GRID 시스템을 위한 온라인 스케쥴링 알고리즘
(An On-line Scheduling Algorithm for a GRID System)

김 학 두 † 김 진 석 †† 박 형 우 †††
(Hak Du Kim) (Jin Suk Kim) (Hyoungwoo Park)

요 약 이질적인 계산자원들로 구성된 분산 컴퓨팅 환경에서 의존성이 존재하지 않는 독립적인 작업들을 자원들에 배치하기 위한 방법은 NP-Complete 문제로 알려져 있다[1]. 이질적인 자원으로 구성된 시스템의 대표적인 예가 GRID[2]이다. 현재까지 그리드 시스템에서 스케쥴링 문제를 풀기 위한 다양한 휴리스틱 스케쥴링 방법이 연구되어 왔다[1,3,4,5]. 스케쥴링 방법은 정적인 방법과 동적인 방법으로 나뉘어진다. 동적 스케쥴링 방법은 작업의 선후 관계를 예측할 수 없는 상황에서 사용되며 동적 스케쥴링 방법은 스케쥴링 시기에 따라 온라인방식과 배치방식으로 나뉘어진다[1,6]. 본 논문에서는 새로운 온라인 휴리스틱 스케쥴링 알고리즘을 제안하였으며 제안된 스케쥴링 알고리즘의 성능이 기존의 스케쥴링 알고리즘의 성능보다 뛰어남을 시뮬레이션을 통하여 보였다.

키워드 : GRID, 동적 스케쥴링 알고리즘, 온라인 스케쥴링

Abstract The scheduling problem that maps independent tasks to heterogeneous resources in distributed computing systems is known as NP-complete[1]. GRID[2] is an example of distributed systems that consisted of heterogeneous resources. Many algorithms to solve this problem have been presented[1,3,4,5]. The scheduling algorithm can be classified into static scheduling algorithms and dynamic scheduling algorithms. A dynamic scheduling algorithm can be used when we can not predict the priority of tasks. Moreover, a dynamic scheduling algorithm can be divided into on-line mode algorithm and batch mode algorithm according to the scheduling time[1,6]. In this paper, we propose a new on-line mode scheduling algorithm. By extensive simulation, we can see that our scheduling algorithm outperforms previous scheduling algorithms.

Key words : GRID, Dynamic Scheduling Algorithm, On-line Scheduling

1. 서 론

자연과학 및 공학분야에서 사용되어지는 대규모 시뮬레이션은 많은 계산량을 요구하기 때문에 병렬처리 시스템이나 슈퍼컴퓨터와 같은 고속의 컴퓨터들이 필요하다. 과거에는 이러한 시스템을 일정한 공간에 같은 성질을 가진 자원들로 구성하였다. 그러나 동일한 자원으로 한곳에 시스템을 구성하는 데는 많은 비용이 소요되며 많은 물리적인 공간이 필요하게 된다. 이런 이유로 최근 지역적으로 분산되어 있는 이질적인 자원들을 하나로 묶어 큰 규모의 시스템을 구성하고자 하는 GRID[2] 관련 연구가 진행되고 있다. GRID는 지역적으로 분산되어 있는 대용량의 저장장치, 고성능의 슈퍼컴퓨터, 클러스터링 시스템, 과학 설비 등을 네트워크로 연결하여 큰 규모의 시스템을 만들고 그 시스템을 이용하여 자연과학의 문제들을 해결하기 위한 프로젝트이다. 이러한 시스템을 구성하는 자원들은 특성이 다르기 때문에 같은 작업을 실행하더라도 실행 능력의 차이를 보이게 된다. 예를 들면 입출력 속도가 빠른 자원은 입출력이 많은 작업을 빠르게 수행시킬 수 있다. 그러므로 작업들을 어떤 자원에 배치할 것인가가 문제가 되고 현재까지 이 문제에 대한 여러 연구가 진행되고 있다[1,3,4,5]. 하지만 서로 독립적인 작업들을 이질적인 자원들로 구성된 환경에서 어떻게 배치하여 실행할 것인가 하는 것은 NP-Complete 문제로 알려져 있다[1]. 그래서 많은 연구에서는 휴리스틱 방법을 사용하여 해결하고 있다. 본 논문에서 제안한 알고리즘은 독립적인 작업을 스케쥴링하기 위한 동적 스케쥴링 알고리즘이다. 제2절과 3절에서는 알고리즘을 기술하고 평가하는데 필요한 기본 용

† 비 회 원 : 한국전자통신연구원 연구원
rexxion@etri.re.kr
†† 종신회원 : 서울시립대학교 컴퓨터과학부 교수
jskim@venus.uos.ac.kr
††† 비 회 원 : KISTI 슈퍼컴퓨팅센터 그리드 연구실장
hwpark@hpcnet.ne.kr
논문접수 : 2002년 11월 29일
심사완료 : 2003년 9월 26일

어에 대한 설명과 기존의 동적 스케줄링의 온라인 휴리스틱 알고리즘을 소개하고 간략하게 그 방법을 설명하였다. 제 4절에서는 본 논문에서 제안한 알고리즘을 구체적으로 설명하고 시뮬레이션 및 결과에서는 시뮬레이션을 하기 위한 기본 가정들과 구성 방법들을 기술하였으며 시뮬레이션에 따른 결과를 그래프를 통하여 보이고 분석하였다.

2. 관련용어

이질적인 시스템(heterogeneous system)은 특성이 다른 자원들로 구성된 시스템을 말한다. 동질의 자원으로 구성된 시스템과는 달리 작업의 요구사항에 따라 그 성능의 차이를 보이는 자원들로 구성되어 있다. 알고리즘의 성능을 측정하는 방법으로는 makespan[7]이 사용된다. makespan은 전체 작업 중 마지막 작업이 스케줄링 되었을 때 자원들의 준비시간(ready time) 중 가장 큰 값이다. 그림 1은 본 논문에서 용어들을 그림으로 나타낸 것이다.

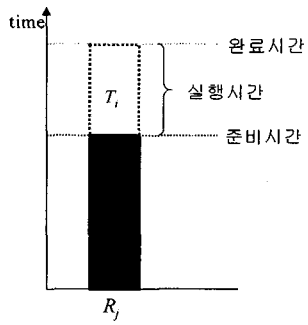


그림 1 실행시간, 완료시간, 준비시간

자원의 준비시간은 그 자원에 주어진 모든 작업의 수행을 마친 후에 대기하는 시간이다. n개의 작업으로 구성된 작업집합을 $T=\{T_1,T_2,...,T_n\}$, m개의 자원으로 구성된 자원집합을 $R=\{R_1,R_2,...,R_m\}$이라고 할 때, 작업 T_i의 자원 R_i에서의 완료시간(completion time)은 T_i를 R_i에서 실행했을 때 작업이 완료되는 시간이며 R_i의 준비시간에 T_i의 R_i에서의 실행시간을 합한 값이며 ct_{ij}로 나타낸다. 이것을 식으로 나타내면 다음과 같다[8].

$$ct_{ij} = b_j + e_{ij}$$

위의 식에서 b_j는 자원 R_i 준비 시간이다. e_{ij}는 작업 T_i을 자원 R_j에서 실행했을 때의 실행시간(execution time)이며 시스템을 구성하는 자원마다 차이를 보인다. 실행시간은 다시 시뮬레이션이 그 작업의 정보를 이용하여 계산한 기대실행시간(expected execution time)과 시스템의 자원에서 실제로 그 작업을 실행했을 때 소요되는 실제실행시간 (actual execution time)으로 나뉘어진다[8]. 시스템을 구성하는 자원들은 그 특성이 제각기 다르며 그 특성은 자원 이질성(resource heterogeneity)으로 나타낼 수 있다. 자원 이질성은 하나의 작업을 여러 자원에서 실행했을 때 나타나는 차이라고 할 수 있다. 이는 상대적인 값으로 1이라는 작업 이질성을 갖는 자원은 10이라는 작업 이질성을 갖는 자원에 비해 10배의 빠른 성능을 보인다. 시스템에 주어지는 작업들 간의 차이는 작업 이질성으로 나타낸다. 작업 이질성은 여러 작업을 하나의 자원에서 실행했을 때 나타나는 차이로써 자원 이질성과 마찬가지로 상대적인 값이다.

3. 관련 연구

본 논문에서는 의존성이 존재하지 않는 독립적인 작업을 다룬다. 독립적인 작업은 내부에 세부작업들로 나누어 질 수 있으며 그 세부작업간에는 의존성이 존재할 수 있다. 또한 스케줄링 시스템은 작업의 스케줄링을 담당하는 위치에 따라 중앙집중적인 시스템과 분산적이 시스템으로 나눌 수 있다. 본 논문에서 제안한 알고리즘은 GRID 연구의 일환으로 개발되고있는 글로벌 큐잉 시스템과 같은 중앙집중적인 시스템에서 사용될 수 있다. 이질적인 자원으로 구성된 환경에서 독립적인 작업의 스케줄링에 관한 연구는 크게 정적 스케줄링 방식과 동적 스케줄링 방식으로 나눌 수 있다[6]. 정적 스케줄링 방식은 작업의 선후관계와 자원의 변동이 없는 환경에서 사용되는 방법이며 동적 스케줄링 방식은 작업들 간의 선후관계가 불분명할 때 사용된다. 동적 스케줄링 방법은 스케줄링 시기에 따라 작업이 도착하는 즉시 스케줄링하는 온라인방식과 간격을 두고 그 간격에 도착한 작업들을 모두 스케줄링하는 배치방식으로 나눌 수 있다. 온라인방식의 알고리즘에는 MET(Minimum Execution Time), MCT(Minimum Completion Time), SA(Switching Algorithm), KPB(K-Percent Best) 등이 있다[8]. 본 논문에서는 MET, MCT, KPB 알고리즘과 본 논문에서 제안한 알고리즘을 비교하였다. MET는 작업의 실행시간의 기대 값이 가장 작은 자원을 찾아 그 자원에 작업을 할당한다. 즉, T_i를 스케줄링할 때, R에서 e_{ij}가 가장 작은 자원을 찾아 T_i에 할당한다. MET는 실행시간이 가장 작은 자원을 작업에 할당하기 때문에 자원사용의 불균형을 초래할 수 있다. MET 스케줄링의 시간 복잡도는 $O(m)$이다. MCT는 해당 작업에 대하여 완료시간이 가장 작은 자원을 작업에 할당한다. 즉, T_i을 스케줄링할 때 그 작업에 대하여 R에서 ct_{ij}가 가장 작은 자원을 T_i에게 할당한다. MCT는 간단하면서도 좋은 성능을 보여주며 이 분야에 관한 연구에서 새로운 알고리즘을 제안할 때 비교하는 대상으로 많이 이

용되고 있다. MCT의 시간 복잡도는 $O(m)$이다. KPB는 자원 중 실행시간이 작은 순서로 $km/100$개의 자원을 선택하여 그 중에서 완료시간이 가장 작은 자원을 할당하게 된다. 여기서 k는 스케줄링 전에 미리 주어지는 상수 값으로 만약, k=100이면 KPB알고리즘의 실행 결과는 MCT와 같게 되고 k=100/m이면 MET와 같게 된다. KPB의 시간 복잡도는 $O(m\log m)$이다. 이 외에도 최대 준비시간과 최소준비시간의 비율에 대한 한계 값을 적용하여 MET와 MCT를 교대로 사용하는 SA(Switching Algorithm)과 가장 빠른 준비시간을 갖는 자원을 작업에 할당하는 OLB(Opportunistic Load Balancing) 방법 등이 있다[8].

4. On-line 방식 알고리즘

4.1 MECT 알고리즘

본 논문에서 제시한 MECT(Minimum Execution and Completion Time)알고리즘은 동적 스케줄링의 온라인 방식에 속하며 독립적인 작업들을 그 대상으로 한다. 자원의 이질성이 존재한다는 것은 특정 작업의 실행시간이 각 자원의 특성에 따라 다르다는 것을 의미한다. 작업이 요구하는 자원의 특성에 맞는 자원에 작업이 주어지게 된다면 전체적인 준비시간을 단축시킬 수 있다. 이는 이질적인 자원으로 구성된 시스템에서 작업의 실행시간이 성능을 좌우하는데 중요한 역할을 할 수 있다는 것을 말한다. MECT 알고리즘은 이러한 가정에서 출발한다. 최우선으로 완료시간을 고려한다는 점에서는 MCT와 유사하지만 이질적인 자원으로 구성된 환경을 위해서 실행시간을 고려한다는 점에서 MCT와 다르다. 그림 2는 MECT 알고리즘의 pseudo code이다. 알고리즘을 살펴보면, 먼저 스케줄링 하고자 하는 작업인 T_i와 T_i에 대한 각 자원의 실행시간을 나타내는 벡터가 입력된다. Ⅰ에서는 현재 자원들의 준비시간 (b_i) 중 가장 큰 값을 찾는다. Ⅱ에서는 Ⅰ에서 구한 값보다 작은 완료시간 (ct_{ij})를 갖는 자원들을 찾는다. Ⅲ에서는 만약 Ⅱ의 조건을 만족하는 자원이 존재할 경우 그 자원 중 T_i에 대한 실행시간이 가장 작은 자원을 찾고 Ⅳ에서는 그 자원의 인덱스를 출력한다. 만약 Ⅱ에서 조건을 만족하는 자원이 없을 경우 MCT와 같은 원리로 시스템의 전체 자원 중 가장 작은 완료시간을 찾아 그 값을 갖는 자원의 인덱스를 출력한다.

MECT의 시간 복잡도는 다음과 같다. 시스템에서 최대 준비시간을 찾기 위한 시간 복잡도와 최대 준비시간보다 짧은 완료시간을 갖는 자원을 검색하기 위한 시간 복잡도는 각각 $O(m)$이다. 검색된 자원 중에서 최소 실행시간을 갖는 자원을 찾기 위한 시간 복잡도는 $O(m)$이다. 검색된 자원이 존재하지 않을 경우, 전체 자원에서 최소 완료시간을 갖는 자원을 검색하기 때문에 시간 복잡도는 $O(m)$이다. 따라서 알고리즘을 수행하는데 소요되는 전체 시간 복잡도는 $O(m)$이 된다.

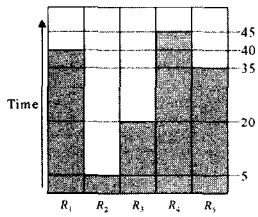
```
Algorithm MECT
Input: T_i, {e_i1, e_i2,...,e_im} , m= |R|
Output: k (resource index)
Ⅰ  find b_max = max_{R_j∈R} b_j
Ⅱ  find a set of resources, R̂ that have completion time smaller than b_max
   if ( |R̂| >0 )
Ⅲ      find k, such that  e_ik = min_{R_j∈R̂} e_ij
   else
Ⅲ'     find k, such that  ct_ik = min_{R_j∈R̂} ct_ij
   endif
Ⅳ  return k
```

그림 2 MECT 알고리즘

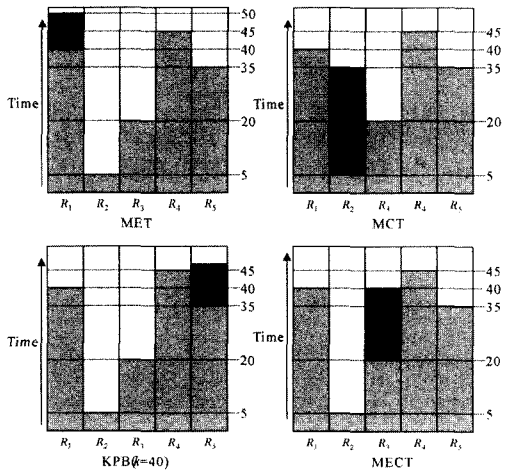
4.2 각 알고리즘 예제

그림 3은 MECT 알고리즘과 기존의 알고리즘이 5개의 이질적 자원으로 구성된 시스템에서 어떻게 작업을 스케줄링 하는지 보이는 예이다. 그림 3의 (a)는 스케줄링하고자 하는 작업에 대한 각 자원에서의 실행시간(e)을 나타낸 표이다. 스케줄링의 대상이 되는 작업을 T_i라

	R_1	R_2	R_3	R_4	R_5
T_i	10	30	20	15	12



(a) 작업의 각 자원에서의 실행시간과 스케줄 이전의 자원상황



(b) (a)의 표에 값을 참고하여 각 알고리즘을 적용했을 때의 결과

그림 3 각 스케줄링 알고리즘의 수행 결과

고 했을 때, 실행시간은 각각 e_{i1}=10, e_{i2}=30, e_{i3}=20, e_{i4}=15, e_{i5}=12이다. 그림 3의 (b)는 각 스케쥴링 알고리즘이 작업을 스케쥴링한 후의 결과를 나타내고 있다. MET는 실행시간 중에서 가장 작은 값을 찾아 그 자원을 작업에 할당하게 되는데 표에서 R_1의 실행시간, e_{i1}=10가 가장 작으므로 R_1를 T_i에 할당한다. MCT는 각 자원의 완료시간 중에서 가장 작은 값을 갖는 자원을 찾아 할당하게 되는데 그림 3의 예에서는 R_2의 완료시간, ct_{i2}=5+30=35이 가장 작으므로 R_2를 선택하여 T_i에 할당한다. KPB는 k=40이므로 전체 자원 중에서 가장 작은 실행시간을 갖는 자원 2개를 선택하게 된다. 그림 3에서는 R_1과 R_5가 선택된다. 최종적으로 두 자원 중 완료시간이 작은 R_5가 선택된다. MECT는 먼저 각 자원의 준비시간 중에서 최대 값을 찾는다. 그림 3에서 그 값은 R_4의 준비시간, b_4=45이다. 다음으로 자원 중에서 완료시간이 45보다 작은 값을 갖는 것들을 찾는다. ct_{i2}= 5+30=35, ct_{i3}=20+20=40이므로 R_2와 R_3이 이에 속한다. 이 자원 중에서 가장 작은 실행시간을 갖는 자원인 R_3(e_{i3}=20)을 선택하게 되고 R_3을 T_i에 할당하게 된다.

5. 시뮬레이션 및 결과

본 연구에서 사용된 시뮬레이션 프로그램은 이산형 모델의 시뮬레이션에 많이 이용되고 있는 SimJava [9]를 사용하여 작성되었다. 또한, 논문 [8]에 의하면 KPB의 성능이 k=20일 때, 가장 좋은 결과를 나타낸다. 따라서 본 논문의 모든 시뮬레이션은 KPB의 k값을 20으로 고정하여 수행하였다.

5.1 작업 · 자원행렬

시뮬레이션을 하기 위해 필요한 값들인 작업의 각 자원에서의 실행시간은 미리 계산되어 진다는 가정에 의해서 작성된 작업 · 자원행렬을 이용하였다. 각 행은 작업을 나타내고 각 열은 자원을 나타내며 행렬의 각 원소는 특정 작업에 대한 특정 자원에서의 실행시간을 나타낸다. 자원의 기대실행시간은 작업의 프로파일 등 여러 가지 방법을 통하여 구할 수 있다[8,10]. 시뮬레이션에 사용된 작업 · 자원행렬은 inconsistent 모델, semi-consistent 모델 그리고 consistent 모델로 나뉘어진다 [8,11]. inconsistent 모델은 각 행에 대하여 열의 값들이 무작위로 이루어져 있으며, semi-consistent 모델은 각 행에 대하여 특정 열들이 그 값의 오름차순으로 정렬되어 있다. consistent 모델은 모든 열이 오름차순으로 정렬된 것이다[8,11]. 앞으로의 시뮬레이션을 통해서 알 수 있지만, consistent 모델과 같은 경우 작업 스케쥴링에 있어 특정 자원에 편중되는 정도가 다른 모델에 비해 크다. 그러므로 자원 사용의 불균형을 초래할 수 있다. 실제로 자원의 특성의 종류는 제한적이며 또한 작업이 요구하는 자원의 특성이 다양하지 않다. 즉, 파일 입출력 속도가 빠른 자원을 요구하는 작업들은 파일 입출력 속도가 빠른 자원에서 빠른 속도를 나타내게 된다. 그러므로, 현실 세계에 가장 적합한 모델은 semi-consistent 모델 작업 · 자원행렬이라고 볼 수 있다[8]. 작업 · 자원행렬의 각 원소의 값은 작업 이질성의 최대 값을 1000, 자원 이질성의 최대 값이 20이라고 했을 때, 작업을 1과 1000사이에서 얻은 난수를 이용하여 산출하고 각 값에 다시 1과 20사이에서 얻은 난수의 값을 곱하는 식으로 구할 수 있다. 표 1은 작업의 개수를 10, 자원의 개수를 5, 작업 이질성을 1000, 그리고 자원 이질성을 100으로 하여 작성한 inconsistent모델의 작업 · 자원행렬이다. 각 행은 작업을 나타내고 열은 자원을 나타낸다. 각 원소는 작업에 대한 자원의 실행시간이다. 예를 들면, T_1의 R_1에서의 실행시간은 50,160이며 T_1의 R_2에서의 실행시간은 50,920이다. 작업 · 자원행렬이 자원 이질성을 50으로 하여 작성되면 자원 이질성을 100으로 하였을 때보다 특정 작업에 대한 자원들간의 실행시간의 격차는 줄어들게 된다. 표 1은 각 작업의 기대실행시간을 나타내는 것으로 실제로 작업을 실행시켜서 나온 실행시간과는 미소한 차이의 오차가 발생하게 된다. 오차는 평균이 작업의 실행시간과 같고 분산이 작업의 실행시간의 3배와 같은 정규분포를 따르는 난수를 이용하여 계산하였다[8]. 그러므로 앞으로 살펴볼 각 알고리즘을 적용한 결과는 표의 수치와 미소한 차이의 오차를 가지게 된다.

표 1 10×5 작업자원행렬

	50,160	50,920	36,480	36,480	41,800
	46,345	7,130	69,161	50,623	54,901
	30,480	30,480	7,112	45,720	26,416
	42,720	39,840	6,240	20,640	5,760
	53,136	25,272	27,216	12,312	6,480
	8,520	5,396	14,768	6,816	22,152
	23,954	56,994	20,650	4,130	43,778
	61,490	86,086	51,084	17,974	65,274
	3,168	4,950	1,287	3,762	2,376
	1,428	4,046	6,664	1,904	5,831

5.2 자원 이질성에 따른 성능평가

다음 시뮬레이션은 자원 이질성의 증가에 따른 각 알고리즘의 성능 변화를 살펴보기 위한 시뮬레이션이다. 자원의 개수를 20, 작업의 개수를 1000, 작업의 도착 시간 간격을 100, 그리고 작업 이질성을 3000으로 고정하

여 시뮬레이션을 수행하였다. 자원 이질성을 10부터 120까지 10씩 증가시키면서 각 조건에 대하여 50회의 시뮬레이션 결과의 평균을 구한 것이다. 그림 4는 inconsistent 모델에서 자원 이질성에 따른 알고리즘의 성능을 비교한 시뮬레이션의 결과 그래프이다. 자원 이질성이 10일 때는 MCT와 KPB, MECT의 결과가 MET보다 2배 이상 좋은 결과를 보인다는 것을 알 수 있다. KPB가 근소한 차이로 MCT, MECT보다 좋은 결과를 보이며 MCT는 MECT보다 근소한 차이로 좋은 결과를 보인다. 하지만 자원 이질성이 점차적으로 증가함에 따라 MET와 다른 알고리즘간의 차이가 줄어드는 것을 알 수 있다. 또한 자원 이질성이 증가함에 따라 KPB가 MCT 보다 미소한 차이로 좋은 성능을 보이고 있다. MECT는 KPB나 MCT보다 좋은 성능을 보이고 있으며, 자원의 이질성이 증가함에 따라 성능의 차이가 증가하는 것을 알 수 있다.

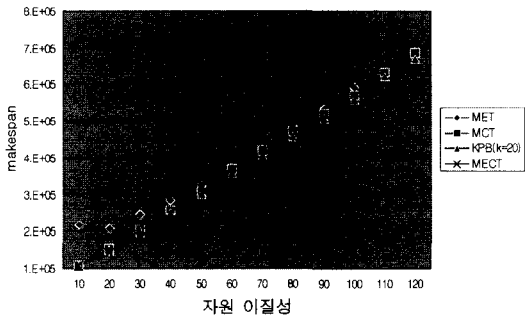


그림 4 inconsistent 모델 작업·자원행렬에서의 성능 비교

그림 5는 semi-consistent 모델에서 자원 이질성의 변화에 따른 각 알고리즘의 성능을 비교한 것이다. MET가 다른 알고리즘에 비해서 현저하게 뒤떨어진 성능을 보이며 다른 알고리즘은 근소한 차이를 보인다. 본 시뮬레이션에서 구성한 semi-consistent 모델의 작업·자원행렬은 고정 열에 따라 정렬을 한 방식을 취하고 있으며 따라서 KPB의 결과는 MCT와 MECT보다 좋지 않은 것으로 나타난다. 자원 이질성이 10일 때 MECT가 KPB와 MCT에 비해 근소한 차이로 좋은 결과를 보인다. 또한 자원 이질성이 증가함에 따라 MECT가 다른 알고리즘에 비해 뛰어난 성능을 보이며 성능의 차이는 자원 이질성이 증가함에 따라 증가하는 것을 알 수 있다.

그림 6은 consistent 모델에서 자원 이질성의 변화에 따른 각 알고리즘의 성능을 비교한 것이다. MET가 다른 알고리즘에 비해 큰 차이로 좋지 않은 결과를 보인다. 이는 MET의 특성이 실행시간만을 고려하기 때문에

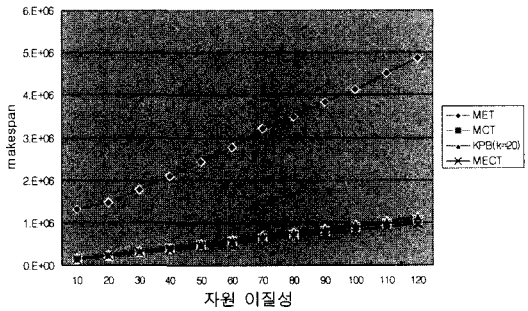


그림 5 semi-consistent 모델 작업·자원행렬에서의 성능 비교

작업들을 한 자원에 편중되게 스케줄링하여 나타나는 결과이다. 다음으로 KPB의 성능이 좋지 않은 결과를 보이며 그 원인은 KPB 알고리즘의 특성상 MCT와 MECT보다 특정 자원에 편중되는 스케줄링하기 때문이다. 그림 2에서 II의 조건을 만족하는 자원이 존재하지 않는다면 이는 MCT와 같은 스케줄링 결과를 나타내게 된다. consistent 모델 작업·자원행렬은 작업의 실행시간이 오름차순으로 정렬되어 있기 때문에 특정 자원에 편중되는 스케줄링 결과를 보이게 되고 앞에서 기술한 현상이 더욱 자주 나타나게 된다. 그러므로 그림 6에서와 같이 두 알고리즘은 비슷한 성능을 나타나게 되는데 시뮬레이션에서는 MECT가 MCT보다 미소한 차이로 좋은 결과를 보이는 것을 알 수 있다.

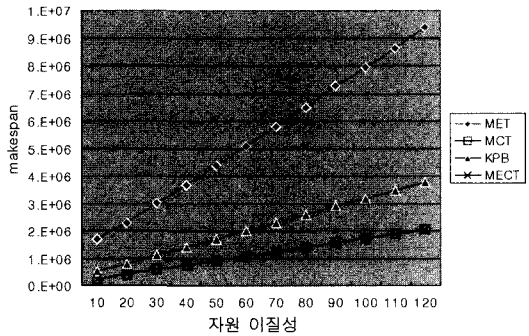


그림 6 consistent 모델 작업·자원행렬에서의 성능 비교

5.3 작업 이질성에 따른 성능평가

그림 7은 20개의 자원, 자원 이질성을 20으로 하고 inconsistent모델 작업 자원행렬을 사용하여 작업 이질성을 500에서 3000으로 500씩 증가시키면서 측정한 결과를 그래프로 나타낸 것이다. 이 시뮬레이션은 작업 이질성에 따라 각 알고리즘이 어떠한 성능의 차이를 보이는지 알아보기 위한 시뮬레이션이다. 그림 7에서 보는

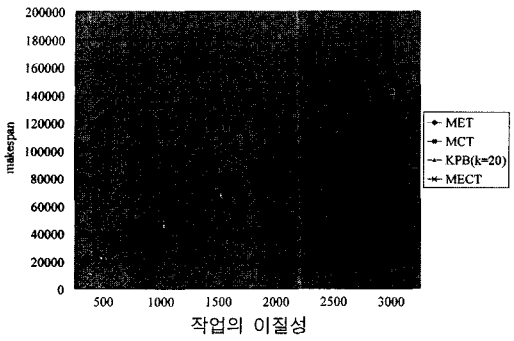


그림 7 작업이질성에 따른 알고리즘 성능 비교

바와 같이 MET의 성능이 가장 떨어지는 결과를 보이고 MECT, KPB, MCT 순으로 좋은 성능을 나타내는 것을 알 수 있다. MECT가 작업의 이질성과 자원의 이질성이 큰 시스템에서 다른 알고리즘보다 뛰어난 성능을 보이고 있다. 작업의 이질성이 차이가 크다는 것과 자원의 이질성이 크다는 것은 시스템에 주어지는 작업의 성격이 다양하다는 것과 시스템을 구성하는 자원이 가지는 특성들이 그 만큼 다양하다는 것을 말한다. 작업의 이질성이 500일 때와 1000일 때는 MECT, KPB, MCT의 성능 차이가 극히 작다는 것을 알 수 있다. 이는 작업의 전체적인 이질성이 작아지면서 KPB나 MECT 알고리즘에서 고려한 부분의 효과가 줄어든다는 것을 말하며 비슷한 작업시간을 갖는 작업들이 주어지는 시스템에서는 MECT가 다른 알고리즘에 비해 좋다고 말할 수 없다.

6. 결 론

이질적 자원으로 구성된 환경은 작업을 실행할 때 그 작업이 요구하는 특성에 따라 자원마다 실행시간의 차이를 보인다는 점에서 동질의 자원으로 구성된 환경과 구별된다. 실행시간이 이질적 자원으로 구성된 환경에서의 독립적인 작업들을 스케쥴링하는 알고리즘을 연구하는데 중요한 부분을 차지하게 됨을 시뮬레이션을 통하여 보였다. 시뮬레이션은 inconsistent, semi-consistent, consistent모델 작업 자원 행렬을 이용하여 자원 이질성에 따라 어떤 성능의 차이가 있는지 수행하였다. 본 논문에서 제안한 알고리즘은 MCT와 유사한 방식을 취하고 있지만, 시스템을 구성하는 자원의 실행시간을 고려하여 설계되었다. consistent 모델 작업에서는 KPB, MCT와 비슷한 성능을 보인다. semi-consistent 모델 작업에서는 consistent 모델에서보다 다른 알고리즘에 비해 보다 좋은 성능을 보이며 특히, inconsistent 모델 작업에서는 다른 알고리즘과의 성능차이가 다른 작업 모델에 비해 크다는 것을 알 수 있다. 시뮬레이션의 결과를 종합해 보면 MECT 알고리즘은 다른 알고리즘에 비해 makespan 측면에서 좋은 성능을 보이며 자원이 이질성이 증가함에 따라 다른 알고리즘과의 성능 차이가 커진다. 즉, 자원의 이질성과 작업의 이질성이 작은 시스템에서는 다른 알고리즘에 비해 이 알고리즘의 효과를 기대할 수는 없지만 시스템이 자원의 이질성과 작업의 이질성의 차이가 크고 작업의 분포가 inconsistent 모델로 구성되어 있을 때 MECT 알고리즘은 좋은 선택이 될 것이다. 앞으로의 계획은 작업의 도착시간을 고려하여 작업의 도착 주기를 변화 시켜 가면서 MECT알고리즘의 다른 알고리즘과의 어떤 성능의 차이를 보이는지 시뮬레이션을 통하여 알아보고자 한다.

참 고 문 헌

[1] O. H. Ibarra and C. E. Kim, "Heuristic Algorithm for Scheduling Independent Tasks on Nonidentical Processors," *Journal of the ACM,* vol. 24, no. 2, pp. 280~289, April, 1977.

[2] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Journal of High-Performance Computing Applications*, vol. 15, no. 3, pp. 200~222, 2001.

[3] R. Buyya, J. Giddy, and D. Abramson, "An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications," *Proc. of the 2nd International Workshop on Active Middleware Services,* August, 2000.

[4] H. Barada, S. M. Sait, and N. Baig, "Task Matching and Scheduling in Heterogeneous Systems using Simulated Evolution," *Proc. of the 15th Parallel and Distributed Processing Symposium*, pp. 875~882, 2001.

[5] B. Hamidzadeh, Lau Ying Kit, and D.J. Lilja, "Dynamic Task Scheduling using Online Optimization," Journal of Parallel and Distributed Systems, vol. 11, pp. 1151~1163, 2000.

[6] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous Distributed Computing," *Encyclopedia of Electrical and Electronics Engineering*, J. G. Wdbster, editor, John Wiley & Sons, vol. 8, pp. 679~690, 1999.

[7] M. Pinedo, Scheduling: *Theory, Algorithms, and Systems,* Prentice Hall, NJ, 1995.

[8] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic Matching and Scheduling of a Class of Indenpendent Tasks onto Heterogeneous Computing Systems," *Proc. of the 8th Heterogeneous Computing Workshop*, pp. 30~44, April, 1999.

[9] F. Howell and R. McNab, "SimJava: A Discrete

Event Simulation Package For Java With Applications In Computer Systems Modelling," *Proc. of the 1st International Conference on Web-based Modelling and Simulation,* January, 1998.

[10] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. L. Wang, "Heterogeneous Computing: Challenges and Opportunities," *Journal of the IEEE Computer,* vol. 26, pp. 18~27, June, 1993.

[11] T. D. Braun, H. J. Siegel, and Noah Beck, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *Journal of Parallel and Distributed Computing,* vol. 61, pp. 810~837, 2001.

김 학 두

1994년 3월~2000년 2월 서울시립대학교 전산통계학 학사. 2001년 3월~2003년 2월 서울시립대학교 전산학 석사. 2003년 2월~현재 한국전자통신연구원 연구원 관심분야는 병렬 처리 시스템, 자바 카드, UICC

김 진 석

1986년 3월~1990년 2월 과학기술대학 전산학 학사. 1990년 3월~1992년 2월 KAIST 전산학 석사. 1992년 3월~1997년 2월 KAIST 전산학 박사. 1997년 3월~1999년 2월 KAIST 인공지능연구센터 Postdoc 연구원. 1997년 10월~1998년 8월 미국 M.I.T. Laboratory for Computer Science Postdoc Fellow. 1998년 10월~1999년 2월 전자통신연구원(ETRI) 수퍼컴퓨터센터 초빙 연구원. 1999년 3월~현재 서울시립대학교 컴퓨터과학부 부교수. 2002년 2월~현재 서울시립대학교 컴퓨터과학부 학부장. 관심분야는 병렬 처리 시스템, 멀티미디어, 인터넷과 정보 검색

박 형 우

1981년 3월~1985년 2월 서울시립대학교 전자공학 학사. 1994년 3월~1996년 2월 성균관대학교 정보공학 석사. 1996년 3월~2001년 8월 성균관대학교 전산학 박사. 1977년 12월~1987년 5월 KIST 전산센터 기능원. 1987년 6월~1992년 10월 KAIST 전산개발센터 기술원. 1992년 11월~1993년 2월 KIST 슈퍼컴퓨터센터 연구원. 1994년 3월~1997년 1월 SERI 슈퍼컴퓨터센터 선임연구원. 1997년 2월~1998년 5월 SERI 슈퍼컴퓨터센터 고성능전산망개발실장/선임연구원. 1998년 6월~1999년 9월 ETRI 슈퍼컴퓨터센터 선임연구원. 1999년 10월~2001년 4월 KISTI 슈퍼컴퓨팅센터 선임연구원. 2001년 5월~현재 KISTI 슈퍼컴퓨팅센터 그리드 연구실장/책임연구원. 관심분야는 슈퍼컴퓨팅, GRID, 초고속 인터넷