클러스터링을 이용한 경험적 태스크 할당 기법

김 석 일† · 전 중 남† · 김 관 유††

요 약

본 논문에서는 여러 개의 머신으로 구성된 분산 환경에서 비 방향성 태스크 그래프를 구성하는 태스크들을 클러스터링 기법을 이용하여 할당하는 경험적 태스크 할당기법을 연구하였다. 이 기법은 우선 태스크-머신 그래프를 구성하고 각 간선의 가중치가 가장 큰 간선의 양단에 연결된 태스크들을 병합하거나 태스크를 머신에 할당하는 클러스터링 과정을 거친다. 이 과정에서 제안한 기법은 머신적합성이 다소 떨어지더라도 이웃한 태스크들과의 통신비용이 줄어들어 머신적합성의 저하를 상쇄할 수 있는 머신이 있다면 이 머신을 찾아 태스크를 할당하도록 하였다. 다양한 형태의 태스크 그래프에 대한 시뮬레이션 결과, 본 논문에서 제시한 기법이 기존의 기법에 비하여 우수함을 확인할 수 있었다. 또한 최적 기법과의 비교에서도 제안한 기법이 기존의 기법들에 비하여 최적에 준하는 결과를 더 많은 경우에 보여주고 있음을 확인할 수 있었다.

A Heuristic Task Allocation Scheme Based on Clustering

Sukil Kim† · Joong-Nam Jeon† · Kwan-You Kim††

ABSTRACT

This paper proposes a heuristic, clustering based task allocation scheme applicable to non-directed task graph on a distributed system. This scheme firstly builds a task-machine graph, and then applies a clustering process wherein a pair of tasks that are connected to the highest cost edge is merged into a big one or a task is allocated to a machine. During the process, the proposed scheme figure out a machine onto which the task allocation may cause deduction of large communication overhead that has incurred between the task and tasks that are already allocated to the machine while the computation costs is slightly increased in the machine. Simulation for the various task graphs shows that the scheduling using the proposed scheme result far better than ones by using the traditional schemes. A comparison with optimal task scheduling also promises that our scheme derives optimal results more occasionally than the traditional schemes do.

1. 서 론

컴퓨터 활용이 급속히 증가하고 거대 과제(grand challenge problem) 수행을 위한 강력한 연산 능력의 필요성이 대두됨에 따라 지역적으로 광범위하게 분포되어있는 여러 가지 컴퓨터 자원들을 네트워크로 연결하고 이를 이용하여 분산처리를 수행하는 기법에 관한 연구가 활발히 일어나고 있다[1-5].

분산 환경에서 응용 문제를 수행하기 위해서는 문제 분할, 태스크 할당, 스케줄링 및 동기화를 위한 전략이 필요하다. 즉 하나의 응용 문제를 여러 개의 태스크로 분할하고 이들을 가용한 머신에 할당하되, 문제를 해결하는데 필요한 총 작업 시간이 최소가 되도록 태스크간의 통신비용과 태스크를 할당할 머신의 성능을 고려하여 태스크를 적절한 머신에 할당하여야 한다. 그

※ 이 논문은 97-98년도 학술진흥재단 자유공모과제의 지원을 받았음.
† 종신회원 : 충북대학교 컴퓨터과학과 교수
†† 준 회 원 : 충북대학교 대학원 전자계산학과
논문접수 : 1999년 2월 18일, 심사완료 : 1999년 9월 20일

러나 이러한 목표는 NP-complete문제이므로 적은 비용으로 최적의 결과를 얻을 수 있는 방법을 찾는 것이 매우 어렵다[6]. 따라서 그 동안 경험적(heuristic)인 방법에 기초를 둔 태스크 할당에 관한 연구가 광범위하게 진행되어 왔다.

분산환경에서의 태스크 할당에 관한 연구는 크게 두 가지 성능 평가 요소에 기초하여 진행되고 있다. 첫 번째 부류는 태스크간의 명시적인 자료의존 관계가 존재하는 방향성 태스크 그래프(directed task graph)를 대상으로 하는 태스크 할당 문제이다[7-11, 20-23]. 이 부류의 연구는 태스크 할당의 우선 순위를 부여하고 우선 순위가 높은 태스크를 머신에 먼저 할당하는 리스트 스케쥴링 기법[7-11], 또는 통신량이 많은 태스크들을 더 큰 태스크로 합병하는 클러스터링 기법[20-23]을 바탕으로 발전하여 오고 있다. 근래에는 과도한 통신을 유발하는 태스크들을 여러 개의 머신에 동시에 할당하여 수행하도록 하는 복사 기법에 관한 연구도 진행되고 있다[11].

두 번째 부류는 태스크간의 수행 순서가 지정되지 않아 임의로 실행될 수 있으나, 태스크들이 수행되는 과정에서 정해진 통신이 발생하는 비 방향성 태스크 그래프를 대상으로 하는 태스크 할당 문제이다[12-19, 28]. 이 문제는 또한 모든 통신비용과 모든 태스크의 수행비용의 합을 최소화하는 것을 목표로 하고 있다. 그 중에서 Stone[12]은 2개의 머신으로 구성된 환경에서 최적의 태스크 할당 결과를 얻을 수 있는 방법을 제안 한 바가 있으며, Lee 등[17]은 Stone의 기법을 n개의 머신으로 구성된 선형 배열 시스템으로 확장하여 최적의 결과를 얻었음을 보고하고 있다. 그러나 임의의 머신 그래프에서는 최적의 할당결과를 얻을 수 있는 polynomial 기법은 아직까지 제안된 바 없으며 다만 Branch & bound 기법[28]을 이용한 최적기법이 연구되었을 뿐이다.

한편, 여러 가지 경험적 기법들이 연구되었는데, 그 중에서 Lo[14]는 임의의 토폴로지를 가지는 머신 환경을 2 머신 환경으로 변환하여 Stone의 2-머신 알고리즘을 반복적으로 적용하는 경험적 기법을 제안한 바 있으며, Kopidakis[15]는 방향성 태스크 그래프의 경우에 적용되었던 클러스터링 기법을 비방향성 태스크 그래프에 적용하여 Lo의 결과보다 우수한 결과를 보여주는 태스크 할당 기법을 제안한 바 있다.

본 논문에서도 후자의 비방향성 태스크 그래프에 대하여 응용 프로그램을 수행하는데 따른 수행 비용 및 통신비용의 합을 줄일 수 있는 태스크 할당 기법을 연구하였다. 특히, 본 논문에서는 Kopidakis의 태스크 할당 기법이 태스크 할당 기법의 복잡도를 줄이기 위하여 단지 간선의 가중치만을 클러스터링의 우선 순위로 삼으므로 말미암아 태스크와 머신의 적합성이 상대적으로 무시됨에 따라 통신비용이 줄어드는 대신 계산비용이 과도하게 증가하여 전체적으로 성능이 떨어질 수 있는 문제점을 개선하도록 하였다.

본 논문의 구성은 다음과 같다. 제 2장에서는 본 논문에서 다룰 비방향성 태스크 그래프를 토대로 태스크 할당의 목적함수를 제시하고 비방향성 태스크 그래프를 대상으로 하는 기존의 태스크 할당 기법을 보다 심도 있게 고찰하였다. 제 3장에서는 본 논문의 핵심인 경험적 태스크 할당 기법을 제안하고, 제 4장에서는 제안한 기법과 기존의 기법을 시뮬레이션을 통하여 비교하였다. 마지막으로 제 5장에서는 이 논문의 결론을 도출하고 향후 연구 계획을 기술하였다.

2. 문제 정의 및 기존 연구

2.1 문제 정의

여러 개의 머신을 네트워크로 연결한 분산 환경은 머신 그래프 $G_P(V_P, E_P)$로 표현할 수 있다. 여기서 V_p는 n개로 구성된 머신들의 집합으로 $V_P = \{p_1, p_2, \cdots, p_n\}$이다. 또한, E_P는 머신간을 직접 연결하는 네트워크의 구성을 의미하는 간선의 집합이다. 한편, 응용 프로그램은 여러 개의 병렬태스크로 분할되고 이들 병렬태스크들은 태스크간에 메시지를 주고받으면서 계산이 진행된다. 따라서 태스크간에 직접 메시지를 주고받는 경우에 이를 태스크간의 간선으로 표시하면 주어진 응용 문제에 대한 태스크 그래프 $G_T(V_T, E_T)$가 구성된다. 여기서 V_T는 m개로 구성된 응용 문제의 태스크 집합 $V_T = \{t_1, t_2, \cdots, t_m\}$이며, E_T는 태스크간의 간선의 집합이다.

본 논문에서 다룰 분산환경과 문제의 범위는 다음과 같다. 즉,

1) 시스템을 구성하는 머신 p_k는 이기종 시스템(heterogeneous system)이라고 간주한다. 즉 태스크 t_i는 머신별로 수행 비용이 다르다고 가정한다. 여기서 머신 p_k

에서 태스크 t_i를 실행시킬 때의 태스크의 실행 비용을 τ_{ik}로 표시하기로 한다.

2) 모든 머신은 완전연결(fully connected) 네트워크로 연결되었으며, 따라서 일정한 크기의 메시지를 한 쌍의 머신간에 송수신하는 비용은 어느 머신간에도 동일하다고 간주한다. 즉, 태스크 그래프 상에 간선이 존재하는 한 쌍의 태스크간의 통신비용은 이들이 할당되는 머신의 종류나 머신의 계산속도에 무관하다. 또한, 태스크 그래프에서 t_i와 t_j간의 간선 e_{ij}의 가중치 $|e_{ij}|$는 t_i와 t_j간에 교환하여야 하는 메시지의 길이에 근거한 통신비용이다. 만약 t_i와 t_j 사이에 간선이 존재하지 않으면 $|e_{ij}|=0$ 이다.

이상으로부터, 주어진 작업을 수행하는데 필요한 총 비용은 식 (1)과 같이 각 머신이 나누어 수행하는 작업의 실행 비용과 작업을 수행하는데 필요한 통신비용의 합으로 정의한다.

$$s=\sum_{i=1}^{m}\sum_{k=1}^{n}\tau_{ik}\zeta_{ik}+\sum_{e_{ij}\in E_T}\sum_{k=1}^{n}|e_{ij}|\tau_{ik}(1-\zeta_{jk}) \qquad (1)$$

여기서 ζ_{ik}는 태스크 t_i가 머신 p_k에 할당되었으면 1, 아니면 0인 값이다. 즉,

$$\zeta_{ik}=\begin{cases}1 & \text{if } t_i \text{ is assigned to } p_k\\ 0 & \text{otherwise}\end{cases}$$

2.2 태스크 할당에 관한 기존 연구

식 (1)이 최소가 되도록 하는 polynomial 복잡도를 가지는 최적 기법은 매우 제한적인 경우에만 가능하다. Stone[12]은 2개의 머신으로 구성된 분산 환경에서 최적의 태스크 할당 기법을 제안한 바 있다. 이 기법은 우선 태스크 그래프에 두 개의 머신 노드를 추가하고 머신과 태스크간을 잇는 간선의 가중치를 해당 태스크가 다른 머신에서 수행될 때의 계산비용으로 표시한 태스크-머신 그래프를 구성한 다음, 이 그래프에 maximum-flow 알고리즘[24-27]을 이용하여 두 개의 부 그래프(subgraph)로 나눌 때 잘리게되는 간선의 가중치의 합이 최소인 간선의 집합(min-cut)을 선택하도록 한다. 이때, min-cut에 의해 분리되는 간선의 가중치의 합은 각 태스크의 실행 비용 및 통신비용의 합이자 최소이므로 최적의 할당 결과를 얻을 수 있다.

Lee 등[17]은 Stone의 기법을 n개의 머신으로 구성된 선형 배열(linear array) 시스템까지 확장하는 기법을 제안하였다. 이 기법은 일렬로 배열된 선형 배열 시스템의 특징을 이용하여 머신 그래프를 두 그룹의 머신으로 구분하고 각각의 그룹을 두 개의 커다란 머신으로 간주하여 Stone의 기법에서 사용하였던 maximum flow알고리즘을 적용하고, 그 결과를 토대로 선형 방정식을 풀어 해를 계산하여 최적의 태스크 할당 결과를 얻는다. 따라서 이 기법은 polynomial 시간에 최적 할당 결과를 얻을 수 있으나, 제한된 형태의 시스템만을 대상으로 하는 기법이므로 임의의 토폴로지를 가진 머신 그래프에는 적용될 수 없다.

Sinclair[28]는 Branch & bound 기법을 이용하여 최적의 해를 구하는 방법을 제안하였다. 그러나 이 알고리즘은 문제의 크기가 큰 경우에는 매우 오랜 시간이 소요되는 단점이 있으므로 본 논문에서는 이 부류의 방법은 배제하도록 하고, polynomial 복잡도를 가지는 경험적 기법을 위주로 연구를 수행하였다. 경험적 기법의 대표적인 연구 성과는 Lo[16]와 Kopidakis[15]에 의하여 제시되었다.

Lo[16]는 Stone의 기법을 임의의 토폴로지로 구성된 n개의 머신 환경에 적용하는 방법을 제안하였다. 이 기법은 우선 태스크 그래프에 머신의 수만큼 노드를 추가하고 머신과 태스크간의 간선을 이어 태스크 머신 그래프를 구성한다. 이 때, 태스크와 머신간을 이어주는 간선의 가중치는 이 태스크가 나머지 머신들에게 할당되어 실행될 때 필요한 실행비용의 평균값으로 결정한다. 이렇게 구성된 태스크-머신 그래프는 모든 머신을 대상으로 하나의 머신과 나머지 머신을 하나의 머신 그룹으로 간주하는 가상의 2 머신 그룹으로 구성된 새로운 태스크-머신 그룹 그래프로 변환되고 이에 Stone의 기법과 같이 maximum-flow 알고리즘을 적용하여 태스크를 할당하게 된다. 만일 이 과정을 거치고도 할당되지 않은 태스크가 남아있다면, 이들 태스크는 태스크간의 통신량을 고려하여 강제적으로 각 머신에 할당한다.

Lo 기법은 태스크-머신 그래프를 또다시 모든 머신별로 가상의 2머신 태스크-머신 그룹 그래프로 재변환하고, 이 그래프를 대상으로 maximum-flow 알고리즘을 반복적으로 적용하기 때문에 알고리즘의 복잡도가 $O(nm^4\log m)$로 크며, 태스크-머신 그래프를 구성하는 경우에도 태스크와 머신을 잇는 간선의 가중치를 태스크의 수행시간만을 고려하여 결정하므로 할당 결과가 과도한 통신을 초래할 가능성이 존재한다.

한편 Kopidakis[15]는 Lo의 기법에서 발생할지도 모를 과도한 통신비용을 줄이기 위하여 태스크 머신 그래프의 모든 간선을 가중치에 의거하여 정렬한 다음, 가중치가 가장 큰 간선부터 차례로 간선의 양단에 있는 태스크와 머신 또는 한 쌍의 태스크를 하나의 클러스터로 묶는 기법을 제안하였다. 즉, 클러스터링의 대상으로 선택된 간선의 양단이 모두 태스크인 경우에는 이 간선이 태스크간의 통신이 매우 큰 통신을 의미하므로 이들 태스크들을 하나의 클러스터로 병합(merge)할 수 있다. 또한, 간선의 양단이 각각 태스크와 머신인 경우에는 이 태스크는 이 머신에 할당하므로 이 태스크를 가장 빠르게 수행하도록 하므로 결국 태스크의 머신 적합성이 충분히 고려된 태스크 할당 결과를 얻을 수 있다. 따라서 이 기법은 통신비용이 고려되지 않았던 Lo의 기법에 비하여 태스크의 할당에 따른 성능이 더 우수한 특징을 지닌다. <표 1>은 이상의 세 가지 태스크 할당 기법을 비교 정리한 것이다.

3. 새로운 태스크 할당기법

본 절에서는 Kopidakis의 기법에서 수행되는 클러스터링 과정을 변형한 새로운 클러스터링 기법을 제안하였다. 제안한 태스크 할당기법은 주어진 태스크 그래프와 머신 그래프를 토대로 태스크-머신 그래프를 구성하고, 구성된 태스크-머신 그래프 상에서 간선의 값이 큰 것부터 순서대로 태스크들을 병합하거나 태스크를 머신에 할당하는 클러스터링 과정을 모든 태스크에 대하여 반복적으로 수행한다.

3.1 태스크-머신 그래프 구성

주어진 태스크 그래프와 머신 그래프로부터 태스크-머신 그래프로 확장하는 과정은 Lo[16] 또는 Kopidakis[15] 기법에서와 같이 각 머신을 태스크 그래프의 노드로 추가하고 추가된 노드와 모든 태스크들간을 간선으로 잇고, 태스크와 머신간의 간선의 가중치를 부여한다. 이 때 태스크 t_i와 머신 p_k간의 가중치는 Kopidakis의 경우와 같이 t_i를 p_k를 제외한 나머지 머신에서 수행할 때 필요한 수행시간의 평균치로 삼는다. 즉, 태스크-머신 그래프에 새로 추가된 태스크와 머신을 잇는 간선 e'_{ik}의 가중치 $|e'_{ik}|$는 식 (2)와 같이 결정된다.

$$|e'_{ik}| = \frac{\sum_{j=1}^{n} \tau_{ij} - \tau_{ik}}{n-1} \quad (2)$$

따라서 태스크-머신 그래프에서 어떤 태스크 t_i와 모든 머신 $p_1, p_2, \cdots, p_n$을 잇는 모든 간선의 가중치를 비교하여 머신 p_j간의 가중치가 가장 크다는 것은 결국 t_i와 다른 태스크간의 통신을 고려하지 않는다면 태스크 t_i를 머신 p_j에서 수행하는 것이 다른 머신에서 수행하는 것에 비하여 t_i를 가장 빨리 끝낼 수 있다. 이러한 판단에 따라 Kopidakis 기법[15]에서는 클러스터링 과정에서 t_i를 p_j에 할당한다. 그러나 본 논문에서 제안하는 기법에서는 t_i를 p_j에 할당하기 전에 p_j에 이미 할당된 태스크들과 t_i와의 통신비용이 얼마이었는가를 토대로, 이 태스크를 p_j에 할당함으로 말미암아 줄어드는 통신비용이 큰 경우에만 t_i를 p_j에 할당하는 과정을 거치도록 하였다.

3.2 클러스터링 과정

본 논문에서 제시하는 기법의 두 번째 단계는 태스크-머신 그래프에서 태스크 또는 머신을 나타내는 노드간을 연결하는 간선의 가중치가 가장 큰 간선부터 차례대로 간선의 양단에 연결된 태스크와 머신 또는 태스크와 태스크를 클러스터로 병합하는 과정이다. 이

〈표 1〉 기존 연구에서의 간선의 가중치 비교

	태스크-머신 그래프의 간선의 가중치	태스크 분할 방법
Stone 기법	$\|e'_{ik}\| = \tau_{ir}, r \neq k$	2-머신 환경에서 max-flow min-cut 알고리즘을 적용
Lo 기법	$\|e'_{ik}\| = \frac{\sum_{j=1}^{n}(\tau_{ij} - \tau_{ik})}{n-1}$	n-머신으로 구성된 환경에서 Stone의 기법을 반복적용
Kopidakis 기법	$\|e'_{ik}\| = \frac{\sum_{p=1}^{n} \tau_{ip} - \tau_{ik}}{n-1}$	태스크 머신 그래프에서 간선의 가중치가 큰 것부터 차례로 클러스터링 수행

과정을 반복하면 모든 태스크의 할당이 완료된다.

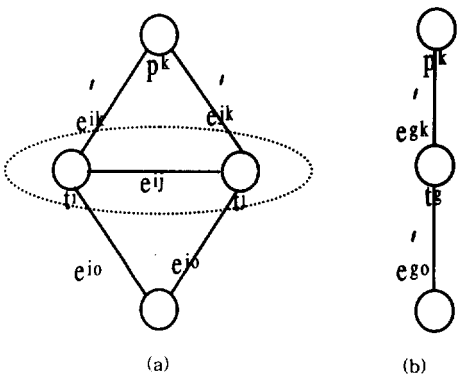
클러스터링의 순서는 가중치가 가장 큰 간선의 태스크들을 먼저 병합하거나 우선 머신에 할당하도록 하였다. 즉, 태스크를 어느 머신에 할당하는가에 따라서 작업시간의 변동폭이 크기 때문에 클러스터링 과정에서 가중치가 큰 간선을 우선적으로 처리하도록 하였다. 또한, 클러스터링은 두 가지 간선에 따라서 그 처리과정이 나뉜다.

3.2.1 태스크-태스크를 잇는 간선

태스크-머신 그래프에서 태스크와 태스크를 잇는 간선은 태스크간의 통신비용을 나타내므로 가중치가 큰 이들 간선의 태스크들을 병합하여 하나의 머신에 할당하면 통신비용을 줄일 수 있다. 따라서 이 간선의 경우에는 태스크를 더 큰 태스크로 병합하고 새로 생성된 큰 태스크와 각 머신들을 연결하는 간선의 가중치를 갱신한다. 예를 들어 (그림 1)(a)와 같이 두 개의 태스크 t_i와 t_j가 병합되어 태스크 t_g를 생성하였다고 하고, 이 때 t_i와 t_j가 각각 태스크 t_o와 머신 p_k간에 간선이 존재한다고 하면, 병합이 이루어진 태스크 t_g와 태스크 t_o및 머신 p_k간은 (그림 1)(b)와 같이 새로운 간선이 생성되며, 이 간선의 가중치는 식 (3)과 같이 각각 태스크 t_i, t_j와 태스크 t_o간의 간선의 가중치의 합 머신 p_k간의 간선의 가중치의 합으로 갱신된다.

$$|e_{go}| = |e_{io}| + |e_{jo}|, \quad |e_{gk}| = |e_{ik}| + |e_{jk}| \tag{3}$$

두 개의 태스크가 병합된 이후에는 새로운 태스크-머신 그래프를 대상으로 가중치가 가장 큰 간선을 선택하여 1) 또는 다음의 2) 과정을 반복하게 된다.



(그림 1) 태스크의 병합에 따른 그래프의 재구성

3.2.2 태스크-머신을 잇는 간선

태스크 머신 그래프에서 태스크와 머신을 잇는 간선 중에서 가중치가 가장 큰 간선에 연결된 머신은 이 태스크가 이 머신에 할당되어서 수행하는 것이 다른 머신에 할당되어 수행되는 것보다 더 빠르게 계산을 할 수 있음을 의미한다. 즉, 이 태스크와 이 머신은 다른 머신들보다 머신 적합성이 크다는 것을 뜻한다. 따라서 Kopidakis의 기법[15]에서는 클러스터링 과정에서 머신적합성을 충분히 고려하여 태스크를 할당한다고 할 수 있다.

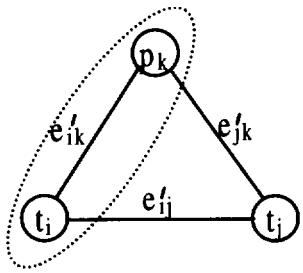
머신적합성이 크다고 하더라도 다른 머신에 할당되는 태스크들과의 통신비용이 과도히 증가하는 경우에는 태스크를 수행하는 시간이 제일 작더라도 통신비용이 크므로 통신시간과 태스크의 수행시간의 합으로 표현되는 작업시간이 늘어나게 된다. 즉, 머신적합성이 다소 떨어지더라도 이웃한 태스크들과의 통신비용이 줄어들어 머신적합성의 저하를 상쇄할 수 있는 머신이 있다면 이 머신을 찾아 태스크를 할당하는 것이 전체적으로 빠른 작업이 가능하다. 따라서 본 논문에서 제안하는 기법은 선정된 간선에 연결된 태스크를 할당할 머신을 결정할 때 각 머신에서의 태스크의 실행비용에서 이미 이 머신에 할당된 태스크들과 간에 제거된 통신비용을 뺀 값이 최소가 되는 머신을 선택하도록 하였다. 즉, 태스크 t_i이 p_k에 할당되기 위해서는 식 (4)과 같이 할당에 따른 이익 ρ_{ik}를 구하고 이 값이 최소가 되어야 한다.

$$\rho_{ik} = \tau_{ik} - \sigma_{ik} \tag{4}$$

여기서 σ_{ik}는 태스크 t_i가 p_k에 이미 할당된 모든 태스크들간의 통신비용의 합이다.

3.2.3 태스크-머신 그래프의 재구성

어떤 태스크를 할당할 머신이 결정되면 해당 태스크를 선정한 머신에 할당하고 태스크-머신 그래프에서 할당된 태스크가 제거되고, 태스크-머신 그래프가 재구성되어야 한다. 예를 들어 (그림 2)와 같이 태스크 t_i가 머신 p_k에 할당되었다고 가정하고, t_i와 태스크 t_j간의 간선 e_{ij}가 존재하며 머신 p_m간에도 간선 e_{im}이 존재한다고 하면, 태스크 t_i가 머신 p_k에 할당되므로 인하여 t_i를 그래프에서 제거하여야 한다. 이 때, 만일 간선 e_{ij}의 가중치가 e_{ik}보다는 작으나 그 외의



(그림 2) 태스크의 할당에 따른 그래프의 재구성

간선에 비하여 그 가중치가 가장 크다면 태스크 t_j는 다음차례에서 태스크 t_i와 병합이 되었을 것이다. 그런데 태스크 t_i가 그래프에서 제거됨으로 인하여 e_{ij}가 제거되면 태스크 t_j와 머신 p_k를 잇는 간선 e'_{jk}의 가중치가 t_j와 다른 머신을 잇는 간선의 가중치나 다른 태스크와의 간선의 가중치보다 작은 경우에 t_j는 t_i와 병합되어 p_k에 할당될 가능성이 낮아진다. 따라서 본 논문에서는 t_i가 p_k에 할당되는 경우 t_i를 제거하기에 앞서 t_i와 연결된 태스크 t_j와 머신 p_k를 잇는 간선의 가중치를 $|e'_{jk}|$와 $|e'_{ij}|$중에서 큰 값으로 변경하여 t_j로 하여금 p_k에 할당되어 t_i와 병합될 수 있는 가능성을 높이도록 하였다. 즉,

$$|e'_{jk}| = \max(|e'_{jk}|, |e'_{ij}|), \ \forall \ j \text{ such that } e_{ij} \in G_T \quad (5)$$

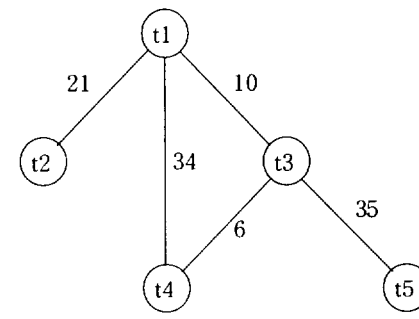
그래프가 재구성이 되면 태스크-머신그래프에서 간선의 가중치가 가장 큰 것을 골라 1) - 2)의 과정을 반복한다. 이상의 방법을 정리하면 알고리즘은 (그림 3)과 같다. 또한, 이 알고리즘의 복잡도는 Kopidakis기법의 경우와 마찬가지로 $O(nm(m+n))$이다. 따라서 복잡도가 $O(nm^4 \log m)$인 Lo의 기법에 비해서 복잡도가 매우 낮은 것을 알 수 있다.

3.3 예제

(그림 4)와 <표 2>는 태스크 그래프와 태스크 연산 비용의 한 예이다. (그림 4)에서 노드는 태스크를 나타내며 노드간의 간선은 두 태스크가 메시지를 교환해야 함을 의미한다. 또한, 간선상의 숫자는 두 태스크간의 통신에 필요한 비용을 의미한다. <표 2>는 5개의 태스크를 머신 p_1, p_2, p_3에서 수행 할 때의 각각의 실행 비용을 나타낸 것이다.

Input : task graph G_T, machine graph G_P;
Output : task-machine mapping;
BEGIN
Create G' adding V_p to G_T;
Draw edges such that e'_{ij} is an edge between t_i and every machine p_j;
Calculate edge weight $|e'_{ij}|$ according to Eq. (2);
repeat
  Find the most weight edge from G';
  /* let e'_{ij} be the most weight edge of G' */
  if e'_{ij} is an edge between a pair of tasks **then**
    /* merge t_i and t_j into big task t_g */
    Cluster t_i and t_j into a big task t_g;
    Adjust edge weight $|e'_{gk}|$, between every machine p_k according to Eq. (3);
    Adjust edge weight $|e'_{go}|$ between every other task t_o according to Eq. (3);
  else if e'_{ij} is an edge between a task t_i and a machine p_j **then**
    /* examine every machine to find out the best machine to allocate t_i */
    Find p_m such that ρ_{im}, according to Eq (4), is minimum;
    Adjust edge weight $|e'_{km}|$, such that $e_{ik} \in G_T$, according to Eq. (5);
    Allocate t_i to p_m;
  endif
until unallocated task exists
END

(그림 3) 제안한 클러스터링 알고리즘

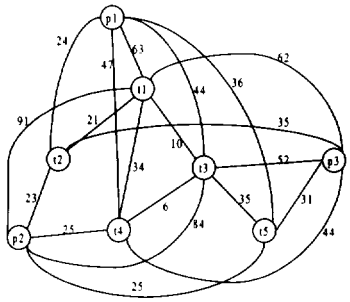


(그림 4) 태스크 그래프

〈표 2〉 태스크 연산비용행렬

	p_1	p_2	p_3
t_1	90	34	92
t_2	34	36	13
t_3	92	13	76
t_4	23	66	28
t_5	20	42	31

(그림 4)의 태스크 그래프와 <표 2>를 토대로 태스크-머신그래프를 구성하면 (그림 5)와 같다. 즉, 5개의 태스크와 3개의 머신이 각각 노드를 구성하고 태스크와 머신노드간의 간선이 추가되었다. 이들 간선의 가중치는 식 (2)과 같이 결정된다. 예를 들어 e_{11}의 가중치는 t_1을 머신 p_2 및 p_3에서 수행하는데 필요한 시간의 평균치인 63으로 결정된다.



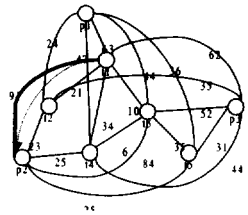
(그림 5) 태스크-머신 그래프

(그림 5)에 대하여 태스크 분할 과정을 살펴보자. 첫 번째로 선택되는 간선은 가중치가 가장 큰 t_1과 p_2를 연결하는 간선이다. 그런데 이 간선의 한쪽이 머신이므로 이 태스크를 할당할 머신을 결정해야 한다. 즉, 모든 머신에 대하여 태스크 t_1의 실행시간에서 각 머신에 할당된 태스크간의 통신시간을 뺀 시간을 계산 한 결과 p_2에서의 값이 가장 작으므로 (그림 6)(a)와 같이 p_2에 할당된다. 마찬가지 방법으로 t_3와 t_4도 각각 (b) 및 (c)와 같이 머신 p_2 및 p_1에 할당된다. (d)는 태스크 t_5를 할당하기 위하여 모든 머신을 조사한 결과, $\rho_{52}=7$이므로 p_2에 할당하는 과정을 설명한 것이다. 즉, 선택된 간선은 t_5의 머신적합성이 p_1에 합당하나 통신시간의 의한 이득을 계산한 결과 p_2할당하는 것이 바람직한 것을 보여주고 있다. 마찬가지 방법으로 t_2는 p_3에 할당되어 (f)와 같이 할당되어 전체 실행 비용 및 통신비용의 합이 186이 된다. 한편 Kopidakis 기법에 의한 결과와 Lo의 기법에 의한 태스크 할당에 따른 작업시간은 각각 199와 213임으로 본 논문에서 제안한 기법의 결과가 가장 좋은 것을 알 수 있다.

4. 실험 및 고찰

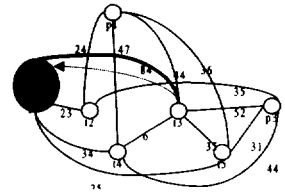
4.1 실험

본 논문에서 제안한 태스크 할당 기법의 성능분석을



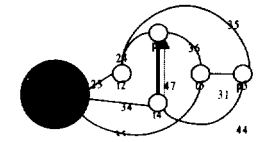
$\rho_{11}=94$, $\rho_{12}=34$, $\rho_{13}=92$

(a)



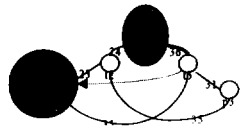
$\rho_{31}=92$, $\rho_{32}=13$, $\rho_{33}=76$

(b)



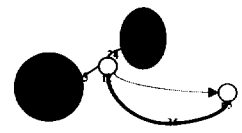
$\rho_{41}=23$, $\rho_{42}=26$, $\rho_{43}=28$

(c)



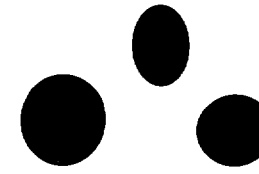
$\rho_{51}=20$, $\rho_{52}=7$, $\rho_{53}=31$

(d)



$\rho_{21}=34$, $\rho_{22}=14$, $\rho_{23}=13$

(e)



(f) 최종상태

(그림 6)

위하여 임의로 여러 가지 태스크 그래프를 발생시켜 기존의 알고리즘과 본 논문에서 제안한 기법의 수행시간을 비교하는 모의실험을 수행하였다. 본 논문에서 비교의 대상으로 삼은 태스크 할당 기법은 Kopidakis 기법[15]과 Lo 기법[16]이다.

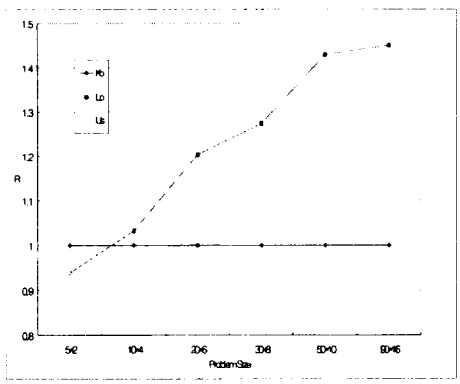
우선, 태스크 그래프를 생성하는 과정에서 각 태스크의 실행 비용은 [1, 100] 의 범위에서의 정규 분포로 생성하였고 통신비용은 총 통신비용(모든 통신비용의 합)과 총 실행 비용(모든 태스크의 실행 비용의 평균의 합)의 상대적 비율인 CCR(Communication to Computation Ratio)의 값을 토대로 정규 분포를 가지는 임의로 발생시켰다.

또한, 각 태스크 할당 기법의 상대적 성능을 비교하기 위해서 Kopidakis의 기법을 기준으로 Lo 및 본 논문에서 제안한 알고리즘간의 상대적 성능비 r_a을 식 (6)과 같이 정의하고 이를 이용하여 각 태스크 할당 기법의 성능을 상대평가 하였다.

$$r_a = \left(\frac{s_a}{s_{Ko}}\right) \tag{6}$$

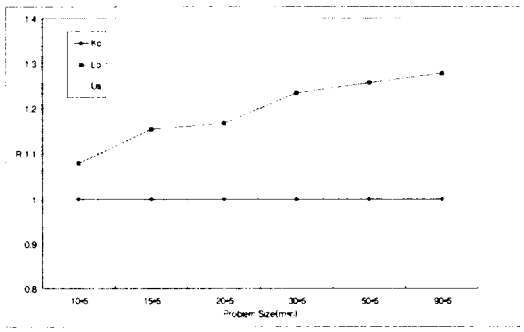
여기서 s_{Ko}는 Kopidakis 기법 적용했을 때 태스크를 수행하는데 필요한 총 비용이며, s_a는 태스크 할당 기법 a를 적용했을 때의 총 비용이다. 따라서 상대적 성능비 r_a가 1 보다 작다는 것은 이 알고리즘이 Kopidakis기법에 비하여 우수함을 의미한다.

(그림 7)은 CCR 값을 0.5로 하였을 때, 문제 크기를 5×2에서 90×16까지 변화시키면서 세 가지 기법의 성능을 측정한 것이다. 여기서 문제의 크기 $m \times n$은 m개의 태스크로 구성된 문제를 n개의 머신에서 태스크를 할당하는 경우를 의미한다. 또한, x축은 문제 크기($m \times n$)를 나타내었으며, y축은 상대적 성능비(r_a)를 표시하였다. 그림에서 문제 크기가 작은 경우에는 Lo의 기법이 Kopidakis의 기법(Ko) 보다 우수하지만 문제 크기가 증가할수록 성능이 상대적으로 점차 떨어지는 것을 보여준다. 그 이유는 Lo의 기법이 태스크-머신 그래프를 가상의 2-태스크-머신 그래프로 전환하여 maximum-flow 알고리즘이 반복적으로 적용되면서 정확성이 저하되기 때문이다. 이에 반하여 본 논문에서 제안하는 클러스터링기법(Us)은 문제 크기가 증가함에도 불구하고 모든 경우에 대하여 Kopidakis기법보다 성능이 좋은 것을 알 수 있다.



(그림 7) 문제 크기 변화에 따른 성능 비교

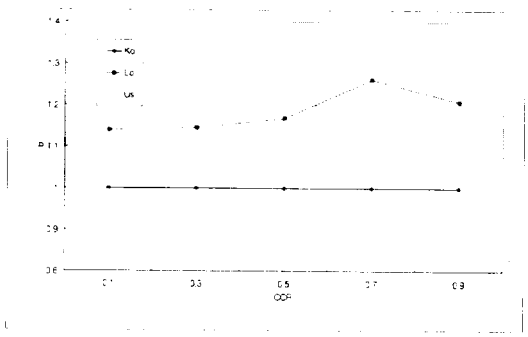
(그림 8)은 머신의 수를 5개, CCR을 0.5로 고정시키고 태스크의 수를 10에서 90까지 변화시키면서 각 알고리즘에 의한 태스크 할당 결과를 비교한 것으로, 본 논문에서 제안한 기법이 모든 문제 크기에 대해서 다른 방법에 비하여 우수함을 알 수 있다.



(그림 8) 태스크 수의 변화에 따른 성능 비교

(그림 9)는 문제 크기를 일정하게 하고 CCR을 변화시키면서 태스크 할당 기법의 성능을 비교한 것이다. 그림에서 CCR이 증가할수록 본 논문에서 제안한 방법이 Kopidakis나 Lo의 기법에 비하여 성능이 점차 좋아지는 것을 볼 수 있다. 이는 본 논문에서 제안한 기법이 태스크 할당시 태스크의 머신 적합성 뿐 아니라 통신비용이 충분히 고려되기 때문인 것으로 판단된다. 반대로 Lo의 기법은 CCR이 증가함에 따라 성능이 점차 나빠지는 것을 알 수 있다.

또한 각 태스크 할당 기법들을 이용한 태스크 할당 결과와 최적 할당 결과도 비교하여 보았다. 최적의 경우에 대한 성능비 s_a/s_o를 토대로 각각의 태스크 할당

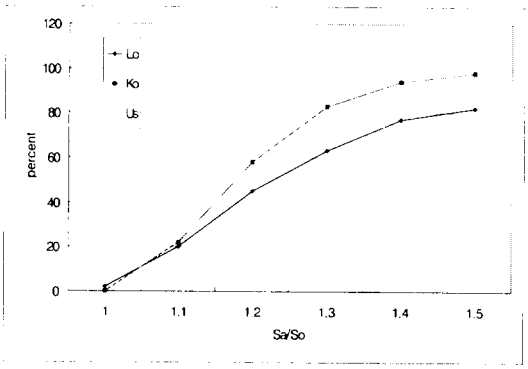


(그림 9) CCR 변화에 따른 성능 비교

〈표 3〉 최적 할당과의 태스크 할당 결과 비교

		s_a/s_o=1.0	s_a/s_o<=1.1	s_a/s_o<=1.2	s_a/s_o<=1.3
4×3	Lo	65	88	96	98
	Ko	41	72	87	96
	Us	83	96	99	100
6×3	Lo	43	78	90	95
	Ko	34	55	81	92
	Us	71	93	100	100
9×3	Lo	27	64	89	96
	Ko	16	70	95	99
	Us	53	94	100	100
13×4	Lo	4	31	61	77
	Ko	10	68	92	98
	Us	7	93	100	100
18×4	Lo	2	20	45	62
	Ko	0	22	58	83
	Us	6	83	98	100

기법을 이용하였을 때 몇 번이나 성능비를 만족시키는 경우수가 나타났는가를 비교하였다. 성능비에서 s_a는 태스크 할당 기법 a를 적용했을 때 산출되는 작업시간이며, s_o는 최적 할당 기법을 이용하였을 때의 작업시간이다. 태스크의 최적 할당 기법으로는 Sinclair[28]가 제안한 Branch & bound 알고리즘을 이용하였다. 실험 결과는 <표 3>과 같다. 표에서 각 열은 최적에 대한 성능비 s_a/s_o가 각각 1.0, 1.1이하, 1.2이하 및 1.3이하인 경우를 의미한다. 예를 들어 s_a/s_o가 1.0 경우는 실험 결과가 최적의 경우와 일치하는 횟수를 총 횟수로 나눈 값의 백분율이다. (그림 10)은 <표 3>을 토대로 문제 크기가 18×4인 경우를 그래프로 나타낸 것으로 본 논문에서 제안한 기법에 의하여 태스크 그래프를 할당하였을 때의 작업시간이 최적 할당시의 경우와 비교하여 항상 30%이내임을 보여준다. 즉, 본 논문에서 제안한 기법이 다른 기법에 비하여 최적 할당 결과에 근접한 할당 결과를 더 많이 발생시킴을 알 수 있다.



(그림 10) 문제 크기가 18×4 인 경우 최적 할당과의 비교

5. 결 론

본 논문에서는 여러 개의 머신으로 구성된 분산 환경에서 비방향성 태스크 그래프를 구성하는 태스크들을 효율적으로 할당하는 경험적 태스크 할당기법을 연구하였다. 본 논문에서 제안한 기법은 태스크 그래프에 가용한 머신을 노드로 추가하고 머신과 태스크를 간선으로 연결한 태스크-머신 그래프를 구성하고 각 간선의 가중치를 토대로 가중치가 가장 큰 간선의 양단에 연결된 태스크들을 병합하거나 태스크를 머신에 할당하는 과정을 거친다. 이 과정에서 제안한 기법은 Kopidakis의 기법이 클러스터링 과정에서 머신적합성만을 고려하여 태스크를 할당할 머신을 결정하는 것에 반하여 머신적합성이 다소 떨어지더라도 이웃한 태스크들과의 통신비용이 줄어들어 머신적합성의 저하를 상쇄할 수 있는 머신이 있다면 이 머신을 찾아 태스크를 할당하도록 한다. 다양한 형태의 태스크 그래프에 대한 시뮬레이션을 해 본 결과, 본 논문에서 제시한 클러스터링 기법이 기존의 태스크 할당 기법에 비하여 우수함을 확인할 수 있었다. 또한 최적의 기법과의 비교에서도 제안한 기법이 최적에 준하는 결과를 더 많은 경우에 보여주고 있음을 확인할 수 있었다.

참 고 문 헌

[1] B.N. Bershad and H.M. Levy, "A remote computation facility for a heterogeneous environment," *IEEE Computer*, pp.50-60, May. 1988.

[2] A. Beguelin, J. Dongrra, A. Geist and V. Sunderam, "Visualization and debugging in a heterogeneous environment," *IEEE Computer*, pp.88-95, Jun. 1993.

[3] A. Khokhar, V.K. Prasanna, M. Shaaban and C.L. Wang, "Heterogeneous supercomputing: problem and issue," *Proc. Workshop on Heterogeneous computing*, pp.3-12, 1992.

[4] A. Khokhar, V.K.Prasanna, M. Shaaban and C.L. Wang, "Heterogeneous computing : challenge and opportunities," *IEEE Computer*, pp.18-27, Jun. 1993.

[5] C.M. Pancake, "The changing face of supercomputing," *IEEE Parallel and Distributed Technology*, pp.12-15, Nov. 1993.

[6] M.J. Quinn, *Parallel Computing : Theory and Practice*. McGraw-Hill Book Company, 1993.

[7] H.E. Rewini and T.G. Lewis, "Scheduling Parallel Program Tasks into Arbitrary Target Machines." *Journal Parallel Distributed Computing*, Vol.9, pp.38-153, 1990.

[8] C.Y. Lee, J.J. Hwang, Y.C. Chow, and F.D. Anger, "Multiprocessor scheduling with interprocessor communication delays," *Oper. Res. Lett. 7*, pp.141-147, 1988.

[9] M.A. Iverson and G.J. Follen, "Parallel existing applications in a distributed heterogeneous environment," *Proc. 1995 Workshop Heterogeneous Processing*, pp.93-100, 1995.

[10] Adam T., Chandy K., and Dickson J., "A comparison of list schedulers for parallel processing systems," *Comm. ACM*, pp.685-690, Dec. 1974.

[11] B. Kruatrachue and T. Lewis, "Grain size determination for parallel processing," *IEEE Software*, pp.23-32, Jan. 1988.

[12] H.S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Trans. SE.*, Vol.SE-3, pp.85-93, Jan. 1977.

[13] H.S. Stone, "Critical load factor in distributed computer systems," *IEEE Trans. SE.*, Vol.SE-4, pp.254-258, May. 1978.

[14] V.M. Lo, "Heuristic algorithms task assignment in distributed system," *IEEE Trans. Comput.*, Vol.37, No.11, pp.1384-1397, Nov. 1988.

[15] Y. Kopidakis, M. Lamari, and V. Zissimopoulos, "On the task assignment problem : Two new efficient heuristic algorithms," *Journal Parallel Distributed Computing*, Vol.42, pp.21-29. 1997.

[16] O.J. El-Dessouki, *Program partitioning and load balancing in network computers*, Ph.D. dissertation, Illinois Instit. Technol., Dec. 1978.

[17] C.H. Lee, D. Lee, M. Kim, "Otimal task assignemt in linear array networks," *IEEE Trans. Comput.*, Vol.41, No.7, pp.877-880, Jul. 1992.

[18] S.H. Bokahari, "Dual processor scheduling with dynamic reassignment," *IEEE Trans. SE.*, Vol. 5, No.4, pp.341-349, Jul. 1979.

[19] S.H. Bokahari, "A shortest tree algorithms for optimal assignments across space and time in distributed processor system," *IEEE Trans. SE.*, Vol.7, No.6, pp.583-589, Nov. 1981.

[20] K. Efe, "Heuristic models of task assignment scheduling in distributed systems," *IEEE Computer*, Vol.15, pp.50-56, Jun. 1982.

[21] Hu, T. C., "Parallel Sequencing and Assembly Line Problem," *Oper. Res.*, Vol.9, No.6, pp.244-257, 1961.

[22] S.J. Kim and J.C. Browne, "A general approach to mapping of parallel computation upon multiprocessor architectures," *Proc. Int'l Conf. Parallel Processing*, Vol.3, pp.23-32, 1988.

[23] T. Yang and A. Gerasoulis, "DSC: Scheduling tasks on an unbounded number of processors," *IEEE Trans. Para. Dist. Sys.*, Vol.5, pp.951-967, Sep. 1994.

[24] L.R. Ford, Jr., and D.R. Fulkerson, *Flow in Networks*. Princeton, NJ : Princeton Univ. Press, 1962.

[25] E.A. Dinic, "Algorithms for solution of a problem of maximum flow in a network with power estimation," *Soviet Math. Doklady*, Vol.11, pp.1277- 1280, 1970.

[26] J. Edmonds and R.M. Karp, "Theoretical improvements

in algorithm efficiency for network flow problems," *J. Ass. Comput.*, Vol.19, pp.248-264, Apr. 1972.

[27] A.V. Karazanov, "Determining the maximal flow in a network by the method of preflows," *Soviet Math. Doklany*, Vol.15, No.2, pp.434-437, 1974..

[28] J.B. Sinclair, "Efficient computing of optimal assignments for distributed tasks," *Journal Parallel Distributed Computing*, Vol.4, pp.342-362, 1987.

김 석 일

e-mail : ksi@cbucc.chungbuk.ac.kr

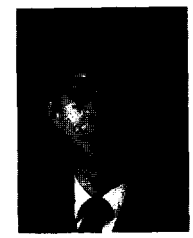
1975년 서울대학교 전기공학과(학사)

1975년~1990년 국방과학연구소 선임 연구원으로 근무

1985년~1989년 미국 North Carolina State University에서 공학박사 취득

1990년~현재 충북대학교 컴퓨터과학과 부교수로 재직 중

관심분야 : 병렬처리 컴퓨터 구조, 슈퍼컴퓨팅, 병렬처리 언어, 이기종 분산처리, 시각장애인 사용자 인터페이스 등임



전 중 남

e-mail : joongnam@cbucc.chungbuk.ac.kr

1981년 연세대학교 전자공학과 학사 취득

1985년 연세대학교 전자공학과 석사 취득.

1990년 연세대학교 전자공학과 박사 취득.

1990년~현재 충북대학교 컴퓨터과학과 부교수로 재직 중.

관심분야 : 컴퓨터 구조, 병렬처리 알고리즘, 공장자동화 등임

김 관 유

e-mail : byblue@john.chungbuk.ac.kr

1997년 충북대학교 컴퓨터과학과 학사취득

1999년 충북대학교 대학원 전자계산학과 석사취득

관심분야 : 병렬처리 알고리즘, 컴퓨터 구조