분산 실시간 시스템에서 통신시간 개선을 위한 타스크 중복 스케줄링

박미경 김창수
부경대학교 전산정보학과

Task Duplication Scheduling to improve Communication Time in Distributed Real-Time Systems

Mi-Kyong Park Chang-Soo Kim
Dept. of Computer and Information, PuKyong National University

요 약

다른 지역에 존재하는 자원이나 데이터들을 이용가능하게 하고, 지정된 마감시간내에 결과를 제공해야 하는 시간적 특성을 가진 분산 실시간 시스템은 시스템의 성능과 신뢰성을 향상시킬 수 있는 장점을 가진다. 이러한 시스템에서 수행되는 타스크는 크게 주기적 타스크와 비주기적 타스크로 나누어지는데, 빠른 수행시간을 위해 대부분의 타스크들은 병렬로 처리되기 위해 여러 개의 서브 타스크들로 분할되어 실행된다. 본 연구에서는 분산 실시간 환경에서 임의의 시간에 마감시간을 가지고 도착한 주기적 타스크에 서브 타스크의 유형에 따라 서브 타스크간의 통신시간과 수행시간을 고려한 EST(Earliest Start Time)기법을 이용하여 서브 타스크들의 효율적인 마감시간 할당 알고리즘과 ITC(Inter Task Communication)시간을 개선하기 위한 처리기 중복 할당 알고리즘을 제시하고 있다. 수행된 결과는 기존의 방법과 비교하여 타스크 전체의 마감시간 위반 최소화와 처리기의 이용률 개선 및 처리기간의 통신시간과 수행 완료시간을 개선하고 있다.

1. 서론

실시간 시스템들의 근본적인 특성들은 논리적 정확성뿐만 아니라, 지정된 시간내에 정확한 결과를 제공해야 된다. 실시간 시스템에서의 기본적인 요구는 시스템 운영의 중대만으로 그 목적을 달성했다고 할 수 없으며, 어떻게 타스크들과 자원들이 제시된 시스템에 효율적으로 스케줄되는지에 대한 신중한 고려를 요구한다[6,8].

실시간 시스템의 작업은 크게 주기적과 비주기적 타스크들로 구성되며, 주기적 타스크들은 고정된 시간 간격으로 수행되는 기본적인 부하인데 반해, 비주기 타스크들은 임의적으로 도착하는 일시적인 부하라고 말할 수 있다. 실시간 시스템들에서, 제공되는 속성은 수행되기 전에 기본적으로 알려져 있다. 주기적 타스크의 가장 중요한 파라미터들은 주기, 연산시간과 상대적인 마감시간들이 있다[8]. 연성 마감시간(soft deadline)을 가진 작업들은 경성 마감시간(hard deadline)을 가진 작업들과 달리 지정된 시간내에 수행되지 못해도 시스템에 중대한 장애를 가져오지 않으므로 마감시간 위반의 최소화가 요구된다.

분산 환경에서, 병렬 수행을 위해서는 일반적으로 여러개의 서브 타스크들로 나누어져서 처리되는데, 전체 마감시간을 어떻게 각 서브 타스크의 마감시간으로 할당할것인지의 문제는 타스크의 시간적 특성들과 수행 우선순위등이 고려되어야 하기 때문에 최적의 방법을 얻기가 어렵다.

본 논문에서는 임의의 시간에 마감시간을 가지고 도착한 주기적 타스크에 서브 타스크간의 통신시간과 수행시간을 고려하여 서브 타스크의 유형에 따라 효율적으로 마감시간을 할당하고 시간적 특성에 따라 효율적인 처리기의 이용률 개선을 위해 서브 타스크들을 하나이상의 처리기에 중복하는 방법을 제안하고 있다.

2. 관련연구

서브 타스크들 각각에 마감시간을 할당하는 방법은 달리 정의된 것을 제외하고는, 일반적으로 전체 타스크와 동일하게 마감시간을 서브 타스크들에 할당한다[1,2]. 서브 타스크의 마감시간을 구하는 연구는 본 연구와 유사하지만 방법상으로 약간의 차이가 있다: 마감시간과 연관된 파라미터들은 도착시간과 예상 수행시간만을 고려하고 있어, 실제적으로 서브 타스크들의 총 수행 요구사항과 그들간의 통신시간을 효율적으로 적용시키지 못하였다. 또한, 타스크 중간에 병행 수행되는 서브 타스크들의 복잡한 상태를 다루고 있지는 않다.

본 연구는 각 서브 타스크들의 마감시간 위반 최소화를 위해 보다 효율적인 할당 방법을 제시하고 있다. 처리기에 할당하기 위한 모든 파라미터들이

생성되면 시간적 특성을 고려한 할당과 병렬성을 최대한 이용하면서 스케줄링하기 위해 DFRN(Duplication First Reduction Next)[3]에서 중복단계를 개선시킨 FUTD(Fully connected, Unbounded Task Duplication)[11]이 직계 부모가 LT(Last Task)가 아닐 경우, 계속적으로 새로운 처리기에 할당되므로 처리기의 증가를 가져온다. 이때 EDFRN(Effective Duplication First and Reduction Next)은 유휴 처리기가 존재한다면 그 유휴 처리기에 할당을 시도한다. 즉, 제안된 방법은 FUTD보다 효율적으로 처리기를 이용할 수 있게 한다. 수행방법은 타스크들의 EST, 수행시간 순으로 우선순위를 할당하여, 우선순위 값을 각 타스크에 부여한 후, 이들을 준비 큐에 삽입한다. 이때 큐내의 타스크들은 우선순위의 내림차순으로 정렬되게 된다. 그런 다음, FIFO 방식으로 큐내의 타스크들을 선택한 후에 처리기에 스케줄하는데, 선택된 타스크가 둘 이상의 선행자를 가진 Join 타스크이면 중복을 적용하고 아니면 자신의 직계 부모(Immediate Parent)와 동일 처리기에 할당된다.

3. 타스크 모델 및 시스템 환경

본 절에서는 서로 다른 주기를 가진 두 개이상의 타스크들에 대해 병행 수행할 수 있는 타스크 모델을 제시하고, 분산 시스템 환경에 대해 간단히 정의한다.

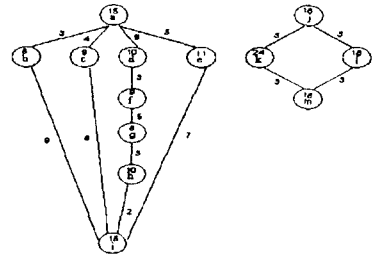
3.1 타스크 모델

분산 시스템에는 m 개의 서로다른 노드(처리기)들의 집합 $P=\{P_1, P_2, \cdots, P_m\}$와 m개 노드들에서 수행되는 n개 타스크 집합이 존재할 수 있으며, 각 타스크 집합 $T=\{T_1, T_2, \cdots, T_n\}$로 구성되고 각 타스크 $T_i=\{\tau_i, \tau_{i+1}, \cdots, \tau_k\}$는 k 개의 서브 타스크들로 구성된다. 본 연구에서 사용되는 $\tau_i=[\tau_1\tau_2\cdots\tau_p]$ 의 표기는 연속적으로 수행될 p 개 서브 타스크를 나타내며, 서브 타스크 τ_i ($i>1$) 는 서브 타스크 τ_{i-1}의 완료전에는 수행을 할 수 없다. 또한, $\tau_i=[\tau_1 \| \tau_2 \| \cdots \| \tau_q]$의 표기는 병렬로 수행될 q개 서브 타스크들, $\tau_1, \tau_2, \cdots, \tau_q$ 들로 구성된 타스크 T_i를 나타낸다. n개의 타스크들은 같은 시간에 도착하고, 만일 타스크 T_i에 대해 서브 타스크들 모두가 수행이 완료되면 타스크 T_i의 수행이 완료된다고 간주한다. 기본적인 서브 타스크들의 유형은 아래의 규칙에 따라 정의한다.

- ST$_i$: 하나의 주기적 타스크에 대해 선행자와 후행자가 존재하지 않는 단일 타스크를 simple subtask라 한다.
- SST$_i$: $T_i=[\tau_1\tau_2\cdots\tau_m]$로 표기되면, τ_i(단, $1\le i\le m$)는 serial subtask라 한다.
- PST$_i$: $T_i=[\tau_1 \| \tau_2 \| \cdots \| \tau_m]$로 표기되면, τ_i(단, $1\le i\le m$)는 parallel subtask로 정의된다.

그림1의 주기적 타스크들에서 T_1의 b, c, d, e 그리고 T_2의 k, l은 각각 병렬 서브 타스크를, T_1에서 d, f, g, h는 연속 서브 타스크를 나타내며, 선행자가 존재하지 않는 a, j 와 후행자가 존재하지 않는 i, m은 단순 서브 타스크로 분류한다. 본 연구에서 고려하고 있는 타스크 T_i 는 다음의 속성들을 가진다. $ar(T_i)$ = T_i의 도착시간, $dl(T_i)$ = T_i의 마감시간, $ex(\tau_i)$ = τ_i의 수행시간, $comm(\tau_{i-1}, \tau_i)$ = τ_{i-1} 과 τ_i 간의 통신 시간들과 관련된다.

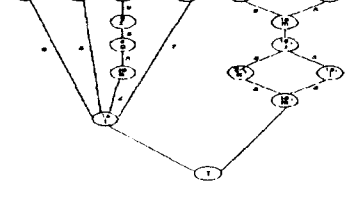
본 논문에서는 서로 다른 주기를 가진 두 개의 주기적 타스크를 고려하는데, 이 주기적 타스크들은 연속-병렬 서브 타스크들로 DAG(Directed Acycle Graph)를 구성하기 위해 서로 다른 타스크들을 타스크 주기들의 최소공배수(Least Common Multiple:LCM) 범위내에 여러 서브 타스크들로 확장된다[5,7,9]. 따라서, 새로이 확장된 타스크 그래프는 각 타스크의 다수 인스턴스들을 포함할 수 있다. 이제 한 타스크 T_i와 j번째 인스턴스 I_j 의 시작시간은 I_j * 주기(T_i)이고, 마감시간은 (I_j-1) * 주기(T_i)가 된다. 확장된 그래프를 입력으로 사용해서 한번의 스케줄이 생성되면, LCM의 주기로 계속 반복되는 타스크 모델은 그림1과 같다.



<그림 1> 주기적 타스크 T1, T2

두 타스크의 주기는 각각 120, 60으로 마감시간은 타스크의 주기보다 작거나 같다는 RM(Rate Monotonic)분석 방법[6]을 가정해서 각각 110, 60으로 가정한다. 그림1에서 원은 각 서브 타스크를 나타내고 원안의 상단에 있는 숫자는 해당 서브 타스크의 수행시간, 서브 타스크로부터 다음 서브 타스크까지 통신시간은 간선에 나타내고 있다.

각 타스크 T_i 는 도착시간 $ar(T_i)$ 이후에 수행을 준비할 수 있고, 마감시간 $dl(T_i)$ 내에 반드시 수행이 끝나야 한다. 그리고, 각 타스크들은 서로 독립적이며 서로 다른 타스크간에 선행 관계나 통신은 존재하지 않는다. 각 서브 타스크는 시간적 특성을 고려한 우선순위 비중단 스케줄링 알고리즘에 의해서 처리기에 할당된다. 그림2는 서로 다른 주기를 가진 T_1, T_2의 스케줄 길이를 구하기 위해 유사노드 R과 T가 추가되어 주기의 최소공배수인 LCM길이로 확장된 그림1의 타스크 그래프이다.

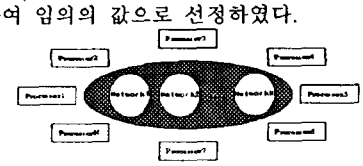


<그림2> LCM길이로 확장된 그림1의 타스크 그래프

3.2 시스템 환경

분산 실시간 시스템의 구성은 그림3과같이 네트워크를 경유하여 다수의 처리기들이 상호 연결된다고 가정하며, 시스템은 요청된 타스크들의 순서에 따라 수행되며, 처리기는 중복스케줄링 알고리즘을 고려하여 제한하지 않고 있다[4].

본 논문에서 제시한 타스크 그래프의 속성과 수행 선행조건, 시간적요구사항들은 기존의 논문[5,7,9]을 참고하여 임의의 값으로 선정하였다.



<그림 3> 분산 시스템 환경의 예

4. 서브 타스크 마감시간 할당

일반적으로 스케줄링 과정 동안 각 서브 타스크에 대한 처리기 할당을 찾기위해 서브 타스크의 수행시간과 통신시간을 고려한 보다 정확한 마감시간이 요구된다. 본 절에서는 서브 타스크의 마감시간 할당에 대한 알고리즘을 기술하고자 한다[10].

서브 타스크의 마감시간을 찾기하기 위한 초기작업으로 그림2의 LCM길이로 확장된 타스크 그래프의 시간적 특성들을 이용한 각 EST(τ_i)값은 선행 서브 타스크 τ_{i-1} 수행과 τ_{i-1} 와 τ_i 간에 통신시간이 경과한 후에 서브 타스크 τ_i 가 가장 빨리 시작 가능한 시간으로 선행자의 EST(τ_{i-1}), 수행시간(τ_i)과 통신시간(τ_{i-1}, τ_i)의 합이 된다.

$$EST(\tau_i) = EST(\tau_{i-1}) + ex(\tau_i) + comm(\tau_i, \tau_i) \cdots\cdots\cdots \text{식(1)}$$

위의 수식(1)을 적용시킨 서브 타스크들의 EST(τ_i) 값은 다음과 같다. EST(τ_i)=a=0, b=18, c=19, d=24, e=20, f=37, g=51, h=62, i=74 j=0, k=21, l=21, m=48 j′=60, k′=81, l′=81, m′=108

위의 결과는 둘 이상의 선행자를 가진 서브 타스크 i, m, m′의 경우에, EST값은 각각 계산된 값중에서 가장 큰값을 취한다. 여기서, 선행자를 가지지 않는 단순 서브 타스크인 a, j는 해당 타스크의 시작시간과 수행시간의 합을 마감시간으로, 후행자를 가지지 않는 i는 T_1의 마감시간인 110을 할당하고 연속-병렬 수행하는 서브 타스크들은 식(2),(3)과 같이 마감시간을 계산한다.

효율적인 연속 서브 타스크 마감시간(Effective Serial subtask Deadline:ESD)할당은 전체 할당된 마감시간에서 임의의 서브 타스크에 할당된 유휴 처리기 시간을 현재 서브 타스크로부터 마지막 서브 타스크까지의 총 남은 유휴 처리기 시간을 곱해서 수행시간으로 나눈값을 EST와 ex(수행시간)에 더하면 통신시간과 수행시간을 고려한 효율적인 마감시간을 구할 수 있다.

$$dl(\tau_i) = \lceil EST(\tau_i) + ex(\tau_i) + \{(deadline(T_i) - EST(\tau_i) - \sum_{j=i} ex(\tau_j)) * \frac{ex(\tau_i)}{\sum_{j=i} ex(\tau_j)}\} \rceil \cdots\cdots\cdots\cdots\cdots \text{식(2)}$$

(τ_i에 할당된 유휴 처리기 시간; 남은 총 유휴 처리기 시간; 수행시간 비)

그림2의 확장된 그래프에서 T_1의 첫 인스턴스내 연속 서브 타스크 d, f, g, h의 마감시간은 수식(2)에 의해 dl(d)≃48, dl(f)≃62, dl(g)≃78, dl(h)=110이 된다.

다음은 효율적인 병렬 서브 타스크 마감시간(Effective Parallel subtask Deadline:EPD)할당을 위해 타스크 그래프의 임의 레벨에서 병렬 수행을 하는 서브 타스크 τ_i 들의 마감시간은 전체 타스크의 마감시간에서 EST를 차감한 값에 병렬 수행 서브 타스크 수를 나누어 균등하게 구한다. 타스크 T_1의 병렬 서브 타스크 b,c,d,e와 T_2의 k, l, k′, l′의 마감시간은 수식(3)에 의해 구해진다.

$$dl(\tau_i) = \lceil [deadline(T_i) - EST(\tau_i)] / subtask\text{수} + EST(\tau_i) \rceil \cdot \text{식(3)}$$

dl(b)=41, dl(c)≃42, dl(d)≃46, dl(e)≃43,
dl(k)≃41, dl(l)≃41, dl(k′)≃101, dl(l′)≃101

하나이상의 마감시간이 존재할때에는 마감시간이 가장 큰 것을 선택한다. 두 번째 인스턴스에서 서브 타스크 j, k, l, m은 EST(τ_i, m τ_{i+1}) = EST(τ_i) + p_i 이므로 EST(τ_j)값이 60으로 설정하여 마감시간을 계산한다.

앞서 기술한 ESD할당과 EPD할당에 대한 총괄적인 서브 타스크 마감시간 할당(Subtask Deadline Assignment:SDA)알고리즘은 그림4에 나타내고 있다.

SDA 알고리즘

```
(입력 X :task; D:deadline)
begin
  if X is a simple subtask then
      if X is a root then dl(X)=start_time(X)+ex(X)
      else                dl(X):= D
  elseif X=[X_j, X_{j+1}, ..., X_n] then
          assign deadline to task X_j according to
          the ESD strategy.
          SDA (X_j, dl(X_j));
       elseif X = {X_1 || X_2 || ... X_n} then
              for i = 1 to n
              assign deadline to X_i according to
              the EPD strategy.
              SDA (X_i, dl(X_i));
  endif
end
```

<그림 4> ESD와 EPD로 구성된 SDA 알고리즘

5. 처리기 할당 알고리즘

본 절에서는 앞서 계산된 실시간 시스템의 시간적 특성들인 파라미터들을 이용하여 휴리스틱 방법에 의한 처리기 할당 알고리즘을 제시한다. 할당원칙은 먼저, 우선순위 큐내에 있는 순서화된 준비 타스크들이 차례로 처리기에 할당된다.

5.1 비중복 할당 알고리즘

여기서 제시된 타스크에 대해 중복을 고려하지 않고 타스크간의 우선순위와 통신시간, 그리고 마감시간을 고려한 휴리스틱 할당 알고리즘을 적용하고 있다. 그림 5는 중복을 고려하지 않은 시간적 특성만을 고려한 할당 알고리즘을 나타내고 있다.

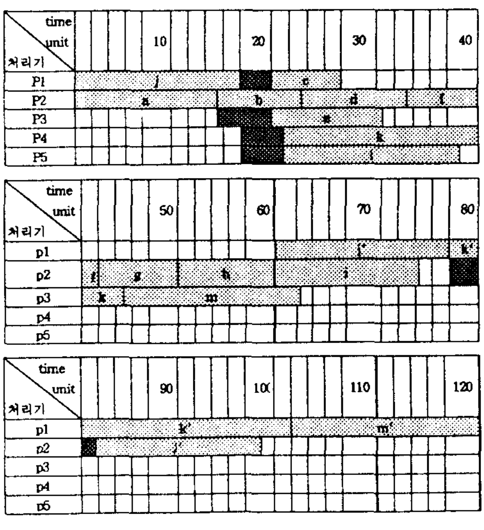
```
priority queue task[](
          subtask_id
          EST(earliest start time of any subtasks)
          ex(execution time)
          pred(predecessor)
          comm(communication time)
          )
pro[].start_time
```

```
Do_subtask[](
    subtask_id;
    finish_time;
    proc_id;
    )
for(i=0, j=0; i<max_num; i++)(
  if(subtask[i].pred==nil)(    //선행자가 없는 경우
        Do_subtask[j].subtask_id=subtask[i].id;
        사용중인 처리기중에서 start_time이
        가장 빠른 것을 선택
        min.proc_start_time에 넣고
        min.proc_id에 처리기 id를 넣는다.
        Do_subtask[j].finish_time =
         min.proc_start_time+subtask[i].ex;
        Do_subtask[j].proc_id = min.proc_id;
        )
  else ( //선행자가 있는 경우
        Do_subtask[] 배열에서 subtask[i].pred를 찾는다.
         pred.finish_time과 pred.proc_id를 구한다.
       //서브 타스크가 처리기 k에서 시작 가능한 시간
        min.time=∞
        min.proc=nil
        for(k=0; k<할당된 처리기+1; k++)(
           if(k==pred.proc_id) //통신시간=0
             start_enable_time=proc[k].start_time;
             )
         else(//통신시간이 존재
           if(proc_start_time>=
             (pred.finish_time+subtask[i].comm)(
              start_enable=proc[k].start_time;
                        )
           else(start_enable=
              pred.finish_time+subtask[i].comm;)
                  )
             if(min.time>start_enable)(
                    min.time=start_enable;
                    min.proc=k;
                    )
       Do_subtask[i].subtask_id=subtask[i].id;
       Do_subtask[i].finish_time=min.time+subtask[i].ex;
       Do_subtask[i].proc_id=min.proc_id;
```

<그림 5> 비중복 할당 알고리즘

할당된 서브 타스크들의 시간적 특성을 고려해서 처리기에 할당한 결과는 그림5와 같고, 서브 타스크들은 그 마감시간 위반이 발생하지 않았다. 처리기에 각 서브 타스크를 먼저-할당(ϕ)한 결과 각 처리기의 이용률 U_i 는 처리기 p_i 에서 수행시간 합과 LCM의 비로 계산된다

$$처리기\ 이용률(U_i) = \frac{P_i[\sum_{j=1}^{m}\phi(\tau_j)]}{LCM} \quad (j=1,2,\cdots\cdots,m) \cdots\cdots\cdots\cdots 식(4)$$



<그림 6> 비중복 할당 결과

처리기에 타스크들을 스케쥴링한 결과, 그림6은 5개의 처리기에서 U_1=0.725, U_2=0.775, U_3=0.092, U_4=0.35, U_5=0.15의 처리기 이용률을 얻을 수 있다. 평균 처리기 이용률은 약 0.418로, 최대이용과 최저 이용에서 U_i 는 각각 0.775~0.092이고 부하간의 차는 0.683이 된다. 그림내의 검은색 부분은 IPC(Inter Processor Communication) time을 나타내고, 모든 서브 타스크들이 마감시간을 위반하지 않았음을 알 수 있다.

5.2 중복을 고려한 할당 알고리즘

기존의 실시간 시스템내의 타스크 중복(duplicatio-n)은 결함-허용 요구사항을 위해서 사용되었으나 [1,8], 여기서는 연속-병렬 서브 타스크들의 처리기 이용률 개선을 위해서 중복을 고려한다. 예를들어, 그림6에서 서브 타스크 c, d, e의 선행자 a를 중복시킨다면 처리기간 통신시간이 없어지면서 수행 완료 시간은 4, 8, 5time unit만큼씩 빨라지게 된다.

본 논문에서 적용한 서브 타스크 중복 방법은 분산된 다중처리기 시스템에 근거한 중복 방법인 FUTD[11]를 개선한 EDFRN을 사용한다. EDFRN은 본 논문에서 제시된 EST계산 기법을 이용하여 계산된 시간적 특성(파라미터)들과 중복 알고리즘을 함께(그림9) 고려하고 있다.

5.2.1 FUTD 방법

FUTD는 각 타스크들을 선형 클러스터링을 이용하여 클러스터링하고 BC값은 타스크들의 우선순위를 결정하기 위해 사용된다. 클러스터내의 타스크들은 실행준비가 되면 준비 큐에 삽입된다. 생성된 타스크의 BC값에 의해 내림차순으로 삽입되며, 타스크가 할당될 처리기 번호는 클러스터 번호와 일치한다. 그후 타스크들을 처리기에 할당하여 준비 큐가 비워질때까지 계속 수행된다.

FUTD 알고리즘

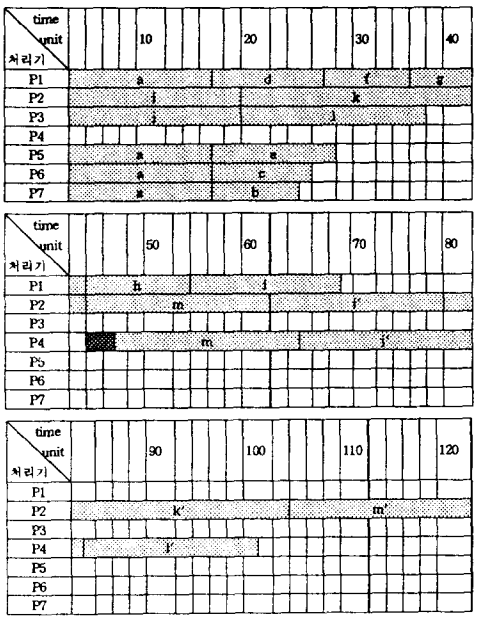
```
While not empty ready_queue do
  if τ_i is not a Join task
     if  not scheduled IP onto P_a and unused P_a
         copy the scheduled up to IP onto P_a
         //P_a is corresponding the processor C_a
     endif
         schedule τ_i to P_a
  else  try_duplication(P_a, τ_i)
        schedule τ_i to P_a
  endif
endwhile

try_duplication(P_a , τ_i )
  for each n_j, (MAT(n_j, n_i) ≥ MAT(n_q, n_i), n_j => n_i , n_q => n_i,
            p≠q, n_j and n_q are not on P_a
            n_x, ((MAT(n_x, n_j) ≥ MAT(n_y, n_j), yet)
     if there is any n_x => n_j , n_y=> n_j, x≠y, n_x and n_y are not on P_a yet)
        EST (P_a, n_j) = MAT (n_x, n_j)
        schedule n_j onto P_a
     else
           schedule n_j onto P_a //duplication n_j on P_a
     endif
     try_deletion (P_a, n_j)
  end for

try_deletion( P_a , τ_i )
  delete any duplicated tasks n_k if ECT (n_k, P_a) > MAT(n_k, n_i)
```

<그림 7> FUTD 알고리즘

그림7의 알고리즘에 따라 그림8은 FUTD를 적용해서 처리기에 각 서브 타스크를 할당(ϕ)한 결과, 각 처리기의 이용률 U_i 가 식(4)와 같은 방식으로 계산될 수 있다. 각 처리기에 타스크들을 할당한 결과 7개의 처리기에서 U_1=0.56, U_2=1, U_3=0.3, U_4=0.45, U_5=0.22, U_6=0.2, U_7=0.19의 처리기 이용률을



<그림 8> FUTD를 적용한 할당 결과

얻었다. 평균 처리기 이용률은 약 0.417이었으며 최대 이용과 최저 이용된 처리기의 U_i 는 각각 1~0.19이고 두 부하간의 편차는 0.81이된다. T_2의 전체 수행 완료시간은 중복전후에 변동이 없었지만, T_1의 경우 67 time unit에 수행을 완료하였으며, IPC time은 3 time unit으로 15 time unit이 감소하였다.

그러나, FUTD는 만일 직계 부모가 LT(Last Task)가 아닐 경우, 계속적으로 새로운 처리기에 할당므로 타스크 그래프의 깊이가 깊어질수록 처리기 수가 증가한다는 단점을 가지고 있다.

5.2.2 EDFRN 방법

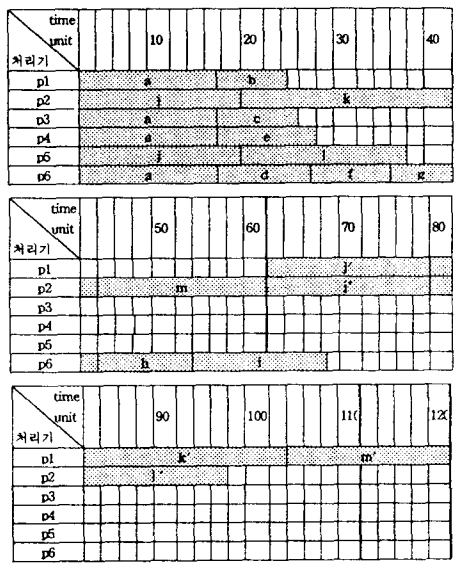
EDFRN 알고리즘

```
initialize()                          //build a priority queue using EST//
for each subtask r_i in the queue //in FIFO manner//
    if r_i is not a Join task        // r_i has only one IP//
          identify the IP
       if the IP is LT
                schedule r_i to the P having the IP
                                      //if the IP is not LT
       else if there exists having the P_u
                copy the schedule up to the IP onto P_u
          else copy the schedule up to the IP onto P_a
                                      //now the IP is LT in P_u
                schedule r_i to P_u.
       endif
    else                              //if r_i is a Join task
          identify CIP and P_c
          if CIP is LT
                DFRN(P_c , r_i )      //apply DFRN to P_c
             schedule r_i to the P_c having the CIP
          else                        //if CIP is not LT
                copy the schedule up to CIP onto P_u
                DFRN(P_u , r_i )      //apply DFRN to P_u
             schedule r_i to the P_u
          endif
    endif
endfor

DFRN(P_a , r_i )
try_duplication(P_a , n_i )
       for each n_p, (MAT(n_p , n_i) ≥ MAT(n_q , n_i), n_p => n_i , n_q => n_i,
                p≠q , n_p and n_q are not on P_a yet)
    if there is any n_x, (MAT(n_x , n_i) ≥ MAT(n_y , n_i), n_x => n_i , n_y => n_i,
                x≠y , n_x and n_y are not on P_a yet)
                EST (n_i , P_a) = MAT(n_x , n_i)
          schedule n_i onto P_a
    else
          schedule n_i onto P_a //duplication n_i on P_a
    end if
    try_deletion (P_a , n_i)
end for

try_deletion( P_a , r_i )
  delete any duplicated tasks n_k if ECT (n_k , P_a) ) MAT(n_k , n_d)
                          // n_d is the ichild of (duplicated) n_k
```

<그림 9> EDFRN 알고리즘



<그림 10> EDFRN을 적용한 할당 결과

가정한 비제한 처리기의 계속적인 사용을 감소시키고 현재 처리기들의 유휴 시간을 충분히 활용하기 위해 FUTD와 달리 EDFRN은 직계 부모가 LT(Last Task)가 아닐 경우, 현재 각 처리기에서 대상 서브 타스크가 할당된 시점에서 유휴 시간을 가진 처리기를 내림차순으로 먼저 고려한 후 새로운 처리기를 고려한다.

그림10은 EDFRN을 적용해서 처리기에 각 서브 타스크를 할당(ϕ)한 결과, 각 처리기의 이용률 U_i 가 식(4)와 같은 방식으로 계산될 수 있다. 각 처리기에 타스크들을 스케줄링한 결과 6개의 처리기에서 U_1=0.69, U_2=0.8, U_3=0.2, U_4=0.22, U_5=0.3, U_6=0.56 의 처리기 이용률을 얻었다. 평균 처리기 이용률은 약 0.462이었으며 최대 이용과 최저 이용된 처리기의 U_i 는 각각 0.8~0.2이고 두 부하간의 편차는 0.6이된다. T_2의 전체 수행 완료시간은 중복전후에 변동이 없었지만, T_1의 경우 67 time unit에 수행을 완료하였다.

5.3 할당 결과 비교

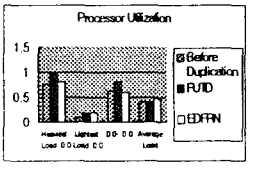
제안된 서브 타스크 마감시간 할당 알고리즘과 중복 할당 알고리즘의 할당 성능을 측정하기 위하여 임의 타스크 그래프에 대해 서로 다른 주기를 가진 주기적 타스크 1~10개, 서브 타스크 4~10개를 가

정하고, 각 서브 타스크간의 수행시간은 1~25 time unit을 통신시간은 1~10 time unit을 각각 임의로 선정하여 시뮬레이션을 수행하였다.

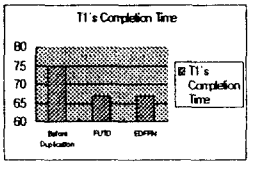
중복전후의 할당 결과는 표1과 같이 처리기 이용률면에서 상당히 개선되었으며, 중복전의 평균 처리기 이용률은 FUTD의 경우, 1%의 감소를 가져왔으나 제안방법은 10%의 증가를 가져왔다. 또한, T_1의 수행 완료시간은 FUTD와 제안방법이 동일하였으나, IPC time에서 제안방법은 FUTD보다 더 적게 발생되었다.

<표 1> 중복전후의 할당 결과 비교

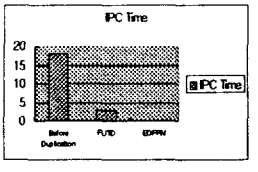
		Before Duplication	After Duplication	
			FUTD	제안방법
Processor Utilization	Heaviest Load ①	0.775	1	0.8
	Lightest Load ②	0.092	0.19	0.2
	① - ②	0.683	0.81	0.6
	Average Load	0.418	0.417	0.462
T_1's Completion Time		75 time unit	67 time unit	67 time unit
IPC Time		18 time unit	3 time unit	0 time unit



<그림 11> 중복전후의 처리기 이용률



<그림 12> 중복전후의 T_1 수행 완료 시간



<그림 13> 중복전후의 IPC Time

그림11에서, 중복 전후의 처리기 이용률을 살펴보면, EDFRN은 각각 0.03배의 처리기 최대 이용률과 1.17배의 최저 이용률이 개선되었는데, 이는 처리기의 이용률이 상대적으로 증가하였음을 나타낸다. 평균 처리기 이용률도 중복전보다 10%의 증가하였다. 그림12는 T_1의 수행 완료시간이 중복전후를 비교해서 8 time unit 단축되었으며, 그림13은 중복전후로 18 time unit의 IPC time을 줄일 수 있음을 보인다. 수행된 결과는 처리기간의 통신시간을 줄임으로써 처리기의 이용률을 개선시키고 타스크의 수행 완료시간을 단축시키는 장점을 가지고 있으나 중복 함수 계산을 위한 시간 복잡도가 존재하고 중복을 위한 타스크 이동 및 기억장치 사용량이 증가하는 단점을 가지고 있다.

6. 결론

본 논문에서는 분산 실시간 시스템에서 주기적 타스크들의 각 서브 타스크 마감시간을 고려한 효율적인 알고리즘을 제안하였으며, 서브 타스크의 수행완료 시간을 개선하기 위해 처리기간의 통신시간을 최소화함으로써 서브 타스크의 중복 할당 알고리즘을 제안하였다. 할당결과, 중복 전후의 처리기간 통신시간을 줄일 수 있었고, 처리기 이용률 및 부하 균등도 개선되었다.

향후 연구과제로는 보다 일반적인 타스크 모델들을 적용하여 알고리즘들을 시뮬레이션하고, 비주기 타스크를 고려한 중복 방법과 다양한 시뮬레이션 환경을 고려한 소프트웨어 툴을 개발하는 것이다.

참고문헌

[1] Ben Kao and Hector Garcia-Molina, "Deadline Assignment in a Distributed Soft Real-Time System", TR, Department of Computer Science, University of Stanford at stanford, April 1993.

[2] Ben Kao and Hector Garcia-Molina, "Subtask Deadline Assignment for Complex Distributed Soft Real-Time tasks", The 14th ICDCS, Poznam, Poland, June 1994.

[3] Gyung-Leen Park, Behrooz Shirazi and Jeff Marquis, "DFRN: A New Approach for Duplication Based Scheduling for Distributed Memory Multiprocessor Systems", In the Proceedings of 11th International Parallel Processing Symposium, pp. 157-166, 1997.

[4] Jose Javier Gutierrez Garcia and Michael Gonzalez Harbour, "Optimized Priority Assignment for Tasks and Messages in Distributed hard real-time systems", IEEE pp.124-132, 1995.

[5] Krithi Ramamritham, "Allocation and Schedu ling of Precedence-Related Periodic Tasks", IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No. 4, April 1995.

[6] Mark H. Klein, John P. Lehoczky, and Ragunathan Rajkumar, "Rate-Monotonic Analysis for Real-Time Industrial Computing", IEEE Transactions on Computer, pp. 24-32, January 1994.

[7] Ramamritham, K., "Allocation and Scheduling of Complex Periodic Tasks", IEEE 10th International Conference on Distributed Computing Systems, pp. 108-115, 1990.

[8] Rhan Ha and Jane W.S. Liu, "Validating Timing Constraints in Multiprocessor and Distributed Real-Time Systems", In the Proceedings of IEEE 14th International Conference on Distributed Computing Systems, 1994.

[9] Sheng-Tzong Cheng and Ashok K.Agrawala, "Allocation and Scheduling of Real-Time Periodic Tasks with Relative Timing Constraints", TR, Department of Computer Science, University of Maryland at College Park, January 1995.

[10] Too-Seng Tia and Jane W. S. Liu, "Assigning Real-Time Tasks and Resources to Distributed Systems", TR, Department of Computer Science, University of Illinois at Urbarna, 1994.

[11] 정경훈, "분산 및 병렬 시스템에서 타스크 중복 스케줄링 알고리즘 연구", 석사학위논문, 부경대학교 전자계산학과, 2월 1998.