실시간 태스크의 마감시간 만족을 위한 캐쉬 최적 분할 형태의 분석

김 명 희[†] · 주 수 종[††]

요 약

본 논문은 실시간 시스템에서 주기 및 비주기적 태스크들의 마감시간 만족을 위한 캐쉬(캐쉬 메모리)의 최적 분할 형태를 분석한다. 연구 목적은 태스크들의 가용비용을 최소화하여 태스크들의 마감시간 위반율을 줄일 뿐 아니라 캐쉬의 유휴공간을 다른 태스크에게 할당시키기 위함이다. 이를 위해 캐쉬의 분할공간에 태스크들을 할당시키기 위한 캐쉬 분할공간 할당 알고리즘을 제시한다. 여기에서 태스크들이 할당된 캐쉬 분할공간들의 집합을 캐쉬 분할 형태라고 한다. 태스크들이 분할공간에 어떻게 할당되는가에 따라 다양한 캐쉬 분할 형태들을 얻을 수 있다. 이러한 캐쉬 분할 형태들로부터 스케쥴링 가능한 태스크들의 가용비용의 한계범위와 태스크들이 최소 가용비용으로 실행할 수 있는 캐쉬의 최적 분할 형태를 분석한다.

An Analysis on The Optimal Partitioning Configuration of Cache for Meeting Deadlines of Real-Time Tasks

Myung-Hee Kim[†] · Su-Chong Joo[††]

ABSTRACT

This paper presents an analysis on the optimal partitioning configuration of cache(memory) for meeting deadlines of periodic and aperiodic real-time task set. Our goal is not only to decrease the deadline missing ratio of each task by minimizing the task utilization, but also to allocate another tasks to idle spaces of cache. For this reason, we suggest an algorithm so that tasks could be allocated to cache segments. Here, the set of cache segments allocated tasks is called a cache partitioning configuration. Based on how tasks allocate to cache segments, we can get various cache partitioning configurations. From these configurations, we obtain the boundary of task utilization that tasks are possible to schedule, and analyze the cache optimal partitioning configuration that can be executed to minimize the task utilization.

1. 서 론

실시간 태스크의 마감시간을 만족시키기 위한 연구로는 그 동안 많은 태스크 할당 및 스케쥴링 방법들이 개발되었다[1, 2, 9, 10, 11]. 특히 멀티미디어 실시간 처리 관점에서는 캐쉬 또는 주메모리에 처리할 해당 페이지들을 많이 올려놓으므로 써 페이지 폴트율을 줄인 결과로 태스크의 수행시간을 최소화하여야 할 필요가 있다. 그러나 캐쉬나 주메모리의 한정

※본 논문은 '96학년도 원광대학교 교내연구비의 지원에 의해 연구됨.
† 준 회 원:원광대학교 대학원 컴퓨터공학과
†† 정 회 원:원광대학교 컴퓨터공학과 부교수
논문접수:1997년 1월 23일, 심사완료:1997년 9월 22일

된 크기로 인해 요구 페이징(demand paging)이 불가피하다. 따라서 본 연구는 **캐쉬 분할공간(segment)**들을 마감시간이 촉박한 태스크들에게 **할당하여** 페이지 폴트로 인한 요구 페이징의 회수를 줄이도록 유도한다. 이로서 태스크의 가용비용(utilization)을 줄일 뿐 아니라, 태스크의 마감시간 위반율을 최소화하는 데 연구의 목적을 둔다.

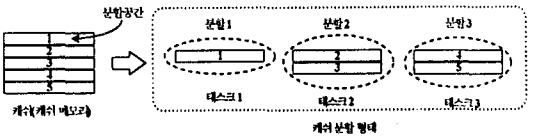
대부분 시스템에서는 캐쉬와 주메모리를 계층적으로 구성하고 있다. 이러한 구조의 이용은 시스템 성능과 고속 메모리의 고가에 대한 절충효과(trade-off)를 균형화 시킬 수 있을 뿐 아니라 실시간 환경에서 결정적(deterministic)성능과 선점(preemptive)의 멀티태스킹을 지원 할 수 있기 때문이다.

이러한 캐쉬(cache memory) 구조하에서, 본 논문은 실시간 멀티태스킹시 마감시간을 위반할 가능성이 큰 태스크에게 보다 많은 캐쉬 영역을 할당하기 위해서, 캐쉬를 일정한 크기로 분할(분할공간이라고 부름)하고, 지정된 분할공간들 내에 하나 또는 그 이상의 태스크가 수행될 수 있도록 보장한다. 본 논문에서 캐쉬에 대해 분할될 분할공간 수는 임의로 정한다. 이때 하나의 실시간 태스크가 할당될 하나 또는 그 이상의 분할공간 집합을 캐쉬 분할 형태(cache partitioning configuration)라고 한다. 태스크의 수행시간은 캐쉬 분할 형태에 따라 조절할 수 있도록 한다.

본 연구는 실시간 주기적 태스크와 비주기적 태스크들이 수행하는 경우에 대한 캐쉬의 분할 형태를 분석하고자 한다. 따라서 주기적 태스크들을 캐쉬 상에 할당할 때에는 동적 프로그램 알고리즘을 이용하여 얻은 결과인 논문[1]의 캐쉬 분할 형태를 이용한다. 그러나 비주기적 태스크들을 같이 고려했을 때는 주기적 태스크와 비주기적 태스크가 동종의 태스크가 아니므로 캐쉬의 분할 형태를 얻는데 논문 [1]의 방법을 사용할 수 없다. 왜냐하면, 주기적 태스크와 비주기적 태스크를 모두 고려한 경우, 태스크들의 가용비용들을 서로 다른 과정으로 얻어야 하기 때문이다. 즉, 주기적인 태스크는 각 주기마다 수행되어야 하므로 비주기적 태스크보다 높은 우선 순위가 부여되기 때문에 주기적 태스크의 가용비용과 비주기적 태스크의 가용비용은 별도로 계산되어야 하며, 두 형태의 태스크들을 모두 고려한 캐쉬 분할 형태는 이들을 조합하여 얻어야 한다. 연구의 결과를 도출하는 과정으로 실시간 태스크들이 할당될 수 있는 가능한 캐쉬의 분할 형태들을 분석하고 이들로부터 태스크들의 가용비용들을 얻는다. 캐쉬 분할 형태들에 대해서 Rate Monotonic 스케줄링 방법[2]을 이용하여 이들의 스케줄링 가능한 가용비용의 한계범위와 분석된 캐쉬 분할 형태들 중에서 최소 가용비용을 갖는 캐쉬 최적 분할 형태를 보인다.

2. 캐쉬 할당 배경

캐쉬는 모두 같은 크기의 분할공간들(segments)로 나누어, 이 곳을 소유한 태스크에게만 접근할 수 있도록 허용한다. 이때 하나의 태스크가 할당될 분할들(partitions)의 집합을 캐쉬의 분할 형태라 하며 (그림 1)과 같이 나타낸다. 아래 그림에서, 태스크 2는 2개의 분할공간들로 구성된 분할 2에 할당됨을 나타낸다.



(그림 1) 캐쉬 분할 형태의 예
(Fig. 1) The Example of Cache Partitioning Configuration

주기 및 비주기적 태스크들을 최적으로 할당하는 캐쉬 최적 분할 형태를 얻기 위해서는, 수행되는 모든 태스크들이 캐쉬 상에서 수행될 때의 최소 가용비용이 계산되어야 한다[3, 4]. 위 가용비용은 태스크 각각의 수행비용을 이용하여 얻어진다. 이를 근거로 하여 최소의 태스크 가용비용을 갖는 캐쉬 최적 분할 형태를 찾게되는데, 이러한 과정은 최적화에 의해 얻어진다. 따라서 최적화 과정에 의해 모든 태스크들이 마감시간 내에 수행을 완료할 수 있도록 스케줄링 가능성이 보장되며, 최소의 태스크 가용 비용을 갖는 캐쉬 최적 분할 형태를 얻을 수 있다.

2.1 태스크의 수행비용

주기 및 비주기적 태스크의 특성은 주기, 상대주기, 부하량으로 정의할 수 있다. 주기적 태스크에 대한 상대주기는 각 태스크의 주기에 대해 주기가 가장 큰

태스크의 주기를 최소공배수로 만들기 위한 곱의 수이며, 비주기적 태스크는 주기를 갖지 못하므로 상대주기를 1로 정의하고 시작시간과 마감시간만을 기술한다. 부하량은 각 태스크의 캐쉬 접근에 필요한 캐쉬 참조 회수(cycles)로 나타낸다. 각 태스크의 수행비용은 페이지 폴트가 많음에 따라 비례적으로 커지게 된다.

본 논문에서 실행되는 대상인 태스크들은 주기 및 비주기적 태스크들이다. 주기적 태스크들 $\tau_1, \tau_2,..., \tau_n$은 최악의 수행비용들 $C_1, C_2,..., C_n$과 주기들 $T_1, T_2,..., T_n$을 갖고, 비주기적 태스크들 $\tau'_1, \tau'_2,..., \tau'_m$은 최악의 수행비용들 $C'_1, C'_2,..., C'_1$과 비주기적 태스크의 시작시간 $A_1, A_2,..., A_m$과 마감시간 $D_1, D_2,..., D_m$을 갖는다. 주기적 태스크들의 마감시간은 그들의 다음 주기가 시작되는 시간이 되며, 비주기적 태스크들의 수행 시간은 마감시간에서 시작시간을 뺀 시간 $(D_i - Ar_i,\ i=1, 2,..., m)$으로서 나타낸다. 다음 〈표 1〉은 본 연구에서 사용하는 주기 및 비주기적 태스크들의 정의를 나타낸다.

〈표 1〉 주기 및 비주기적 태스크들의 정의
〈Table 1〉 The Definition of Periodic and Aperiodic Task Set

주기적 태스크	주기 cycles(T)	상대 주기	부하량(memory operations)
τ_1	100	14	10
τ_2	140	10	18
τ_3	1400	1	60

비주기적 태스크	시작시간(Ar)/ 마감시간(D)	상대 주기	부하량(memory operations)
τ'_1	500/700	1	20
τ'_2	1000/1100	1	8

또한 하나의 τ_i 또는 τ'_i가 할당될 캐쉬 분할 형태를 P라 한다면, P는 하나 또는 그 이상의 분할공간(segment)들의 집합으로 이루어진다. 예로서 τ_i가 분할공간 j개에 할당된 캐쉬 분할 형태를 P_{ij}라고 정의한다. 만일 분할 형태 P_{10}이 있다면 이는 τ_i가 캐쉬 상에 전혀 올라 있지 못한 상태이고 P_{23}이라면 태스크 τ_2가 캐쉬 상에 3개 분할공간들을 할당받고 있으며 τ_2의 내용중 3개 분할공간 크기 양만큼이 캐쉬에 올려져 있음을 나타낸다. 분할 형태 P10에 대해 다시 말하면, 태스크 τ_1은 분할공간을 전혀 할당받지 못한 상태이므로 페이지 요청시 페이지 폴트가 되고, 필요한 페이지들을 주기억장치 또는 보조기억장치에서 스워핑(swapping)해야한다. 반면에 캐쉬 분할 형태 P_{23}은 태스크 τ_2의 실행을 위해 수행에 필요한 일부 페이지들이 3 분할공간들 상에 올려져 있음을 나타낸다. 따라서 태스크에 대한 페이지가 분할공간 상에 올려져 있는 양에 비례해서 페이지 폴트율은 줄어들므로 임의의 태스크를 많은 분할공간상에 할당되도록 한다면, 태스크의 실행시간을 보다 더 줄일 수 있다. 이에 대한 결과는 캐쉬의 가용성을 늘리는 효과를 갖는다.

각 태스크의 수행비용은 태스크가 할당될 분할공간들의 수에 의존하게 된다. 즉 적은 양의 분할공간 수를 할당받은 경우, 페이지 폴트율은 상대적으로 높아지며 태스크의 수행비용도 커진다. 각 태스크의 부하(workload)는 메모리 명령수(memory operations)로 정의하고, 태스크의 수행비용은 메모리 명령수 * 페이지 폴트(fault)나 히트(hit)시 요구되는 사이클의 수로 계산한다. 원하는 페이지가 할당된 분할공간들 상에 이미 존재(hiting)한다면, 이를 실행하기 위해서 단지 1 사이클만이 필요하다고 가정하고, 태스크의 수행비용은 태스크의 부하량에 1 사이클을 곱해서 계산한다. 그러나 주어진 분할공간상에 필요한 페이지가 없다면(page faulting), 이를 분할공간 상으로 스워핑하는데 4 사이클이 요구된다고 가정하고, 태스크 부하 값에 4 사이클을 곱한다. 예로서 부하량이 10인 태스크 τ_i가 임의의 캐쉬 분할 형태에 할당(P_{ij})되었을 때, 페이지 폴트율을 0.2라 한다면, 태스크의 부하량 10의 20%는 페이지 스워핑을 위해 각각 4 사이클이 요구되고, 나머지 80%는 hiting상태이므로 1사이클만을 요구한다. 따라서 이 태스크의 수행비용은 10*0.2*4+10*0.8*1=16이 된다. 태스크의 부하량이 주어질 때, 각 할당되는 분할공간 수에 따른 태스크의 수행시간 C=(태스크의 부하량*페이지 폴트율)*4+(태스크의 부하량*hiting율)*1이다. 여기서 hiting율은 (1-페이지 폴트율)이며, 페이지 폴트율은 하나의 태스크가 할당받는 분할공간 수, 즉 캐쉬 분할 형태에 따라서 가변적이다. 이 페이지 폴트율은 실제 태

스크의 수행 시간에 영향을 주게 된다.

캐쉬가 6개의 분할공간들로 나뉘어졌을 때, 〈표 1〉에서 정의된 태스크들의 수행비용은 아래 〈표 2〉와 같다. 여기에서 주기 및 비주기적 태스크가 수행되기 위해 할당받을 수 있는 분할공간의 수가 많을수록 페이지 폴트율이 작아지므로 태스크의 수행비용도 적어짐을 알 수 있다. 예로서 〈표 2〉에서 주기적 태스크 τ_1이 seg(0), 즉 캐쉬 분할공간를 전혀 할당받지 못한 경우, 페이지 스워핑이 이루어져야 하기 때문에 (부하량*4 사이클)이 요구되어 수행비용은 40인데 반해, seg(3), 즉 3개의 캐쉬 분할공간를 할당받아서 수행하는 경우, 페이지 폴트율이 더욱 줄어들므로 써 태스크의 수행시간은 32가 된다. 여기에서 8 시간단위의 차는 주기적 태스크 τ_1에 할당되는 분할공간의 페이지 폴트율이 반영된 수행비용으로, seg(0)보다 seg(3)이 8만큼의 시간 단위가 절약됨을 나타낸다.

〈표 2〉 주기 및 비주기적 태스크들의 수행비용
〈Table 2〉 The Execution Time of Periodic and Aperiodic Tasks

주기적 태스크	seg (0)	seg (1)	seg (2)	seg (3)	seg (4)	seg (5)	seg (6)
τ_1	40	35	34	32	31	31	30
τ_2	72	63	61	58	57	55	54
τ_3	240	213	204	195	191	186	182

비주기적 태스크	seg (0)	seg (1)	seg (2)	seg (3)	seg (4)	seg (5)	seg (6)
τ'_1	80	71	68	65	63	62	60
τ'_2	32	28	27	26	25	24	24

2.2 태스크의 가용비용

태스크의 가용비용(utilization)은 앞절에서 다룬 태스크의 수행비용과는 달리 주기적 태스크와 비주기적 태스크들에 대해서 각각 다른 과정으로 산출되어야 한다.

먼저, 주기적 태스크 τ_i에 대한 가용비용은 $U_i = C_i / T_i$이다. 여기에서 C_i는 τ_i의 수행비용을, T_i는 τ_i의 주기를 나타내며 $i = 1, 2, ..., n$이다. 이를 이용하여 주기적 태스크들과 비주기적 태스크들의 모두에 대한 총 가용비용은 아래 식 (1)과 같다.

$$U_{total} = \sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{k=1}^{m} \frac{C'_k}{D_k - Ar_k} \quad (1)$$

이 식에서 첫째 항은 주기적 태스크들의 총 가용비용을 나타내며, 두 번째 항은 비주기적 태스크들의 총 가용비용을 나타낸다. 여기에서 C'_k는 비주기적 태스크 τ'_k의 수행비용이며, D_k와 Ar_k는 각각 τ'_k의 마감시간과 도착시간을 나타내며 k = 1, 2,..., m이다. 프로세서 관점에서 태스크의 수행시간(C)은 세부적으로 두 개의 시간요소로 나눌 수 있는데, 한가지 시간요소인 C_{ideal}은 프로세서에 의해 캐쉬로부터 태스크에 속한 명령어나 데이터들을 불러오는데 요구되는 시간이고, 또 다른 시간요소인 $C_{waiting}$은 페이지 폴트로 인하여 주메모리로부터 캐쉬로 태스크의 내용을 가져온 후, 엑세스를 하기까지의 시간이다. 식 (1)에 C_{ideal}과 $C_{waiting}$을 대입하면 식 (2)와 같다.

$$\begin{aligned} U_{total} &= \sum_{i=1}^{n} \frac{C_{ideal_i} + C_{waiting_i}}{T_i} \\ &\quad \sum_{i=1}^{m} \frac{C'_{ideal_k} + C'_{waiting_k}}{D_k - Ar_k} \\ &= U_{total-ideal} + U_{total-waiting} \end{aligned} \quad (2)$$

위 식 (2)로부터 태스크의 총 가용비용(U_{total})을 최소화하기 위해서는 $U_{total-waiting}$을 가능한 줄이도록 해야 할 것이다. 결국 페이지 폴트가 적게 일어나도록 각 태스크에게 분할공간들을 더 많이 할당해야한다. 여러 개의 태스크가 동시 수행되고 있다면, 마감시간이 가까운 태스크에게 분할공간들을 많이 할당해 주므로 써 $U_{total-waiting}$을 감소시키도록 한다. 주기적 및 비주기적 태스크 모두를 고려한 경우, $U_{total-waiting}$ 산출식은 식 (3)과 같다.

$$U_{total-waiting} = \sum_{i=1}^{n} \frac{C_{waiting_i}}{T_i} + \sum_{i=1}^{m} \frac{C'_{waiting_k}}{D_k - Ar_k} \quad (3)$$

다음으로는 태스크들과 이들이 할당될 분할공간들을 고려하여 태스크의 최소 가용비용을 얻는 과정을 살펴보면, 태스크를 할당한 캐쉬 분할 형태(즉, 한 개 이상의 고정크기로 나뉜 분할공간들의 집합)는 몇 개의 분할공간으로 구성되느냐에 따라 다양하다. 캐쉬

분할 형태에 따라 태스크의 $U_{total-waiting}$의 비용도 다르게 된다.

〈표 2〉에서와 같이, 태스크 수와 각 태스크가 할당된 분할공간 수의 조합으로 캐쉬 분할 형태를 만들 수 있다. 만일 태스크 τ_i가 j개의 분할공간들로 구성된 캐쉬 분할 형태(P_{ij})에서 실행될 때, 태스크 τ_i의 가용비용은 $U_{i,j}$라고 한다면 〈표 1〉과 〈표 2〉를 이용하여 계산한 주어진 태스크들의 가용비용은 〈표 3〉과 같다. 하나의 예로 τ_2가 3개의 분할공간를 할당받아, 즉 분할 형태 P_{23}, 수행될 때 τ_2의 가용비용은 0.42이다.

〈표 3〉 태스크들에 대한 분할공간의 가용비용
〈Table 3〉 The Utilizations of the Segment for Task Sets

주기적 태스크	seg (0)	seg (1)	seg (2)	seg (3)	seg (4)	seg (5)	seg (6)
τ_1	0.40	0.35	0.34	0.32	0.32	0.31	0.30
τ_2	0.51	0.46	0.44	0.42	0.41	0.40	0.39
τ_3	0.17	0.15	0.15	0.14	0.14	0.13	0.13

주기적 태스크	seg (0)	seg (1)	seg (2)	seg (3)	seg (4)	seg (5)	seg (6)
τ'_1	0.40	0.35	0.34	0.32	0.32	0.31	0.30
τ'_2	0.32	0.28	0.27	0.26	0.26	0.25	0.24

각 캐쉬 분할 형태에서 태스크들이 할당되어 실행될 때 태스크들의 총 가용비용은 아래 식 (4)와 같이 얻을 수 있다.

$$U_{total}=\sum_{i=1}^{n} U_{i,a_i}+\sum_{k=1}^{m} U'_{k,a'_k} \tag{4}$$

여기에서 a_i(또는 a'_k)는 τ_i(또는 τ'_k)에 할당되는 분할공간의 수이다. 이는 캐쉬 분할 형태 P_{ij}에서 분할공간 수를 나타내는 j의 의미 첨자들의 혼돈을 피하기 위해서 정의한 매개변수이다. 총 분할공간의 수를 S라 할 때 $S=\sum_{i=1}^{n} a_i+\sum_{k=1}^{m} a'_i$이다. 캐쉬에 할당된 n개의 주기적 태스크들과 m개의 비주기적 태스크들이 갖는 가용비용들로부터 최소의 총 가용비용은 아래 식 (5)와 같이 얻을 수 있다.

$$\min(U_{total})=\min_{(a\in A)}\left(\sum_{i=1}^{n} U_{i,a_i}\right)+\min_{(a'\in A')}\left(\sum_{k=1}^{m} U'_{k,a'_k}\right) \tag{5}$$

여기에서 총 분할공간의 수를 S라 할 때, A는 모든 주기적 태스크들이 할당 가능한 분할공간수의 집합이고 A′는 S-A로써 비주기적 태스크들이 할당 가능한 분할공간들의 집합을 나타낸다. 태스크의 최소 가용비용을 갖는 캐쉬 분할 형태가 최적 분할 형태가 된다.

위 식 우변의 첫째 항은 주기적 태스크들이 $A \geq \sum_{i=1,n} a_i$개의 분할공간를 할당받아 실행할 때의 캐쉬의 최소 가용비용을 나타내며, 둘째 항은 비주기적 태스크들이 $A' \geq \sum_{k=1,m} a_k$개의 분할공간를 할당받아 실행할 때의 최소 가용비용을 나타낸다.

3. 캐쉬 분할 형태

3.1 주기적 태스크의 캐쉬 분할 형태

S개의 분할공간들로 분할된 캐쉬 상에서 주기적 태스크들의 각각의 가용비용 즉, 태스크 τ_i가 j개의 분할공간를 할당받아 실행할 때 태스크의 가용비용을 $U_{i,j}$라고 한다면, τ_i에 할당될 수 있는 분할공간 수가 많을수록, 즉 j값이 커질수록 $U_{i,j}$는 작아짐을 〈표 3〉에서 보았다. 주기적 태스크 τ_i(i = 1, 2,..., n)에게 할당 가능한 j개의 분할공간을 최적 분할하여 실행시킬 때, 이때의 최소 가용비용을 $M_{i,j}$라 하고 식 (6)과 같이 나타낸다. 여기에서 a는 각 태스크가 할당받을 수 있는 분할공간의 수이며, 주기적 태스크 τ_i에게 할당 가능한 분할공간의 총 수인 A개를 초과할 수 없다.

$$M_{i,j}=\min_{a\in A}\left(\sum_{t=1}^{i} U_{t,a_t}\right), \text{ for } \sum_{t=1}^{i} a_t \leq A \tag{6}$$

위 식에 의해 모든 주기적 태스크들을 고려한 최소의 총 가용비용 $M_{n,S}$를 얻을 수 있다. 다음은 각 태스크가 할당 될 최적 캐쉬 분할 형태(하나의 태스크가 몇 개의 분할공간들을 할당받아야 할 것인가)를 찾기 위해, 태스크 τ_i가 임의의 분할공간를 할당받았을 때의 가용비용과 τ_i가 아닌 태스크가 나머지 분할공간들을 할당받아 수행했을 때의 가용비용을 각각 계산

하여 가장 작은 값을 $M_{i,j}$로 취한다. 이 때 τ_i에 할당된 분할공간의 수 $P_{i,j}$는 j와 같거나 적어야 한다. 이 때 차가 되는 분할공간양은 비주기적 태스크를 위해 할당될 것이다. $M_{i,j}$를 얻는 과정은 다음 식 (7)과 같다.

$$j=\sum_{t=1}^{i} a_t = a_i + \sum_{t=1}^{i-1} a_t$$

$$\begin{aligned} M_{i,j} &= \min_{a\in A}\left(\sum_{t=1}^{i} U_t, a_t\right) \\ &= \min_{a\in A}\left(U_{i,a_i} + \sum_{t=1}^{i-1} U_{t,a_t}\right) \\ &= \min_{a_i\in\Phi_j}\left(U_{i,a_i} + \min_{b\in A}\left(\sum_{t=1}^{i-1} U_{t,a_t}\right)\right) \\ &= \min_{a_i\in\Phi_j}\left(U_{i,a_i} + M_{i-1,j-a_i}\right) \end{aligned} \quad (7)$$

여기에서 $M_{0,j}=0$이 된다. 왜냐하면 $i=0$이라는 것은 태스크가 없다는 의미이므로 가용비용을 나타낼 수 없다. 그리고 $\Phi_j=\{0, 1, 2,..., j\}$는 캐쉬 분할공간들의 수를 나타내며, j=3인 경우 $\Phi_3=\{0, 1, 2, 3\}$으로 임의의 태스크는 최대 3개까지 분할공간들을 할당받을 수 있음을 나타낸다. 또한 $P_{i,j}$를 이용해 첫 번째 태스크 τ_i에 대한 최적 할당을 한 후에 나머지 분할공간들을 $i-1$개의 남은 태스크들에게 차례로 할당을 해나간다.

다음 과정은 각각의 태스크에 대해 할당하게 될 분할공간의 수와 가용비용을 얻는 과정을 나타낸다.

```
for (i = 1 to n) {
    for (j = 0 to S) {
      M_{i,j} = min_{a_i ∈ Φ_j}(U_{i,a_i} + M_{i-1,j-a_i});
      P_{i,j};
      output(M_{i,j}, P_{i,j});
}     }
```

$M_{i,j}$와 $P_{i,j}$값의 계산이 끝나면, $M_{n,S}$은 주기적 태스크들의 최소 가용비용을 갖게되고, $P_{n,S}$로부터 n번째 태스크가 갖는 최적 할당 분할공간들의 수를 얻게 된다. 〈표 3〉을 이용하여 각 태스크와 할당되는 분할공간을 고려하여 각각 최소 가용비용을 구하면 〈표 4〉와 같다. 여기에서 ( )내의 숫자는 해당 태스크가 할당받을 수 있는 분할공간들의 수이다. 〈표 4〉의 각 최소 가용비용의 계산과정을 예로 들면, τ_3가 6개의 분할공간으로 나뉘어진 캐쉬 상에서 수행될 때의 최소 가용비용 $M_{3,6}$과 이때의 할당받을 분할공간의 수 $P_{3,6}$은

$$\begin{aligned} M_{3,6} &= \min_{a_i\in\{0,1,2,3,4,5,6\}}(U_{3,a_i} + M_{2,6-a_i}) \\ &= \min_{a_i\in\{0,1,2,3,4,5,6\}}(U_{3,0} + M_{2,6}, \\ &\quad U_{3,1} + M_{2,5}, U_{3,2} + M_{2,4}, U_{3,3} + M_{2,3}, \\ &\quad U_{3,4} + M_{2,2}, U_{3,5} + M_{2,1}, U_{3,6} + M_{2,0}) \\ &= \min(0.92, 0.91, 0.92, 0.93, \\ &\qquad 0.94, 0.98, 1.04) \\ &= 0.91 \end{aligned}$$

$P_{3,6}=1$과 같이 계산된다.

〈표 4〉 주기적 태스크들의 캐쉬 최소 가용 비용
〈Table 4〉 The Minimum Utilization of the Periodic Task Set

주기적 태스크	seg (0)	seg (1)	seg (2)	seg (3)	seg (4)	seg (5)	seg (6)
τ_1	0.40 (0)	0.35 (1)	0.34 (2)	0.32 (3)	0.32 (4)	0.31 (5)	0.30 (6)
τ_2	0.91 (0)	0.86 (1)	0.81 (1)	0.79 (2)	0.77 (3)	0.76 (3)	0.74 (3)
τ_3	1.09 (0)	1.03 (0)	0.98 (0)	0.96 (0)	0.94 (0)	0.92 (0)	0.91 (1)

위 〈표 4〉에 의해 $M_{3,6}$ 값은 3개의 주기적 태스크에 대한 최소 가용 비용으로 0.91이다. 이 최소 가용비용을 토대로 $P_{i,j}$ 값에 따라 각각의 태스크를 다음과 같이 주어진 분할공간내에 할당한다. 〈표 4〉를 참조하여 분할공간을 할당하게 되는데 이에 대한 과정을 보면, 처음에 태스크 τ_3이 할당받을 수 있는 분할공간은 6개이다. 그 중 τ_3은 1개의 분할공간만을 할당받아도 최소의 가용비용(0.91)을 얻을 수 있다. 나머지 5개의 분할공간을 가지고 τ_2에게 할당했을 때 3개의 분할공간만을 할당받아도 최소 가용비용(0.74)를 얻을 수 있다. 마지막으로 τ_1에게 나머지 2개의 분할공간을 할당하게 된다. 다시 말해서, τ_1, τ_2와 τ_3 태스크에 대한 캐쉬 분할 형태들은 P_{12}, P_{23}, P_{31}이고, 이들은 각각 2개, 3개, 1개의 분할공간들로 이루어진다. 이 캐쉬 분할 형태에서의 각 태스크들에 대한 가용비용의 합은 모든 다른 캐쉬 분할 형태에서 얻어진 가

용비용보다 작은 값이 되어 이들이 최적 캐쉬 분할 형태가 된다. 결국 이 결과 값대로 τ_1에게 2개, τ_2에 3개, τ_3에 1개의 분할공간을 각각 할당시켰을 때 태스크 스케쥴링이 보장될 수 있다.

위에서 설명한 태스크들을 분할공간들 상에 할당하는 과정은 (그림 2)와 같다.

```
allo = S;
for ( i = n ; i = 1 ; i-- ) {
  output( i , P_{i,alloc} ) ;
        /* 각 주기적 태스크의 최소 가용비용을
        얻기위해 할당되는 분할공간들의 수*/
  allo  -= P_{i,alloc};
}
```

(그림 2) 주기적 태스크들의 분할공간 할당 과정
(Fig. 2) The Progress of the Segment allocation of Periodic Task Set

각 태스크가 (그림 2)에 따라 분할공간들을 최적으로 할당받은 후, 주기적 태스크들에 대해 스케쥴링이 가능한지를 검사한다. 본 논문에서는 비주기적 태스크를 같이 고려해야하므로 주기적 태스크들에 대한 스케쥴링이 가능한 분할공간 수를 제한시킬 필요가 있다. 따라서 여분의 분할공간을 비주기적 태스크에게 할당한다. 이 문제를 고려하기 위하여 비주기적 태스크에 대한 $M'_{i,j}$, $P'_{i,j}$가 필요하다. 여기에서 $M'_{i,j}$는 여분의 분할공간들을 할당받은 비주기적 태스크 τ'_i의 가용비용이며, $P'_{i,j}$는 τ'_i에게 할당될 분할공간들의 수이다. 비주기적 태스크들은 주기적 태스크와 달리 독립적으로 실행시켜야 하므로 〈표 3〉의 비주기적 태스크들로부터 $M'_{i,j}$, $P'_{i,j}$를 얻는다.

일반적으로 주기적 태스크의 우선 순위가 비주기적 태스크의 우선 순위보다 높다고 가정하므로, 주기적 태스크가 마감시간을 만족하지 못하면 비주기적 태스크에 대한 분할공간 할당과정은 무의미하다. 주기적 태스크들에게 할당된 분할공간들의 구조 즉, 캐쉬 분할 형태에서 스케쥴링이 가능하면 스케쥴링이 가능한 캐쉬 분할 형태가 된다. 만일 주기적 태스크에 대한 분할 형태가 스케쥴링이 가능하지 않으면, 새로운 캐쉬 분할 형태를 찾기 위하여 분할공간 재할당과 재스케쥴링이 요구된다. 이때 재할당 과정은 $M_{n,S}$ 값의 계산 방법을 이용한다.

이미 언급한바와 같이 $M_{n,S}$ 값은 $a_{n\in\Phi}(U_{N,a_i}+M_{N-1,S-a_i})$값들 중에서 가장 작은 값으로 선정하게 된다. 이 값들을 오름차순으로 정렬을 해서 작은 값부터 $M_{n,S}$ 값으로 선정하여 재할당이 요구될 때마다 위 값들을 순서적으로 사용한다. 물론 이때 $M_{n,S}$ 값을 변경함에 따라 $P_{n,S}$도 변경된다. 주기적 태스크들의 스케쥴링 가능 여부에 따라 캐쉬 분할 형태들을 재 선정하는 과정은 다음 (그림 3)의 알고리즘과 같다.

3.2 비주기적 태스크의 캐쉬 할당 방법

주기적 태스크들의 분할공간 할당시, 주기가 가장 긴 태스크부터 시작하므로 마지막 태스크는 남아있는 분할공간를 모두 할당받게 된다[9, 10, 11]. 여기에서 마지막 주기적 태스크가 마감시간을 위반하지 않을 최소의 분할공간들만을 마지막 태스크에 할당하고 여분의 분할공간들은 비주기적 태스크들에게 할당하도록 한다. 이 과정은 마지막에 분할공간들을 할당받는 주기적 태스크(예 τ_1)의 분할공간 수를 조절하므로 써 이루어진다. 그런 후, 주기 및 비주기적 태스크들을 혼합하였을 때, 스케쥴링이 가능한지를 본다. 만일 마감시간을 만족하면 이 할당에 대한 $\min(U_{total})$를 갖는 캐쉬 분할 형태가 주기 및 비주기적 태스크의 마감시간을 만족할 수 있는 최적 캐쉬 분할 형태이다.

앞의 예로 보면, 주기적 태스크 τ_1의 분할공간 수만을 감소시키면서 주기적 태스크에 대해 그 분할 형태가 스케쥴링이 가능한지를 본다. 만일 주기적 태스크들 중에 마감시간을 만족하지 못하는 태스크가 있다면 주기적 태스크의 분할공간 할당들을 (그림 3)의 알고리즘처럼 재할당시키고, 재 할당된 캐쉬 분할 형태에 대해 주기적 태스크들을 다시 스케쥴링 한다. 만일 주기적 태스크가 마감시간을 만족하면, 비주기적 태스크까지 고려해서 스케쥴링을 한다. 비주기적 태스크들은 τ_1에게 할당하고 남은 분할공간을 할당받는다. 비주기적 태스크들의 분할 형태는 비주기적 태스크 각각에 대한 할당을 모두 고려한다. 이때, 마감시간을 만족하게 되면 주기적 태스크의 가용비용과 비주기적 태스크들의 가용비용의 합을 $\min(U_{total})$과 비교하여 작은 값으로 둔다. 이 값에 대응된 캐쉬 분할 형태상에서 주기 및 비주기적 태스크들이 마감시간을 만족하는지의 스케쥴링 검사를 고려하고, 재 할당

```
if ( 선정된 캐쉬 분할 형태에 할당된 주기적 태스크들의 스케줄링)
    then  비주기적 태스크 스케줄링(그림 4);
    else { M_{n,S} <- 현재 할당시 참조된 M_{m,A} 다음 으로 작은 값 ;
                /* 위에서 수행된 오름차순으로 정렬된  a_M ∈ Φ_j(U_{i,a_i} + M_{i-1,j-a_i})중에 앞에
                   서 할당된 값 다음으로 작은 값임 */
           P_{n,S} ;                    /* M_{n,S} 값의 변화에 따른 P_{n,S} 값 변경 */
           goto  주기적 태스크들을 분할공간에 할당(그림 2);
                                        /* 태스크들이  할당될 캐쉬 분할 형태를 재 고려 */
    }
```

(그림 3) 캐쉬 분할 형태의 재 선정 알고리즘

(Fig.3) The Rechoosing Algorithm of Cache Partitioning Configuration

```
/* 비주기적 태스크를 고려한 태스크들의 최소 가용비용 및 할당 분할공간들의 수 계산*/
 p_allo = 총 분할공간수 - 마지막 주기적 태스크를 제외한 태스크들에 할당된 분할공간수;
 for ( i = p_allo ; i >= 0 ; i-- ) {
       τ_1 주기적 태스크에 i개의 분할공간을 할당 ;
       allo = p_allo - i ;
       for( j = 0; j <= allo ; j++ ) {                    /* allo개의 분할공간들로 가능한
           τ'_1 비주기적 태스크에 j개의 분할공간을 할당;      분할 형태들을 비주기적 태스크
           τ'_2 비주기적 태스크에 (allo-j)개의 분할공간을 할당;      들에게 할당 */
           if (주기 및  비주기적 태스크에 대한 분할 형태가 스케줄링 가능)
              then { 주기적 태스크들과 비주기적 태스크들의 최소 가용비용을 얻는다 ;
                     기존에 얻은 캐쉬 분할 형태에 대한 총 최소 가용비용과 비교하여 그 중
                     작은 값을 새로운 총 최소 가용비용으로 설정한다; }
 }     }
```

(그림 4) 비주기적 태스크들의 분할공간 할당 알고리즘

(Fig.4) The Segment Allocation Algorithm of Aperiodic Task Set

```
/*  주기적 태스크들에 대한 스케줄링 검사 */
 for( i = 0; i < N ; i++){
τ_i 관점에서 Σ_{k=1}^{i} τ_i 태스크를 고려하여 태스크들이
        마감시간을 만족하는지 검사;
 }
```

(그림 5) 주기적 태스크들의 스케줄링 검사 알고리즘

(Fig.5) The Scheduling Test Algorithm of the Periodic Task Set

```
/* 주기 및 비주기적 태스크 모두에 대한 스케줄
   링 검사 */
 for( i = 0; i < N ; i++){
τ_i 관점에서 Σ_{k=1}^{i} τ_i 태스크를 고려하며,
   비주기적 태스크의  실행시간을 삽입하여
   마감시간을 만족하는지 검사 ;
 }
```

(그림 6) 주기 및 비주기적 태스크들의 스케줄링 검사 알고리즘

(Fig.6) The Scheduling Test Algorithm of the Periodic and Aperiodic Task Sets

가능한 모든 캐쉬 분할 형태에 대해 반복 실행한다.

결과적으로, $\min(U_{total})$ 값이 주기 및 비주기적 태스크의 최소 가용비용이 되고 이를 토대로 한 분할공간 할당이 주기 및 비주기적 태스크 모두의 마감시간을 만족시키게 된다. 위 전반적인 과정은 다음 (그림 4)의 알고리즘으로 나타낸다.

4. 결과 분석

4.1 스케쥴링 검사

주기 및 비주기적 태스크들의 분할공간 할당이 타당한지 밝히기 위해 스케쥴링 가능 여부를 검사해야 한다. 주어진 태스크들의 스케쥴링이 가능하다는 것은 모든 태스크들이 마감시간을 만족함을 의미한다. 주기적 태스크들에 대한 스케쥴링 검사는 Rate Monotonic 스케쥴링 알고리즘을 이용하여 모든 주기적 태스크들이 마감시간을 만족하는지 먼저 검사한 후, 비주기적 태스크들을 추가하여 재스케쥴링 검사를 한다. 이때 주기적 태스크의 스케쥴링 검사에서 최소의 분할공간들이 할당되도록 하여 여분의 분할공간들을 비주기적 태스크에 할당시키므로 써 주기적 태스크의 실행시간 중 유휴시간을 비주기적 태스크가 실행하도록 한다. 주기적 태스크들의 스케쥴링 검사 알고리즘은 (그림 5)와 같으며, 주기 및 비주기적 태스크들을 모두 고려한 스케쥴링 검사 알고리즘은 (그림 6)과 같다.

위 알고리즘을 〈표 1〉에서 기술한 주기 및 비주기적 태스크들에 대해 〈표 4〉를 이용하여 얻은 분할 형태대로 각 태스크들의 분할공간 할당 형태를 보면, τ_1엔 2개, τ_2엔 3개, τ_3은 1개, τ'_1엔 0개, τ'_2엔 0개가 할당된다. 제일 처음 주기가 가장 짧은 τ_1 관점에서 스케쥴링 검사를 하면, 이때는 고려되어야 할 태스크가 τ_1 뿐이므로 τ_1이 2개의 분할공간를 할당받아 실행한 시간을 고려한다. τ_1의 실행시간은 〈표 2〉를 참조하면 34가 되고 이 시간은 τ_1주기 안에 이루어지므로 마감시간을 만족한다.

다음으로 τ_2 관점에서 스케쥴링 검사를 하면, 이때는 τ_1과 τ_2를 고려해야한다. 실행시간 계산은 τ_1의 실행 시간 34와 τ_2의 3분할공간 할당의 실행시간 58로 계산한다. τ_3 관점에서 스케쥴링 검사를 하면, 이때는 τ_1, τ_2와 τ_3을 고려한다. τ_1의 34와 τ_2의 58, 그리고 τ_3의 1분할공간 할당의 실행시간 213으로 계산한다. τ_3에 대해서 스케쥴링을 설명하면, τ_3이 고려해야할 주기(실행시간)는 τ_1, τ_2와 τ_3 모두에 대한 주기이다. 그러므로 각 태스크들의 주기들은 {100, 140, 200, 280, 300, 400, 420, 500, 560,..., 1260, 1300, 1400}이다. 이때 각각의 태스크에 대한 실행시간 계산 값이 주기 값보다 작거나 같게 되면($xC_1 + yC_2 + zC_3 \le T_i$), 그 태스크들은 스케쥴링이 가능하다고 본다. 스케쥴링 검사는

$$C_1 + C_2 + C_3 \le T_1 \Rightarrow 34 + 58 + 213 > 100$$
$$2C_1 + C_2 + C_3 \le T_2 \Rightarrow 68 + 58 + 213 > 140$$
$$2C_1 + 2C_2 + C_3 \le 2T_1 \Rightarrow 68 + 116 + 213 > 200$$
$$3C_1 + 2C_2 + C_3 \le 2T_1 \Rightarrow 102 + 116 + 213 > 280$$
$$\vdots$$
$$13C_1 + 10C_2 + C_3 \le 13T_1 \Rightarrow 442 + 580 + 213 < 1300$$
$$14C_1 + 10C_2 + C_3 \le T_3 \Rightarrow 476 + 580 + 213 < 1400$$

위의 계산과정을 보면 이 주기적 태스크는 1100 이후 스케쥴링이 가능하므로 비주기적 태스크까지 고려해서 스케쥴링 검사를 확장한다. 비주기적 태스크를 포함한 계산과정은 위와 비슷하고 다만 비주기적 태스크의 시작시간을 고려하여 검사한다.

$$C_1 + C_2 + C_3 \le T_1 \Rightarrow 34 + 58 + 213 > 100$$
$$2C_1 + C_2 + C_3 \le T_2 \Rightarrow 68 + 58 + 213 > 140$$
$$\vdots$$
$$6C_1 + 4C_2 + C_3 + C'_1 \le 4T_2 \Rightarrow 204 + 232 + 213 + 80 > 540$$
$$11C_1 + 8C_2 + C'_1 + C'_2 \le 11T_1 \Rightarrow 374 + 464 + 213 + 80 + 32 > 1100$$
$$14C_1 + 10C_2 + C_3 + C'_1 + C'_2 \le T_3 \Rightarrow 476 + 580 + 213 + 90 + 32 < 1400$$

과 같이 계산이 되고, 위 스케쥴링 검사에서도 1400에서 주기적 태스크와 비주기적 태스크 모두가 마감시간을 만족함을 알 수 있다.

4.2 스케쥴링 가능한 캐쉬 분할 형태의 분석

앞에서 주어진 주기 및 비주기적 태스크들에 대해서 캐쉬 분할 형태들을 구해보고 스케쥴링이 가능하면서 태스크들의 최소 가용비용을 갖는 최적 캐쉬 분할 형태를 얻는다.

〈표 4〉를 이용해서 맨 처음에 할당되는 주기적 태스크의 캐쉬 분할 형태는 앞에서 언급한대로 τ_1엔 2개, τ_2엔 3개, τ_3은 1개, τ'_1엔 0개, τ'_1엔 0개 분할공간들이 할당된다. 이 캐쉬 분할 형태에 대해서 주기적 태스크와 비주기적 태스크 모두에 대한 스케쥴링이 가능하다. 그러므로 (그림 4)와 (그림 5)의 알고리즘

〈표 5〉 주기 및 비주기적 태스크들의 캐쉬 분할 형태 및 스케쥴링

〈Table 5〉 The Cache Partitioning Configurations and the Scheduling Test of the Periodic and Aperiodic Task Sets

	주기적 태스크 (τ_1, τ_2 τ_3)			스케쥴링	주기적 태스크의 가용비용	비주기적 태스크 (τ'_1, τ_2)		전체 태스크들의 스케쥴링	총 최소 가용비용
	태스크 τ_1	태스크 τ_2	태스크 τ_3			태스크 τ'_1	태스크 τ'_2		
캐쉬 메모리 분할 형태	2	3	1	O	0.91	0	0	O	1.63
	1	3	1	O	0.92	0	1	O	1.60
						1	0	O	1.59
	0	3	1	O	0.97	0	1	X	–
						1	1	X	
						2	0	X	
	3	3	0	O	0.92	0	0	O	1.63
	2	3	0	O	0.93	0	1	O	1.61
						1	0	O	1.60
	1	3	0	O	0.94	0	2	X	–
						1	1	X	
						2	0	X	
	0	3	0	O	0.99	0	3	X	
						1	2	X	
						2	1	X	
						3	0	X	
	1	3	2	O	0.92	0	0	O	1.64
	0	3	2	O	0.97	0	1	X	–
						1	0	X	
	1	2	3	O	0.93	0	0	O	1.65
	0	2	3	X	0.98	–	–	X	–
	1	1	4	O	0.94	0	0	X	–
	0	1	4	X	1.0	–	–	X	
	0	1	5	X	0.98	–	–	X	–
	0	0	6	X	1.04	–	–	X	–

에 의해 스케쥴링 가능한 캐쉬 분할 형태에 대한 태스크들의 최소 가용비용인 $\min(U_{total})$ 값을 설정해 두고, 또 다른 분할 형태를 찾기 위해 분할공간에 재 할당한다. 다음에 나타날 수 있는 태스크의 할당은 τ_1엔 1개, τ_2엔 3개, τ_3은 1개의 분할공간들을 갖는 캐쉬 분할 형태를 고려한다. 여기에서 이 주기적 태스크들의 할당에 대한 스케쥴링 검사를 하여 스케쥴링이 가능하다면, 주기적 태스크들을 분할공간들 상에 할당하고 난 후, 여분의 분할공간(여기에서는 1개)을 비주기적 태스크들을 위해 할당한다. 이렇게 주기적 태스크의 최소 가용비용을 이용해서 비주기적 태스크까지 고려한 할당 가능한 캐쉬 분할 형태들은 〈표 5〉와 같이 얻어진다. 〈표 5〉에서 스케쥴링의 가능여부를 O(가능)와 X(불가능)로 구별하여 표시한다.

〈표 5〉는 주기적 태스크들과 비주기적 태스크들 각각에 대해 할당된 분할공간들의 수와 태스크들의 스케쥴링 가능여부, 주기적 태스크들의 **가용비용**, 비주기적 태스크들을 포함하는 경우에서 **스케쥴링**이 될 때의 태스크들의 총 최소 가용비용을 나타내고 있다.

〈표 5〉에서 나타낸 것처럼 주기적 태스크들에게 할당되는 분할공간들은 〈표 3〉과 〈표 4〉의 최소 가용비용을 이용하여 순서적으로 실행한다. 주기적 태스크들의 분할공간 할당에 대한 스케쥴링 검사가 먼저 행해지고 스케쥴링이 가능하면 비주기적 태스크들까지 고려한 캐쉬 분할 형태들을 보인다.

〈표 5〉에서 스케쥴링의 가능 여부의 관점에서 분석해보면, 주기적 태스크들의 가용비용이 0.91에서 0.93 사이일 때는 비주기적 태스크들까지 고려하더라도 모든 캐쉬 분할 형태들 상에 태스크들의 마감시간이 보장됨을 알 수 있다. 그러나 주기적 태스크들의 가용비용이 0.94이상일 때는 주기적 태스크들이 스케쥴링 불가능하거나 주기적 태스크들에 대해서는 스케쥴링이 가능하지만 비주기적 태스크들까지 고려할 때는 스케쥴링이 불가능함을 알 수 있다. 주기적 태스크들이 스케쥴링 가능인 경우에, 이것을 태스크의 최소 가용비용과 상대적 크기로 고찰해보면, 주기적 태스크의 가용비용이 최소 가용비용에 대해 55%에서 58%사이의 비율을 가지고 있을 때, 주기적 태스크와 비주기적 태스크들 모두에 대해 스케쥴링이 가능함을 알 수 있다. 또 한가지 분석된 내용으로 마지막 할당되는 주기적 태스크(τ_1)가 분할공간을 하나도 할당받지 못할 때도 스케쥴링이 불가능하게 됨을 알 수 있다.

〈표 5〉에서 보는바와 같이, 캐쉬 최적 분할 형태는 τ_1:1 분할공간, τ_2:3 분할공간, τ_3:1 분할공간, τ'_1:1 분할공간, τ'_2:0 분할공간을 할당하는 경우이다. 이 때, 3개의 주기적 태스크의 최소 가용비용은 0.92이고, 2개의 비주기적 태스크의 최소 가용비용은 0.67이다.

5. 결 론

실시간 태스크의 처리를 요구하는 경우에, 가능한 한 캐쉬 상에서 페이지 폴트율을 최소화해야 한다. 따라서 캐쉬 분할공간들을 고정 크기로 분할하여, 마감시간이 촉박한 태스크들에게 할당 공간을 우선적으로 할당하므로 써 마감시간을 만족시켜야 한다.

본 논문은 실시간 주기적 태스크와 비주기적 태스크들의 데드라인 만족을 고려한 캐쉬의 최적 분할 형태를 얻음으로써 태스크들에 대한 캐쉬의 가용성을 늘릴 뿐 아니라 마감시간 위반율을 최소화하는데 목적을 두었다. 비주기적 태스크들까지 고려한 캐쉬 분할 형태는 주기적 태스크가 데드라인을 만족하면서 최소한의 분할공간들을 할당받도록 하고, 주기적 태스크에 할당된 분할공간들을 제외한 나머지 분할공간들에 비주기적 태스크들을 할당시킨다. 주기적 태스크들과 비주기적 태스크들에 대해서 데드라인을 만족하는 캐쉬 분할 형태들 중에서 총 가용비용이 최소가 되는 분할 형태가 캐쉬 최적 분할 형태가 된다. 캐쉬 최적 분할 형태로 주기적 태스크들과 비주기적 태스크들이 할당되므로 써, 캐쉬의 페이지 폴트율을 줄이게 되고 태스크들이 모두 데드라인을 만족하게 된다.

본 논문은 주기적 태스크들과 비주기적 태스크들을 예로 들어 각각의 가용비용과 수행비용을 계산하여 주기적 태스크들에 대해서는 주기적 분할공간 할당 알고리즘을 이용하고, 비주기적 태스크들에 대해서는 비주기적 분할공간 할당 알고리즘을 이용하여 다양한 캐쉬 분할 형태들을 얻고, 캐쉬 분할 형태들로부터 스케쥴링 가능한 태스크들의 가용비용의 범위와 태스크들이 최소 가용비용으로 실행할 수 있는 캐쉬의 최적 분할 형태를 분석하였다.

앞으로의 추가되어야 할 연구로는, 시스템에 본 연

구를 응용하기 위해서 시스템 관점에서 캐쉬의 분할 공간의 크기의 결정 방법과 캐쉬에 영향을 주는 버스 프로토콜, I/O수행, 페치(fetch)시간, Buswidth들의 매개변수를 고려한 정확한 태스크의 가용비용 계산이 요구되어야 할 것이다.

참 고 문 헌

[1] John E. Sainnowski and Jay K. Strosnnider, "A Dynamic Algorithm for Cache/Memory partioning for Real-Time Systems", IEEE Trans. on Computer, Vol. 42, No. 8, pp 997-1001, August, 1993.

[2] John Lehoczky, Lui Sha and Ye Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization And Average Case Behavior", in Proc. Real-Time Systems Symp., IEEE, Santa Monica, CA, pp 166-171, Dec. 1989.

[3] D. B. Kirk, "Predictable cache design for real-time systems," Ph. D dissertation, Carnegie Mellon Univ., Dec. 1990.

[4] Alan Jay Smith, ""Line (Block) Size Choice for CPU Cache Memories", IEEE Trans. on computers, Vol. C-36, No. 9, pp 1063-1075, Sep. 1987.

[5] G. Brassard and P. Bratley, "Algorithmics: Theory and Practice", Englewood Cliffs, NJ: Prentice-Hall, pp 142-144, 1988.

[6] A. M. van Tilborg and G. M. Koob, "Foundations of Real-Time Computing", Boston, MA. Kluwer, ch. 9, 1991.

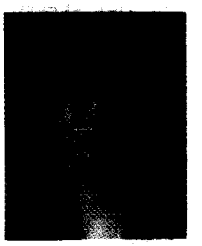
[7] David B. Kirk, "Process Dependent static cache partitioning for Real-Time Systems", in Proc. Real-Time Systems Symp., IEEE, pp 181-190, 1988.

[8] Mark Dalton. "Operating System Concepts", Addison Wesley, Reading, Massachusetts, 1985.

[9] John P. Lehoczky, Lui Sha, Jay K. Strosnider, "Enhanced Aperiodic Scheduling in Hard Real-Time Environments", Proc. of the Real-Time Systems Symp., IEEE, San Jose, CA, pp 261-270, Dec.. 1987.

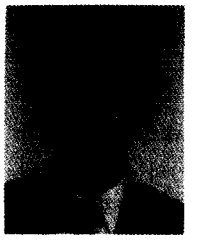
[10] 김형일 외 3인, "실시간 환경하에서 혼합 우선순위 할당에 의한 비주기적 태스크 스케줄링 알고리즘", 한국정보과학회 논문지, 22권, 5호, 1995.

[11] 이승룡 외 3인, "혼합 우선 순위 시스템에서 경성 비주기적 태스크 스케줄링 알고리즘", 정보과학회논문지(A), 22권, 10호, 1995.



김 명 희

1993년 원광대학교 공과대학 컴퓨터공학과(공학사)
1996년 원광대학교 대학원 컴퓨터공학과(공학석사)
1996년 원광대학교 대학원 컴퓨터공학과 박사과정중
관심분야: 멀티미디어 데이터베이스, 분산 실시간 컴퓨팅, 시스템 최적화, 운영체제



주 수 종

1986년 원광대학교 전자계산공학과(공학사)
1988년 중앙대학교 대학원 전자계산학과(공학석사)
1992년 중앙대학교 대학원 전자계산학과(공학박사)
1993년~1994년 미국 Univ. of Massachusetts at Amherst, 전기 및 컴퓨터공학과, Post-Doc.
1990년~현재 원광대학교 공과대학 컴퓨터공학과 부교수
관심분야: 멀티미디어 데이터베이스, 분산 실시간 컴퓨팅, 분산객체모델, 시스템 최적화