pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2013.2.6.237

Real-time Scheduling for (m,k)-firm Deadline Tasks on Energy-constrained Multiprocessors

Yeonhwa Kong† · Hyeonjoong Cho††

ABSTRACT

We propose Energy-constrained Multiprocessor Real-Time Scheduling algorithms for (m,k)-firm deadline constrained tasks (EMRTS-MK). Rather than simply saving as much energy as possible, we consider energy as hard constraint under which the system remains functional and delivers an acceptable performance at least during the prescribed mission time. We evaluate EMRTS-MKs in several experiments, which quantitatively show that they achieve the scheduling objectives.

Keywords : Real-time Scheduling, Static or Dynamic Voltage Scaling, Multimedia, Weakly Hard Rreal-time Constraints

한정된 전력량을 가진 멀티프로세서 시스템에서 (m,k)-firm 데드라인 태스크를 위한 실시간 스케줄링 기법

공 연 화† · 조 현 중††

요 약

본 연구에서는 전력량 제약을 가진 멀티프로세서 시스템에서 (m,k)-firm 데드라인을 갖는 실시간 태스크를 효율적으로 스케줄링 할 수 있는 방법으로 EMRTS-MK(Energy-constrained Multiprocessor Real-Time Scheduling algorithms for (m,k)-firm deadline constrained tasks) 를 제안한다. EMRTS-MK는 단지 전력 소모량을 최소로 줄이는 것이 목표가 아니라 한정된 전력량을 고려하여 시스템이 주어진 임무 시간(Mission Time) 동안 최소한의 서비스 품질을 보장하고 동시에 가능하면 최대한의 서비스 품질을 제공함을 목표로 한다. 본 연구에서는 상용 멀티코어 환경에서 EMRTS-MK를 구현하여 성능을 평가하였으며, 제안된 알고리즘이 (m,k)-firm 데드라인을 갖는 멀티미디어 서비스를 효과적으로 지원해 줄 수 있다는 것을 보였다.

키워드 : 실시간 스케줄링, 정적/동적 전압 조절, 멀티미디어, 약경성 실시간성

1. 서 론

단일 프로세서에 비해 적은 전력 소모량과 낮은 제조 비용 등의 장점으로 최근 멀티프로세서에 대한 연구가 활발히 진행되고 있다. 특히, 제한된 전력량으로 동작하는 휴대용 기기들은 전력을 효율적으로 사용하는 것이 중요한 설계 요소이므로 멀티프로세서가 널리 탑재되는 추세이다. 본 연구는 멀티프로세서를 탑재한 휴대용 기기에서 배터리 등 제한된 전력량을 고려하여 멀티미디어 등을 처리하기 위한 효과적인 실시간 스케줄러를 제안한다.

멀티미디어 서비스가 지니는 특징 중 하나는 서비스를 처리하기 위한 실시간 태스크들이 경우에 따라서 데드라인을 놓치는 것을 허용할 수 있다는 점이다. 예를 들어 초당 수십 개의 비디오 프레임을 디코딩해야하는 실시간 태스크가 실행 도중 잠시 데드라인을 놓쳐 몇몇 프레임이 처리되지 못하더라도, 서비스의 품질은 조금 떨어지지만 사용자는 처리되지 못한 프레임의 앞뒤 영상으로부터 전체 영상의 내용을 인지할 수 있다. 이러한 멀티미디어 서비스의 품질(QoS)은 전통적으로 통계적 방법으로 표현되어 왔다.

반면 본 연구는 멀티미디어 등의 서비스 품질을 통계적이 아닌 결정적인 (Deterministic) 방법으로 표현할 수 있는 (m,k)-firm 데드라인 모델을 기반으로 하고 있다[1]. (m,k)-firm 데드라인 모델은 긴 시간 동안의 평균적인 서비스 품질을 표현하는 확률적인 방법과는 달리 시간 구간의 길이와

※ 이 논문은 2012년도 정부 (교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.2011-0011534).
† 비 회 원 : 고려대학교 전산과학 석사
†† 종신회원 : 고려대학교 컴퓨터정보학과 부교수
논문접수 : 2013년 1월 25일
수 정 일 : 1차 2013년 4월 12일, 2차 2013년 4월 24일
심사완료 : 2013년 4월 24일
* Corresponding Author : Hyeonjoong Cho(raycho@korea.ac.kr)

무관하게 정확한 QoS 표현을 가능하게 한다. (m,k)-firm 데드라인 모델에서 $\delta_{i,j}$를 i번째 태스크의 j번째 작업(Job)이라고 하고, $\delta_{i,j}$의 (m,k)-firm 데드라인 변수를 각각 m_i, k_i라고 할 때, $\delta_{i,j}$을 포함하여 이전에 도착한 가장 가까운 k_i개의 작업을 나타낸 열을 $(\delta_{i,j-k+1}, ..., \delta_{i,j-1}, \delta_{i,j})$로 나타낼 수 있다. 이때 작업열의 각 작업이 데드라인을 만족시킴을 M, 만족시키지 못할 경우를 m이라고 표시할 경우, M의 개수가 m_i보다 작다면 동적 실패 (Dynamic Failure)가 발생했음을 나타낸다. 또 동적 실패가 발생하기까지 연속적으로 놓쳐야 하는 데드라인의 개수를 동적 실패까지의 거리로 표현한다. Hamdaoui 등은 이와 같은 거리를 이용하여 동적 실패까지 짧은 거리에 있는 태스크를 먼저 스케줄링 하는 DBP (Distance Based Priority)을 제안하였다[1].

Barnet[11]은 (m,k)-firm 데드라인 등의 시간 제약들을 정리하여 실시간 시스템의 데드라인 만족 정도를 결정적으로 나타낼 수 있는 약경성 실시간 제약(Weakly hard real-time constraint, WHRT) 을 제안하였다.

Ramanathan은 [2]에서 (m,k)-firm 데드라인을 갖는 태스크의 작업들을 의무 (Mandatory) 작업과 선택 (Optional) 작업으로 나누었고 G.Quan 는 의무 및 선택 작업을 선정하는 다양한 (m,k) 패턴을 정의하였다[3]. 또한 Koren은 (m,k) 패턴 중 하나로 "Deeply-red pattern"을 정의하였는데[4], 본 연구에서는 이를 이용하여 의무 및 선택 작업을 나눈다.

AlEnawy 등은 단일 프로세서에서 한정된 전력량이 주어졌을 때 (m,k)-firm 데드라인을 갖는 실시간 태스크들을 스케줄링하기 위한 탐욕 (Greedy) 알고리즘과 전력량 밀도 (Energy-Density) 알고리즘을 제안하였다.

Wu는 멀티프로세서에서 (m,k)-firm 데드라인 태스크를 스케줄링 하기 위해 DBP를 확장한 MPDBP (Multi-Processor DBP)를 제안하였다[6]. 하지만 MPDBP는 태스크들의 (m,k)-firm 데드라인을 만족 정도에 대한 보장을 하지 않는 Best-effort 이다. 또, 내장형 시스템 적용을 위해 전력 소모를 고려한 MPDBP 확장에 대한 후속 연구도 아직 알려진바 없다.

Gong은 멀티프로세서에서 (m,k)-firm 데드라인을 가지는 실시간 태스크를 스케줄링하기 위한 GMRTS-MK를 제안하였다[9]. GMRTS-MK는 MPDBP와는 달리 저부하 상태에서 동적 실패가 일어나지 않음을 보장한다. 또 부하에 관계없이 태스크들의 데드라인을 최대한 만족시키도록 설계되어 QoS를 최대화한다. 하지만 GMRTS-MK는 전력 소모를 고려하지 않았는데 반해 본 연구에서는 한정된 전력량을 가지는 휴대용 장치가 주어진 임무 시간 (Mission Time) 동안 운영되고 (m,k)-firm 데드라인으로 표현되는 최소한의 QoS를 제공하기 위해 GMRTS-MK를 확장하고자 한다.

멀티프로세서에서 전력 소모를 줄이기 위한 시도가 많이 있어왔다[13,14,15]. 하지만, 본 연구에서는 제한된 전력 자원을 갖는 휴대용 장치 초점을 맞추고, 단순히 전력 소모를 줄이는 것이 아니라, 시스템의 한정된 전력량을 고려하여 주어진 임무 시간 동안 최소한의 QoS를 보장하고, 필요한 경우 최대한의 QoS를 제공하기 위한 멀티프로세서 환경에서의 실시간 스케줄링, EMRTS-MK (Energy-constrained Multiprocessor Real-Time Scheduling algorithms for (m,k)-firm deadline constrained tasks) 을 제안한다.

멀티프로세서 환경에서 (m,k)-firm 태스크를 위한 EMRTS-MK의 스케줄링 목표는 다음과 같다.

(1) 태스크들의 계산 부하 (Utilization)가 멀티프로세서의 계산 용량(Capacity)보다 적은 저부하 상태에서 초기 시스템의 전력량이 허락할 경우, 모든 태스크들이 동적 실패 없이 임무 시간을 만족함을 보장한다.

(2) 태스크들의 계산 부하가 저부하 혹은 과부하 상태 모두의 경우에 시스템 전력량이 허락하는 한 태스크들의 데드라인 만족도를 최대화하여 멀티미디어 서비스의 QoS를 최대화한다.

2. 시스템 모델

2.1 태스크 모델

본 논문에서는 n개의 주기적인 태스크로 이루어진 태스크 집합 $\{T_1, T_2 \cdots T_n\}$을 스케줄링 대상으로 가정한다. 주기 태스크 T_i는 일정 주기마다 반복되는 작업들로 구성되어 있다. T_i는 네 개의 파라메터 (C_i, P_i, m_i, k_i)로 정의될 수 있는데 여기서 C_i는 최악 실행 시간(Worst Case Execution Time), P_i는 주기, m_i과 k_i는 (m,k)-firm 데드라인의 파라메터를 나타낸다.

각 태스크는 동적 실패까지의 거리 $Dist_i$에 따라 긴급 태스크와 비긴급 태스크로 나뉜다. 긴급 태스크는 $Dist_i=1$인 태스크로 한번의 데드라인 어김으로도 동적 실패가 일어나는 태스크이고, 비긴급 태스크는 $Dist_i>1$인 태스크이다.

2.2 전력 모델

본 연구에서는 내장형 시스템이 멀티프로세서를 탑재하고 한정된 전력량을 갖는 배터리로 운영되며, 주어진 임무 시간, X까지 배터리를 재충전할 수 없다고 가정한다. 제안된 스케줄링을 표현하기 위해 다음과 같은 용어들을 사용한다.

- E_{max}: 시스템이 가질 수 있는 전력량의 최대값
- E_{min}: 시스템의 성능을 보장할 수 있는 전력량의 최소값
- s: 시스템의 최대 속도로 현재 속도를 나눈 표준화된 속도로 1보다 작거나 같은 값을 가짐.
- $E_{budget}(t)$: 시간 t부터 시스템이 사용할 수 있는 전력량 값. 만약 배터리가 최대로 충전 되어있다면 $E_{budget}(0)$은 $E_{max}-E_{min}$로 정의됨.
- $E_{total}(t, s)$: 프로세서 속도 s로 모든 태스크의 작업들을 시간t로부터 임무 시간X까지 수행하기 위해 필요한 전력량.
- $E_{limit}(t,s)$: 프로세서 속도 s에서 모든 태스크의 의무 작업들을 시간 t로부터 임무 시간 X까지 수행하기 위해 필요한 전력량.

따라서 $E_{limit}(0,s)$은 다음과 같은 공식으로 구할 수 있다.

$$E_{limit}(0,s) = \alpha s^3 \sum\left(N_i^m \frac{C_i}{s}\right)$$

여기서 N^m_i은 임무 시간까지 T_i의 의무 작업 개수, α는 상수를 나타낸다. 본 연구에서는 CMOS 기반의 코어가 단일 클럭을 공유하는 멀티프로세서 설계를 가정한다. 이때 소모 전력 함수 g(s)는 프로세서 속도 s의 함수 $g(s) \propto s^3$로 표현된다[7,8].

EMRTS-MK는 이와 같은 모델을 사용하여 $E_{total}(t,s)$ 및 $E_{limit}(t,s)$를 스케쥴링 이벤트마다 계산하고 현재 전력량 E_{budget} (t)와 비교하여 태스크의 스케쥴링 방법을 동적으로 결정한다. 즉, $_{Etotal}(t,s)$는 다음의 수식으로 구할 수 있다.

$$E_{total}(t,s) = s^3 \sum\left(\left\lceil \frac{X-t}{P_i} \right\rceil C_i - C_i^p \right)$$

$E_{limit}(t,s)$은 t시점에서 스케줄링을 결정해야 하는 태스크가 긴급 태스크인 경우와 비긴급 태스크인 경우로 나누어 구할 수 있다. 만약 t시점에서 태스크가 긴급 태스크라면,

$$E_{limit}(t,s) = s^2 \sum\left(\left\lceil \left(\frac{X-US(t)}{P_i}\right)\left(\frac{m_i}{k_i}\right) \right\rceil C_i - C_i^p \right),$$

비긴급 태스크라면,

$E_{limit}(t,s) = s^2 \sum \left\lceil \left(\frac{X-US(t)}{P_i}\right)\left(\frac{m_i}{k_i}\right) \right\rceil C_i$ 로 구한다. 여기서 X는 의무 시간을 나타내고, 은 현재 작업에서 수행된 실행 시간을 나타낸다. US(t)은 t시점에서 가장 가까이 긴급 태스크가 릴리즈되는 시간을 나타낸다.

3. 문제 정의와 성능 측정

EMRTS-MK가 목표로 하는 스케줄링 문제는 다음과 같이 정의할 수 있다. "한정된 전력량을 가지는 배터리로 동작하는 멀티프로세서 시스템에서 모든 태스크들이 주어진 (m,k)-firm 데드라인을 임무 시간 동안 동적 실패 없이 만족시킴을 보장하는 동시에 최대한 태스크의 데드라인을 만족시키도록 스케줄링 할 수 있을까?"

본 연구에서는 임무 시간 만족 정도 (MTS, Mission Time Satisfaction), 동적 실패 비율 (DFR, Dynamic Failure Ratio), 데드라인 만족율(DSR, Deadline Satisfaction Ratio)의 세 가지 측면에서 알고리즘의 성능을 측정한다. MTS는 주어진 임무 시간까지 동적 실패 발생 여부로 결정되고, DFR은 총 작업의 개수 대비 동적 실패 발생 횟수의 비율로, DSR은 총 작업 개수 중 데드라인을 만족시킨 개수의 비율로 구해진다.

4. 알고리즘

EMRTS-MK 알고리즘은 정적 EMRTS-MK과 동적 EMRTS-MK 등 두 가지 방법으로 구현된다. 정적 EMRTS-MK는 고정된 프로세서 속도로 태스크들을 스케줄링 하는 방법으로 시스템이 시작될 때 프로세서 속도가 단 한번 정해지는 반면 동적 EMRTS-MK는 각 태스크들의 주기마다 프로세서 속도가 동적으로 변화한다. 즉 정적 EMRTS-MK는 알고리즘 1의 CalculateSpeed()를 시스템 시작 시 한번 호출하고, 동적 EMRTS-MK는 같은 함수를 각 태스크의 주기마다 매번 호출한다.

CalculateSpeed()는 프로세서 속도를 결정한다. Calculate Speed()의 입력 값은 T_i = (C_i, P_i, m_i, k_i) 이고 출력결과는 표준화된 프로세서 속도 s이다. 알고리즘1로 프로세서 속도가 새로운 s로 변하면 태스크의 실행시간의 값 역시 변하게 되는데, 이를 C_i와 구별하여 C_i'으로 표시한다.

알고리즘 1의 동작 원리는 다음과 같다. 하위 스케쥴러로 사용된 P-fair는 저부하에서 주기 태스크들이 데드라인을 100% 만족시키는 실시간 스케쥴링 알고리즘으로 알려져 있다[10]. 하위 스케쥴러가 M보다 작거나 같은 계산 요구량을 갖는 태스크 집합의 실시간성을 보장하므로 프로세서 속도는 알고리즘 1의 줄1과 같이 최대한 낮게 결정될 수 있다. 하지만 각 태스크의 개별 계산 요구량이 1을 초과할 수 없으므로 프로세서 속도는 줄 22과 같이 결정된다. 하위 스케쥴러 P-fair는 퀀텀 기반 스케쥴러로 각 태스크의 실행시간을 정수로 가정한다. 따라서 줄 2부터 줄 20까지는 C_i'를 정수로 보정하기 위한 과정이다. 알고리즘 1의 결과값 s는 0과 1사이의 실수값으로 목표 시스템이 프로세서 속도를 연속적으로 가변할 수 있음을 가정한다. 하지만 불연속적인 프로세서 속도를 지원하는 실제 시스템에 적용할 때는 알고리즘 1의 결과값 s보다 빠른 최소의 프로세서 속도가 최종적으로 선택된다.

알고리즘 1. CalculateSpeed(T_i)

1: $S_{temp} = \dfrac{\sum \frac{C_i}{P_i}}{M}$; /* M: 프로세서의 개수*/

2: Umax = max{U_i } /* 태스크들의 최대 계산요구량 */

3: for(i=0;$i<n$;i++)

4: $C_i'=C_i/$Stemp

5: $U_{temp} = \dfrac{\sum\left(\left\lceil \frac{C_i}{S_{temp}} \right\rceil\right)}{P_i}$;

6: while(U_{temp}>M){

7: for(i=0, least=0 ; $i<n$ i++){

8: if($\lceil C'_i \rceil - C'_i \neq 0$){

9: if(($C'_i - (\lceil C'_i \rceil - 1)) < C'_{least} - (\lceil C'_{least} \rceil - 1)$){

10: least = i;

11: }

12: }

13: }

14: $S_{temp} = \frac{C_{least}}{(\lceil C'_{least} \rceil - 1)}$;
15: U_{temp} =0;
16: for(i=0;i<ni++){
17: $C_i' = C_i / S_{temp}$;
18: $U_{temp} = U_{temp} + \frac{\lceil C'_i \rceil}{P_i}$;
19: }
20: }
21: if(S_{temp}>1) S_{temp}=1;
22: s = max{S_{temp}, U_{max}} ;
23: return s;

정적 EMRTS-MK는 내장형 시스템의 한정된 전력량를 고려하여 GMRTS-MK[9]를 확장한 알고리즘으로, 상 하위로 나눌 수 있는 계층적 구조로 되어있다. 상위 스케쥴러은 각 태스크의 주기마다 실행되는 스케쥴러로 하위 스케쥴러로 보낼 태스크를 선택하고, 하위 스케쥴러에서는 상위에서 선택된 태스크들을 P-fair 알고리즘으로 스케쥴링 한다. 따라서 상위 스케쥴러에서 동적 실패를 최대한 줄이면서 전력소모를 최소화하기 위해 선택된, M보다 작은 연산 요구량을 갖는 태스크 집합은 하위 스케쥴러 P-fair에 의해 실시간성을 보장받는다.

EMRTS-MK는 세 개의 큐를 사용한다. QA는 하위 스케쥴러로 보내질 태스크가 저장되는 큐, QU는 긴급 태스크가, 그리고 QNU는 비긴급 태스크가 저장되는 큐이다.

알고리즘2에서 getTask()는 태스크의 주기에서 스케쥴링 이벤트를 일으킨 태스크를 불러오는 함수이다. updateDist()는 스케쥴링 이벤트를 일으킨 태스크의 동적 실패까지의 거리를 업데이트한다. 만약 스케쥴링 이벤트를 일으킨 태스크가 긴급이면 이를 QU에, 비긴급이라면 QNU에 저장한다.

이후 EMRTS-MK는 E_{total}, E_{limit}, EA을 계산한다. E_{total}은 모든 태스크를 수행하는데 필요한 전력량이고, EA은 하위 스케쥴러에 의해 운영되고 있는 태스크를 수행하는데 필요한 전력량이다. ET_i는 태스크 T_i를 임무 시간까지 실행시킬 수 있는 전력량을 나타낸다.

EMRTS-MK는 현재 시스템의 잔존 전력량에 따라서 스케줄링 정책을 변경하는데, 만약 지금 가진 전력량이 모든 태스크의 모든 작업들을 실행하기에 충분하다면 GMRTS-MK와 동일하게 작동하지만 전력량이 부족하다면 임무 시간과 (m,k)-firm 데드라인 등을 고려하여 태스크의 작업들을 선택적으로 실행한다.

먼저 E_{budget}이 E_{total}보다 클 경우, 긴급 태스크는 계산 부하가 작은 순으로, 비긴급 태스크는 동적 실패에 가장 가까운 태스크 순으로 총 계산 부하의 합이 M보다 작도록 QA에 저장되고 하위 스케쥴러로 전달된다. 두 번째로 E_{budget}이 E_{limit}보다 클 경우에는 긴급 태스크는 데드라인이 임박한 순으로 QA에 저장되고, QA의 태스크들의 계산 부하가 M보다 작을 경우, 데드라인이 임박한 비긴급 태스크 중에서 전력소모량이 앞으로의 의무 작업들을 수행하기 위한 전력량을 해치지 않는 태스크가 QA에 저장되어 하위 스케쥴러로 전달된다. 세번째로 E_{budget}이 E_{limit}보다 작을 경우에는 QA의 태스크들이 임무 시간까지 수행되기 위한 전력량을 해치지 않을 만큼의 긴급 태스크를 데드라인이 임박한 순으로 QA에 포함시켜 하위 스케쥴러로 전달한다. 알고리즘2에서 sort(Q,A)는 큐 Q에 있는 모든 태스크를 A 규칙에 따라 정렬하는 함수이다. A는 LeastUtilizationFirst, LeastEDFFirst 등이 될 수 있다. move(T_i,Q)는 태스크 T_i를 큐 Q에 옮기는 함수이다. U는 모든 태스크의 계산 부하의 합, UT_i는 태스크 T_i의 계산 부하이고 UA는 하위 스케쥴러로 보내질 태스크가 저장되는 큐의 계산 부하를 나타낸다.

알고리즘 2. EMSRT-MK
1:At start, CalculateSpeed()
2:At boundary of each task T_k
3: T_k= getTask();
4: updateDist(T_k);
5: if (T_k is urgent) insert(T_k, QU);
6: else insert(T_k,QNU);
7: calculate E_{total};
8: calculate E_{limit};
9: calculate EA;
10: if(E_{total} $\leq$ E_{budget}) {
11: sort(QU, LeastUtilizationFirst);
12: sort(QNU, LeastDistanceFirst);
13: for($T_i \in$ QU)
14: if (UA+ UT_i $\leq$ M){
15: move($T_i \in$ QU, QA);
16: UA = UA + UT_i ;}
17: else break ;
18: for(T_i $\in$QNU)
19: if (UA+ UT_i $\leq$ M){
20: move($T_i \in$ QNU, QA);
21: UA =UA+ UT_i ;}
22: }
23: else if(E_{limit} $\leq$$E_{budget}$){
24: sort(QNU, LeastEDFirst);
25: sort(QU, LeastEDFirst);
26: for($T_i \in$ QU)
27: if(UA + UT_i $\leq$M){
28: move($T_i \in$ QU, QA);
29: UA = UA + UT_i ;
30: }
31: E_{temp} = EA;
32: if(QU ==NULL){
33: for($T_i \in$ QNU){
34: if(E_{limit} +E_{temp}+ET_i $\leq$$E_{budget}$ &&
UA+$UT_i$$\leq$M){

```
35:             move(Ti∈ QNU, QA);
36:             Etemp=Etemp+ETi ;
37:             UA =UA+ UTi ;}
38:         else break;
39:         }
40:    }
41: }
42: else if(Elimit>Ebudget){
43:    sort(QU, LeastEDFirst);
44:    Etemp = EA;
45:    for(Ti∈ QU){
46:      if( Etemp + ETi ≤ Ebudget &&
          UTi+UA≤M){
47:          move(Ti∈ QU, QA);
48:          Etemp = ETi+Etemp ;
49:          UA = UA + UTi ;}
50:      else break;
     }
51: }
52:At every quantum
53:    Pfair(QA);
```

5. 실험 및 결과

성능 평가를 위해 EMRTS-MK는 Linux 기반의 커널 레벨 실시간 스케쥴링 프레임워크인 LITMUS에 구현되었다 [12]. OS는 Linux 2.6.32를, 멀티프로세서는 800, 1200, 1600, 1800, 2100MHz 다섯 단계의 프로세서 속도를 제공하는 AMD 12 코어 프로세서를 사용했다. 실시간 태스크의 주기는 1~20 msec 구간에서 임의로 선택하였고 각 태스크의 최악 실행 시간 또한 임의로 선택하여 전체 시스템의 계산 부하를 조정하였다. 임무 시간은 실험의 편의를 위해 모든 경우 300 sec로 고정하였다. 각 태스크들의 (m,k)-firm 데드라인은 (2,3)-firm 데드라인으로 고정하였다. 이때 계산 부하는 4~14의 범위를 갖도록 변화시켜 12 프로세서에서 저부하와 과부하 상태 모두를 포함하도록 했다. E_{budget}은 150~20% 범위에서 실험하여 시스템의 전력량이 E_{total}보다 큰 경우부터 E_{limit}보다 작은 경우를 모두 포함하도록 했다. 특히 본 실험 태스크들은 (2,3)-firm 데드라인을 가지므로 모든 실험에서 E_{total}은 E_{limit}의 150%에 해당한다. 또한 전력량 측정은 파워미터(WT-210, Yokogawa)를 사용하였고, 매 실험은 10번씩 수행하여 그 평균값을 구하였다.

Fig. 1과 2는 계산 부하가 8일 때 DSR과 DFR를 나타낸 것이다. E_{budget} (0)이 E_{limit}보다 클 경우 DFR이 0으로 EMRTS-MK는 (m,k)-firm 데드라인 태스크들의 최소 QoS를 보장함을 의미한다. 또 E_{budget} (0)이 E_{limit}보다 작을 경우에도 DFR은 급격하지 않게 증가하는데, 이는 주어진 전력량이 부족하더라도 (m,k)-firm 데드라인을 최대한 만족시키고자 하는 EMRTS-MK의 특성이다. 한편 DSR은 E_{budget}가 감소함에 따라 서서히 감소하는데 이는 EMRTS-MK가 주어진 전력량 안에서 QoS를 최대화함을 의미한다.

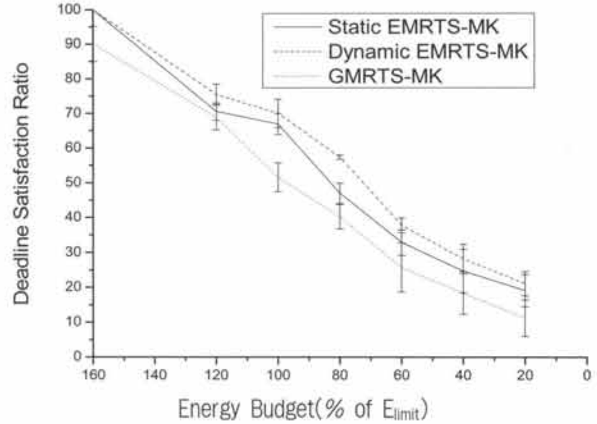

Fig. 1. DSR, when total utilization is 8

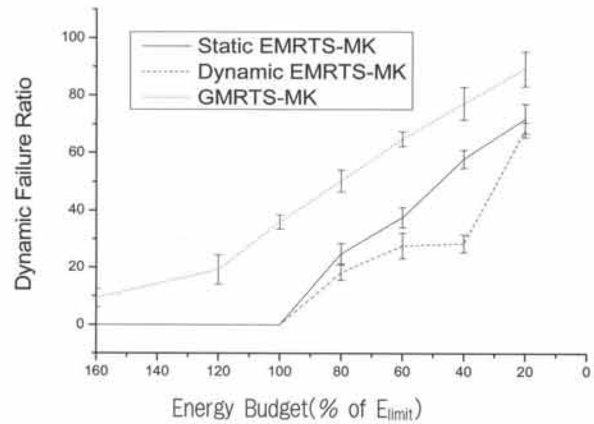

Fig. 2. DFR, when total utilization is 8

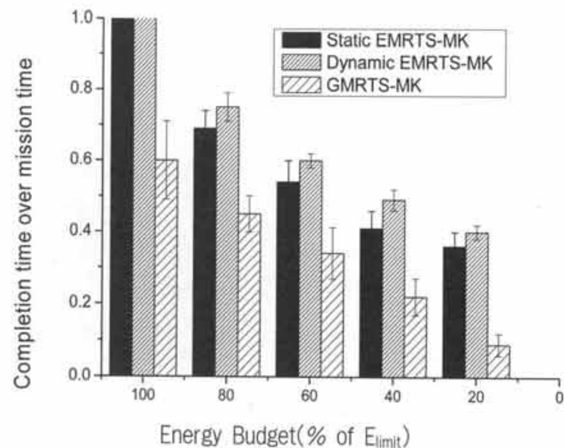

Fig. 3. Completion time over mission time, when total utilization is 8

Fig. 3는 임무 시간 대비 수행 시간을 비율로 임무 시간 달성율을 보여준다. 계산 부하가 8, 즉 저부하 상태이고 주어진 전력량이 E_{limit}보다 적을 경우에 EMRTS-MK가 GMRTS-MK보다 임무 시간에 가까운 수행시간을 가짐을 알 수 있다.

또 Fig. 4과 5는 계산 부하가 14, 과부하 상태일 때 DSR과 DFR를 나타낸 것으로, EMRTS-MK는 과부하 상태에서도 DSR을 최대화하고, DFR를 최소화함을 보이고 있다.

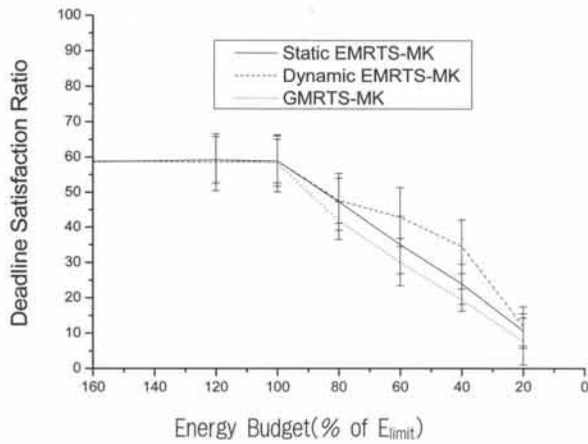

Fig. 4. DSR, when total utilization is 14

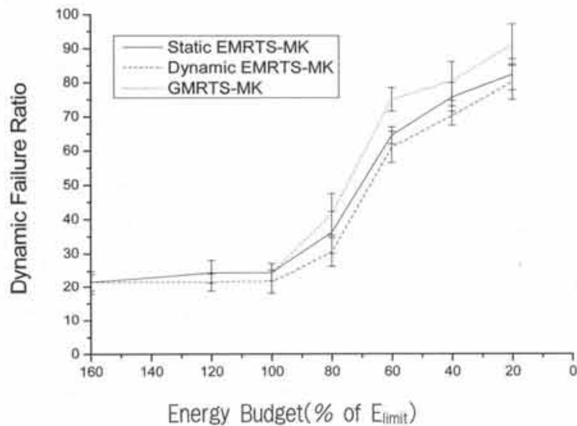

Fig. 5. DFR, when total utilization is 14

Fig. 6는 EMRTS-MK와 GMRTS-MK의 실제 전력 소모를 임무 시간 동안 비교 측정한 것이다. EMRTS-MK는 GMRTS-MK에 비해 동적 전력 소모 (=전체 전력 소모 - 프로세서 휴지(Idle) 시간 전력 소모) 면에서 저부하의 경우 이득이 있음을 알 수 있다. 이는 저부하일 때 프로세서의 속도를 좀더 느리게 할 수 있어 높은 전력 이득을 보기 때문이다.

Fig. 7은 GMRTS-MK 소모 전력량 대비 EMRTS-MK의 전력 소모 비율을 나타낸 그래프이다. 이에 따르면 EMRTS-MK는 GMRTS-MK에 비해 최대 63%까지 전력 소모를 줄일 수 있다.

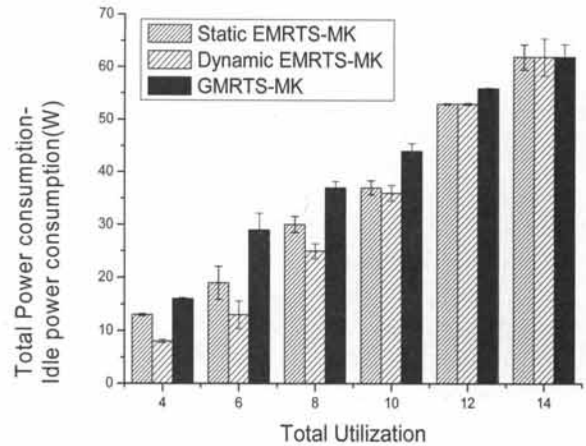

Fig. 6. Total power consumption-idle power consumption, when energy budget is 80% of E_{limit}

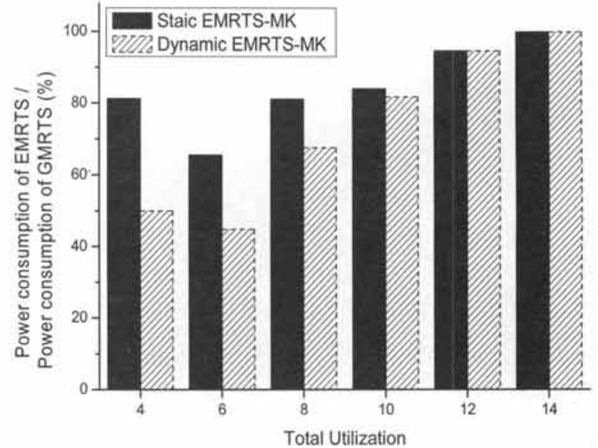

Fig. 7. Power consumption of EMRTS / GMRTS (%), when energy budget is 80% of E_{limit}

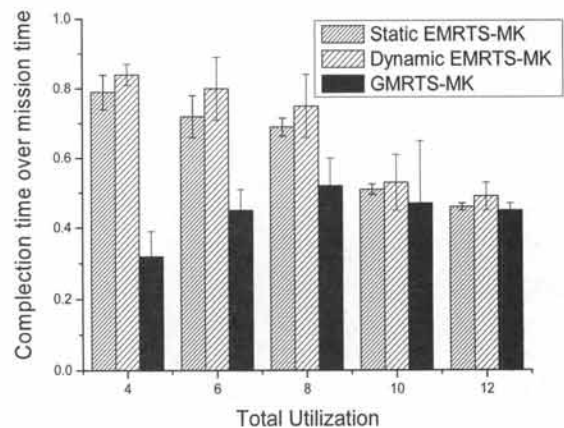

Fig. 8. Completion time over mission time, when energy budget is 80% of E_{limit}

Fig. 8은 $E_{budget}(0)$이 E_{limit}의 80%일 때 계산 부하에 따른 임무 시간 달성률로 EMRTS-MK는 $E_{budget}(0)$이 모든 의무

작업들을 실행하기에 부족한 전력량을 가지고 있음에도 불구하고 GMRTS-MK에 비해 임무 시간에 가깝게 시스템을 운영함을 보인다. 또 계산 부하가 낮을수록 동적 EMRTS-MK와 정적 EMRTS-MK가 GMRTS-MK보다 임무 시간에 근접하게 도달한다는 것을 확인할 수 있는데 이는 계산 부하가 작을 때 프로세서 속도를 조정하여 전력 효율을 크게 높일 수 있기 때문이다. 반면 GMRTS-MK는 계산 부하가 작을 때 QoS만을 높이기 위해 DSR을 높이게 되고 이에 따라 임무시간은 오히려 만족을 못 시키는 현상을 보인다.

Fig. 9은 EMRTS-MK의 처리 오버헤드(Overhead)를 측정한 값으로 태스크가 증가하면 오버헤드가 늘어나는 것을 알 수 있다. 또 단일 수행 시 상위 스케쥴러의 오버헤드가 하위 스케쥴러의 오버헤드보다 크고, 수행 빈도를 고려하면 평균적으로 오버헤드가 비슷함을 확인할 수 있다.

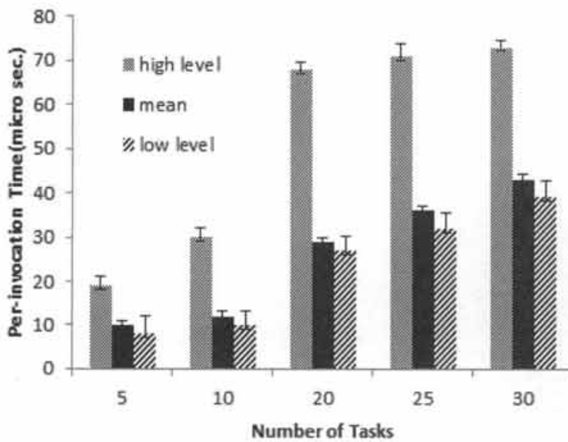

Fig. 9. Per-invocation time overhead

6. 결론 및 향후 연구

EMRTS-MK는 내장형 시스템의 한정된 전력량을 고려하여 멀티미디어 디코딩 등 (m,k)-firm 데드라인 제약을 가지는 태스크들의 임무 시간 동안의 동작을 보장한다. 뿐만 아니라 전력량이 부족하거나 과부하 상태인 경우에도 향상된 QoS를 위해 DSR를 최대화하고 임무 시간을 최대한 만족시킨다. 즉, EMRTS-MK는 시스템이 가지고 있는 전력량에 따라서 최대한 (m,k)- firm 데드라인을 만족시키고 시스템이 임무 시간까지 동작하도록 설계되었다. 본 논문에서 정적, 그리고 동적 EMRTS-MK를 제안하고 12코어 멀티프로세서를 사용한 구현 실험으로 최소 QoS를 보장하고 기존 스케쥴링 알고리즘보다 향상된 QoS를 보여줌을 입증하였다.

참 고 문 헌

[1] M. Hamdaoui, P. Ramanathan, "A dynamic priority assignment technique for streams with (m,k)-firm deadlines," IEEE T. on Computers, pp.1443-1451, 1995.

[2] P. Ramanathan, "Overload management in real-time control applications using (m,k)-firm guarantee," IEEE T. on Parallel and Distributed Systems, pp.549-559, June, 1999.

[3] G. Quan, X. Hu, "Enhanced Fixed-priority scheduling with (m,k)-firm guarantee," IEEE RTSS, pp.79-88, 2000.

[4] G. Koren, D.Shasha, "Skip-over: Algorithm and complexity for overloaded systems that allow skips," RTSS, 1995.

[5] T. A. Alenawy, H. Aydin, "Energy-constrained scheduling for weakly-hard real-time systems," IEEE RTSS, 2005.

[6] T. Wu, S. Jin, "Weakly hard real-time scheduling algorithm for multimedia embedded system on multiprocessor platform," IEEE International Conference on Ubi-Media Computing, 2008.

[7] P.Pillai, K.G Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," ACM Symposium on Operating System Principles, 2002.

[8] S. Saewong, R. Rajkumar, "Practical Voltage-Scaling for Fixed-Priority Real-time Systems," IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03), May, 2003.

[9] Y. Kong, H. Cho, "Guaranteed Scheduling for (m,k)-firm Deadline-Constrained Real-time Tasks on Multiprocessors," PDCAT, Oct., 2011.

[10] S. K. Baruah, N. K. Cohen, C. G. Plaxton, D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation", Algorithmica, Vol.15, No.6, pp.600-625, 1996.

[11] G. Bernat, A. Burns, A. Llamosi. "Weakly-hard real-time systems," IEEE Trans. on Computers, April, 2001.

[12] J. Calandrino, H. Leontyev, A. Block, U. Devi, and J. Anderson, "LITMUSRT: A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers", IEEE RTSS, 2006.

[13] W. Shieh, B. Chen, "Energy-Efficient Tasks scheduling Algorithm for Dual-core Real-time Systems," Computer Symposium (ICS), 2010.

[14] H. Yu, B. Veeravalli, Y. Ha, "Leakage-Aware Dynamic Scheduling for Real-Time Adaptive Applications on Multiprocessor Systems," DAC, 2010.

[15] Y. Wang, H. Lku, D. Liu, E. H.-M. SHA, "Overhead-Aware Energy Optimization for Real-Time Streaming Applications on Multiprocessor System-on-Chip," ACM Trans. on Design Automation of Electronic Systems, Vol.16, issue 2, March, 2011.

공 연 화

e-mail : misticlime@naver.com
2010년 고려대학교 컴퓨터정보학과(학사)
2012년 고려대학교 전산과학 석사
관심분야 : 실시간 컴퓨팅, 멀티프로세서

조 현 중

e-mail : raycho@korea.ac.kr

1996년 경북대학교 전자공학과(학사)
1998년 포항공과대학교 전기전자공학과 (공학석사)
1998년~2003년 (주)삼성전자 선임연구원
2006년 버지니아공대 컴퓨터공학과 (공학박사)
2006년~2009년 한국전자통신연구원 선임연구원
2009년~현 재 고려대학교 컴퓨터정보학과 부교수
관심분야 : 실시간 시스템, 멀티프로세서 스케쥴링, HCI, 제스처 인식 등