이질형 다중 프로세서 시스템에서 휴리스틱 기법을 이용한 타스크 할당 알고리즘

임 선 호† · 이 종 성†† · 채 수 환†††

요 약

동질형 다중 프로세서 시스템에서는 시스템의 성능을 향상시키기 위하여 타스크 수를 가능한 한 균등하게 배분하는 타스크 할당 알고리즘이 일반적으로 사용되고 있다. 그러나, 이질형 다중 프로세서 시스템에서는 이런 알고리즘에 의해 효과적인 타스크 할당이 이루어질 수 없다. 따라서, 이질형 다중 프로세서 시스템에서는 JSQ(Join the Shortest Queue) 알고리즘이 일반적으로 사용되고 있다. 그러나 JSQ 알고리즘은 프로세서 간에 타스크의 처리 능력의 차이가 클 경우에는 효율적이지 못하다. 본 논문에서는 타스크의 도착 시간, 타스크의 서비스 시간, 수행되어진 타스크의 수 등의 동적 데이터에 의해 습득된 프로세서의 처리 능력과 로컬 큐(local queue)의 길이를 동시에 고려한 휴리스틱(heuristic) 타스크 할당 알고리즘을 제시한다. 시뮬레이션 결과, 제안한 휴리스틱 할당 알고리즘에 의해 시스템 성능을 크게 향상 시킬 수 있음을 보여 주었다.

Task Allocation Algorithm for Heterogeneous Multiprocessor Systems Using Heuristic Technique

Sun-Ho Lim† · Jong-Sung Lee†† · Soo-Hoan Chae†††

ABSTRACT

In homogeneous multiprocessor systems, the task allocation algorithm which equally assigns tasks to processors if possible, is generally used. But this algorithm is not suitable to accomplish effective task allocation in heterogeneous multiprocessor systems. JSQ (Join the Shortest Queue) algorithm is often used in heterogeneous multiprocessor systems. Unfortunately, JSQ algorithm is not efficient when the differences of capabilities of processors are far large. To solve this problem, we suggest a heuristic task allocation algorithm that makes use of dynamic information such as task arrival time, task service time, and number of finished tasks. The results of simulation show that the proposed heuristic allocation algorithm improves the system performance.

1. 서 론

타스크 할당 문제(task allocation problem)는 활발히 연구되어온 분야 중에 하나이며 OR(Operating Research), 통신(communication), 네트워크(network), 분산 시스템(distributed system), 분산 데이터 베이스(distributed database), 그리고 공유 메모리 다중 프로세서 시스템(shared memory multiprocessor system, 이하 다중 프로세서 시스템) 등에서 폭 넓게 응용 되고있다. 분산 시스템에서 타스크 할당 문제는 부하 분

† 준 회 원 : Pennsylvania State Univ. Dept. of Computer Science & Engineering 대학원
†† 준 회 원 : 한국항공대학교 대학원 컴퓨터공학과
††† 정 회 원 : 한국항공대학교 컴퓨터공학과 교수

논문접수 : 1998년 9월 14일, 심사완료 : 1999년 1월 29일

배(load sharing)와 부하 균등(load balancing)은 분배 시스템의 성능을 최적화 하는데 사용된다. 특히, 다중 프로세서 시스템에서 타스크 할당 문제는 시스템의 전체적인 성능에 영향을 미칠 수 있기 때문에 부하 분배[1], 부하 균등[2], 그래프 이론 방법[3][4], 그리고 유전자(genetic) 알고리즘[5][6][7] 등 다양한 이론과 방법을 이용한 연구가 활발하게 진행되고 있다.

다중 프로세서 시스템의 성능에 영향을 줄 수 있는 요소는 크게 작업 분할(work partitioning), 데이터 지역성(data locality), 작업 분배(work distribution)로 구분될 수 있다[8].

작업 분할 문제에서는 병렬 수행할 타스크를 정하는 것과 타스크의 크기를 정하는 것이 중요하다. 데이터 지역성 문제에서는 다중 프로세서 시스템의 성능이 메모리 시스템에 의존적이므로 데이터 할당(data allocation) 방법이 중요하다. 작업 분배 문제에서는 처리되어질 타스크를 이상적으로 모든 프로세서에 배분해야 하므로 시스템의 성능을 높이기 위해 타스크 할당(task allocation) 방법이 중요한 문제가 된다.

프로세서 p_j에 타스크 t_i를 할당하는 일은 식 (1)과 같이 함수로 표현할 수 있다.

$$f : t_i \rightarrow p_j \qquad (1)$$

여기에서 $0 \le i < T$ (T는 타스크 수), $0 \le j \le P$ (P는 프로세서 수)

식 (1)에서 T개의 타스크가 P개의 프로세서에 할당될 수 있는 가지 수는 P^T이므로, 이중에서 최적의 할당 경우를 찾아 내는 것은 프로세서 수 P와 타스크 수 T가 커질수록 더욱 어려워지며 이는 NP-hard 문제로 알려져 있다[3][9]. 또한, 다중 프로세서 시스템을 구성하는 각각의 프로세서들이 이질적인 경우, 시스템의 성능을 최적으로 하기 위한 타스크 할당 문제는 더욱 어렵다. 그러므로, 본 논문에서는 sub-optimal을 추구하며 타스크가 프로세서의 로컬 큐에 대기하는 시간을 고려한 상대적인 응답 시간(response time)을 최소로 하는 휴리스틱 타스크 할당 알고리즘을 제시한다. 상대적인 응답시간은 4장에서 정의한다.

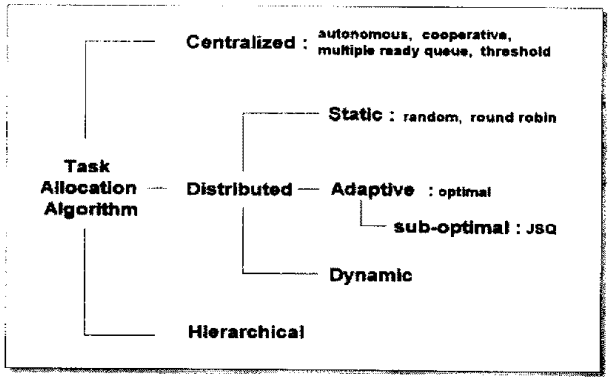
본 논문의 구성은 다음과 같다. 먼저 2장에서는 관련 연구에 대하여 살펴보고, 3장에서는 본 논문에서 제시하는 휴리스틱 타스크 할당 알고리즘을 다루며, 4장에서는 시뮬레이션을 통해 기존의 알고리즘과 본 논[illegible] 하고 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

지금까지 연구된 다중 프로세서 시스템에 타스크를 할당하는 방법은 (그림 1)과 같이 글로벌 큐(global queue)을 이용한 타스크 할당 방법과, 로컬 큐(local queue)를 이용한 타스크 할당 방법, 그리고 글로벌 큐와 로컬 큐를 함께 사용하는 계층 큐(hierarchical queue)를 이용한 타스크 할당 방법으로 대별할 수 있다.

한편, 로컬 큐(local queue)를 이용한 타스크 할당 방법은 미리 정해진 일련의 법칙(rule)에 의하여 할당하는 정적 타스크 할당(static task allocation)과 프로세서와 시스템의 상태에 관한 정보를 통해서 할당하는 동적 타스크 할당(dynamic task allocation)으로 다시 구분할 수 있다

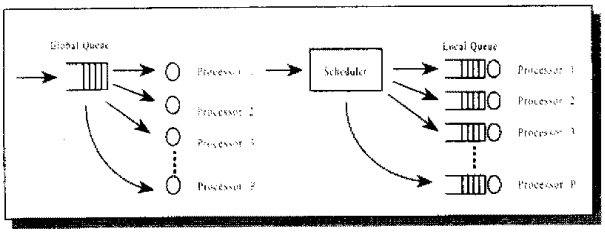
(그림 1)의 분류를 통해 살펴본 종래의 타스크 할당 방법들의 특징을 간단히 살펴보면 다음과 같다.



(그림 1) 타스크 할당 알고리즘의 분류
(Fig. 1) Classification of Task Allocation Algorithms

글로벌 큐를 사용하는 경우는 (그림 2)의 (a)와 같이 시스템의 모든 프로세서들 중 자신의 현재 상태가 유휴(idle) 상태인 프로세서가 글로벌 큐를 접근하여 타스크를 수행한다. 그러므로 자연스럽게 부하 분배(load sharing)가 이루어지나, 프로세서의 수가 큰 시스템에서는 프로세서가 글로벌 큐를 접근하는데 다른 프로세서와 접근 충돌(access contention)이 증가하게 되어 대기시간이 길어지는 병목(bottleneck) 현상이 발생한다. 이와 같은 문제는 동질형이나 이질형 프로세서에서 동일하게 발생하고, 프로세서간의 접근 충돌을

최소화 하기 위해 글로벌 큐의 접근 횟수를 줄이는 방법들이 제시 되었다[10][11][12][15][16] .

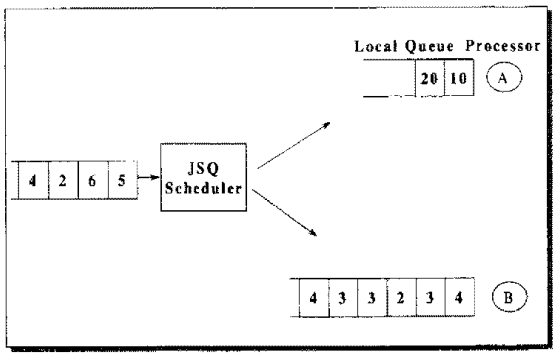


(그림 2) 타스크 할당 방법
(Fig. 2) Task Allocation Methods

글로벌 큐가 시스템의 병목이 되는 것을 방지 하기 위하여 (그림 2)의 (b)와 같이 로컬 큐를 사용한다. 각 프로세서는 로컬 큐를 가지고 있고, 처리될 타스크를 각 프로세서에게 할당을 전담하는 스케줄러가 있다. 스케줄러는 처리될 타스크를 일정한 타스크 할당 정책이나 알고리즘에 따라 적절히 프로세서에게 할당한다. 그러므로 스케줄러는 도착하는 타스크에 대해 언제 어느 프로세서에게 보낼 것인지 결정하거나 처리속도가 빠른 프로세서가 나타날 때까지 가지고(hold) 있을 수 있다. 동질형 시스템에서는 타스크를 가지고 있는 것이 이득이 되지 않지만, 이질형 시스템에서는 가용한 프로세서가 상대적으로 처리속도가 낮을 때 스케줄러가 타스크를 가지고 있는 편이 더욱 효율적일 경우가 있다. 스케줄러가 타스크 할당 시 처리속도가 빠른 프로세서가 나타날 때까지 기다려서 처리하는 것과 다소 처리속도가 늦은 프로세서에게 할당하여 처리하는 것 사이에는 트레이드 오프(trade-off) 관계가 있다[13].

일반적으로 타스크들을 각 프로세서에 할당하는데 있어, 어느 프로세서의 로컬 큐에는 처리를 기다리는 타스크가 많고 다른 프로세서에는 큐에 타스크가 없는 부하 불균형(load imbalance)이 발생할 수 있다. 그러므로, 프로세서간 부하 불균형으로 인해 시스템의 성능이 떨어지는 것을 줄이기 위해 타스크 할당 정책이나 알고리즘이 매우 중요하며 이에 대한 연구가 활발히 진행되고 있으나 대부분의 연구는 동질형 다중 프로세서 시스템을 기반으로 하고 있다[10][11][12][14] .

한편, 최근에 제시된 이질형 다중 프로세서 시스템의 타스크 할당 알고리즘인 JSQ(Join the Shortest Queue) 알고리즘은 스케줄러가 처리될 타스크를 로컬 큐 길이가 가장 짧은 프로세서에게 할당하는 방법이다[17][18]. 이 방법은 스케줄러가 각 프로세서의 로컬 큐 길이에 대한 테이블을 유지해야 하며 타스크를 할당할 때마다 테이블을 이용하여 최소의 로컬 큐 길이를 가지고 있는 프로세서를 찾는다.



(그림 3) JSQ 알고리즘에 의해 이질형 다중 프로세서들에 타스크 할당
(Fig. 3) Task Allocation at Heterogeneous Multiprocessors by JSQ Algorithm

그러나, JSQ 알고리즘은 프로세서의 성능을 고려하지 않고 단지 로컬 큐 길이만 고려하여 스케줄링을 하므로 다음과 같은 문제점이 있다.

즉, 스케줄러가 어떤 프로세서에 타스크를 할당할 때 현재의 모든 프로세서들의 로컬 큐를 조사하여 가장 짧은 로컬 큐에 할당하는 것이 가장 빠른 서비스를 받을 수 없을 수도 있다는 것이다.

예를 들어, (그림 3)에 도시된 바와 같이 처리능력이 1인 프로세서 A의 로컬 큐에 서비스 시간이 각각 10,20인 타스크들이 대기하고 있고, 처리능력이 4인 프로세서 B의 로컬 큐에 서비스 시간이 각각 4,3,2,3,3,4인 타스크들이 대기하고 있다고 가정할 경우 JSQ 알고리즘을 사용하여 타스크를 할당할 때 프로세서 성능을 고려하지 않고 프로세서 A에 타스크를 할당하므로, 만일 서비스 시간이 적은 타스크(이를테면, 5,6,2,4)가 많이 발생하면 프로세서 A에 일시적으로 타스크들이 편중되는 문제점이 발생하여 균등한 타스크 스케줄링을 할 수 없다는 문제점이 있었다.

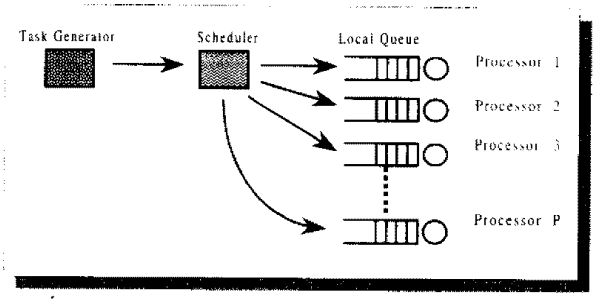
3. 휴리스틱 타스크 할당 알고리즘

본 논문에서 사용한 모델은 (그림 4)와 같으며 타스크 생성기, 스케줄러, 이질형 프로세서, 그리고 로컬 큐들로 구성된다. 타스크 생성기에서 서비스 시간의 차이가 큰 타스크를 생성하기 위해 도착 간격이 지수 분

부를 바탕으로, 요구하는 서비스 시간은 랜덤 함수를 사용한다. 스케줄러는 본 논문에서 제시한 휴리스틱 타스크 할당 알고리즘(이하 휴리스틱 알고리즘)에 따라 도착하는 타스크를 적절한 프로세서에 할당하고, 이때 프로세서의 수를 P라고 한다. 각 프로세서들은 독립적인 처리능력을 갖으며, 처리될 타스크가 임시 저장되는 로컬 큐를 가지고 있고 선입 선출(FCFS)을 따른다. 이때 큐의 길이에 대한 제한은 고려하지 않는다. 각 프로세서들은 이질형이고 이것은 타스크에 대해 처리 능력이 다름을 나타내며 식 (2)와 같이 표현할 수 있다[13].

$$\mu_i \neq \mu_j, \quad i \neq j \tag{2}$$

여기에서, μ는 각 프로세서의 처리능력을 나타내며, $0 < i, j \leq P$이다.



(그림 4) 타스크 할당 모델
(Fig. 4) Task Allocation Model

본 논문에서 제시하는 휴리스틱 알고리즘은 타스크의 도착 시간, 타스크의 서비스 시간, 수행되어진 타스크의 수 등의 동적 데이터를 이용한다. 이를 바탕으로 스케줄러는 각 프로세서의 처리 능력과 로컬 큐(local queue)의 길이를 동시에 고려하여 각 프로세서의 상태에 대한 가중치를 부여하고, 이에 따라 타스크를 프로세서에 차등 분배한다. 각 프로세서의 상태에 대한 가중치는 주기적으로 갱신된다.

프로세서의 상태에 대한 가중치를 부여하기 위하여, 스케줄러는 프로세서의 타스크 처리 능력과 대기 큐 길이를 동시에 고려한다. 스케줄러는 도착하는 타스크에 대하여 프로세서에 할당한 시간(allocation time)을 기록하고, 프로세서는 할당된 타스크를 처리하고 처리 종료 시간(finish time)을 스케줄러에게 전송한다. 초기에는 스케줄러가 프로세서에 대한 타스크 처리 능력에 대한 정보를 갖지 못하므로 타스크를 차등 분배할 수 없다. 그러므로 타스크를 프로세서에 라운드 로빈 방식으로 할당한다. 각 프로세서들은 이질형이므로 단위 시간에 처리하는 타스크의 수가 다르다. 각 프로세서가 처리한 타스크 수의 상대적 비율을 계산하여 이를 각 프로세서 성능으로 인식한다. 스케줄러는 초기에 프로세서의 성능을 예측할 수 없지만, 처리되는 타스크의 수가 증가할수록 점점 더 정확하게 예측할 수 있다. 그리고 각 프로세서는 주기적으로 스케줄러의 요청을 받아서 현재 자신의 로컬 큐에 쌓여 있는 타스크의 수를 스케줄러에게 전송한다.

위의 처리 과정을 통하여 스케줄러는 각 프로세서별로 타스크 처리능력(W)과 로컬 큐 길이(L)를 알 수 있다. 이러한 정보를 식 (3)을 이용하여 계산함으로써 스케줄러는 각 프로세서의 상태(PS)를 인식할 수 있다.

$$PS_i = W_i - C * L_i \tag{3}$$

여기에서, C 는 임의의 가중치이고 $0 < i \leq P$

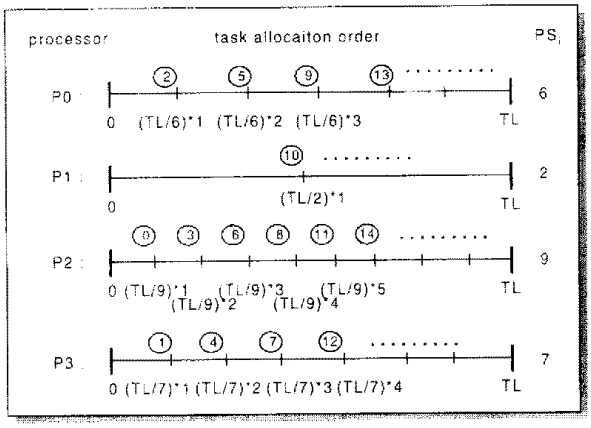
식 (3)에서 가중치 C는 각 프로세서의 PS를 계산하는데 있어 프로세서의 W 와 L에 대한 고려 정도를 조절한다. 다음 4가지 상태에 의해 식 (3)을 고려한다.

1) W가 높고, L이 많은 경우
2) W가 높고, L이 적은 경우
3) W가 낮고, L이 많은 경우
4) W가 낮고, L이 적은 경우

만약 가중치 C를 $C \geq 1.0$으로 하면, 스케줄러는 프로세서의 PS를 계산하는데 L에 대한 비중을 높인 결과가 된다. 그러므로, 한 프로세서의 타스크 처리 능력이 좋아도 자신의 로컬 큐에 대기하는 타스크의 수가 많다면 PS값은 낮아지게 되어 작은 수의 타스크가 할당이 된다. 이 결과, 스케줄러는 프로세서의 PS에 따라 타스크 처리 능력이 조금은 떨어져도 로컬 큐에 대기하는 타스크의 수가 적은 프로세서에 타스크를 할당할 수 있다. 반대로, 가중치 C가 $C < 1.0$ 이면 스케줄러는 프로세서의 W에 대한 비중을 높이게 된다. 그러므로 프로세서간의 로컬 큐에 대기하는 타스크가 서로 크게 차이가 나지 않으면, 프로세서의 W가 상대적으로 높은 프로세서가 높은 PS를 갖게 된다.

스케줄러는 식 (3)을 통하여 얻은 각 프로세서의 PS를 부하 분배를 하는데 있어서 프로세서의 타스크 할당 가중치로 사용한다. 이를 바탕으로 스케줄러는 타스크를 각 프로세서에게 동적으로 할당한다. 예를 들어 P = 4인 경우, 프로세서의 PS의 비율이 $PS_0 : PS_1 : PS_2 : PS_3 = 6 : 2 : 9 : 7$이라고 하자. 프로세서

의 PS에 따라 타스크를 자동 분배하기 위하여 (그림 5)와 같이 특정 시간 간격(TL)을 각 프로세서의 PS로 분할한다. 프로세서 P2와 같이 PS가 높은 프로세서는 분할 간격이 촘촘하며, 반대로 프로세서 P1과 같이 PS가 낮은 프로세서는 분할 간격이 넓다. 타스크가 할당될 프로세서의 순서는 TL을 각 프로세서의 PS로 나눈 값들과 비교하여 오름차순으로 정렬한 순이다. 만약 2개 이상의 프로세서가 같은 위치에 있다면 프로세서의 오름차순으로 순서를 정한다. 따라서 PS가 높은 프로세서에게는 타스크가 빈번히 할당되고, PS값이 낮은 프로세서에게는 간헐적으로 할당된다.



(그림 5) PS에 따른 타스크 할당 순서
(Fig. 5) Order of Task Allocation according to PS

스케줄러는 일정 수의 타스크가 처리될 때마다 각 프로세서의 PS를 갱신한다. 갱신한 각 프로세서의 PS는 다음 갱신 시기까지 타스크 할당 기준으로서 사용된다. 스케줄러가 PS 갱신 주기를 짧게 할 경우, 프로세서의 PS에 따른 타스크의 차별 할당 효과를 미처 보기 전에 갱신되므로 성능의 증가를 볼 수 없다. 반대로, 스케줄러가 PS 갱신 주기를 길게 하는 경우는, 설정된 각 프로세서의 PS를 타스크 할당 기준으로서 너무 오래 사용하므로 부하 불균형을 초래할 수 있다. 본 논문에서는 시뮬레이션을 통하여 처리한 타스크의 수(TN)가 100 개일 때마다 각 프로세서의 PS를 갱신하였을 때 성능이 가장 우수하였다.

위에서 설명한 과정을 알고리즘으로 표현하면 (그림 6)과 같다.

```
PROCEDURE : HEURISTIC_TASK _ALLOCATION ( )
P : total number of processors
Qn : local queue length of processor
T : total number of tasks
TN : total number of processed tasks
TNPPn : total number of processed tasks per processor
IN : initial number of processed tasks
UI : update interval of processor state
{
  while ( T > TN ) {
    /* determine the processors to execute tasks. */
    if ( TN < IN) {
      scheduler allocates tasks to processors by the round
      robin method ;
    }
    else if ( (TN mod UI) == 0 ) {
      scheduler sends messages to send Qn ;
      scheduler calculates and updates states of processors
      by use of Qn and TNPPn ;
    }
    else {
      scheduler determines processors to allocate tasks
      according to the estimated state values of
        processors ;
    } ; /* end if */
    /* each processor receives a task and processes it */
    if (a task is arrived) {
      scheduler checks the task's arrival time ;
      scheduler allocates the task to the processor with the
      highest state value ;
    } ; /* end if */
    for (i = 0 ; i < P ; i ++ ) {
      each processor executes a task in its local queue ;
      if (a processor completes a task) {
        the processor sends the finish time of task to the
        scheduler ;
        the scheduler receives the finish time from the
        processor and updates the TNPPn ;
      } ; /* end if */
    } ; /* end for */
  } ; /* end while */
} /* end procedure */
```

(그림 6) 휴리스틱 타스크 할당 알고리즘
(Fig. 6) Heuristic Task Allocation Algorithm

4. 시뮬레이션

본 논문에서 사용한 성능 평가의 기준은 상대적인 응답 시간(RT : Response Time)[1]으로 정하며 식 (4)

1) 상대적인 응답시간이란 모든 타스크가 프로세서들에 할당되어서 큐에 대기한 후 처리가 완료되는데 소요되는 전체 처리시간에서 각각의 타스크 서비스 시간(즉, 순수 프로세서 처리 시간)을 나누어 합한 값으로 결국 적용된 알고리즘을 통해 타스크들을 할당하여 처리할 때 대기 큐에서 소비되는 시간의 중요성을 부각시키기 위해 적용된 정의이다.
이는 종래의 평균 응답시간이 단지 타스크 도착시간과 타스크 완료시간만을 고려하고 대기 큐에서 소비되는 시간을 고려하지 않으므로 발생하는 즉, 큐 대기시간이 길고 처리시간이 짧은 타스크를 처리하는 프로세서와 대기시간은 짧으나 처리시간이 긴 타스크를 처리하는 프로세서를 분리하여 성능을 평가하지 못하는 문제점을 극복하기 위함이다.

와 같다.

$$RT = \sum_{i=0}^{T}\left(\frac{TR_i - TA_i}{L_i}\right) \quad (4)$$

여기에서, i = 0, 1, 2, ... T (T는 타스크 수)

TR은 프로세서가 한 타스크의 처리를 끝내고 스케줄러에게 메시지를 보낸 시간이고, TA는 프로세서의 로컬큐에 타스크가 도착한 시간이며, L은 타스크 서비스 시간이다. TR에는 식 (5)와 같이 큐 대기시간(WT : Waiting Time)과 프로세서에서 처리 시간(ET : Execution Time)이 포함되어 있다.

$$TR = WT + ET \quad (5)$$

여기에서, i = 0, 1, 2, ... T (T는 타스크 수)

타스크 할당 알고리즘의 성능을 서로 비교하기 위해 이상적 할당, 랜덤 할당, 그리고 JSQ 할당 알고리즘을 사용하였다. 이상적 할당과 랜덤 할당 알고리즘을 성능의 기준(baseline)으로 정한다. 이상적 할당(IA: Ideal Allocation) 알고리즘에서 스케줄러는 각 프로세서들의 성능과 시스템의 상태를 사전에 알고 있다고 가정하고, 타스크가 도착함에 따라 지금까지 응답 시간이 가장 짧은 프로세서에게 할당 하며, 타스크가 프로세서의 로컬 큐에 쌓임에 따라 각 프로세서가 처리할 시간을 계속 유지한다.

본 논문에서 스케줄러는 시뮬레이션 시에 처리될 타스크에 대하여 사전에 정보를 갖고 있지 않다고 가정한다. 타스크는 서로 서비스 시간의 차이가 큰 랜덤 분포를 하며, 스케줄러에 도착하는 타스크의 간격은 식 (6)의 지수 분포(exponential distribution)를 따른다.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & (x \geq 0) \\ 0 & (x < 0) \end{cases} \quad (6)$$

여기에서, λ는 도착 간격 파라미터이며 $\lambda > 0$이다.

각 프로세서들은 타스크 처리 능력이 서로 동일하지 않은 이질형이며 프로세서의 타스크 처리능력은 <표 1>과 같이 랜덤 분포를 따랐고 시뮬레이션 파라미터는 <표 2>와 같다.

본 논문에서는 타스크의 도착 간격 와 프로세서의 수 P를 달리하고 타스크 수 T = 5,000개일 때, 타스크 할당 알고리즘간 응답 시간과 대기 시간을 비교하였다. 여기에서, RRA는 라운드 로빈 할당(RRA : Round Robin Allocation) 알고리즘을, HA는 휴리스틱 할당 (HA : Heuristic Allocation) 알고리즘을 각각 나타낸다.

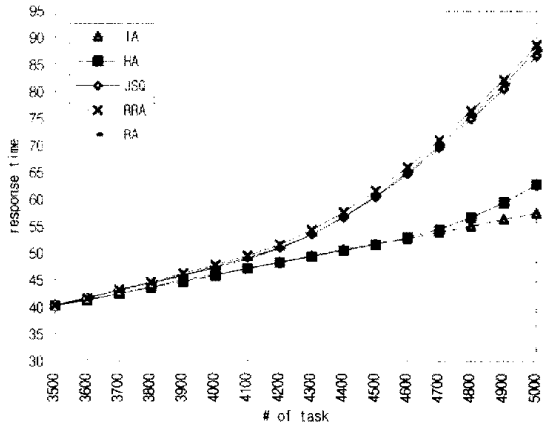
〈표 1〉 프로세서의 타스크 처리 능력
〈Table 1〉 Capabilities of Processors

프로세서	P0	P1	P2	P3	P4	P5	P6	P7
처리능력	3	13	10	17	9	2	10	12
프로세서	P8	P9	P10	P11	P12	P13	P14	P15
처리능력	4	8	18	11	17	10	2	14
프로세서	P16	P17	P18	P19	P20	P21	P22	P23
처리능력	1	8	13	11	3	3	10	8
프로세서	P24	P25	P26	P27	P28	P29	P30	P31
처리능력	10	15	15	7	16	12	3	4

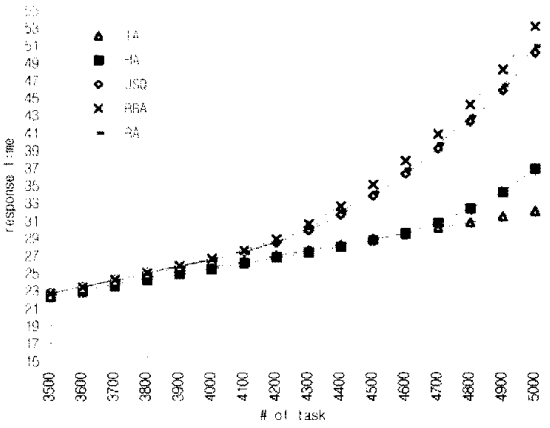
〈표 2〉 매개변수의 값
〈Table 2〉 Parameters for Simulation

프로세서 수	최대 32개
타스크 수	최대 5,000개
타스크 길이	10-5000 사이의 랜덤 값
타스크 도착 분포	지수 분포
타스크간 평균 도착 간격	최대 5
프로세서의 상태 정보 갱신 주기	처리된 타스크가 100일 때마다

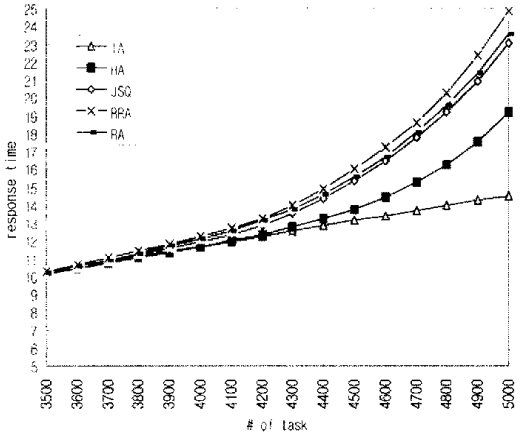
[시뮬레이션 1] λ = 1이고 P = 4, 8, 16, 32인 경우, 각 알고리즘간 응답 시간을 비교한다.



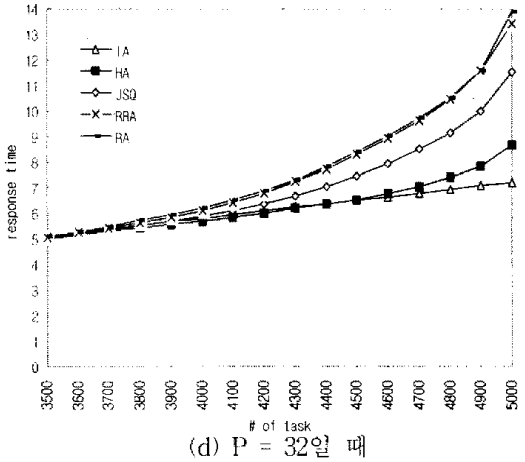
(a) P = 4일 때



(b) P = 8일 때



(c) P = 16일 때

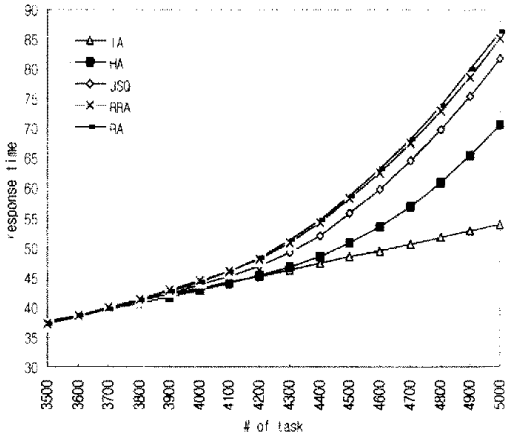


(d) P = 32일 때

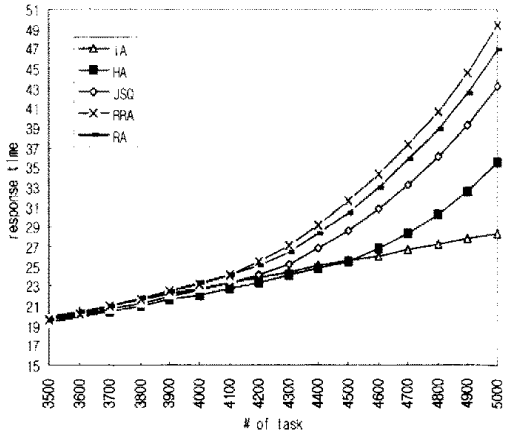
(그림 7) λ = 1일 때, 알고리즘간 응답 시간
(Fig. 7) Response Time in the case of λ = 1

태스크들의 도착 간격이 매우 짧으므로 (그림 7)에서 보인 바와 같이 P가 적은 경우에 JSQ 알고리즘은 RA 나 RRA할당 알고리즘의 응답 시간과 별 차이가 없다. (그림 7)의 (d)와 같이 P가 큰 경우에는 JSQ 알고리즘이 RA와 RRA 할당 알고리즘보다 약간 개선된 것을 볼 수 있다. 하지만, 본 논문에서 제안한 HA는 P와는 상관없이 JSQ 알고리즘보다 우수한 성능을 보이고 IA알고리즘에 근접한 응답 시간을 나타낸다.

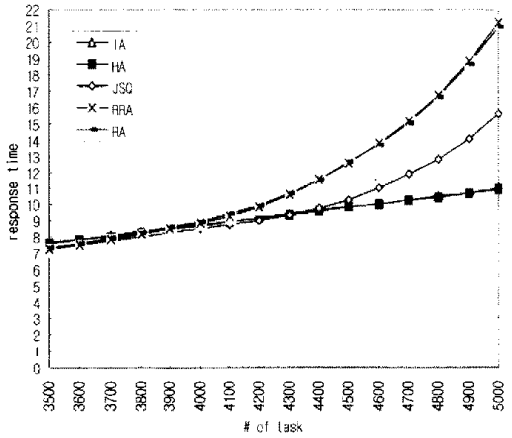
[시뮬레이션 2] λ = 2이고 P = 4, 8, 16, 32인 경우, 각 알고리즘간 응답 시간을 비교한다.



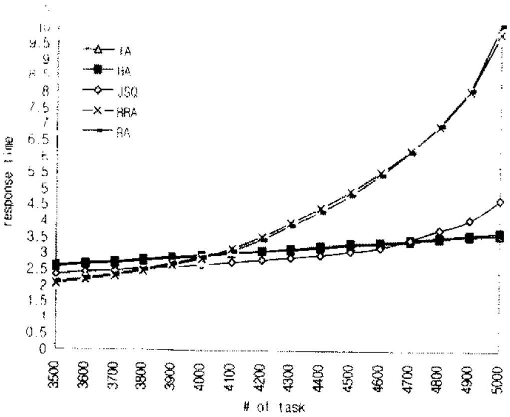
(a) P = 4일 때



(b) P = 8일 때



(c) P = 16일 때

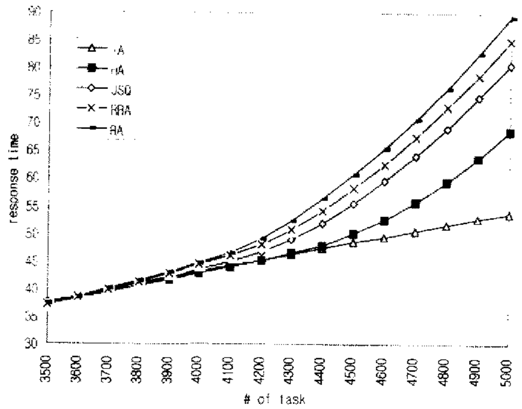


(d) P = 32일 때

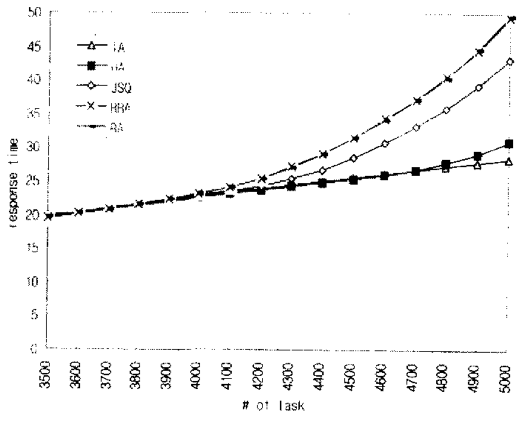
(그림 8) λ = 2일 때, 알고리즘간 응답 시간
(Fig. 8) Response Time in the case of λ = 2

JSQ 알고리즘은 P = 4, 8인 경우에는 RA나 RRA알고리즘과 별 차이가 없고, P = 32인 경우에 IA 알고리즘에 접근한다. 그러나, HA 알고리즘은 (그림 8)과 같이 P와 상관없이 IA알고리즘에 근접한 응답 시간을 나타낸다.

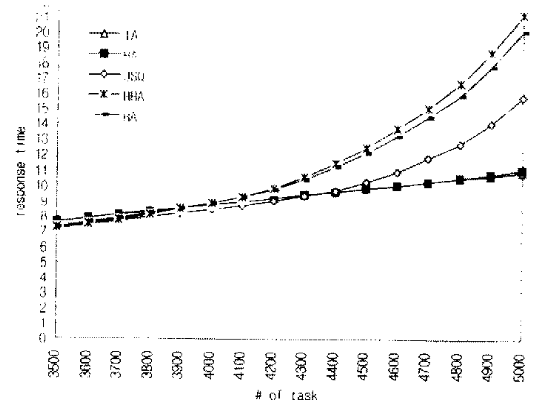
[시뮬레이션 3] λ = 5이고 P = 4, 8, 16, 32인 경우, 각 알고리즘간 응답 시간을 비교한다.



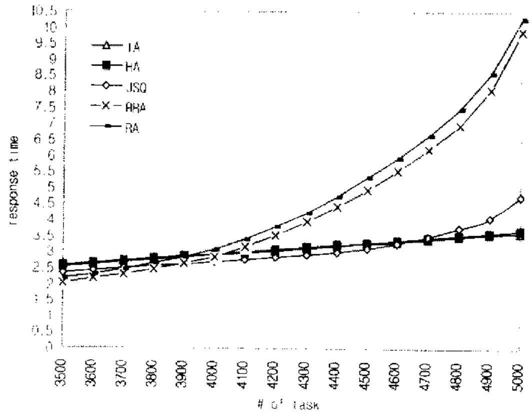
(a) P = 4일 때



(b) P = 8일 때



(c) P = 16일 때

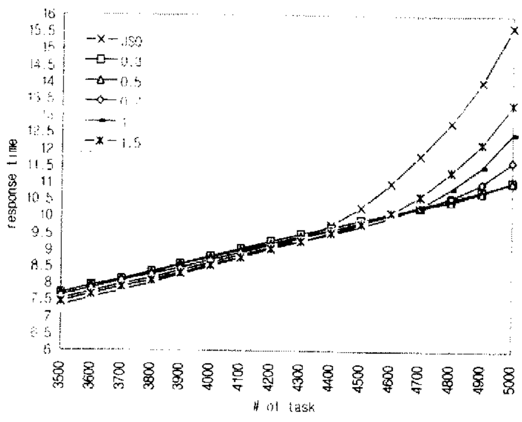


(d) P = 32일 때

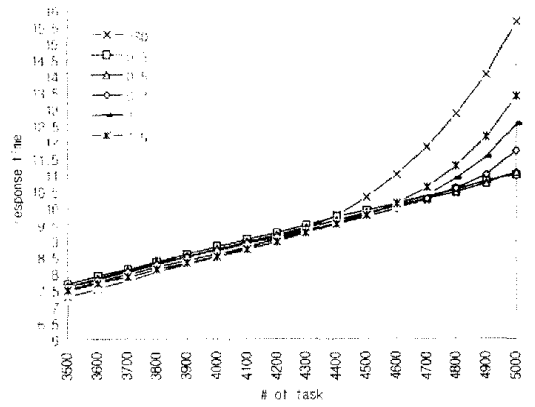
(그림 9) λ = 5일 때, 알고리즘간 응답 시간
(Fig. 9) Response Time in the case of λ = 5

HA 알고리즘은 (그림 9)와 같이 P와 상관없이 IA 알고리즘에 근접한 응답 시간을 나타낸다. [시뮬레이션 1] [시뮬레이션 3]을 통해 HA 알고리즘이 JSQ 알고리즘보다 우수한 응답 시간을 보인을 알 수 있다.

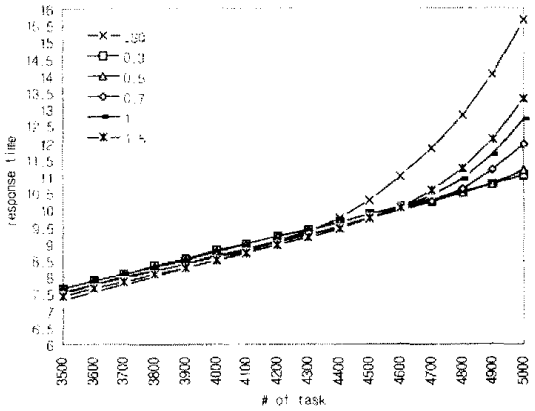
[시뮬레이션 4] λ = 2 이고 P = 16, 그리고 가중치 C 0.3, 0.5, 0.7, 1.0, 1.5일 때, 갱신 주기 UI = 50, 100, 200인 경우 JSQ 알고리즘과 응답 시간을 비교한다.



(a) UI = 50일 때



(b) UI = 100일 때



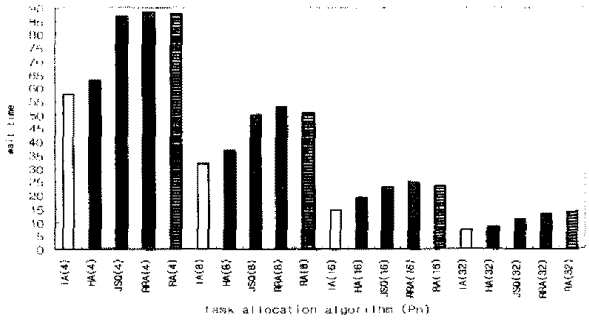
(c) UI = 200일 때

(그림 10) λ = 2, P = 16일 때, 응답 시간
(Fig. 10) Response Time in the case of λ = 2 and P=16

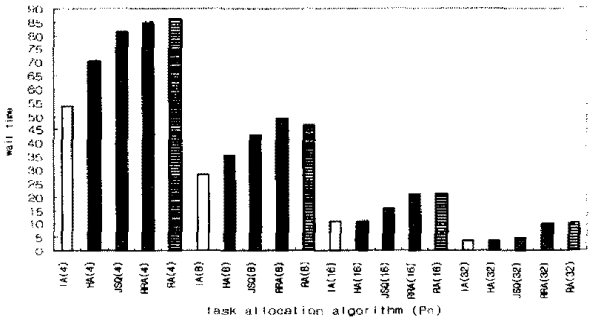
(그림 10)과 같이 전체적으로 HA 알고리즘은 JSQ 알고리즘보다 좋은 응답 시간을 보이며, 가중치 C 값에 따라 응답 시간이 약간의 차이를 보인다. C = 0.3 이고 UI = 100인 경우에 가장 빠른 응답 시간을 보인다. 즉 큐의 길이보다 측정된 프로세서의 능력이 더욱 응답시간에 미치는 영향이 큼을 알 수 있다.

시뮬레이션 3) 타스크 도착 간격이 λ = 1, 2, 5이고 프로세서의 수가 P = 4, 8, 16, 32인 경우, 알고리즘간 대기 시간을 비교한다. (그림 11)에서 괄호 안의 수는 프로세서의 수(즉, P)이다.

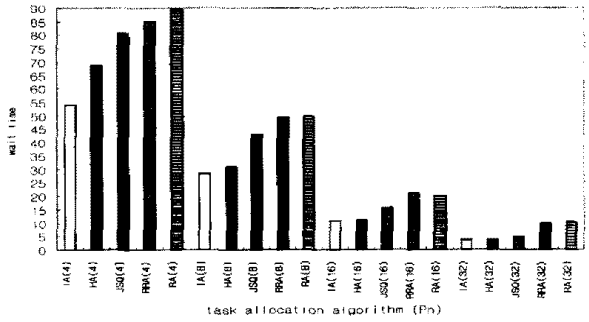
(그림 11)과 같이 JSQ 알고리즘은 타스크 도착 간격이 λ = 1, 2인 경우에 RA나 RRA 알고리즘과 별 차이가 없다. λ = 5인 경우에는 약간의 시간을 줄이는 반면에, HA는 도착 간격에 상관없이 JSQ 알고리즘 보다 낮은 대기 시간을 보이며 적은 프로세서 수에서도 IA 알고리즘의 대기시간에 근접하는 것을 볼 수 있다.



(a) λ = 1일 때



(b) λ = 2일 때



(c) λ = 5일 때

(그림 11) 알고리즘간 대기 시간
(Fig. 11) Waiting Time

초기에 스케줄러가 프로세서의 PS에 대하여 정확히 예측하지 못하지만, 시뮬레이션이 진행되는 동안 각 프로세서에 대한 누적된 정보를 이용하여 점차 최적에 가까운 타스크 할당을 할 수 있음을 알 수 있다. 따라서 제안된 휴리스틱 알고리즘은 시뮬레이션을 통하여 처리되는 타스크의 수가 증가될수록 JSQ 할당 알고리즘보다 우수한 성능을 나타내고 IA 알고리즘에 접근되어가는 것을 알 수 있다.

5. 결 론

기존에 단순히 프로세서의 로컬 큐 길이만을 고려하는 JSQ 알고리즘과 달리 제안한 휴리스틱 알고리즘

고 동적인 정보를 이용하여 각 프로세서의 태스크 처리 능력과 로컬 큐 길이를 동시에 고려하였다. 이를 바탕으로 스케줄러는 각 이질형 프로세서의 상태를 파악하여 태스크를 프로세서에 동적으로 할당함으로써 시스템의 전체적인 성능을 향상시키고 태스크가 로컬 큐에 대기하는 대기시간을 줄였다. 시뮬레이션에서 휴리스틱 알고리즘은 프로세서 수와 상관없이 응답 시간과 로컬 큐에 대기하는 태스크의 대기시간을 줄일 수 있고 이상적 할당 알고리즘의 응답 시간과 대기시간에 근접해간다. 특히, 프로세서의 수가 P≤16인 경우와 태스크의 도착 간격이 작은 경우에 뛰어난 성능을 발휘함을 알 수 있다.

결과적으로 프로세서 간에 태스크의 처리 능력이 크게 차이가 나는 이질성 다중 프로세서 시스템에서는 휴리스틱 알고리즘이 JSQ 알고리즘보다 더욱 적합함을 알 수 있다. 또한 본 논문에서 제안한 휴리스틱 알고리즘은 다양한 다중 처리 시스템, 분산 처리 시스템, 그리고 병렬 처리 시스템에 적용이 가능하다.

참 고 문 헌

[1] Yung-Terng Wang, Robert J.T. Morris, "Load Sharing in Distributed Systems," IEEE Transactions on Computers, Vol.c 34, No.3, pp.204-217, March 1983.

[2] Lionel M. Ni, Kai Hwang, "Optimal Load Balancing Strategies for A Multiple Processor System," IEEE Computer, pp.352-357, 1981

[3] Virginia Mary Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," IEEE Transactions on Computer, Vol.37, No.11, pp. 1384 1397, November 1988.

[4] Wesley W. Chu, Leslie J. Holloway, Min Tsung Lan, Kemal Efe, "Task Allocation in Distributed Data Processing", IEEE Computer, Vol.13, No.11, November, 1980.

[5] Edwin S. H. Hou, Nirwan Ansari, Hong Ren, "A Genetic Algorithm for Multiprocessor Scheduling," IEEE Transaction on Parallel and Distributed Systems," Vol.5, No.2, pp.113 120, February, 1994.

[6] Anup Kumar, Rakesh M. Pathak, Yash P. Gupta, "Genetic Algorithm based Approach for File Allocation on Distributed Systems," Computers & Operations Research, Vol.22, No.1, pp. 41-54, 1995.

[7] Venkateswaran, Zoran Obradovic, C. S. Raghavendra, "Cooperative Genetic Algorithm for Optimization Problems in Distributed Computer Systems," Technical Report TR-EECS-93-018, School of EECS, Washington state Univ., 1993.

[8] Per Stenstrom, Fredrik Dahlgren, "Applications for Shared Memory Multiprocessors," IEEE Computer, Vol.29, No.12, pp.29-31, December, 1996.

[9] Sol M. Shatz, Jia-Ping Wang, Masanori Goto, "Task Allocation for Maximizing Reliability of Distributed Computer Systems," IEEE Transaction on Computer, Vol.41, No.9, September, 1992.

[10] Lionel M. Ni, Ching-Farn E. Wu, "Design Tradeoff for Process Scheduling in Shared Memory Multiprocessor Systems," IEEE Transaction on Software Engineering, Vol.15, No.3, pp.327 334, March, 1989.

[11] Sivarama P. Dandamudi, "Reducing Run Queue Contention in Shared Memory Multiprocessors," IEEE Computer, Vol.30, No.3, pp.82-89, March, 1997.

[12] Sivarama P. Dandamudi, Philip S. P. Cheng, "A Hierarchical Task Queue Organization for Shared Memory Multiprocessor Systems," IEEE Transactions on Parallel and Distributed Systems, Vol.6, No.1, pp.1-16, January, 1995.

[13] Scott Shenker, Abel Weinrib, "The Optimal Control of Heterogeneous Queuing Systems : A Paradigm for Load Sharing and Routing," IEEE Transaction on Computer, Vol.38, No.12, pp.1724-1735, December, 1989.

[14] Seung W. Yoo, "Dynamic Scheduling Algorithms in a Homogeneous Multiple Processor System," Journal of the Korea Information Science Society, Vol.16, No.1, pp.93-102, January, 1989.

[15] R. Nelson, M. Squillante, "Analysis of Contention in Multiprocessor Scheduling," Proc. Int'l

Symp. Computer System Modeling, Measurement and Evaluation, North-Holland, Amsterdam, pp.391-405, 1990.

[16] Venkat J. Malla, Sayed A. Banawan, "Threshold Policies for Load Sharing in Heterogeneous Systems," Phoenix Conference on Computers and Communications, pp.100-105, Conference Proceedings 1991.

[17] Hwa-Chun Lin, C. S. Raghavendra, "Approximating the Mean Response Time of Parallel Queues with JSQ Policy," Computers & Operations Research, Vol.23, No.8, pp.733-740, 1996.

[18] Hwa-Chun Lin, C. S. Raghavendra, "An Approximate Analysis of the Join the Shortest Queue(JSQ) Policy," IEEE Transactions on Parallel and Distributed Systems, Vol.7, No.3, pp.301-307, March, 1996.

[19] Flavio Bonomi, Anurag Kumar, "Adaptive Optimal Load Balancing in a Nonhomogeneous Multiserver System with a Central Job Scheduler," IEEE Transaction on Computer, Vol.39, No.10, October, 1990.

[20] Niranjan G. Shivaratri, Philip Krueger, Mukesh Singhal, "Load Distributing for Locally Distributed Systems," IEEE Computer, Vol.25, No.12, pp.33-44, December, 1992.

임 선 호

e-mail : slim@cse.psu.edu

1996년 한국 항공대학교 전자계산학과 졸업(이학사)

1998년 한국 항공대학교 컴퓨터공학과 대학원 졸업(공학석사)

1998년~현재 미국 Pennsylvania State Univ. (University Park) Dept. of Computer Science & Engineering 대학원 유학

관심분야 : scheduling, interconnection network, evolution algorithm 등임

이 종 성

e-mail : jslee@hanul.hangkong.ac.kr

1994년 한국 항공대학교 전자계산학과 졸업(이학사)

1996년 한국 항공대학교 전자계산학과 대학원 졸업(이학석사)

1996년~현재 한국 항공대학교 컴퓨터공학과 대학원 박사과정

관심분야 : 병렬/분산처리, High Performance Computing, Computer Security, Intrusion Detection System 등임

채 수 환

e-mail : chae@mail.hangkong.ac.kr

1973년 한국 항공대학교 항공전자공학과 졸업(공학사)

1985년 미국 Univ. of Alabama 전산공학과 졸업(공학석사)

1988년 미국 Univ. of Alabama 전기공학과 졸업(공학박사)

1973년~1977년 공군교육사령부 통신학교 교관

1977년~1983년 금성통신 근무(연구원)

1996년 9월~1997년 8월 영국 Newcastle upon tyne 교환교수

1989년~현재 한국항공대학교 컴퓨터공학과 정교수

1998년~현재 한국항공대학교 컴퓨터신기술연구소장

관심분야 : 컴퓨터 구조, 병렬처리시스템 등임