록 평균가치 또한 증가한다. 이는 동일한 조건 하에서 w_u가 증가할수록 평균적으로 각 태스크의 가치함수 가중치 w_i가 증가하여, 단위 시간당 가치 증가율이 그 만큼 더 커지기 때문이다. 이후 제시될 제안 알고리즘의 다양한 실험 결과는 λ와 w_u에 의존하기보다는 생성된 평균가치(또는 총가치)에 의존하는 경향을 보였다. 즉, λ와 w_u가 달라도 생성된 평균가치가 동일하면 제안 알고리즘은 항상 비슷한 실험 결과를 보였다. 따라서 본 논문에서는 다양한 평균가치를 생성하는 태스크 집합을 만들기 위해 λ를 1로 고정시키고 w_u를 <표 1>과 같이 변경하였다.

<표 1> 기존 온-라인 최적 알고리즘에 의해 생성된 평균가치

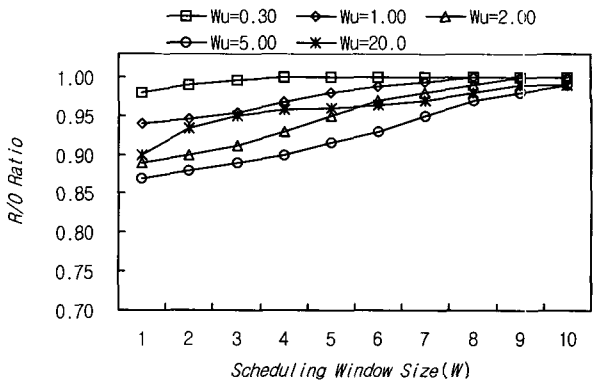
w_u	0.3	0.5	1.0	1.5	2.0	3.0	5.0	8.0	20
RT	0.169	0.252	0.391	0.483	0.562	0.663	0.779	0.865	0.958

(그림 1)과 (그림 2)는 서로 다른 w_u를 가진 태스크 집합들에 대해 스케줄링 창 크기(W)별로 실시한 모의실험 결과인 비율 R/O와 ST/T를 보인 것이다. 이 실험에서 태스크 선택정책(SP)은 HRR, 시스템 내에 상존하는 평균

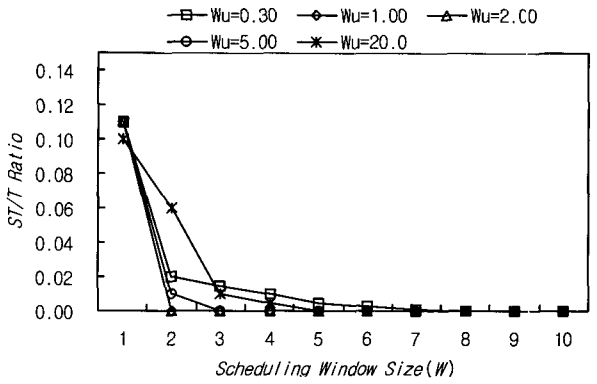
태스크 개수 ρ는 10, λ는 1로 고정하였다. (그림 1)에서 총 가치 비율 R/O는 w_u와 W에 상관없이 항상 0.88보다 크다. 이는 ρ가 10이고 HRR 선택정책을 사용할 경우, 제안 알고리즘은 온-라인 최적 알고리즘이 생성하는 총가치의 88% 이상을 항상 생성함을 의미한다. 예측되는 바와 같이 W가 증가할수록 총가치 또한 함께 증가한다. 그림에서 평균가치가 매우 적을 경우($w_u = 0.3$), 창의 크기와 상관없이 온-라인 최적 알고리즘과 유사한 총가치를 생성함을 알 수 있다. (그림 2)에서 창의 크기가 3이상일 경우, HRR 정책을 채택한 제안 알고리즘은 w_u에 상관없이 새로운 태스크가 도착하는 시점을 제외하고는 추가로 재스케줄링을 거의 하지 않음을 알 수 있다(ST/T가 0에 근접함). 이는 곧 W가 3이상인 경우, 평균적으로 이전에 스케줄된 모든 태스크들의 실행 완료 전에 새로운 태스크가 도착한다는 것을 의미한다. 이는 제안 알고리즘이 스케줄링 대상을 W개로 축소해도 생성되는 총가치는 약간 떨어지지만, 새로운 태스크가 도착하는 경우를 제외한 추가로 재스케줄링하는 경우는 거의 없다는 것을 의미한다. 이러한 실험 결과를 통해 제안 알고리즘의 시간 복잡도가 최악의 경우 $O(W_n)$일지라도 평균적으로는 거의 $O(W)$에 가깝다는 것을 알 수 있다. 이는 곧 알고리즘 복잡도가 최악의 경우 시스템 내의 태스크 개수에 의존하지만, 평균적으로 스케줄링 창의 크기에 의존한다는 것을 의미한다.

(그림 3)과 (그림 4)는 선택정책을 ED로 변경하고 다른 실험 변수들은 (그림 1)과 동일하게 설정한 후 실시한 모의 실험 결과인 비율 R/O와 ST/T를 보인 것이다. 그림에서 W가 작아질수록 HRR 선택정책을 채택한 (그림 1)과 (그림 2)의 결과보다 R/O는 더 감소하고, ST/T는 더 증가한다. 이는 W가 작을수록 ED 정책이 HRR 보다 성능이 더 떨어짐을 의미한다. 그러나 W가 증가함에 따라 R/O는 빠르게 1에 접근한다. 또한 HRR 정책의 경우 W가 증가함에 따라 R/O 비율이 w_u에 따라 다르게 증가하지만, ED 정책의 경우 w_u에 상관없이 R/O 비율은 거의 동일한 창의 크기에서 1에 접근한다. 또한 흥미로운 것은 w_u가 증가할수록 R/O 비율은 HRR (그림 1)과 반대로 오히려 증가한다는 것이다. 이를 통해 동일 조건에서 평균가치가 낮을수록 HRR 정책이 더 많은 총가치를 생성하고, 평균가치가 높아질수록 ED 정책이 더 많은 총가치를 생성한다는 것을 알 수 있다. ST/T 비율 역시 HRR (그림 2)에 비해 상대적으로 높게 나타나지만 창의 크기가 증가하면서 0에 접근한다. 특히, 창의 크기가 6이상일 경우, R/O와 ST/T는 w_u에 상관없이 온-라인 최적 알고리즘의 결과와 거의 동일해진다. 따라서 ED 정책은 창의 크기가 작으면 HRR 정책에 비해 성능이 떨어지지만, 창의 크기가 일정 이상 커지면 오히려 HRR 보다 더 좋은 성능을 발휘하고, 온-라인 최적 알고리즘의 결과와 거의 동일해진다는 것을 알 수 있다.

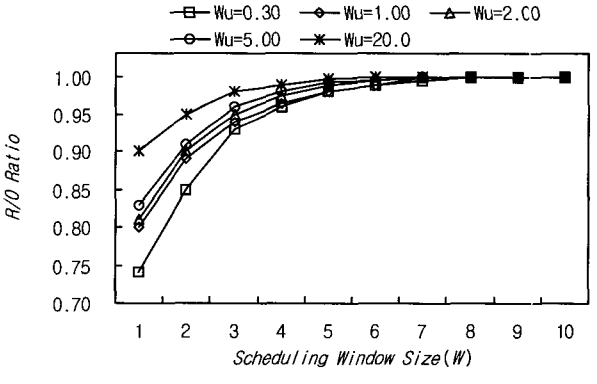
시스템에 상존하는 평균 태스크의 개수 ρ가 제안 알고리



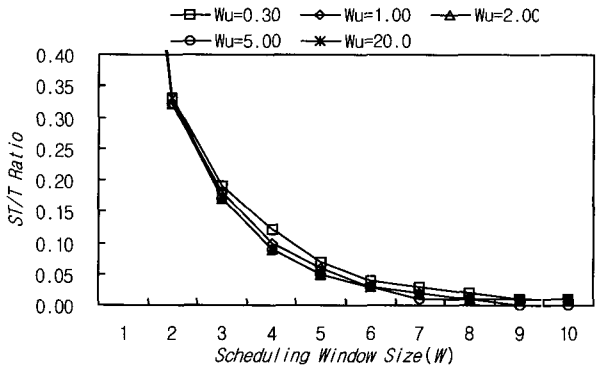
(그림 1) R/O 비율(ρ = 10, λ = 1, SP = HRR)



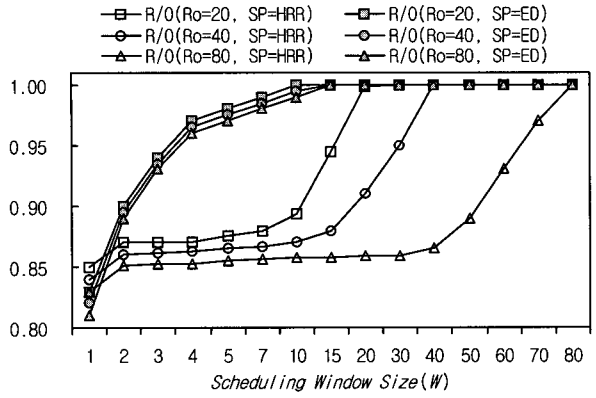
(그림 2) ST/T 비율(ρ = 10, λ = 1, SP = HRR)



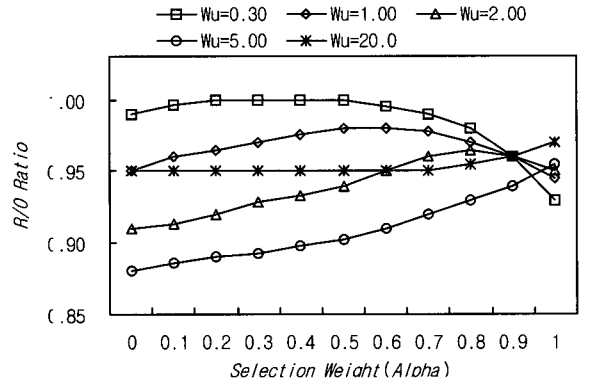
(그림 3) R/O 비율(ρ = 10, λ = 1, SP = ED)



(그림 4) ST/T 비율(ρ = 10, λ = 1, SP = ED)



(그림 5) R/O 비율(λ = 1, w_u = 8)



(그림 6) R/O 비율(ρ = 10, λ = 1, W = 3)

즘의 성능에 어떻게 영향을 미치는지 확인하기 위해 서로 다른 ρ를 가진 태스크 집합들에 대해서도 모의실험을 실시하였다. (그림 5)는 서로 다른 ρ(그림에선 Ro로 표기 되었음)를 가진 태스크 집합들에 대해 서로 다른 선택정책과 스케줄링 창 크기(W)별로 실시한 모의실험 결과 R/O 비율을 보인 것이다. 이 실험에서 λ는 1로, w_u는 8로 고정하였다. 그림에서 선택정책이 ED인 경우, 세 개의 서로 다른 ρ (20, 10, 80)에 대해 거의 동일한 총가치를 생성한다. 이는 ED 선택정책을 가진 제안 알고리즘이 생성하는 총가치는 시스템 내의 상존하는 태스크의 개수에 거의 영향을 받지 않는다는 것을 의미한다. 뿐만 아니라, 스케줄링 창의 크기가 증가할수록 ρ에 상관없이 빠르게 온-라인 최적 알고리즘의 총가치에 접근한다. 이것은 제안 알고리즘이 ED 선택정책과 특정 값((그림 5)에서는 15) 이상의 스케줄링 창을 가질 경우 기존 온-라인 최적 알고리즘과 동일한 총가치를 생성한다는 것을 의미한다. 반면 HRR 선택정책의 경우 ρ에 상당한 영향을 받는다.

본 연구에서는 또한 만기가 가장 빠른 태스크들과 순간 가치 증가율이 가장 큰 태스크들을 적절히 섞어 W개의 태스크를 선택했을 때의 스케줄링 결과를 확인하기 위해 통합 선택정책의 선택 가중치 α를 다양하게 변경시키면서 실험을 실시하였다. (그림 6)은 ρ는 10, λ는 1, W는 3으로 고정시킨 상태에서, 각 α별 실험 결과인 총가치 R/O 비율이 다. 그림에서 α = 0(HRR 선택정책)과 α = 1(ED 선택정책)일 때는 각각 (그림 1)과 (그림 3)의 W = 3일 때의 결과와 일치한다. 전체적으로 각각의 w_u에 대해 α가 증가함에 따라 R/O 비율도 함께 증가하다가, α가 일정 이상이 되면 R/O 비율은 다시 감소하기 시작한다. 특히, 각각의 w_u에 대해 가장 좋은 R/O 비율을 보이는 α 역시 w_u가 증가함에 따라 함께 증가 한다. 즉, 태스크 선택시 적은 w_u에 대해서는(생성된 평균가치가 적은 경우) 순간 가치 증가율을 더 많이 반영하고, 높은 w_u에 대해서는 만기를 더 많이 반영해야 하는 것이 보다 많은 총가치를 생성한다. 이 실험결과는 스케줄링 창의 크기가 작을 경우(그림에선 W = 3), 단순히 태스크들의 만기나 또는 순간 가치 증가율만을 고려하여 태스크를 선택하기 보다는 이들 두 요소를 적절히 섞어 선택하는 것이 보다 효율적이라는 것을 보여준다.

이상의 모의실험을 통해, HRR 선택정책을 채택한 제안 알고리즘은 도착율 λ, 창의 크기 W, 시스템 내의 상존하는 태스크 개수 ρ에 상관없이 항상 온-라인 최적 알고리즘이 생성하는 총가치의 83% 이상을 생성하고, 추가로 실행되는 스케줄링 횟수는 항상 태스크 도착 횟수의 12% 이하라는 사실을 알 수 있다. 그러나 최적의 총가치(R/O = 1)를 생성하기 위해선 창의 크기를 시스템 내에 존재하는 태스크 개수만큼 증가시켜야 하는 단점이 있다. 반면, ED 정책을 채택한 제안 알고리즘은 창의 크기가 작을수록 HRR 정책보다도 더 낮은 성능을 보이지만, 창의 크기가 일정 이상((그림 3)과 (그림 5) 참조) 커지면 λ, W, ρ에 상관없이 온-라인 최적 알고리즘의 결과와 거의 동일한 스케줄링 결과를 생성한다. 이는 곧 그 만큼의 스케줄링 복잡도를 증가시킨다. 또한 제안 알고리즘이 통합 선택정책을 채택할 경우, 선택 가중치를 태스크들의 평균가치에 따라 적절히 조절함으로써 스케줄링 성능을 보다 향상시킬 수 있음을 확인했다. 그러나 통합 선택정책은 태스크 선택시 모든 태스크들에 대해 선택 수식을 계산해야 하는 스케줄링 부하를 가진다.

5. 결 론

IRIS 태스크를 위한 기존 스케줄링 알고리즘들은 많은 태스크들을 가진 실제 시스템에 적용하기에는 매우 높은 시간 복잡도를 가진다. 이러한 높은 시간 복잡도는 최적의 총가치를 생성하기 위해 시스템 내의 모든 태스크들을 스케줄링 대상으로 고려하기 때문이다. 본 논문에서는 스케줄링 복잡도를 줄이기 위한 방안으로 스케줄링 창을 가진 휴리스틱 온-라인 스케줄링 알고리즘을 제안하였다. 제안 알고리즘은 시스템 내의 모든 태스크들을 대상으로 스케줄링 하는 것이 아니라, 사전에 정의된 선택정책에 의해 선택된 W개의 태스크들만을 대상으로 스케줄링 한다. 본 논문에서는 두 개의 기본적인 선택정책과 이 둘을 적절히 반영한 통합 선택정책을 제시하였다.

선택된 태스크들로 스케줄링 대상을 축소함으로써 제안

알고리즘의 복잡도를 $O(Wn)$로 줄일 수 있었다. 다양한 성능실험 결과 알고리즘 복잡도는 평균적으로 $O(W)$에 더 가깝다는 것을 확인할 수 있었다. 또한, 본 연구팀은 제안 알고리즘의 실제 활용 가능성을 확인하기 위하여 네트워크를 통한 화상회의 시스템에 제안 알고리즘을 적용하였으며 그 실험결과를 발표하였다[13]. 이 시스템은 Wavelet 방식에 의해 압축된 영상을 복원하여 화면에 보여주는 시스템으로 실험결과 상대적으로 낮은 스케줄링 복잡도를 가지고도 기존 최적 알고리즘들과 유사한 화질의 프레임을 복원할 수 있음을 확인할 수 있었다.

제안 알고리즘은 스케줄링 창의 크기와 선택정책에 따라 그 성능이 달라지며, 이는 총가치와 스케줄링 복잡도의 손익(trade-off)을 고려해 결정해야 한다. 총가치를 증가시키기 위해선 스케줄링 창의 크기를 늘려야 하지만, 이는 또한 스케줄링 복잡도의 증가를 초래한다. 적절한 창의 크기를 선택하는 것은 응용에 의존적이지만, 제안 알고리즘은 시스템 설계자로 하여금 요구되는 총가치와 시간 복잡도를 선택할 수 있는 유용성을 제공한다.

참 고 문 헌

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

[2] 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.

[3] 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, pp.802-813, July, 1996.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

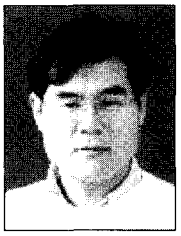
[9] K. Choi and G. Jung, "Comment on On-line Scheduling Policies for a Class of IRIS Real-Time Tasks," IEEE Trans. Computers, Vol.50, No.5, pp.526-528, May, 2001.

[10] G. Jung, T. Kim, S. Park and K. Choi, "A Low Complexity Dynamic Scheduling Algorithm for Real-Time Tasks, IEE Electronic Letters," Vol.35, No.24, pp.2106-2108, Nov., 1999.

[11] 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.

[12] 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.

[13] Jaehong Shim, Kangbin Yim, Kyunghee Choi and Gihyun Jung, "An On-line Frame Scheduling Algorithm for the Internet Video Conferencing," IEEE Transactions on Consumer Electronics, Vol.49, No.1, pp.80-88, Feb., 2003.



심 재 홍

e-mail : jhshim@chosun.ac.kr

1987년 서울대학교 전산과학과(학사)

1989년 아주대학교 컴퓨터공학과(석사)

2001년 아주대학교 컴퓨터공학과(박사)

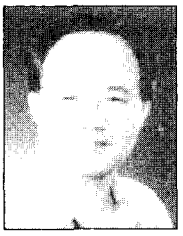
1989년~1994년 서울시스템(주) 공학연구소

1999년~2000년 University of Arizona 객원연구원

2001년~2001년 아주대학교 정보통신전문대학원 BK21 전임연구원

2001년~현재 조선대학교 인터넷소프트웨어공학부 전임강사

관심분야 : 운영 체제, 분산시스템, 실시간 및 멀티미디어시스템



최 경 희

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

1976년 서울대학교 수학교육과(학사)

1979년 프랑스 그랑데꼴 Enseeiht대학(석사)

1982년 프랑스 Paul Sabatier대학 정보공학부(박사)

1982년~현재 아주대학교 정보통신전문대학원 교수

관심분야 : 운영 체제, 분산시스템, 실시간 및 멀티미디어 시스템 등

정 기 현

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

1984년 서강대학교 전자공학과(학사)

1988년 미국 Ilinois주립대 EECS(석사)

1990년 미국 Purdue대학 전기전자공학부(박사)

1991년~1992년 현대반도체 연구소

1993년~현재 아주대학교 전자공학부 교수

관심분야 : 컴퓨터구조, VLSI 설계, 멀티미디어 및 실시간 시스템 등