워크플로우 작업의 효율적인 배치를 위한 다단계 워크플로우 그래프 분할 기법

(A Multilevel Workflow Graph Partitioning Scheme for Efficient Placement of Workflow Tasks)

최 경 훈 † 손 진 현 †† 김 명 호 †††
(Kyung Hoon Choi) (Jin Hyun Son) (Myoung Ho Kim)

요 약 워크플로우는 자동화 및 전산화된 업무 프로세스로 정의되며, 서로 관련성을 가지는 여러 개의 워크플로우 작업들로 구성된다. 오늘날 대부분의 업무 프로세스들은 지리적으로 떨어져 있는 서로 다른 부서 및 회사에서 수행되는 작업들을 포함하기 때문에 워크플로우는 본질적으로 분산성을 가진다. 분산 워크플로우 시스템에서 각 워크플로우 작업은 원격 호스트에 있는 공유 자원들을 활용하여 주어진 역할을 수행하고, 워크플로우 정의에 따라 다음 작업들을 수행하기 위해 제어를 전달한다. 따라서 고성능을 요구하는 워크플로우 환경을 지원하기 위해서는 워크플로우 작업들을 적절한 호스트에 배치해야 한다. 본 논문에서는 효율적인 워크플로우 작업 배치를 위한 다단계 워크플로우 그래프 분할 기법을 제안한다. 이 방법은 워크플로우의 수행 과정에서 발생하는 원격 통신 비용을 최소화하여 워크플로우의 처리 성능을 향상시킬 수 있다.

키워드 : 워크플로우, 워크플로우 작업, 작업 배치, 다단계 그래프 분할

Abstract Workflow is defined as the automation of a business process, and consists of interrelated workflow tasks. Because many modern business processes may involve activities that are geographically distributed between different departments or organizations, workflow inherently has the characteristics of distribution. In distributed workflow systems, each workflow task performs its assigned role by utilizing information resources placed at some hosts, and then transmits workflow execution control to the next tasks in a workflow definition. Hence, it is very important to appropriately allocate workflow tasks to hosts for high performance workflow processing. In this paper, we propose a multilevel workflow graph partitioning scheme for efficient placement of workflow tasks. This method can improve the performance of workflow processing by minimizing the remote communication costs occurred during workflow execution.

Key words : Workflow, Workflow Task, Task Placement, Multilevel Graph Partitioning

1. 서 론

워크플로우(workflow)는 업무 프로세스(business process)의 전체 또는 일부분을 자동화하고 전산화하는 개념으로 정의되며, 워크플로우가 수행되는 동안 업무에 참여하는 사람 또는 프로그램 사이에는 문서, 정보, 작업 등이 정해진 절차에 따라 전달된다[1]. 자동화된 업무 프로세스는 워크플로우 정의(workflow definition)를 통해서 표현된다. 워크플로우 정의는 다양한 워크플로우 작업(workflow task)들과 수행 절차, 그리고 워크플로우 관리에 이용되는 제어 데이타로 구성된다. 각 워크플로우 작업마다 워크플로우 프로세스(workflow process)의 한 단계를 진행하기 위해 해야 할 역할이 할당되며, 이는 사람 또는 에이전트(agent)와 같은 소프트웨어 프

· 본 연구는 CEBT의 부분적인 지원을 받아 수행되었음.
이 논문은 2002학년도 한양대학교 공학기술연구소 연구비에 의하여 부분적으로 연구되었음.
† 비 회 원 : KT 연구원
crucian@kt.co.kr
†† 종신회원 : 한양대학교 컴퓨터공학과 교수
jhson@cse.hanyang.ac.kr
††† 종신회원 : 한국과학기술원 전자전산학과 교수
mhkim@dbserver.kaist.ac.kr
논문접수 : 2002년 4월 4일
심사완료 : 2003년 2월 7일

로세스에 의해 수행된다. 워크플로우 관리 시스템(workflow management system)은 워크플로우 정의를 저장 및 해석하고, 워크플로우 인스턴스(workflow instance)를 생성 및 관리하고, 워크플로우 참가자(workflow participant)들과 응용 프로그램들과의 상호 작용을 제어하기 위한 소프트웨어들로 구성된다. 워크플로우 인스턴스는 워크플로우 프로세스가 수행될 때마다 하나씩 생성되며, 워크플로우 관리 시스템이 워크플로우 정의에 따라 관리하고 종료시킨다[1,2].

오늘날의 복잡하고 규모가 큰 업무 프로세스들은 지리적으로 떨어져 있는 서로 다른 부서 및 회사에서 수행되는 작업들을 포함하는 경우가 많다. 따라서 워크플로우는 본질적으로 분산성을 가지며, 분산 워크플로우 시스템은 이를 효과적으로 지원할 수 있다 [3]. 일반적으로 분산 워크플로우 시스템에서는 워크플로우 작업들과 각 작업의 수행에 필요한 공유 자원들이 여러 호스트들에 배치된다. 각 워크플로우 작업은 원격 호스트에 있는 공유 자원들을 활용하여 주어진 역할을 수행하고, 워크플로우 정의에 따라 다음 작업들을 수행하기 위해 제어를 전달한다. 이러한 환경에서 워크플로우 작업들과 자원들을 부적절하게 분산시키면, 인접한 작업들 사이 또는 각 작업과 자원들 사이의 원격 통신 비용으로 인하여 워크플로우 시스템에 심각한 과부하를 초래할 수 있다. 따라서 고성능을 요구하는 워크플로우 환경을 지원하기 위해서는 워크플로우 작업들과 자원들을 효과적으로 분산시켜야 한다. 데이터베이스, 디스크, 프린터 등과 같은 자원들은 일반적으로 유동성이 거의 없으므로 본 논문에서는 자원들이 이미 호스트에 배치되어 있다고 가정하고 워크플로우 작업들의 배치 문제를 고려한다. 이를 위해 우선 본 논문에서고려하고 있는 분산 워크플로우 시스템의 모델을 설명하고, 그래프 분할(Graph Partitioning)의 개념을 활용한 효율적인 워크플로우 작업 배치 방법을 제안한다. 이 방법은 워크플로우의 수행 과정에서 발생하는 원격 통신 비용을 최소화하여 워크플로우의 처리 성능을 향상시킬 수 있다.

분산 워크플로우 시스템의 성능을 향상시키기 위한 몇 가지 연구들이 이루어져 왔다. [4]에서는 분산 워크플로우 환경에서의 작업 배치 문제를 정의하고, 정수 선형 계획법(ILP: Integer Linear Programming)을 활용한 배치 방법을 제안하였다. 이 방법은 작업의 수가100개 이하인 소규모 워크플로우 시스템에서는 최적의 배치를 결정할 수 있지만, 수행 시간이 지수적으로 길어지기 때문에 대규모 워크플로우 시스템에는 적합하지 않다. 따라서 현대의 복잡하고 광범위한 워크플로우 환경을 지원할 수 있는 워크플로우 작업 배치 방법이 요구된다. [5]에서는 워크플로우 시스템의 사용자가 너무 많아 하나의 워크플로우 서버와 부분망으로 워크플로우 서비스를 효과적으로 제공하기 어려운 경우, 워크플로우 서버와 부분망을 추가하여 통신 비용을 줄이고 작업 부하를 분산시킬 수 있는 워크플로우 수행 환경을 제안하였다. 그러나 워크플로우 작업과 공유 자원들 사이의 통신 비용을 고려하지 않았고, 워크플로우 사용자가 부분망에 배치되기 이전에만 적용될 수 있다는 제약이 있다. 한편 [6]에서는 이러한 제약을 해결하기 위해 동적 서버 할당을 이용하여 통신 부하를 최소화할 수 있도록 워크플로우의 제어를 분산시키는 방법을 제안하였다.

그래프 분할 문제는 주어진 그래프의 노드들을 서로 겹치지 않는 부분 집합들로 분할할 때 서로 다른 분할에 속하는 노드들을 연결하는 간선들의 가중치의 총합을 최소화하는 문제이다. 그래프 분할의 개념은 병렬 처리, 작업 스케줄링, VLSI 설계 등과 같은 광범위한 응용 분야에 적용될 수 있다 [7]. 지금까지 제안된 많은 그래프분할 알고리즘들 중에서 몇 가지 알고리즘들은 적절한 수행 시간 안에 효과적인 분할을 결정할 수 있다고 알려진 다단계 분할 방법을 사용한다 [7,8,9]. 일반적으로 다단계 분할 알고리즘은 그림 1과 같이 연속하는 세단계, 즉 코스닝(coarsening), 분할, 언코스닝(uncoarsening)으로 구성된다. 우선 코스닝 단계에서는 주어진 그래프의 여러 노드들을 한 노드로 합치면서 그래프의 크기를 점점 줄이는 과정을 [7]의 실험에 의해 결정된 임계 값이 될 때까지 반복하고, 분할 단계에서는 코스닝 단계에서 크기가 작아진 그래프를 분할한다. 마지막으로 언코스닝 단계에서는 코스닝 단계에서 반복한 과정을 반대로 수행하면서 작아진 그래프에서 결정된 분할을 원래의 그래프로 되돌린다. 워크플로우 작업 배치는 워크플로우 그래프를 통해 표현될 수 있으므로 결국 그래프 분할 문제라고 할 수 있지만, 본 논문에서 제안하는 방법은 워크플로우의 본질적인 특성들을 적절하게 반영한다.

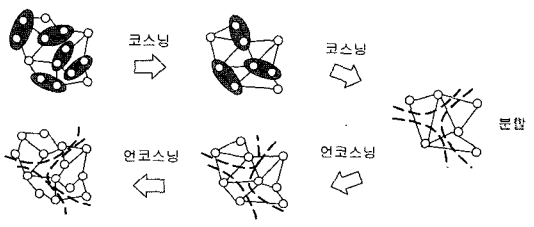


그림 1 다단계 그래프 분할

본 논문은 다음과 같이 구성된다. 2절에서는 본 논문에서 고려하고 있는 분산 워크플로우 시스템의 모델을 설명하고, 3절에서는 이 모델을 기반으로 하여 워크플로우 작업 배치 문제를 정의한다. 4절에서는 분산 시스템의 통신망 구성 상태를 반영하기 위한 호스트 클러스터링에 대해 설명하고, 5절에서는 그래프 기반의 워크플로우 작업 배치 방법을 제안한다. 6절에서는 실험을 통해 제안하는 방법의 효율성을 검증하고, 마지막으로 7절에서는 본 논문의 요약과 공헌을 언급하면서 결론을 맺는다.

2. 분산 워크플로우 시스템

분산 워크플로우 시스템은 그림 2와 같이 워크플로우 서버, 워크플로우 작업, 공유 자원, 워크플로우 정보 저장소 등으로 구성된다. 워크플로우 서버는 작업 스케줄러와 작업 관리자로 구성되며 워크플로우 작업을 수행하는 주체이다. 각 서버는 작업과 일대일 대응 관계를 가지면서 그 작업과 동일한 생명 주기를 갖는다[3,10]. 작업 스케줄러는 워크플로우가 수행되는 동안 인접한 작업들 사이의 제어흐름을 관리하고, 이를 위해 워크플로우 정보 저장소에 저장되어 있는 정보를 이용하여 다른 작업 스케줄러들과 통신한다. 각 작업 스케줄러가 워크플로우 정보 저장소를 접근하는 비용은 항상 동일하다고 가정한다. 이 가정은 워크플로우 정보 저장소를 여러 호스트들에 중복하는 방법으로 만족시킬 수 있다. 인접한 작업들 사이의 제어 흐름은 그림 2의 작업1과 작업 2 사이와 같이 지역 전달이거나 작업2와 작업 3 사이와 같이 원격 전달을 통해 이루어진다. 작업 관리자는 작업 스케줄러에 의해서 선택된 작업이 여러 공유 자원들을 접근하여 그 역할을 수행하고 종료할 수 있도록 관리한다. 자원들의 위치와 각 작업이 접근하는 자원들은 미리 정의되어 있다고 가정한다. 제어 전달과 마찬가지로 자원 접근도 그림2에서 작업 1이 자원 1을 접근할 때와 같이 지역 접근이거나 작업 2가 자원 3을 접근할 때와 같이 원격 접근으로 구분된다. 한편 워크플로우 정보 저장소에는 워크플로우 서버에 의해서 사용되거나 워크플로우 관리 시스템에서 발생하는 모든 정보가 저장된다. 예를 들면 워크플로우 정의, 워크플로우 관련 데이터(workflow relevant data), 워크플로우 제어 데이터(workflow control data), 워크플로우 참가자들의 조직 정보 등이 저장된다.

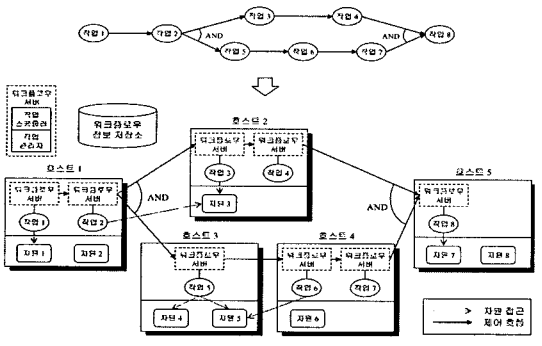


그림 2 분산 워크플로우 시스템

3. 워크플로우 작업 배치

분산 워크플로우 시스템의 처리 성능을 향상시키기 위해서는 워크플로우 작업들을 적절한 호스트에 배치하여 불필요한 원격 접근 및 전달을 줄여야 한다. 워크플로우 작업이 특정한 공유 자원들을 빈번하게 접근하는 경우에는 그 작업을 관련 자원들과 가급적 동일한 호스트에 위치시키고, 워크플로우 정의에서 인접한 작업들도 가능한 한 가까이 배치하여야 한다. 원격 통신 비용뿐만 아니라 호스트가 지원할 수 있는 최대 처리 용량도 고려해 주어야 한다. 만약 호스트의 최대 처리 용량을 초과하여 작업들이 배치된다면 과부하가 발생하여 관련된 워크플로우들의 수행 성능이 떨어지게 되므로 작업 부하를 분산하여 모든 호스트들의 처리 용량을 효율적으로 활용할 수 있어야 한다. 호스트들의 총 처리 용량은 모든 작업들을 수용할 수 있을 만큼 충분하다고 가정한다. 분산 워크플로우 시스템에서 워크플로우 정의와 각 워크플로우 작업이 접근하는 공유 자원들에 대한 정보가 주어졌을 때, 워크플로우 작업 배치 문제는 각 호스트에 배치되는 작업들의 총 처리 용량이 그 호스트의 최대 처리 용량을 넘지 않으면서 원격 자원 접근과 원격 제어 전달로 인하여 발생하는 총 처리 비용을 최소화하도록 작업들을 호스트에 배치하는 문제로 정의된다. 작업 배치 이전에 주어지는 기반 정보는 다음과 같다.

- $freq(t_i)$: 워크플로우 작업 t_i 가 수행되는 빈도
- $num_{res}(t_i, h_j)$: 작업 t_i의 수행에 필요한 공유 자원 들 중에서 호스트 h_j에 위치하는 자원의 수
- $adj(t_i, t_j) = \begin{cases} 1 & \text{작업 } t_i\text{와 작업 } t_j\text{가 워크플로우 정의에서 서로 인접한 경우} \\ 0 & o.w. \end{cases}$
- α : 작업이 원격 자원을 접근할 때 발생하는 추가적인 비용
- β : 작업들 사이의 원격제어 전달로 인하여 발생하는 추가적인 비용
- $\text{cap}_{task}(t_i)$: 작업 t_i를 처리하기 위해 필요한 용량

● $\mathrm{cap}_{host}(h_j)$: 호스트 h_j 가 지원할 수 있는 최대 처리 용량

각 작업이 수행되는 빈도는 그 작업이 속한 워크플로우 프로세스의 수행 빈도와 워크플로우 정의를 이용하여 계산할 수 있다. 2절에서 각 작업이 접근하는 자원들은 미리 정의되어 있다고 가정하였기 때문에 $num_{res}(t_i, h_j)$의 값은 워크플로우 정의에서 구할 수 있고, α와 β의 값은 시스템 환경에 따라 결정된다. 한편 작업 배치의 결과로 결정되는 정보는 다음과 같다.

● $alloc(t_i, h_j) = \begin{cases} 1 & \text{작업 } t_i\text{가 호스트 } h_j\text{에 배치된 경우} \\ 0 & o.w. \end{cases}$

● $remote_{access}(t_i, h_j) = \begin{cases} 1 & \text{작업 } t_i\text{가 호스트 } h_j\text{에 위치한 자원들을 원격 접근하는 경우} \\ 0 & o.w. \end{cases}$

● $remote_{transfer}(t_i, t_j) = \begin{cases} 1 & \text{작업 } t_i\text{와 작업 } t_j \text{ 사이의 제어 흐름이 원격 전달인 경우} \\ 0 & o.w. \end{cases}$

위에서 정의된 표기법들을 사용하여 작업 배치 문제를 정형화하면 다음과 같다.

$$Minimize\ \ Cost_{access} + Cost_{transfer} \tag{1}$$

$$subject\ to\ \sum_{i=1}^{n} \mathrm{cap}_{task}(t_i) alloc(t_i, h_j) \le \mathrm{cap}_{host}(h_j)(1 \le j \le m) \tag{2}$$

$$where\ Cost_{access} = \alpha \sum_{i=1}^{n} \sum_{j=1}^{m} freq(t_i)\ num_{res}(t_i, h_j)\ remote_{access}(t_i, h_j)$$

$$Cost_{transfer} = \beta \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \min(freq(t_i), freq(t_j))\ adj(t_i, t_j)\ remote_{transfer}(t_i, t_j)$$

식 (1)에서 $Cost_{access}$는 원격 자원들을 접근하여 발생하는 비용의 총합이고, $Cost_{transfer}$는 인접한 작업들 사이의 원격 제어 전달로 인하여 발생하는 비용의 총합이다. 따라서 식 (1)은 워크플로우의 수행 과정에서 발생하는 원격 통신 비용의 총합을 최소화하기 위한 목적 함수이고, 작업 배치 방법들을 평가하는 비용 모델이 될 수 있다. $Cost_{transfer}$를 계산하는 식에서 $\min(freq(t_i), freq(t_j))$는 인접한 두 작업들 사이의 제어 흐름에 대한 빈도수이다 [4]. 한편 식 (2)는 호스트가 총 처리 용량을 초과하여 작업들을 수용할 수 없다는 제약조건이다. [4]와 [11]에서 작업 배치 문제는0/1 Knapsack 문제로 변형될 수 있고, 0/1 Knapsack 문제는 NP Complete로 알려져 있으므로 작업 배치 문제는 NP Complete이다[12].

4. 호스트 클러스터링

3절의 목적 함수에서 원격 자원 접근과 원격 제어 전달로 인하여 발생하는 추가적인 비용은 각각 α와 β로 항상 일정하다고 간주하였다. 그러나 이는 분산 워크플로우 시스템의 통신망 구성 상태를 적절하게 반영하지 않은 것이다. 일반적으로 분산 시스템은 근거리 통신망(LAN), 도시 지역 통신망(MAN), 원거리 통신망(WAN), 인터넷(Internetwork) 등으로 구성되어 있다. 통신망들은 그림 3과 같이 계층적으로 구성되며 원격 통신 비용은 통신망의 종류에 따라 달라진다. 예를 들면 같은 LAN 안에 있는 호스트 h_1과 h_3 사이의 원격 통신은 각각 다른 MAN에 속하는 호스트 h_1과 h_6 사이의 원격 통신보다 훨씬 적은 비용으로 처리될 수 있다. 호스트 클러스터링은 워크플로우 작업 배치 과정에서 이러한 환경을 반영하기 위한 개념으로 호스트들이 속하는 통신망의 종류에 따라 네 단계, 즉 LAN, MAN, WAN, 인터넷 단계로 나누어서 클러스터링을 수행하는 것이다. 각 단계마다 같은 통신망 안에 있는 모든 호스트들을 합쳐서 하나의 가상호스트로 만드는 과정을 반복하면 그림 4와 같은 호스트 클러스터링 트리가 구성된다. 같은 통신망 안에서의 원격 통신 비용은 거의 동일하기 때문에 각 클러스터링 단계마다 워크플로우 작업들을 배치하면 목적 함수에서 일정한 α와 β값을 사용할 수 있다. 일반적으로 상위 단계에서의 α, β가 하위 단계보다 더 큰 값을 가지므로 $\alpha_I > \alpha_W > \alpha_M > \alpha_L$이고 $\beta_I > \beta_W > \beta_M > \beta_L$이

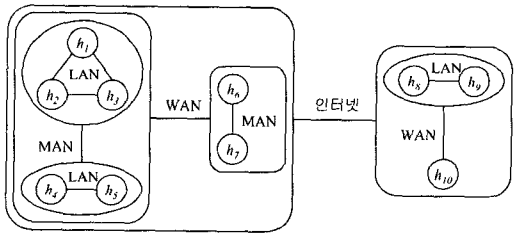


그림 3 분산 시스템의 통신망 구성

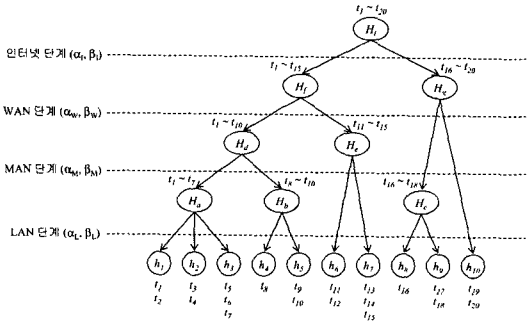


그림 4 호스트 클러스터링 트리

다. 한편 그림 4는 20개의 워크플로우 작업들을 그림 3과 같이 구성된 10개의 호스트들에게 배치하는 과정을 보여준다. 우선 호스트 클러스터링 트리의 루트 노드인 가상 호스트 H_i로 20개의 작업들을 배치하고, 최상위 단계인 인터넷 단계에서 α_I와 β_I를 사용하여 H_j와 H_k로 작업들을 배치한다. 인터넷 단계에서의 배치 결과를 기반으로 하여 WAN 단계에서는 α_w와 β_w를 사용하여 작업들을 배치하고 MAN 단계와 LAN 단계에서도 이 과정을 반복하면최종 결과로 20개의 작업들이 10개의 실제 호스트들에게 배치된다.

5. 그래프 기반의 배치 알고리즘

본 절에서는 워크플로우 작업들을 효율적으로 배치하기 위한 그래프 기반의 방법을 제안한다. 제안하는 방법의 전체 알고리즘은 그림 5에 간략하게 설명되어 있다. 우선 4절에서 언급한 네 단계(LAN, MAN, WAN, 인터넷)로 호스트 클러스터링 트리를 구성하고, 최상위 단계인 인터넷 단계에서 최하위 단계인 LAN 단계까지 차례대로 작업들을 배치한다. 각 단계에서는 작업 배치를 위한워크플로우 그래프(workflow graph)를 구성하고 롤링(rolling) 단계와 언롤링(unrolling) 단계를 수행하여 이를 분할한다. 그래프의 분할이 결정되면 각 작업을 같은 분할에 속하는 호스트에 배치한다. 워크플로우 그래프는 워크플로우 정의와 작업 배치 이전에 주어지는 기반 정보를 표현하는 그래프이며, 다음과 같이 정의된다.

단계 1: 호스트 클러스터링 트리를 구성한다.
단계 2: 구성된 트리의 각 통신망 단계에 대해 상위 단계에서 하위 단계의 순으로 단계 3에서 단계 5까지 반복한다.
단계 3: 워크플로우 그래프를 구성한다.
단계 4: 롤링 단계와 언롤링 단계를 수행하여 워크플로우 그래프를 분할한다.
단계 5: 워크플로우 작업들을 같은 분할에 속하는 호스트에 배치한다.

그림 5 그래프 기반의 배치 알고리즘

워크플로우 그래프 $G=(V, E)$

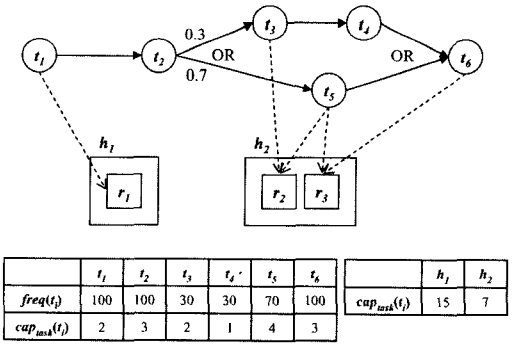
$V(G)$: 워크플로우 작업들과 호스트들로 구성된 노드들의 집합

$E(G)$: 작업과 호스트 사이 또는 작업들 사이의 관계를 나타내는 간선들의 집합

$$w_v(v_i)=\begin{cases} cap_{task}(t_a) & v_i\text{가 작업 노드인 경우} \\ cap_{host}(h_u) & v_i\text{가 호스트 노드인 경우} \end{cases}$$

$$w_e(v_i, v_j)=\begin{cases} \alpha \cdot freq(t_a) \cdot num_{res}(t_a, h_u) \\ \quad (v_i, v_j)=(t_a, h_u)\text{인 경우} \\ \beta \cdot \min(freq(t_a), freq(t_b)) \cdot adj(t_a, t_b) \\ \quad (v_i, v_j)=(t_a, t_b)\text{인 경우} \\ 0 \quad o.w. \end{cases}$$

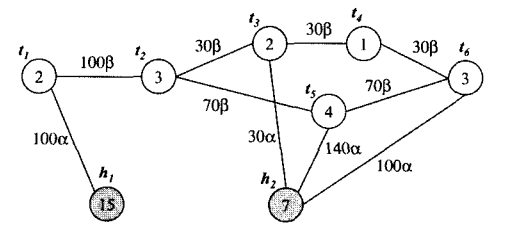
그림 6은 워크플로우 그래프를 구성하는 예이다. 노드들은 작업 또는 호스트를 나타내며 각각 흰색과 검정색으로 구분된다. 작업 노드의 가중치는 그 작업을 처리하기 위해 필요한 용량이고, 호스트 노드의 가중치는 그 호스트가 지원할 수 있는 최대 처리 용량이다. 한편 간선들은 작업이 호스트에 위치하는 자원을 접근하는 경우 작업과 호스트 사이에 연결되고, 워크플로우 정의에서 두 작업들이 서로 인접하는 경우에는 작업들 사이에 연결된다. 간선의 가중치는 두 노드들 사이의 원격 통신 비용을 나타내며, 원격 자원 접근은 $\alpha \cdot freq(t_a) \cdot num_{res}(t_a, h_u)$ 로 계산되고 원격 제어 전달은 $\beta \cdot \min(freq(t_a), freq(t_b)) \cdot adj(t_a, t_b)$ 로 계산된다. 구성된 워크플로우 그래프는 롤링 단계와 언롤링 단계를 통해 적절하게 분할된다. 롤링 단계에서는 워크플로우 그래프의 두 노드들을 한 노드로 합치면서 간선의 수를 줄이는 과정을 총 노드의 수가 호스트의 수와 같아질 때 까지 반복하고, 언롤링 단계에서는 롤링 단계에서 반복한 과정을 반대로 수행하면서 워크플로우 그래프의 분할을 결정한다. 각 단계의 세부과정은 다음 절에 구체적으로 설명되어 있다.



	t_1	t_2	t_3	t_4	t_5	t_6
$freq(t_i)$	100	100	30	30	70	100
$cap_{task}(t_i)$	2	3	2	1	4	3

	h_1	h_2
$cap_{task}(t_i)$	15	7

가) 워크플로우 정의 및 기반 정보



나) 워크플로우 그래프

그림 6 워크플로우 그래프

5.1 롤링 단계

롤링 단계에서는 워크플로우 그래프의 크기를 최소화하여 작업 배치 문제의 복잡도를 줄이기 위해 그래프의 두 노드들을 한 노드로 합치면서 간선의 수를 줄이는 과정을 반복한다. 일반적인 그래프에서 간선 e로 연결된 노드 v_i와 v_j를 합쳐서 새로운 노드 v_k를 만들게 되면 v_i 또는 v_j에 연결된 간선들 중에서 e를 제외한 모든 간선들이 v_k에 연결된다 [13]. 워크플로우 그래프의 노드들을 합칠 때에는 부가적으로 표 1의 세 가지 규칙들이 적용된다. 그림 7은 워크플로우 그래프에서 간선 e_1과 e_2로 연결된 노드들을 합치는 예이다. e_1이 연결하는 노드들은 모두 작업 노드이므로 이들을 합치기 위해서는 규칙 1이 적용되고, 작업 노드와 호스트 노드를 연결하는 e_2에는 규칙 2가 적용된다. 규칙 2에서 생성된 호스트 노드의 가중치는 그 호스트의 남아 있는 처리 용량을 나타낸다. 한편 그림 7의 간선 e_3과 e_4는 규칙 3을 통해 생성된다. 작업 배치 문제의 정의에 따라 위의 규칙들은 두 호스트 노드들이 합쳐지는 경우를 고려하지 않는다.

롤링 단계에서는 워크플로우 그래프의 매칭(matching)을 구하여 그 크기를 줄이고 작아진 그래프에 대해서도 이 과정을 반복한다. 일반적인 그래프의 매칭은 서로 같은 노드를 연결하지 않는 간선들의 집합으로 정의된다 [8,9]. 이 정의를 기반으로 하여 워크플로우 그래프의 최대 매칭(maximal matching)은 표 2의 세 가지 조건들을 만족하는 매칭으로 정의된다. 조건 1은 모든 가능한 간선들을 선택하여 이들이 연결하는 노드들을 합치면서 워크플로우 그래프의 크기를 최대한 줄이기 위한 것이고, 조건 2에서 가중치가 큰 간선은 두 노드들 사이의 원격 통신 비용이 크다는 것을 의미하므로 이러한 노드들을 합쳐서 같은 호스트에 배치하면 총 처리 비용을 효과적으로 줄일 수 있다. 따라서 조건 2는 작업 배치 문제의 목적 함수와 잘 부합된다. 한편 조건 3은 작업 배치 문제의 정의에 따라 두 호스트 노드들이 합쳐지는 것을 막기 위한 것이다. 그림 8은 그림 6에서 구성된 워크플로우 그래프를 이용하여 롤링 단계를 수행하는 과정을 보여준다. 타원으로 묶여 있는 간선들은 최대 매칭을 나타내며, α와 β의 값의 비는 2:1로 가정한다. 각 단계에서 주어진 워크플로우 그래프 G_i의 최대 매칭을 구하여 노드들을 합치고 크기가 작아진 그래프 G_{i+1}을 이용하여 다음 단계를 반복적으로 수행하면 최종 결과로 생성된 그래프에는 2개의 호스트 노드들만 남는다.

표 1 워크플로우 그래프의 노드 통합 규칙

규칙 1	합쳐지는 두 노드들이 모두 작업 노드이면 새로 생성되는 노드는 작업 노드이며 그 가중치는 합쳐진 두 노드들의 가중치의 합과 같다.
규칙 2	작업 노드 t_i와 호스트 노드 h_j가 합쳐지면 새로 생성되는 노드는 호스트 노드이며 그 가중치는 h_j의 가중치에서 t_i의 가중치를 뺀 값을 가진다.
규칙 3	두 노드들을 합치는 과정에서 노드들에 연결된 간선들이 합쳐져 새로운 간선이 생성되면 이 간선의 가중치는 합쳐진 간선들의 가중치의 합과 같다.

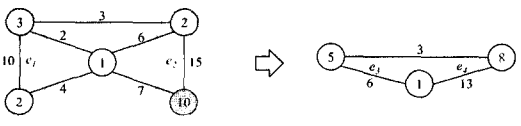


그림 7 워크플로우 그래프의 노드 통합 과정

표 2 최대 매칭의 조건

조건 1	이미 선택된 간선들과 겹치지 않도록 간선을 더 이상 추가할 수 없다.
조건 2	가중치가 큰 간선에서 가중치가 작은 간선의 순으로 선택된다.
조건 3	두 호스트 노드들을 연결하는 간선은 선택될 수 없다.

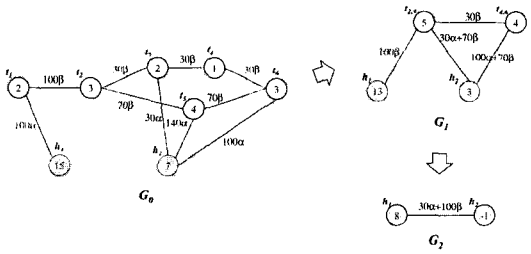


그림 8 롤링 단계의 수행 과정

5.2 언롤링 단계

언롤링 단계에서는 롤링 단계에서 크기가 작아진 워크플로우 그래프를 원래의 그래프로 되돌리면서 분할한다. 워크플로우 그래프의 분할이 결정되면 각 분할은 하나의 호스트 노드와 그 호스트에 배치되는 작업 노드들로 구성된다. 롤링 단계에서 두 노드들을 합쳐서 생성된 노드들이 다시 원래의 노드들로 나누어지기 때문에 각 단계에서 주어진 워크플로우 그래프는 새로 나누어진 노드들과 이전 단계에서 분할이 결정된 노드들로 구성된다. 각 단계의 워크플로우 그래프를 분할할 때 그래프의 모든 노드들에 대해 표 3의 세 가지 규칙들이 적용된다. 그림 9는 그림 8에서 크기가 작아진 워크플로우

표 3 워크플로우 그래프 분할 규칙

규칙 1	작업 노드 t_i를 호스트 노드 h_u에서 h_v로 배치하였을 때 총 처리 비용이 최소화된다면 t_i를 h_v와 같은 분할에 속하게 된다.
규칙 2	t_i를 h_u에서 h_v로 배치하였을 때 총 처리 비용은 증가하지 않고 호스트들의 작업 부하가 더 균등하게 분산된다면 t_i를 h_v와 같은 분할에 속하게 한다.
규칙 3	위의 규칙 1과 규칙 2가 적용되지 않는 경우에는 t_i를 h_u와 같은 분할에 속하게 한다.

(단, t_i는 이전 단계에서 h_u와 같은 분할에 속하거나 h_u와 합쳐졌다가 새로 나누어진다고 가정한다.)

그래프를 이용하여 언롤링 단계를 수행하는 과정을 보여준다. 롤링 단계의 최종 결과인 그래프 G_2는 점선으로 연결된 두 개의 호스트 노드들로 구성되며, 점선으로 연결된 노드들은 서로 배치 관계를 가지지 않는다. 호스트 노드의 가중치는 그 호스트의 남아 있는 처리 용량을 나타내므로 가중치가 -1인 호스트 노드 h_2는 과부하 상태이고 h_1은 처리 용량에 여유가 있음을 알 수 있다. G_2의 다음 단계 그래프 G_1에서 새로 나누어진 작업 노드 $t_{2,3}$은 롤링 단계에서 h_1과 합쳐졌던 노드이지만 h_1과 h_2에 배치하였을 때 발생하는 원격 통신 비용이 각각 $30\alpha + 100\beta$와 100β이므로 규칙 1에 따라 h_2와 같은 분할에 속하게 되며, 이는 $t_{2,3}$과 h_2를 연결하는 실선으로 나타낸다. 한편 $t_{4,6}$은 규칙 3에 따라 h_2에 배치된다. 언롤링 단계의 마지막 그래프 G_0에서는 t_2가 규칙 2에 따라 h_1에 배치되고 다른 작업 노드들은 규칙 3에 따라 배치된다. 규칙 1과 규칙 2가 적용되면 작업 노드들이 롤링 단계에서 결정된 호스트들과 다른 호스트들에게 배치되므로 호스트들의 남아 있는 처리 용량이 갱신되어야 한다. G_0의 분할이 결정되면 각 호스트 노드에 실선으로 연결된 작업 노드들이 그 호스트로 배치된다.

지금까지는 호스트들의 남아 있는 처리 용량을 고려하지 않으면서 원격 통신 비용을 최소화하도록 작업 노드들을 배치하였기 때문에 그림 9의 G_0에서 가중치가

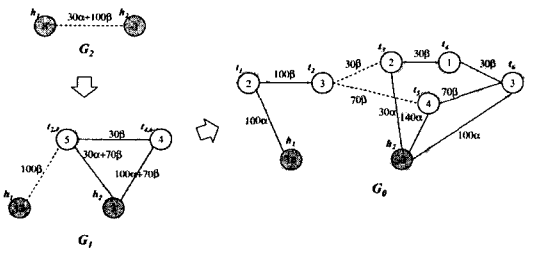


그림 9 언롤링 단계의 수행 과정

음수인 호스트 노드 h_2처럼 과부하 상태에 있는 호스트들이 생길 수 있다. 따라서 가중치가 음수인 호스트들에게 배치된 작업 노드들을 가중치가 양수인 호스트들로 이동시켜야 하며, 이동시킬 작업 노드들을 선택하기 위한 적절한 기준이 필요하다. 이러한 문제를 해결하기 위해 작업 노드 t_i의 이동으로 인하여 발생하는 추가적인 비용과 t_i의 가중치의 비를 나타내는 CPW(cost per weight)를 사용한다. 전체 호스트 노드의 수가 m개이고 호스트 노드 h_u에 k개의 작업 노드들이 배치되어 남아 있는 처리 용량이 음수라고 가정하면, h_u에 배치된 각 작업 노드의 CPW는 다음과 같이 계산된다.

$$CPW(t_i) = \frac{cost_{move}(t_i)}{w_v(t_i)}$$

$$cost_{move}(t_i) = Min\left\{ \sum_{v_a \in h_u} w_e(t_i, v_a) - \sum_{v_b \in h_v} w_e(t_i, v_b) \right\} \quad (1 \le v \le m, v \ne u) \quad (3)$$

식 (3)에서 $w_v(t_i)$는 작업 노드 t_i를 처리하기 위해 필요한 용량이고 $cost_{move}(t_i)$는 t_i를 h_u에서 h_v로 이동시킬 때 발생하는 최소한의 원격 통신비용을 나타낸다. $cost_{move}(t_i)$를 계산하는 식에서 $v_a \in h_u$의 의미는 작업 노드 v_a와 호스트 노드 h_u 사이에 실선으로 연결된 경로가 존재하여 v_a가 h_u에 배치되어있음을 나타낸다. 과부하 상태인 호스트들의 작업 부하를 분산시키기 위해서는 가장 작은 CPW 값을 가지는 작업 노드 t_i를 선택하여 h_u에서 h_v로 이동시키고 각 작업 노드의 CPW 값을 갱신하는 과정을 반복해야 한다. 그림 10은 그림 9의 G_0에서 가중치가 음수인 호스트 노드 h_2에 배치된 작업 노드들을 가중치가 양수인 h_1로 이동시키는 과정을 보여준다. h_2에 배치된 작업 노드들 중에서 가장 작은 CPW 값을 가지는 t_3을 선택하여 G_0'와 같이 h_1에 배치하고, 남아 있는 작업 노드들 중에서 가장 작은 CPW 값을 가지는 t_4을

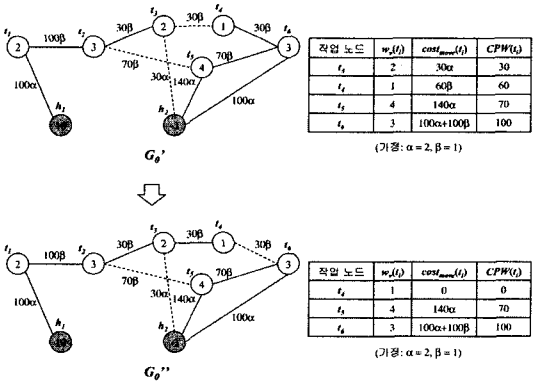


그림 10 호스트들의 처리 용량을 고려한 재배치

선택하여 G_0''와 같이 h_1에 배치한다. G_0''에는 가중치가 음수인 호스트 노드가 없으므로 작업 배치의 모든 과정이 끝나게 된다. 언롤링 단계의 전체 알고리즘은 그림 11에 설명되어 있다.

단계 1: 분할 단계에서 크기가 작아진 워크플로우 그래프와 원래의 그래프로 되돌리면서 각 단계의 그래프에 대해 단계 2에서 단계 5까지 반복한다.
단계 2: 그래프의 분할에 더 이상 변화가 없을 때까지 각 작업 노드 t_i에 대해 단계 3에서 단계 5까지 수행한다. (t_i는 이전 단계에서 h_w와 같은 분할에 속하거나 h_w와 합쳐졌다가 새로 나누어진다고 가정한다)
단계 3: 만약 t_i가 h_w에서 h_r로 배치하였을 때 총 처리 비용이 최소화된다면 t_i가 h_r와 같은 분할에 속하게 한다.
단계 4: 만약 t_i가 h_w에서 h_r로 배치하였을 때 총 처리 비용은 증가하지 않고 호스트들의 작업 부하가 더 균등하게 분산된다면 t_i가 h_r와 같은 분할에 속하게 한다.
단계 5: 단계 3과 단계 4의 조건들을 만족하지 않는 경우에는 t_i가 h_w와 같은 분할에 속하게 한다.
단계 6: 가중치가 음수인 호스트 노드들에게 배치된 작업 노드들의 CPW 값을 계산하여 가장 작은 값을 가지는 작업 노드 t_i를 선택한다.
단계 7: t_i가 CPW 계산 과정에서 결정된 가중치가 양수인 호스트 노드로 재배치한다.
단계 8: 가중치가 음수인 호스트 노드가 없을 때까지 단계 6과 단계 7을 반복한다.

그림 11 언롤링 단계 알고리즘

6. 실 험

본 절에서는 그래프 기반의 워크플로우 작업 배치 방법의 효율성을 몇 가지 실험을 통해 평가한다. 기존의 작업 배치 방법들 중 비용 모델이 그래프 기반의 방법과 유사하여 정량적인 비교가 가능한 것은 정수 선형 계획법(ILP)을 이용한 [4]이므로 몇 가지 예제들에 대해 각 방법의 작업배치 결과를 비교하고, 추가적으로 간단하고 직관적인 두 가지 탐욕(greedy) 기법들의 작업 배치 결과와도 비교한다. ILP 기반의 방법은 소규모 워크플로우 시스템에서는 최적의 작업 배치를 결정할 수 있지만 워크플로우 작업의 수가 늘어나면 수행 시간이 지수적으로 증가한다는 단점이 있다[4]. 탐욕 기법들은 원격 자원들을 접근하여 발생하는 비용을 최소화하도록 워크플로우 작업을 배치하거나 원격 제어 전달로 인한 비용을 최소화하도록 배치한다. 이들을 각각 탐욕(자원) 기법과 탐욕(제어) 기법으로 부르기로 한다. 작업 배치 방법들을 평가하기 위한 비용 모델로 2절에서 언급한 식 (1)과 (2)를 사용하고 α와 β의 값의 비는 2:1로 가정한다. 값의 비를 다르게 가정하면 각 방법의 배치결과도 달라질 수 있다. 정수 선형 계획법의 패키지로는 지금까지 제조 스케줄링, 항공 경로 설정, 공장에서의 원재료 혼합, 교통 정책 등 많은 분야에서 활용된 CPLEX를 사용하며, 펜티엄3 (600MHz)와 384M 메모리의 개인용 컴퓨터에서 실험한다.

그림 12는 실험에 사용된 워크플로우 정의들이다. 각 워크플로우 정의를 구성하는 작업의 수는 6개, 15개, 25개이며, 위에서 언급한 네 가지 방법들을 이용하여 배치된 워크플로우 작업들의 총 처리 비용이 작업의 수가 늘어남에 따라 어떻게 변화하는지 알 수 있다. 각 작업 배치 방법의 총 처리 비용은 그림 13과 같다. 탐욕 기법들은 상반되는 두 가지 비용들 중에서 하나만을 최대한 줄이도록 배치하기 때문에 탐욕(자원) 기법의 경우 원격 자원 접근 비용이 워크플로우 정의 1, 2, 3에서 각각 60, 400, 300으로 네 가지 방법들 중 가장 작지만 원격제어 전달 비용은 가장 크다. 반면에 탐욕(제어) 기법은 원격 제어 전달 비용이 각각 0, 400, 0으로 최소이지만 원격 자원 접근 비용은 다른 방법들보다 커서 결국 자원 접근 비용과 제어 전달 비용을 합친 총 처리 비용은 탐욕

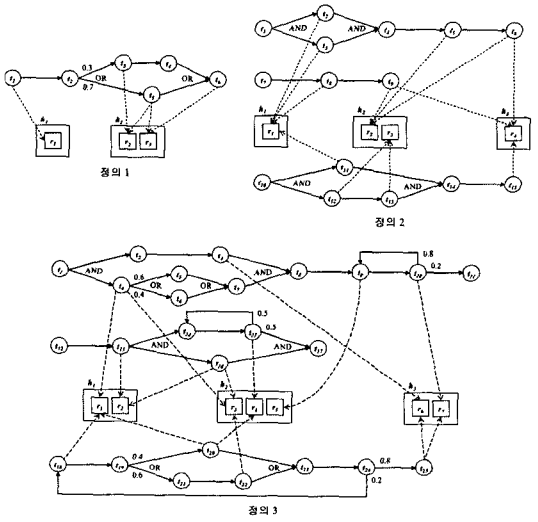


그림 12 실험에 사용된 워크플로우 정의

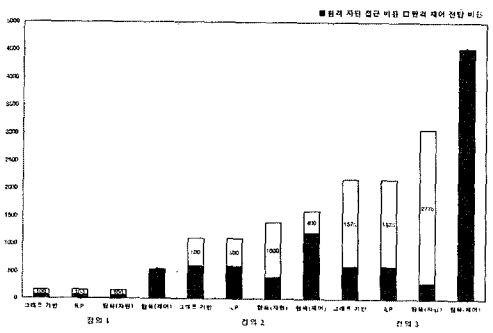


그림 13 각 작업 배치 방법의 총 처리 비용

(자원) 기법과 탐욕(제어) 기법 모두 그래프 기반의 방법보다 크고 작업의 수가 늘어날수록 그 차이가 더 커짐을 알 수 있다. 예를 들면 그래프 기반의 방법과 탐욕(제어) 기법의 총 처리 비용 차이는 작업의 수가 6개, 15개, 25개일 때 각각 380, 500, 2325이다. 한편 ILP 기반의 방법은 총 처리 비용을 최소화할 수 있지만 지수 시간 복잡도를 가지므로 대규모 워크플로우 시스템에 적용하기에는 제한적이다. 본 논문에서 제안한 그래프 기반의 방법은 실험에 사용된 모든 워크플로우 정의에서 ILP 기반의 방법과 총 처리 비용이 동일한 최적의 작업 배치를 결정할 수 있고 다항식 시간 복잡도를 가지므로 다른 작업 배치 방법들보다 더 효율적이고 실용적이라고 할 수 있다.

7. 결 론

분산 워크플로우의 작업 배치는 워크플로우의 처리 성능을 개선하기 위해 각 작업의 수행에 필요한 공유 자원 정보와 워크플로우의 제어 흐름 정보를 기반으로 하여 작업들을 적절한 호스트에 배치하는 문제이다. 본 논문에서는 NP Complete로 알려진 워크플로우 작업 배치 문제를 효과적으로 해결하기 위해 우선 분산 워크플로우 시스템의 모델과 비용 모델에 대해서 기술하고 그래프 분할의 개념을 활용한 경험적(heuristic) 방법을 제안하였다. 이 방법은 워크플로우의 수행 과정에서 발생하는 불필요한 원격 자원 접근과 원격 제어 전달을 최소화하여 고성능을 요구하는 워크플로우 환경을 효율적으로 지원할 수 있고, 호스트들의 작업 부하를 균등하게 분산하여 워크플로우 시스템의 처리 용량을 효율적으로 활용할 수 있다.

참 고 문 헌

[1] P. Lawrence, Workflow Handbook 1997, John Wiley & Sons Ltd, 1997.

[2] F. Leymann and D. Roller, Production Workflow: Concepts and Techniques, Prentice Hall, N.J., 1999.

[3] J. A. Miller, A. P. Sheth, K. J. Kochut, and X. Wang, "CORBA-Based Run-Time Architectures for Workflow Management Systems," Journal of Database Management, 7(1):16-27, 1996.

[4] S. K. Oh, J. H. Son, Y. J. Lee, and M. H. Kim, "An Efficient Method for Allocating Workflow Tasks to Improve the Performance of Distributed Workflows," In Proceedings of the International Conference on Computer Science and Informatics, 2000.

[5] T. Bauer and P. Dadam, "A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration," In Proceedings of the 2nd IFCIS Conference on CoopIS, pages 99-108, 1997.

[6] T. Bauer and P. Dadam, "Efficient Distributed Workflow Management Based on Variable Server Assignments," In Proceedings of the Conference on Advanced Information Systems Engineering, pages 94-109, 2000.

[7] G. Karypis and V. Kumar, "Multilevel k-way Partitioning Scheme for Irregular Graphs," Journal of Parallel and Distributed Computing, 48(1):96-129, 1998.

[8] B. Hendrickson and R. Leland, "A Multilevel Algorithm for Partitioning Graphs," Technical Report SAND93-1301, Sandia National Laboratories, 1993.

[9] T. Bui and C. Jones, "A Heuristic for Reducing Fill-In in Sparse Matrix Factorization," In Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing, pages 445-452, 1993.

[10] J. Miller, D. Palaniswami, A. Sheth, K. Kochut, and H. Singh, "WebWork: METEOR2's Web-based Workflow Management System," Journal of Intelligence Information Management Systems, 10(2): 185-215, 1997.

[11] S. Y. Hwang and C. T. Yang, "Component and Data Distribution in a Distributed Workflow Management System," In Proceedings of the IEEE Software Engineering Conference, pages 244-251, 1998.

[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, Introduction to Algorithms, McGraw-Hill Book Company, 1998.

[13] D. B. West, Introduction to Graph Theory, Prentice-Hall, 2001.

최 경 훈

2000년 한국과학기술원 전산학 학사. 2002년 한국과학기술원 전산학 석사. 2002년 한국과학기술원 ROPAS 연구원. 2003년~현재 KT 전임연구원. 관심 분야는 워크플로우, 분산데이타베이스, XML, 데이타마이닝

손 진 현

1996년 서강대학교 전산학과 학사. 1998년 한국과학기술원 전산학과 석사. 2001년 한국과학기술원 전자전산학과 박사. 2001년 9월~2002년 8월 한국과학기술원 전자전산학과 박사후 연구원. 2002년 9월~현재 한양대학교 컴퓨터공학과 전임강사. 관심 분야는 데이타베이스, e-비즈니스, 워크플로우, 미들웨어, XML

김 명 호

1982년 서울대학교 컴퓨터 공학과 학사. 1984년 서울대학교 컴퓨터 공학과 석사. 1989년 MICHIGAN 주립대 전산학과 박사. 1989년 MICHIGAN 주립대 연구원. 1989년 ~ 1993년 한국과학기술원 조교수. 1993년 ~ 1999년 한국과학기술원 부교수. 1999년 ~ 현재 한국과학기술원 정교수. 1992년~1993년 개방형 컴퓨터 통신 연구회(OSIA) 분산 트랜잭션처리 분과위(TG-TP) 의장. 1993년~1994년 한국통신기술협회(TTA) 분산 트랜잭션처리 실무 위원회 의장. 관심 분야는 데이타베이스, e-비즈니스, 분산트랜잭션, 분산시스템, 워크플로우