論文 96-21-11-6

실시간 다중프로세서 환경에서 프로세서 수의 감소를 위한 효율적인 타스크 배치방식

正會員 愼 明 昊*, 李 廷 泰**, 朴 升 圭**

Efficient Task Allocation Algorithms for Reducing Processors on Real-Time Multiprocessor System

Myung-Ho Shin*, Jung-Tae Lee**, Seung-Kyu Park** *Regular Members*

※본 논문은 아주대학교 연구비의 지원에 의하여 연구되었습니다.

요 약

실시간 시스템을 위한 다중프로세서 환경에서 스케쥴링 문제는 대부분 NP hard 문제로서 최적의 해를 구하는 것은 매우 어려우나, 휴리스틱에 의한 여러 효율적인 방법이 계속 연구되고 있다. 이중 주기적인 타스크들을 여러 프로세서에 어떻게 배치하면서 실시간성을 보장하는 가에 대한 연구도 진행되고 있다. 그 동안 연구되었던 배치 방법인 RMNF, RMFF, FFDUF 및 NEXT-Fit-M 등은 주어진 타스크들을 정렬 또는 그룹을 만들어 프로세서에 배치하는 알고리즘이다. 본 논문에서는 Next-Fit-M과 유사한 그룹에 의한 배치로 보다 적은 프로세서를 요구하는 방식 및 알고리즘 네가지를 제안하고, 주기적인 타스크들을 임의로 발생시켜 시뮬레이션을 수행하였다. 이러한 분석결과 제안한 방법이 기존의 방식보다 프로세서 수를 더 줄일 수 있음을 보였다.

ABSTRACT

Scheduling problems in real-time systems are known to be NP-hard. The heuristic approaches are generally applied to solve a certain class of systems. One of such cases is to allocate periodic tasks to multiprocessors while the method assures the requirement of the deadline constraints of real-time systems. The study on the allocation of per-

*한국교육개발원 부설 멀티미디어교육연구센터
Center for Multimedia Educational Research & Development, KEDI.
**아주대학교 정보 및 컴퓨터공학부
Division of Information & Computer Eng. Ajou Univ.
論文番號:96103-0328
接受日字:1996년 3월 28일

iodic tasks includes RMNF, RMFF, FFDUF and Next-Fit-M algorithms, which make a set of task groups first and then allocate to processors. This paper proposes the various algorithms which are based on the Next-Fit-M. To analyze the four proposed methods, simulation was carried on, in which the sample tasks are randomly generated with the various time intervals. The proposed algorithms reduce the number of processors compared with the conventional methods.

Ⅰ. 서 론

최근에 하드웨어 기술의 발전으로 다중프로세서 컴퓨터는 더욱 일반화되고 있다. 이에 따라 그동안 단일프로세서 환경에서 연구되던 스케쥴링 알고리즘인 RM(Rate Monotonic) 스케쥴링이나 ED(Earliest Deadline) 스케쥴링이 다중프로세서 환경으로 확장되어 연구되고 있다. 이러한 스케쥴링 문제는 다중프로세서 환경에서 대부분 NP hard로 밝혀지고 있다[1].

그러므로 다중프로세서 환경에서 주어진 여러 타스크를 선점 우선순위기반으로 스케쥴링하는 알고리즘도 RM과 ED 스케쥴링 알고리즘을 사용할 때로 구분해서 고려하려고 한다.

첫째, 동적 스케쥴링 알고리즘인 ED을 사용할 경우에는 단일프로세서에 할당된 모든 타스크들에 대한 UF(Utilization Factor)를 더한 값이 1보다 같거나 작으면 스케쥴링이 가능하다[2][3]. 이와같이 ED 스케쥴링 알고리즘을 사용할 때 각 프로세서에 타스크들을 배치하는 문제는 "Bin Packing Problem"[4]과 같다. 이는 각 프로세서에 배치되는 타스크의 수에 관계없이 스케쥴링 가능하므로 본 논문의 고려대상에서 제외하고자 한다.

둘째, 정적 스케쥴링 알고리즘인 RM을 사용할 경우에는 단일프로세서에 배치된 모든 타스크들에 대한 UF를 더한 값이 $n(2^{1/n}-1)$ 보다 같거나 작으면 RM 알고리즘으로 스케쥴링이 가능하다. 이 RM 스케쥴링 알고리즘 사용시 다중프로세서 환경에서 각 프로세서에 대한 타스크들의 배치문제는 매우 복잡한 문제이나 그동안 휴리스틱에 근거한 몇가지 알고리즘들이 제안되고 있다.

RM을 응용한 스케쥴링 알고리즘으로는 RMNF(Rate Monotonic Next Fit), RMFF(Rate Monotonic First Fit), FFDUF(First Fit Decreasing Utilization Factor) 스케쥴링 알고리즘[5]과 RMFF를 개선한 Next-Fit-M 스케쥴링 알고리즘[6] 등이 있다. 상기 연구결과는 RMNF보다는 RMFF가, RMFF보다는 FFDUF가 더 효율적인 알고리즘으로 평가되고 있다.

본 논문은 이중 Next-Fit-M과 유사한 그룹에 의한 타스크 배치를 하지만, 프로세서의 수를 더 줄일 수 있는 방식과 알고리즘을 제안하고자 한다.

Ⅱ. 다중프로세서 환경에서의 타스크 배치

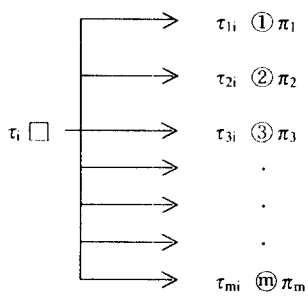
배치방법 기술에 앞서 사용되는 기호들을 표시하면 다음과 같다.

타스크 1, 2, ..., n : $\tau_1, \tau_2, ..., \tau_n$ ($n>0$, n : 정수)
프로세서 1, 2, ..., m : $\pi_1, \pi_2, ..., \pi_m$ ($m>0$, m : 정수)
τ_i의 최악계산시간 : C_i,
τ_i의 주기 : T_i
τ_i의 이용률(UF : Utilization Factor) : $U_i = C_i/T_i$
class-k의 타스크 집합 : $(2^{1/(k+1)}-1) < U_i \leq (2^{1/k}-1)$
class-M의 타스크 집합 : $(0 < U_i \leq (2^{1/M}-1))$
class-k의 j번째 프로세서 : $P_{k,j}$
class-k의 타스크 수 : N_{Ak}

다중프로세서에서 타스크를 스케쥴링하는 알고리즘은 다음과 같이 두가지 경우로 구분해서 고려해야 한다.

첫째, 한개의 타스크가 하나의 프로세서에서만 실행되는 경우에는 각 프로세서에서 RM 또는 ED 알고리즘을 사용하여 각 프로세서에서의 스케쥴 가능성을 검사할 수 있다.

둘째, 한개의 타스크가 여러프로세서에서 실행되는 경우, 즉 아래와 같이 타스크 τ_i가 타스크 τ_{1i}, τ_{2i}, τ_{3i}, τ_{4i}, ..., τ_{mi}로 쪼개어질 경우는 서브타스크의 순서가 일정하면 Flow Shop 알고리즘를 사용하여 스케쥴 가능성을 검사한다.



그렇지 않은 경우, 즉 서브타스크의 순서가 일정치 않을 경우는 Job Shop 알고리즘를 사용하여 스케쥴 가능성을 검사하면 된다.

본 논문에서는 한 개의 타스크가 하나의 프로세서에서 실행되며, RM 스케쥴링 알고리즘을 사용하는 경우로 제한한다. 이와같은 다중프로세서 환경에서 타스크를 배치하는 방법은 이미 기술한 바와 같이 RMNF, RMFF, FFDUF 및 Next-Fit-M 등이 있다.

RMNF 스케쥴링 알고리즘은 먼저 주어진 타스크들을 주기(T_i)에 따라 올림차순으로 순서를 정하고, τ_i가 프로세서 π_j에서 스케쥴링 가능하면 프로세서 π_j에 배치한다. τ_i가 π_j에서 스케쥴 가능하지 않으면 π_{j+1}에 배치한다.

RMFF 스케쥴링 알고리즘은 타스크에 순서를 정하는 것은 RMNF와 같으나 타스크를 프로세서에 배치할 때 이미 배치된 프로세서 π_1 부터 π_j 까지 스케쥴 가능성을 검사하여 스케쥴 가능하면 그 프로세서에 배치하고 그렇지 않으면 π_{j+1}에 배치한다.

반면에 FFDUF 스케쥴링 알고리즘은 주기에 따라 타스크 순서를 정하지 않고 UF(C_i/T_i)값을 가지고 내림차순으로 타스크를 순서화한 후에 τ_i가 프로세서 π_1 부터 π_j에서 스케쥴 가능하면 프로세서 π_1 부터 π_j에 배치하고 그렇지 않으면 π_{j+1}에 배치한다.

또한, Next-Fit-M 스케쥴링 알고리즘은 UF값에 따라 타스크를 class별로 분류한 후에 class-k에 있는 타스크가 프로세서 $P_{k,j}$에서 스케쥴링 가능하면 $P_{k,j}$에 배치하고 그렇지 않으면 $P_{k,j+1}$에 배치하는 알고리즘이다.

1. 타스크 배치 방식

다중 프로세서 환경에서 타스크를 프로세세에 배치하는 RMNF, RMFF, FFDUF, Next-Fit-M 알고리즘 중 Next-Fit-M 알고리즘의 요구되는 프로세서 수, 프로그램의 복잡도 등에서 가장 효율적인 알고리즘으로 평가되고 있다. 그러므로 본 논문은 Next-Fit-M 알고리즘은 수정하지 않고 class별 타스크 분류 방법을 더 효율적으로 수정한 배치방식-1, Next-Fit-M의 class별 분류방법은 그대로 적용하고 알고리즘을 더 효율적으로 수정한 배치방식-3 및 타스크 class별 분류 방법과 알고리즘을 모두 수정한 배치방식-2와 배치방식-4가 있다. 본 논문에서 제안한 이 네가지 알고리즘은 다음과 같이 두단계로 나누어 접근하였다.

첫째, 모든 타스크들을 그룹으로 분류하는 단계에서 Next-Fit-M 알고리즘의 분류 방법을 수정, 새로운 배치방식에 적합하게 분류한다.

둘째, Next-Fit-M 알고리즘을 수정해서 새로운 알고리즘을 생성한다.

1.1 배치방식-1

다중프로세서 상에서 실행될 독립적인 타스크 $\tau_i(1 < i \le n)$를 U_i의 값에 따라 아래와 같이 각 그룹으로 분류한다. N_{Ak}는 class-k에 속하는 타스크의 수로 각 그룹으로 구분할 때 타스크의 수를 갖고 있다. Next-Fit-M 알고리즘의 타스크 분류 방법은 아래와 같다.

타스크의 class	UF의 범위
1	$(2^{1/2}-1, 1]$
2	$(2^{1/3}-1, 2^{1/2}-1]$
3	$(2^{1/4}-1, 2^{1/3}-1]$
4	$(2^{1/5}-1, 2^{1/4}-1]$
.	.
.	.
.	.
M	$(0, 2^{1/M}-1]$
$(3<M$, M : 정수),	$(1 \le K < M$, K : 정수)

본 연구에서 제안하는 배치방식-1은 다음과 같이 위의 class를 재배치한다.

```
for k = 2 to M − 1 do
    N_D = N_Ak − [N_Ak/k] × k;
    if N_D > 0 then
```

assign ND tasks of class-k to class-M;
end-if;

즉, 각 class마다 여분의 타스크(ND개의 타스크)들을 class-M에 분류한다. 이렇게 분류한 후에 Next-Fit-M 알고리즘을 이용하여 타스크를 다중프로세서에 배치한다. 이때 각 class에 분류되는 타스크는 스택이나 큐를 가지고 관리하면 class-M에 있는 타스크들을 U_i 값에 따라 순서있게 유지하는 데 편리하다. 이 경우 실제 프로세서 배치 전략은 Next-Fit-M과 동일하다.

1.2 배치방식-2

이 방식의 타스크 분류 방법은 배치방식-1과 동일하게 하지만 알고리즘은 아래와 같이 수정한다.

배치방식-2 알고리즘

ST1. for k = 1 to N do set $N_k = 1$;
ST2. set i = 1;
ST3. while i ≤ n do
  if τ_i is a task from class-k, $1 \le k \le M$
  then assign τ_i to $P_{k, Nk}$;
    if $P_{k, Nk}$ has currently k tasks assigned to it then
      set $N_k = N_k + 1$
    end-if
  else {τ_i is a task from class-M)
    set j = 1;
L1: if the total utilization factors of all the tasks
      assigned to $P_{M, j}$ is greater than $Ln2 - U_i$ then
        set j = j + 1; goto L1;
    else
      assign τ_i to $P_{M, j}$
    end-if
    if $j > N_M$ then set $N_M = j$;
  end-if;
    set i = i + 1;
  end-while;
ST4. if $P_{k, Nk}$ has no task assigned to it then
    set $N_k = N_k - 1$;

이 알고리즘의 특징은 class-M에 있는 타스크들을 프로세서에 재배치할 때 RMFF알고리즘을 응용했다는 점이다. 이 경우에 있어서 타스크의 순서는 다시 배열하지 않아도 된다.

1.3 배치방식-3

배치방식-3은 타스크를 class 그룹으로 분류하는 방법은 Next-Fit-M과 같다. 배치방식-3이 배치방식-1이나 배치방식-2와 다른 점은 각 class의 여분의 타스크들을 프로세서에 배치하기 위해서 class-M에 속한 타스크처럼 간주하고 있다는 점이다. 이 알고리즘은 다음과 같다.

배치방식-3 알고리즘

ST1. for k = 1 to N do set $N_k = 1$;
ST2. set i = 1;
ST3. while i ≤ n do
  if τ_i is a task from class-k, $1 \le k \le M$
  then
    if $N_k \le [N_{Ak}/k]$ then assign τ_i to $P_{k, Nk}$;
      if $P_{k, Nk}$ has currently k tasks assigned to it then
        set $N_k = N_k + 1$
    else goto L1
  end-if
  else {τ_i is a task from class-M)
L1:   if the total utilization factors of all the tasks
      assigned to $P_{M, Nk}$
      is greater than $Ln2 - U_i$ then
          set $N_m = N_m + 1$
      end-if;
      assign i to $P_{M, Nm}$
    end-if;
      set i = i + 1;
  end-while;
ST4. if $P_{k, Nk}$ has no task assigned
    to it then set $N_k = N_k - 1$;

1.4 배치방식-4

배치방식-4에서는 class-k로 분류하는 방법은 Next-Fit-M과 같고 각 class-k에 속하는 여분의 타스크($N_D = N_{Ak} - [N_{Ak}/k] \times k$)를 재배치하는 알고리즘은 배치방식-2와 같다.

Ⅲ. 비교 분석

기본적으로 각 방식에 따른 비교분석은 실험적 타스크의 발생, 시뮬레이션에 의한 배치 프로세서의 계산, 그리고 실제 시스템에 의한 실험(트랜스퓨터에서 실시간 커널인 VIRTUOSO를 이용)으로 이루어진다[7][8][9].

이장은 이중 타스크 배치시 여러 경우의 수를 고려한 분석결과와 시뮬레이션에 의한 요구 프로세서의 수를 계산한 결과를 제시한다. 또한 모든 타스크의 주기(T_i), 최악의 계산시간(C_i), 이용률(U_i) 등을 발생 및 계산하는 시간과 프로그램 문맥교환(Context Switching) 및 타스크 스케쥴링을 위한 실행시간은 동일하다고 간주하였다.

1. NEXT-FIT-M과의 프로세서 수 비교

타스크 집합이 다음과 같은 3가지 경우에 대하여 프로세서를 배치하는 알고리즘에 따라 프로세서의 수를 구하고 결과를 비교하였다. 비교 대상 알고리즘은 Next-Fit-M, 배치방식-1, 배치방식-2, 배치방식-3 및 배치방식-4 알고리즘이며, 이때 배치방식-i 알고리즘에 의해 요구된 프로세서의 수를 n_p(배치방식-i)로 표현한다. 또한 각 알고리즘별 요구 프로세서 수는 임의의 class에 타스크가 jk, jk +(k −1), jk +1과 같이 일정하게 할당된 경우를 가정하여 class별 타스크의 수를 증가시키면서 각 알고리즘을 적용하여 요구 프로세서 수를 계산하였다. 이 경우 class별 타스크의 수와 관계없이 각 알고리즘별 요구 프로세서 수의 관계는 일정하게 적용됨을 알수 있다.

1.1 각 class-k에 타스크가 jk개 들어 있는 경우

표 1과 같이 타스크들이 각 class에 k의 j정수배 만큼 분포되면(j≥1, j:정수), Next-Fit-M 알고리즘을 사용하여 각 프로세서에 타스크를 배치할 경우 가장 프로세서의 수를 줄일 수 있는 최적의 class별 분포이다.

표 1. class-k에 타스크가 jk개 있는 경우

타스크의 Class	타스크 집합
1	$\tau_2, \tau_{16}, \tau_{17}$
2	$\tau_4, \tau_7, \tau_{18}, \tau_{19}$
3	τ_5, τ_6, τ_8
4	$\tau_1, \tau_9, \tau_{11}, \tau_{12}$
M = 5	$\tau_3, \tau_{10}, \tau_{13}, \tau_{14}, \tau_{15}$

class-k에 타스크가 jk개 있는 경우 class-k의 타스크 수, 전체 타스크 수, 기존 알고리즘과 개선된 알고리즘에서 요구되는 프로세서의 수를 비교하면 다음과 같다.

class-k의 타스크 수:

$$S_k = N_i * K (K \geq 1,\ K: \text{정수})$$

단, N_i는 class-k에 할당된 프로세서 수

전체 타스크 수:

$$S = \sum_{k=1}^{M} S_k = (N_1 * 1 + N_2 * 2 + \ldots + N_M * M)$$

알고리즘별 요구 프로세서의 수:

$$n_p(\text{Next-Fit-M}) = n_p(\text{배치방식-1}) = n_p(\text{배치방식-2}) = n_p(\text{배치방식-3}) = n_p(\text{배치방식-4})$$

즉, 표 1과 같이 주어진 타스크들을 배치하기 위해 각 알고리즘에서 요구되는 프로세서의 수는 기존 알고리즘과 개선된 알고리즘 모두 동일하다.

1.2 각 class-k에 타스크가 jk +(k −1)개 있는 경우

표 2와 같이 타스크들이 class에 분포되면, 각 class에 여분의 타스크가 있는 경우 Next-Fit-M 알고리즘을 사용시 가장 최적의 class별 분포이다.

표 2. 각 class-k에 타스크가 jk +(k −1)개 있는 경우

타스크의 Class	타스크 집합
1	$\tau_2, \tau_{22}, \tau_{24}$
2	$\tau_4, \tau_7, \tau_9, \tau_{21}, \tau_{23}$
3	$\tau_5, \tau_6, \tau_8, \tau_{10}, \tau_{15}$
4	$\tau_1, \tau_{11}, \tau_{12}, \tau_{14}, \tau_{16}, \tau_{17}, \tau_{19}$
M = 5	$\tau_3, \tau_{13}, \tau_{19}, \tau_{20}$

각 class-k에 타스크가 jk +(k −1)개 있는 경우 class-k의 타스크 수, 전체 타스크 수, 기존 알고리즘과 개선된 알고리즘에서 요구되는 프로세서의 수를 비교하면 다음과 같다.

class-k의 타스크 수:

$$S_k = N_i * K + (K-1) (K \geq 1,\ K: \text{정수})$$

단, N_i는 class-k에 할당된 프로세서 수

전체 타스크 수:

$$S=\sum_{k=1}^{M} S_k=(N_1*1+0)+(N_2*2+1)+\ldots+(N_M*M+(M-1))$$

알고리즘별 요구 프로세서의 수:

n_p(Next-Fit-M) $>=$ n_p(배치방식-4) $=$ n_p(배치방식-3) $>=$ n_p(배치방식-1) $>=$ n_p(배치방식-2) $>$ K

즉, 표 2와 같이 주어진 타스크들을 배치하기 위해 요구되는 프로세서의 수는 개선된 알고리즘들이 Next-Fit-M 알고리즘보다 같거나 작다는 것을 보여준다.

1.3 각 class-k에 타스크가 jk +1 개 있는 경우

표 3과 같이 class에 타스크들이 분포되면, 각 class에 여분의 타스크가 있는 경우 Next-Fit-M 알고리즘은 최악의 class별 타스크 분포이다.

표 3. 각 class-k에 타스크가 jk +1개 있는 경우

타스크의 Class	타스크 집합
1	τ_2, τ_3
2	τ_4, τ_7, τ_9, τ_{21}, τ_{23}
3	τ_5, τ_6, τ_8, τ_{15}, τ_{19}, τ_{20}, τ_{21}
4	τ_1, τ_0, τ_{11}, τ_{12}, τ_{14}
M = 5	τ_3, τ_{10}, τ_{13}, τ_{16}, τ_{17}, τ_{18}

각 class-k에 타스크가 jk +1개 있는 경우 class-k의 타스크 수, 전체 타스크 수, 기존 알고리즘과 개선된 알고리즘에서 요구되는 프로세서의 수를 비교하면 다음과 같다.

class-k의 타스크 수:

$$S_k=N_i*K+1(K\geq 1,\ K: 정수)$$

단, N_i는 class-k에 할당된 프로세서 수

전체 타스크 수:

$$S=\sum_{k=1}^{M} S_k=(N_1*1+1)+(N_2*2+2)+\ldots+(N_M*M+M)$$

알고리즘별 요구 프로세서의 수:

n_p(Next-Fit-M) $>$ n_p(배치방식-1) $=$ n_p(배치방식-3) $>$ n_p(배치방식-4) $=$ n_p(배치방식-2) $>$ K

즉, 표 3과 같이 주어진 타스크를 배치하기 위해 요구되는 프로세서의 수는 개선된 알고리즘이 Next-Fit-M 알고리즘보다 적은 프로세서를 필요로 한다.

2. 시뮬레이션에 의한 프로세서 수 비교 분석

이 절은 Next-Fit-M과 배치방식-1, 배치방식-2, 배치방식-3 및 배치방식-4를 시뮬레이션을 이용하여 비교한 결과 및 이에 대한 분석을 기술하고 있다.

2.1 시뮬레이션 방법

시뮬레이션을 위한 타스크들의 최악의 수행시간및 주기는 워크스테이션상에서 C++ 라이브러리 난수함수를 이용하여 생성하였다. 이때, 타스크의 주기 T_i는 다음과 같은 범위에서 난수함수를 이용, 생성하였다.

$$iC_i <= T_i <= jC_i \text{ (단 } j>i,\ i,\ j>0 \text{ 정수)}$$

타스크의 주기를 일정한 범위로 제한하여 시뮬레이션을 수행해야 요구되는 프로세서 수와 M값의 관계를 분석할 수 있고, 각 배치방식의 특징도 알 수 있다. 시뮬레이션시 비교 대상 알고리즘은 Next-Fit-M, 배치방식-1, 배치방식-2, 배치방식-3 및 배치방식-4 이다. 또한 타스크의 갯수는 20에서 부터 500까지 10씩 증가하면서 실험하였다.

2.2 타스크의 주기 범위가 작은 경우

그림 4-1, 그림 4-2, 그림 4-3은 주기가 수행시간의 2배에서 4배, 5배에 6배, 5배에서 7배 사이인 경우, 각 알고리즘별로 요구되는 프로세서의 수를 그래프로 표현한 것이다. 이와같이 타스크의 주기 범위가 작은 경우에는 Next-Fit-M 알고리즘에서의 요구 프로세서의 수는 본 논문에서 제안한 알고리즘에서의 요구 프로세서 수와 큰 차이가 없다. 각 알고리즘에 따른 요구 프로세서 수는 Next-Fit-M, 배치방식-4, 배치방식-3, 배치방식-1, 배치방식-2의 순으로 나타나고 있다. 그리고 배치방식-1과 배치방식-3은 같은 프로세서 수를

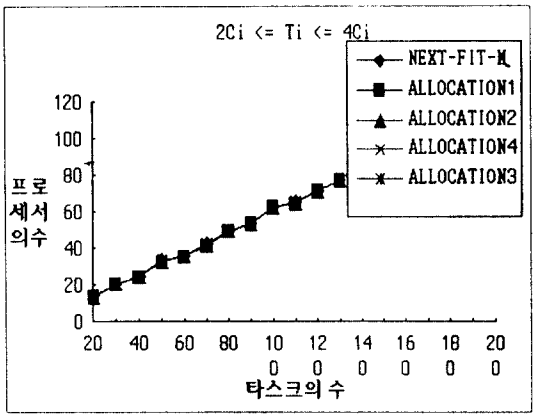


그림 4-1. 주기가 $2C_i <= T_i <= 4C_i$인 경우

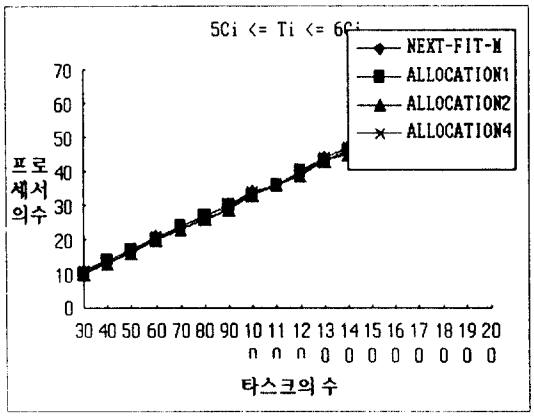


그림 4-2. 주기가 $5C_i <= T_i <= 6C_i$인 경우

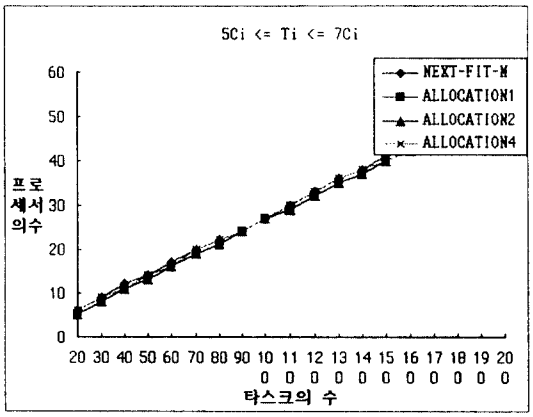


그림 4-3. 주기가 $5C_i <= T_i <= 7C_i$인 경우

요구하고 있으므로 그림 4-1를 제외한 나머지 그림은 배치방식-1에 대한 프로세서 그래프만 비교하였다.

2.3 타스크의 주기범위가 중간정도인 경우

그림 4-4와 그림 4-5는 타스크의 주기범위가 중간정도인 경우, 각 알고리즘별로 요구되는 프로세서 수를 나타낸 것이다. 타스크 주기의 최소값과 최대값의 차는 같지만 Next-Fit-M 알고리즘에서 요구하는 프로세서 수는 본 논문에서 제안한 알고리즘에서 요구하는 프로세서 수 보다 2배 이상 더 많다. 이는 그림 4-5의 타스크 주기가 그림 4-4의 타스크 주기보다 절대값에서 2배 이상 크기 때문에 M값이 그림 4-5가 그림 4-4의 2배 이상 크다. 그러므로 요구 프로세서 수도 2배 이상 요구하게 되는 것이다.

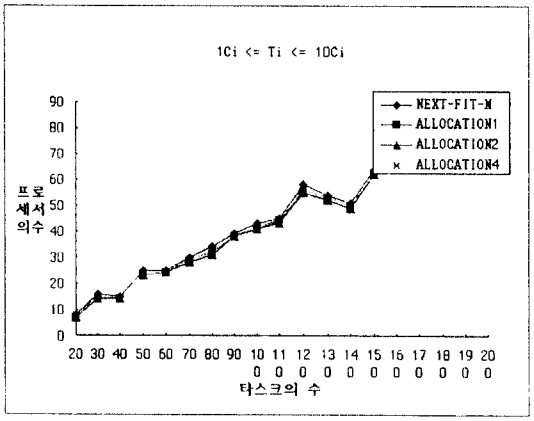


그림 4-4. 주기가 $1C_i <= T_i <= 10C_i$인 경우

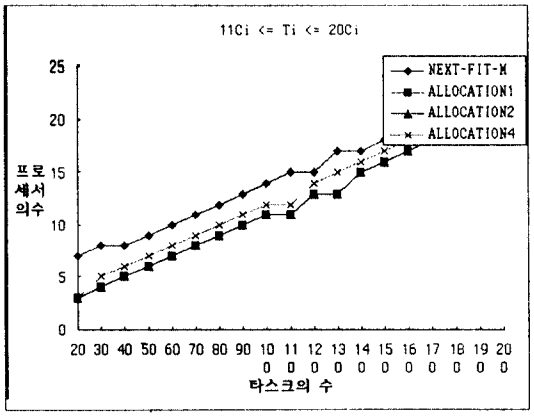


그림 4-5. 주기가 $11C_i <= T_i <= 20C_i$인 경우

2.4 타스크의 주기범위가 큰 경우

그림 4-6, 그림 4-7 및 그림 4-8은 타스크의 주기범위가 큰 경우, 각 알고리즘별로 요구되는 프로세서 수를 나타낸 그래프로 타스크의 주기와 요구 프로세서 수의 관계를 잘 보여주고 있다. 우선 그림 4-7은 그림 4-6의 주기보다 주기범위가 2배 더 큰 경우로 요구 프로세서 수도 Next-Fit-M은 본 논문에서 제안한 배치방식 보다 2배 이상 많이 요구하고 있다. 반면에 그림 4-7과 그림 4-8은 주기의 최대값과 최소값이 폭은 같고 절대값에 있어서 1.5배정도 차이가 있으나 요구 프로세서 수는 더 많은 차이를 보이고 있다. 물론 이 경우도 Next-Fit-M 보다는 본 논문에서 제안한 배치방식의 훨씬 좋은 효율을 보이고 있다.

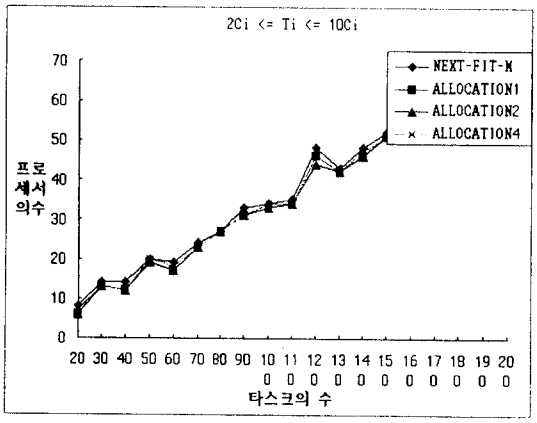


그림 4-6. 주기가 $2C_i <= T_i <= 10C_i$인 경우

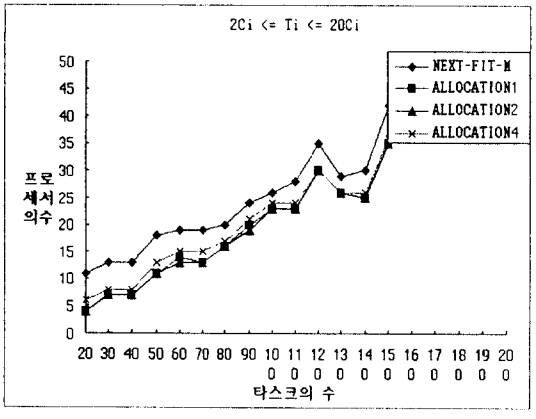


그림 4-7. 주기가 $2C_i <= T_i <= 20C_i$인 경우

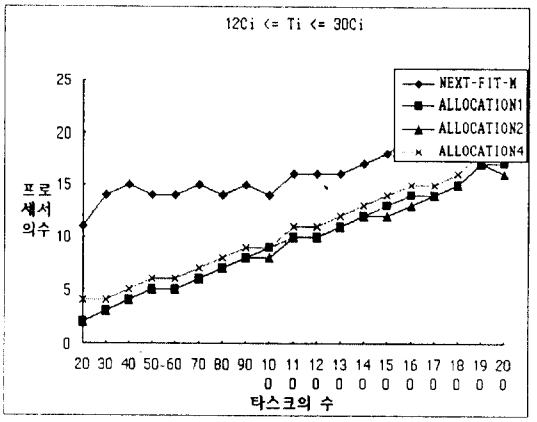


그림 4-8. 주기가 $12C_i <= T_i <= 30C_i$인 경우

2.5 시뮬레이션 분석

시뮬레이션 결과를 분석하면, 각 알고리즘별로 요구되는 프로세서 수와 타스크의 주기 및 수행시간과의 관계는 다음과 같이 요약된다.

첫째, 주기와 수행시간과의 차가 클수록, 즉 UF값이 작을수록 적은 수의 프로세서를 요구할 뿐만 아니라 본 논문에서 제안한 알고리즘의 Next-Fit-M 보다 더 적은 프로세서 수를 요구한다.

둘째, 주기범위가 같을 때는 그 중 가장 작은 UF값에 따라 M값이 결정되므로 타스크 주기의 최소값이 Next-Fit-M 알고리즘과 본 논문에서 제안한 알고리즘 사이의 요구 프로세서 수에 많은 영향을 준다.

셋째, 타스크의 수가 적고 UF값이 크면, 요구되는 프로세서의 수가 상대적으로 많아지고 Next-Fit-M 알고리즘에서의 요구 프로세서 수가 본 논문에서 제안한 알고리즘에서의 요구 프로세서 수 보다 매우 많다.

Ⅳ. 결 론

다중프로세서 실시간 시스템 환경에서 타스크를 프로세서에 어떻게 배치하는 가의 문제는 그동안 많은 연구가 있었으나 본 연구에서는 프로세서 수를 최소화 할 수 있는 휴리스틱 방식을 고려하였다. class별로 RM에 의해 배치된 여분의 타스크를 First-Fit방식으로 재배치하는 방법이 더 효율적임을 시뮬레이션에 의한 비교실험과 이론적인 분석으로 제시하였다. 향후 연구로는 실시간 커널 시스템에 실제 적용하기 위

한 구현 연구 등이 있다[10].

참 고 문 헌

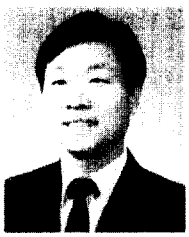
1. Andre M.van Tilborg, Gary M.Koob, "Foundations of Real-Time Computing Scheduling and Resource Management", KLUWER ACADEMIC PUBLISHERS 1991.
2. J.P.E. Sunter, K.C.J. Wijbrans and A.W.P. Bakers, "Cooperative Priority Scheduling in Occam", IOS Press 1990.
3. Sudarshan K. Dhall, C.L.Liu, "On a Real-Time Scheduling Problem", Operations Reserch Vol. 26, No. 1, January-February, pp. 127-140, 1978.
4. D. S. Johnson, A. Demers, J.D.Ullman, M.R. Garey, and R.L. Graham, "Worst-case Performance Bounds for Simple One Dimensional Packing Algorithm," SIAMJ. Comput. 3, 299-325, 1974.
5. S. Davari and S.K. Dhall, "On a Real-Time Task Allocation Problem," 19 Annual Hawaii International Conference on System Sciences, Jan. 8-10 Honolulu, Hawaii,. 1985.
6. Sadefh Davari, Sudarshan K. Dhall, "An online algorithm for Real-Time Tasks Allocation", IEEE Computer Society Press, 1986, 5.
7. Ronald S. Cok, "Parallel Programs for the Transputer", Prentice Hall, 1991.
8. Malvern, UK, "Transputer Applications", Pitman Publishing, 1989.
9. "Virtuoso User Manual", Intelligent Systems International Inc, 1994.
10. Daniel Tschirhart, "Commande en Temps Reel", Dunod Informatique Industrielle, 1990.

愼 明 昊(Myung-Ho Shin) 정회원

1966년 4월 19일생
1992년 : 아주대학교 전자계산학과 졸업(공학사)
1995년 : 아주대학교 대학원 컴퓨터공학과 졸업(공학석사)
1996년 4월~현재 : 한국교육개발원 부설 멀티미디어 교육연구센터 연구원
※주관심분야 : 멀티미디어처리컴퓨터 시스템구조, 실시간처리시스템, 저작도구, 컴퓨터통신시스템

李 庭 泰(Jung-Tae Lee) 정회원

1957년 1월 26일생
1979년 : 서울대학교 원예학과 졸업(이학사)
1981년 : 서울대학교 대학원 계산통계학과 졸업(이학석사)
1988년 : 서울대학교 대학원 계산통계학과 졸업(이학박사)
1981년~1983년 9월 : 울산대학교 전임강사
1983년 9월~현재 : 아주대학교 정보 및 컴퓨터공학부 교수
※주관심분야 : 프로그래밍 언어 이론



朴 升 圭(Seung-Kyu Park) 정회원

1950년 6월 17일생
1974년 : 서울대학교 응용수학과 졸업(이학사)
1976년 : 한국과학기술원 전자계산학과 졸업(공학석사)
1982년 : (프)Institut National Polytechnique de Grenoble 졸업(공학박사)
1976년 3월~1977년 12월 : 한국과학기술연구소
1984년 3월~1985년 9월 : IBM Watson Research Center
1978년 1월~1992년 2월 : 한국전자통신연구소
1992년 3월~현재 : 아주대학교 정보 및 컴퓨터공학부 교수
※주관심분야 : 다중 및 분산처리컴퓨터구조, 멀티미디어처리컴퓨터 시스템구조, 실시간 컴퓨터 시스템 및 성능평가