Journal of The Institute of Electronics and Information Engineers Vol.54, NO.5, May 2017 https://doi.org/10.5573/ieie.2017.54.5.48
ISSN 2287-5026(Print) / ISSN 2288-159X(Online)

논문 2017-54-5-7

환경 특성에 맞는 성능 향상 기법을 사용하는 태스크 스케줄링 알고리즘

(A Task Scheduling Algorithm with Environment-specific Performance Enhancement Method)

송 인 성*, 윤 동 성*, 박 태 신*, 최 상 방**

(Inseong Song, Dongsung Yoon, Taeshin Park, and Sangbang Choi©)

요 약

클라우드 컴퓨팅의 IaaS 서비스는 유지비용 없이 원하는 만큼의 고성능 가상 머신을 사용할 수 있다는 장점 덕분에 대용량 병렬 프로그램을 실행하기 위한 고성능 컴퓨팅 환경으로 주목받고 있다. 이러한 고성능 컴퓨팅 환경에서 병렬 프로그램의 실행에 소요되는 시간은 태스크 스케줄링 알고리즘에 좌우된다. 클라우드 컴퓨팅 환경을 기반으로 하는 태스크 스케줄링 알고리즘에 관한 연구는 사용자 부담 비용을 최소화하는 알고리즘이 주류를 이루었으며, 병렬 프로그램의 실행을 최대한 빨리 끝내기 위한 알고리즘에 관한 연구는 거의 이루어지지 않았다. 본 논문에서는 사용자 부담 비용 등의 제약 없이 병렬 프로그램을 최대한 빨리 끝내기 위한 알고리즘인 HAGD 알고리즘과, HAGD 알고리즘이 사용하는 새로운 성능 향상 기법인 묶음 태스크 복제 기법을 제안한다. 묶음 태스크 복제 기법은 기존 태스크 복제 기법을 단순화하였으며, HAGD 알고리즘은 고성능 컴퓨팅 환경과 병렬 프로그램의 특성에 맞추어 태스크 삽입 기법 혹은 묶음 태스크 복제 기법을 사용한다. 성능 평가 결과, 제안하는 알고리즘이 환경 특성과 관계없이 우수한 표준화한 전체 실행 시간을 제공하는 것을 확인하였다.

Abstract

An IaaS service of a cloud computing environment makes itself attractive for running large scale parallel application thanks to its innate characteristics that a user can utilize a desired number of high performance virtual machines without maintenance cost. The total execution time of a parallel application on a high performance computing environment depends on a task scheduling algorithm. Most studies on task scheduling algorithms on cloud computing environment try to reduce a user cost, and studies on task scheduling algorithms that try to reduce total execution time are rarely carried out. In this paper, we propose a task scheduling algorithm called an HAGD and a performance enhancement method called a group task duplication method of which the HAGD utilizes. The group task duplication method simplifies previous task duplication method, and the HAGD uses the group task duplication method or a task insertion method according to the characteristics of a computing environment and an application. We found that the proposed algorithm provides superior performance regardless of the characteristics in terms of normalized total execution time through performance evaluations.

Keywords : DAG, 병렬 프로그램, IaaS, 고성능 컴퓨팅, 스케줄링

Ⅰ. 서 론

유전자 분석, 기후 변화 분석 등 대규모 데이터를 빠르게 처리하고자 하는 수요가 증가하면서 고성능 컴퓨팅 시스템이 등장했다. 초기의 고성능 컴퓨팅 시스템은 연구소 등이 직접 자신만의 고성능 컴퓨팅 시스템을 구축하여 사용하는 형태였다. 하지만 데이터의 규모가 기하급수적으로 늘어나면서 단일 기관이 보유한 고성능 컴퓨팅 시스템으로는 단시간 내에 데이터를 분석할 수 없는 수준에 이르자, 다양한 형태의 컴퓨팅 시스템이

* 정회원, ** 평생회원, 인하대학교 전자공학과(Department of Electronic Engineering, Inha University)
© Corresponding Author(E-mail : sangbang@inha.ac.kr)
※ 이 논문은 2010년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(2010-0020163)
Received ; December 23, 2016 Revised ; February 6, 2017 Accepted ; April 6, 2017

연구되었다.

최근에는 클라우드 컴퓨팅(cloud computing) 시스템의 다양한 서비스 중 인프라스트럭처 서비스(Infrastructure as a Service, IaaS)가 빠르게 발전하면서 기존 고성능 컴퓨팅 시스템을 대체하고 있다[1~2]. IaaS는 가상 머신 인스턴스(virtual machine instance) 형태로 컴퓨팅 자원을 사용자에게 대여한다. 그 예로, Amazon EC2의 c4.8xlarge 유형 가상 머신 인스턴스는 36개의 가상 프로세서, 60GB의 가상 메모리를 갖는 가상 컴퓨팅 자원을 제공한다. 사용자는 대여한 가상 머신 인스턴스를 이용하여 고성능 컴퓨팅 시스템을 구축하고 대용량 병렬 프로그램을 실행한다. IaaS를 이용하여 구축한 고성능 컴퓨팅 시스템의 예는 물리학 실험, 기계 학습 등의 분야에서 찾아볼 수 있다.

고성능 컴퓨팅 시스템에서 실행하는 대용량 병렬 프로그램은 방향성 비순환 그래프(Directed Acyclic Graph, DAG)로 모델링 한다[3]. 그리고 대용량 병렬 프로그램을 고성능 컴퓨팅 시스템에서 실행하기 위해, DAG에 포함된 태스크를 우선순위에 따라 정렬하고 그 우선순위에 따라 태스크를 컴퓨팅 자원에 할당하는 태스크 스케줄링 과정을 거친다. 하지만 태스크 스케줄링 문제는 NP-complete 문제이므로, 태스크 스케줄링 분야의 연구는 문제의 정확한 해를 찾는 대신 최적의 해를 찾기 위해 노력해왔다.

태스크 스케줄링 알고리즘에 관한 연구는 고성능 컴퓨팅 시스템이 등장한 이래로 이어져 왔다. 전통적으로 대용량 병렬 프로그램을 빠르게 실행하기 위한 알고리즘이 폭넓게 연구되었다[4~6]. 이후 고성능 컴퓨팅 시스템이 IaaS를 기반으로 구축되기 시작하면서, 클라우드 컴퓨팅 시스템의 특성을 고려하여 제한 비용 내에서 대용량 병렬 프로그램의 실행을 빠르게 완료하거나[7], 제한 시간 내에 저렴한 비용으로 대용량 병렬 프로그램의 실행을 완료하기[8] 위한 알고리즘이 연구되었다. 하지만 비용과 관련한 연구가 주를 이룬 나머지, IaaS를 기반으로 구축한 고성능 컴퓨팅 시스템에서 대용량 병렬 프로그램을 빠르게 실행하고자 하는 사용자는 기존 고성능 컴퓨팅 시스템을 대상으로 연구한 구형 태스크 스케줄링 알고리즘을 사용해야만 했다.

따라서 본 논문에서는 IaaS를 이용하여 구축한 고성능 컴퓨팅 시스템에서 사용자 비용 등의 고려 없이 대용량 병렬 프로그램을 빠르게 실행하기 위한 알고리즘인 HAGD (Hybrid Allocation with Group Duplication) 알고리즘을 제안한다. HAGD 알고리즘은 일반적인 태스크 스케줄링 알고리즘과 같이 태스크 우선순위 결정 단계와 자원 할당 단계로 구성된다. 우선순위 결정 단계는 부분 임계 경로 개념을 이용하여 DAG 내의 모든 태스크를 전체 실행 시간에 큰 영향을 끼치는 순서대로 정렬한다. 자원 할당 단계는 고성능 컴퓨팅 시스템과 DAG의 특성을 확인하고, 태스크 삽입 기법 혹은 본 논문에서 제안하는 묶음 태스크 복제 기법을 이용하여 태스크를 가장 빨리 끝낼 수 있는 가상 머신 인스턴스에 할당한다.

본 논문에서 제안하는 알고리즘의 공정한 성능 평가를 위하여 표준 태스크 그래프 모음이 제공하는 DAG와 실제 응용 프로그램을 모델링 한 그래프를 사용하였다. 그리고 다양한 매개변수를 적용하여 실행 환경을 바꿔가며 HAGD 알고리즘의 성능을 평가하였다. 성능 평가는 표준화한 DAG의 전체 실행 시간을 기준으로 하였다. DAG와 매개변수를 조합하여 총 452,785회의 시뮬레이션을 진행한 결과 HAGD 알고리즘이 환경과 관계없이 HCPFD 알고리즘에 비해 3.8%~16.2%, HEFT 알고리즘에 비해 3.1%~7.3%, PETS 알고리즘에 비해 5.0%~8.7% 우수한 성능을 제공하는 것을 확인할 수 있었다.

본 논문의 구성은 다음과 같다. 2장은 클라우드 컴퓨팅 시스템의 IaaS를 이용하여 구성한 고성능 컴퓨팅 시스템과 대용량 병렬 프로그램의 모델을 제시한다. 3장은 본 논문에서 서술하는 연구 내용과 관련있는 기존 연구에 대해 설명하며, 4장은 본 논문에서 제시하는 HAGD 알고리즘에 대해 설명한다. 5장은 HAGD 알고리즘의 실험 환경과 결과에 대해 설명하며, 마지막으로 6장은 본 논문을 마무리한다.

Ⅱ. 문제 정의

본 논문에서 제안하는 알고리즘은 클라우드 컴퓨팅 시스템의 IaaS를 이용하여 구성한 고성능 컴퓨팅 시스템을 기반으로 한다. IaaS는 컴퓨팅 자원을 가상 머신 인스턴스의 형태로 대여하며, 사용자는 컴퓨팅 자원을 사용한 만큼 비용을 지불한다. IaaS가 제공하는 컴퓨팅 서비스 집합은 $S = \{s_1, s_2, ..., s_n\}$ 이다. 각 컴퓨팅 서비스 유형 즉 가상 머신 인스턴스 유형 s_n은 서로 다른 처리 능력을 갖는다. 사용자가 구성한 고성능 컴퓨팅 시스템에 포함된 가상 머신 인스턴스 집합은 $S_{pi} = \{s_{1,1}, s_{1,2}, ... s_{j,k}\}$이다. $s_{j,k}$는 k 번째로 대여한 유형 s_j의 가상 머신을 의미한다.

컴퓨팅 자원을 호스팅하는 서버는 완전 연결 네트워

크로 연결되어[9] 가상 머신 인스턴스는 태스크의 실행이 끝남과 동시에 그 결과를 전송할 수 있으며, 동시에 다수의 데이터 전송 작업이 가능하다. 또한, 가상 머신 인스턴스는 비선점 방식으로 실행 도중 방해 받지 않으며, 태스크의 실행과 데이터의 전송을 동시에 처리할 수 있다. 마지막으로 가상 머신 인스턴스 간에 데이터 전송에 필요한 사용자 부담 비용은 없다[10].

대용량 병렬 프로그램을 모델링 한 방향성 비순환 그래프는 $G=\{V, E\}$로 표현할 수 있다. 여기에서 V는 프로그램을 구성하는 태스크의 집합이며, E는 선행 제약을 나타내는 에지의 집합이다.

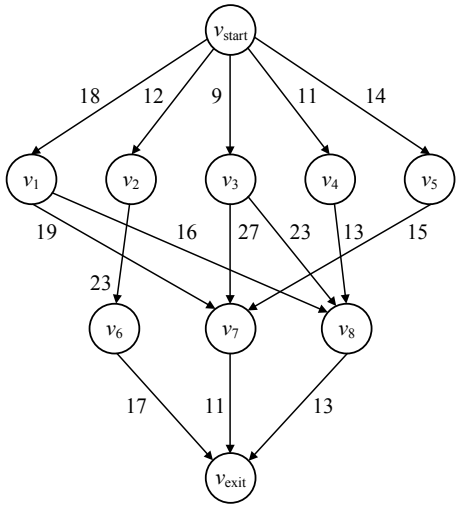


그림 1. 방향성 비순환 그래프의 예
Fig. 1. An example directed acyclic graph.

표 1. 계산비용 표의 예
Table1. An example computation cost table.

v_i	$s_{1,1}$	$s_{2,1}$	$s_{3,1}$
v_{start}	18	20	9
v_1	24	21	15
v_2	23	18	14
v_3	21	17	13
v_4	16	15	13
v_5	19	16	14
v_6	20	16	7
v_7	17	14	6
v_8	21	24	9
v_{exit}	14	19	10

집합 $V=\{v_1, v_2, ..., v_n\}$는 방향성 비순환 그래프의 모든 태스크를 포함하며, i번째 태스크는 v_i로 나타낸다. 집합 $E=\{e_{1,2}, e_{1,3}, ..., e_{i,j}\}$는 방향성 비순환 그래프의 모든 에지를 포함하며, 태스크 v_i를 태스크 v_j로 연결하는 에지는 $e_{i,j}$로 나타낸다. 에지는 방향성을 갖고 두 태스크를 연결하며, 이때 v_i와 v_j는 선행 제약 관계에 있다. 이 경우 v_j는 v_i의 실행이 끝난 후 v_i로부터 데이터를 전달받아야만 실행할 수 있으며, v_i는 v_j의 부모 태스크, v_j는 v_i의 자식 태스크라고 부른다. $cparent(v_i)$는 v_i의 임계 부모 태스크이다. 임계 부모 태스크는 v_i의 여러 부모 태스크 중 가장 늦은 시각에 v_i로 데이터를 전달하는 태스크이다. 부모 태스크가 존재하지 않는 태스크를 시작 태스크 v_{start}, 자식 태스크가 존재하지 않는 태스크를 출력 태스크 v_{exit}로 정의한다. 그림 1은 방향성 비순환 그래프의 예를 보여준다. 그림에서 원은 태스크를 화살표는 에지를 의미하며, 에지 옆의 숫자는 평균 통신비용을 의미한다.

고성능 컴퓨팅 시스템에서 방향성 비순환 그래프를 실행할 때의 계산비용은 $|S|\times|V|$ 크기의 표로 주어진다. 이때, $w(v_i, s_{j,k})$는 태스크 v_i를 가상 머신 인스턴스 $s_{j,k}$에서 실행할 때의 계산비용 즉, v_i의 실행 시간이다. 표 1은 그림 1의 방향성 비순환 그래프를 고성능 컴퓨팅 시스템에서 실행할 때의 계산비용으로, 각 태스크를 각 가상 머신에서 실행할 때의 계산비용을 보여준다. v_i의 평균 계산비용 $\overline{w}(v_i)$는 식 (1)과 같이 계산할 수 있다.

$$\overline{w}(v_i)=\frac{\sum_j w(v_i, s_j)\times j_k}{\sum_j j_k} \quad (1)$$

식 1에서 $w(v_i, s_j)$는 태스크 v_i를 가상 머신 인스턴스 유형 s_j에서 실행할 때의 계산비용을, j_k는 s_j의 개수가 k개임을 의미한다.

$c(v_i, v_j)$는 네트워크를 통해 태스크 v_i로부터 태스크 v_j로 데이터를 전달하기 위해 소요되는 평균 시간 즉, 평균 통신비용이다. 두 태스크가 같은 자원에 할당된 경우 같은 자원 내부에서 데이터를 전달하는 것이므로 평균 통신비용을 0으로 간주한다.

가상 머신 인스턴스가 태스크를 실행 중인 경우 다음 태스크는 현재 실행 중인 태스크의 종료를 기다려야 한다. $IAT(s_{j,k})$는 가상 머신 인스턴스 $s_{j,k}$가 실행 중이던 태스크의 실행을 끝내고 다음 태스크를 실행할 준비가 완료되는 시각 즉, 인스턴스 사용 가능 시각이다.

$EST(v_i, s_{j,k})$는 태스크 v_i가 가상 머신 인스턴스 $s_{j,k}$에

서 실행을 시작할 수 있는 가장 빠른 시각 즉, 실행 시작 가능 시각이다. $EFT(v_i, s_{j,k})$는 태스크 v_i가 가상 머신 인스턴스 $s_{j,k}$에서 실행을 완료할 수 있는 가장 빠른 시각 즉, 실행 완료 시각이다. 실행 시작 가능 시각과 실행 완료 시각은 식 (2), 식 (3)과 같이 계산할 수 있다. 대용량 병렬 응용 프로그램의 전체 실행 시간 *Makespan*은 방향성 비순환 그래프의 전체 실행 시간과 같으며 식 (4)와 같이 계산할 수 있다.

$$EST(v_i, s_{j,k}) = \begin{cases} 0 & if\ v_i = v_{start} \\ \max\limits_{v_l \in parent(v_i)} \{IAT(s_{j,k}), \max\{EFT(v_l) + c(v_l, v_i)\}\} & otherwise \end{cases} \quad (2)$$

$$EFT(v_i, s_{j,k}) = EST(v_i, s_{j,k}) + w(v_i, s_{j,k}) \quad (3)$$

$$Makespan = EFT(v_{exit}, s_{j,k}) \quad (4)$$

Ⅲ. 관련 연구

1. 성능 향상 기법

방향성 비순환 그래프의 태스크 간에는 선행 제약이 존재하며, 이는 대용량 병렬 프로그램의 실행을 지연시킨다. 태스크 스케줄링 알고리즘은 실행 지연을 최소화하기 위해 태스크 삽입 기법, 태스크 복제 기법[11] 등의 성능 향상 기법을 사용한다.

가. 태스크 삽입 기법

태스크 삽입 기법은 컴퓨팅 자원에 이미 할당된 태스크 사이에 새로운 태스크를 할당하는 것이다. 태스크 삽입 기법을 이용하면 태스크를 이미 할당된 태스크 사이의 컴퓨팅 자원의 유휴 시간에 할당하여 실행할 수 있으므로 실행 지연을 최소화할 수 있다. 하지만 태스크를 삽입하려면 컴퓨팅 자원의 유휴 시간을 탐색해야 하므로 알고리즘의 복잡도가 증가한다는 단점이 있다.

나. 태스크 복제 기법

태스크 복제 기법은 컴퓨팅 자원에 이미 할당된 태스크 사이에 기존에 할당된 태스크 중 하나를 복제하여 할당하는 것이다. 태스크 복제 기법을 이용하면 지연을 유발하는 부모 태스크를 복제하여 자식 태스크와 같은 컴퓨팅 자원에 할당할 수 있으므로 실행 지연을 최소화할 수 있다. 하지만 태스크를 복제하려면 컴퓨팅 자원의 유휴 시간을 탐색해야 하므로 알고리즘의 복잡도가 증가한다는 단점이 있다.

2. 기존 태스크 스케줄링 알고리즘

기존 태스크 스케줄링 알고리즘은 대용량 병렬 프로그램의 실행을 빠르게 완료하기 위한 알고리즘과 매개변수를 최적화하기 위한 알고리즘으로 구분할 수 있다. 대용량 병렬 프로그램의 실행을 빠르게 완료하기 위한 알고리즘은 전통적인 태스크 스케줄링 알고리즘으로 가능한 모든 자원을 활용하여 가능한 빠른 시간 내에 대용량 병렬 프로그램의 실행을 완료하는 것을 목표로 한다[4]. 매개변수를 최적화하기 위한 알고리즘은 최근 활발히 연구되는 분야로 사용자 부담 비용, 최종 한계 실행 시간 등 매개변수를 제한 범위 내에서 최적화하기 위한 알고리즘이다[12].

가. HEFT

HEFT (Heterogeneous Earliest Finish Time)[4] 알고리즘은 대용량 병렬 프로그램의 실행을 빠르게 완료하기 위한 삽입 기반 스케줄링 알고리즘이다.

HEFT 알고리즘은 태스크 우선순위 결정 단계에서 $rank_u$를 사용하여 입력 그래프의 출력 태스크부터 시작 태스크까지, 모든 태스크에 우선순위를 부여한다. 각 태스크의 $rank_u$는 각 태스크로부터 출력 태스크까지의 부분 임계 경로가 갖는 비용을 의미한다. 마지막으로 $rank_u$가 큰 태스크부터 내림차순 정렬하여 우선순위 리스트에 삽입한다.

다음으로 HEFT 알고리즘은 자원 할당 단계에서 삽입 기법을 이용한다. 우선순위 리스트에 저장된 태스크 중 첫 번째 태스크부터 각 태스크의 실행을 가장 빨리 끝낼 수 있는 컴퓨팅 자원에 태스크를 할당하며, 이 때 태스크의 삽입을 고려한다.

나. PETS

PETS (Performance Effective Task Scheduling)[5] 알고리즘은 대용량 병렬 프로그램의 실행을 빠르게 완료하기 위한 그래프 계층화 및 삽입 기반 태스크 스케줄링 알고리즘이다.

PETS 알고리즘은 그래프 계층화 단계에서 입력 그래프를 계층별로 분류하고 같은 계층에 있는 태스크를 하나의 그룹으로 묶는다.

다음으로 PETS 알고리즘은 태스크 우선순위 결정 단계에서 *rank*를 사용하여 입력 그래프의 시작 태스크부터 출력 태스크까지, 모든 태스크에 우선순위를 부여한다. 각 태스크의 *rank*는 가장 큰 *rank*를 갖는 부모 태스크의 *rank*에 현재 태스크의 평균 계산비용과 현재

태스크에서 자식 태스크로 향하는 모든 에지의 통신비용을 합하여 계산한다. 마지막으로 낮은 계층부터 *rank*를 기준으로 각 계층 내의 태스크를 내림차순 정렬하여 우선순위 리스트에 삽입한다.

PETS 알고리즘은 자원 할당 단계에서 삽입 기법을 이용한다. 우선순위 리스트에 저장된 태스크 중 첫 번째 태스크부터 각 태스크의 실행을 가장 빨리 끝낼 수 있는 컴퓨팅 자원에 태스크를 할당하며, 이 때 태스크의 삽입을 고려한다.

다. HCPFD

HCPFD (Heterogeneous Critical Parents with Fast Duplicator)[6] 알고리즘은 대용량 병렬 프로그램의 실행을 빠르게 완료하기 위한 복제 기반 스케줄링 알고리즘이다.

HCPFD 알고리즘은 태스크 우선순위 결정 단계에서 *AEST*와 *ALST*를 사용하여 임계 경로 태스크 리스트를 생성한다. *AEST*와 *ALST*는 태스크의 평균 계산비용과 에지의 평균 통신비용을 이용하여 계산한 태스크의 가장 빠른 실행 시작 가능 시각, 가장 늦은 실행 시작 가능 시각이다. 마지막으로 임계 경로 태스크 리스트에 있는 첫 번째 태스크부터 해당 태스크의 모든 부모 태스크를 우선순위 리스트에 삽입한다.

다음으로 HCPFD 알고리즘은 자원 할당 단계에서 복제 기법을 이용한다. 우선순위 리스트에 저장된 태스크 중 첫 번째 태스크부터 각 태스크의 실행을 가장 빨리 끝낼 수 있는 컴퓨팅 자원에 태스크를 할당하며, 이 때 임계 부모 태스크의 복제를 고려한다.

라. BHEFT

BHEFT (Budget-constrained Heterogeneous Earliest Finish Time)[7] 알고리즘은 단일 매개변수를 최적화하는 태스크 스케줄링 알고리즘으로 최대 사용자 부담 비용 내에서 전체 실행 시간을 최소화하는 알고리즘이다.

BHEFT 알고리즘은 태스크 우선순위 결정 단계에서 HEFT 알고리즘을 따라 출력 태스크부터 시작 태스크까지, $rank_u$를 사용하여 각 태스크에 우선순위를 부여한다. 마지막으로 $rank_u$가 큰 태스크부터 내림차순 정렬하여 우선순위 리스트에 삽입한다.

다음으로 BHEFT 알고리즘은 자원 할당 단계에서 최대 사용자 부담 비용을 넘지 않으면서 전체 실행 시간을 줄일 수 있는 최적의 컴퓨팅 자원에 태스크를 할당한다. BHEFT 알고리즘은 최적의 컴퓨팅 자원을 찾기 위해 *Spare Application Budget* 변수와 *Current Task Budget* 변수를 사용하며, 두 변수가 갖는 값의 차이에 따라 태스크의 실행을 가장 빨리 끝낼 수 있는 컴퓨팅 자원 혹은 태스크의 실행을 가장 저렴하게 끝낼 수 있는 컴퓨팅 자원에 태스크를 할당한다.

마. ICPCP

ICPCP (Iaas Cloud Partial Critical Path)[8] 알고리즘은 단일 매개변수를 최적화하는 태스크 스케줄링 알고리즘으로 최종 한계 실행 시각 내에서 사용자 부담 비용을 최소화하는 알고리즘이다.

ICPCP 알고리즘은 부분 임계 경로 구성 단계에서 입력 그래프의 출력 태스크부터 시작하여 다른 부분 임계 경로에 포함되지 않은 부모 태스크 중 임계 부모 태스크를 현재 구성 중인 부분 임계 경로에 삽입한다. 이 과정은 더는 삽입할 부모 태스크가 없을 때까지 반복되며 이 과정을 통해 하나의 부분 임계 경로를 완성한다.

자원 할당 단계에서 ICPCP 알고리즘은 최종 한계 실행 시각을 넘지 않으면서 사용자 부담 비용을 최소화할 수 있는 최적의 컴퓨팅 자원에 태스크를 할당한다. 우선 가장 저렴한 컴퓨팅 자원에 부분 임계 경로를 할당하여 해당 부분 임계 경로의 지연 실행 허용 시각 내에 태스크의 실행을 완료할 수 있는지 확인한다. 지연 실행 허용 시각 내에 태스크의 실행을 완료할 수 있다면 해당 컴퓨팅 자원에 해당 부분 임계 경로를 할당하며, 완료가 불가능하다면 다음으로 저렴한 컴퓨팅 자원에 부분 임계 경로를 할당해보는 과정을 반복한다.

부분 임계 경로의 할당이 끝나면 할당한 부분 임계 경로에 포함된 태스크의 실행 시각과 관련된 정보를 갱신하며, 다시 부분 임계 경로 구성 단계로 진입하여 새로운 부분 임계 경로를 구성한다.

Ⅳ. 제안하는 알고리즘

본 논문에서는 IaaS를 이용하여 구축한 고성능 컴퓨팅 시스템에서 대용량 병렬 프로그램의 실행을 가능한 빨리 실행하기 위한 새로운 태스크 스케줄링 알고리즘인 HAGD 알고리즘과 새로운 태스크 복제 기법인 묶음 태스크 복제 기법을 제안한다. 기존 태스크 스케줄링 알고리즘은 컴퓨팅 환경과 DAG의 특성을 고려하지 않고 태스크를 스케줄하지만, 제안하는 알고리즘은 매 태스크마다 컴퓨팅 환경과 DAG의 특성을 고려하여 태스크를 스케줄한다. 또한, 기존 알고리즘은 태스크의 삽입

혹은 태스크의 복제 중 한 가지 방법만을 선택하여 전체 태스크에 적용하지만, 제안하는 알고리즘은 태스크의 삽입과 태스크의 복제를 모두 사용하며, 각 태스크의 특성을 고려하여 성능 향상 기법을 선택한다. 여기에서 제안하는 알고리즘은 기존 태스크 복제 기법 대신 제안하는 묶음 태스크 복제 기법을 사용한다.

제안하는 HAGD 알고리즘은 태스크 우선순위 결정 단계와 자원 할당 단계로 구성된다.

1. 태스크 우선순위 결정 단계

태스크 우선순위 결정 단계에서 HAGD 알고리즘은 DAG 내의 모든 태스크에 우선순위를 부여하고, 부여한 우선순위에 따라 태스크를 정렬하여 태스크 우선순위 리스트를 생성한다. 우선순위 결정은 $BL(v_i)$를 이용하며 식 (5)와 같이 계산할 수 있다.

$$BL(v_i)=\begin{cases}\bar{w}(v_{exit}) & if\ v_i=v_{exit}\\ \bar{w}(v_i)+\max\limits_{v_l\in child(v_i)}\{c(v_i,v_l)+BL(v_l)\} & otherwise\end{cases} \tag{5}$$

$BL(v_i)$는 태스크 v_i부터 종료 태스크 v_{exit}까지의 부분 임계 경로이다. 즉, $BL(v_i)$가 크다는 것은 v_i 이후로 실행해야 할 태스크가 많다는 뜻, 혹은 v_i가 v_i 이후에 실행되는 태스크의 실행에 큰 영향을 끼친다는 뜻이다. 따라서 큰 $BL(v_i)$를 갖는 태스크가 높은 우선순위를 갖는다. $BL(v_i)$는 종료 태스크부터 시작하여 시작 태스크까지, DAG를 거꾸로 올라가며 계산한다.

HAGD 알고리즘은 DAG 내 모든 태스크의 $BL(v_i)$를 계산하고 난 뒤, $BL(v_i)$ 값의 내림차순으로 태스크를 정렬한다. 태스크를 정렬할 때 같은 우선순위 값을 갖는 태스크가 존재한다면, 평균 계산비용이 큰 태스크에 우선순위를 부여한다. 태스크 정렬이 완료되면, HAGD 알고리즘은 우선순위 리스트 L을 생성하고 정렬된 태스크를 순서대로 우선순위 리스트에 넣는다.

2. 자원 할당 단계

HAGD 알고리즘은 자원 할당 단계에서 컴퓨팅 환경과 프로그램의 특성을 확인하고 그 결과에 따라 태스크 삽입 기법 혹은 묶음 태스크 복제 기법을 이용하여 태스크를 가상 머신 인스턴스에 할당한다.

가. 특성 확인 기준

컴퓨팅 환경과 DAG의 특성을 확인하기 위한 식은 DAG의 통신비용 대 계산비용의 비율(Communication to Computation Ratio, *CCR*)과 한 태스크가 여러 가상 머신 인스턴스에서 가질 수 있는 계산비용의 범위와의 관계로부터 유도할 수 있다. *CCR*은 식 (6)과 같이 계산하며, 한 태스크가 여러 가상 머신 인스턴스에서 가질 수 있는 계산비용의 범위는 식 (7)과 같이 나타낸다.

$$CCR=\frac{\sum_{e\in E}c(v_j,v_k)}{\sum_{v\in V}\bar{w}(v_i)} \tag{6}$$

$$\bar{w}(v_i)\times\left(1-\frac{hf(v_i)}{2}\right)\le w(v_i,s_{j,k})\le\bar{w}(v_i)\times\left(1+\frac{hf(v_i)}{2}\right) \tag{7}$$

위의 식에서 $hf(v_i)$는 태스크 v_i의 이질성도 즉, 한 태스크가 여러 가상 머신 인스턴스에서 갖는 계산비용의 편차를 의미한다. 위의 두 식에서 한 태스크가 가질 수 있는 최대 계산비용과 통신비용 대 계산비용의 비율을 비교하여 아래 식 (8)을 얻을 수 있다.

$$CCR>1+\frac{hf(v_i)}{2} \tag{8}$$

HAGD 알고리즘은 식 (8)을 이용하여 컴퓨팅 환경과 프로그램의 특성을 확인하고, 특성에 따라 태스크를 가상 머신 인스턴스에 할당한다. 식의 좌항은 *CCR*을 이용하여 계산한 태스크 v_i와 태스크 v_j의 간의 예상 평균 통신비용으로부터 만들어진 것이며, 식의 우항은 v_i가 가질 수 있는 최대 계산비용으로부터 만들어진 것이다. 따라서 왼쪽 항이 더 크다는 것은 즉, 부등식이 참이라는 것은 v_i와 v_j의 예상 평균 통신비용이 v_i가 가질 수 있는 최대 계산비용보다 크다는 것이므로, 태스크 복제 기법을 사용하는 것이 성능 향상에 유리하다. 반대로 오른쪽 항이 더 크다는 것은 즉, 부등식이 거짓이라는 것은 v_i와 v_j의 예상 평균 통신비용이 v_i가 가질 수 있는 최대 계산비용보다 작다는 것이므로, 태스크 삽입 기법을 사용하는 것이 성능 향상에 유리하다.

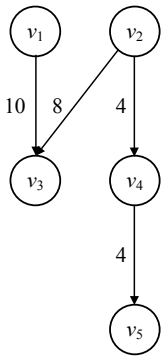
나. 묶음 태스크 복제 기법

묶음 태스크 복제 기법은 기존 태스크 복제 기법의 복잡한 과정을 단순화한 태스크 복제 기법이다. 묶음 태스크 복제 기법은 기존 태스크 복제 기법과 달리 여러 가지 경우의 수를 고려할 필요 없이 태스크 v_i와 v_i의 임계 부모 태스크 v_{cp}를 묶은 태스크 묶음 v_{cp+i}의 실행 완료 시각만을 고려하면 된다.

Algorithm 1. Group task duplication method

1: Calculate $EFT(v_i, s_{j,k})$ and set to $EFT1$
2: Create task group v_{cp+i}
3: Calculate $EFT(v_{cp+i}, s_{m,n})$ and set to $EFT2$
4: **if** $EFT(cparent(v_i), s_{m,n}) < EST(v_i, s_{m,n})$ **do**
5: Adjust $EFT2$
6: **endif**
7: **if** $EFT1 < EFT2$ **do**
8: Allocate v_i on $s_{j,k}$
9: **else**
10: Allocate v_{cp+i} on $s_{m,n}$
11: **endif**

그림 2. 묶음 태스크 복제 기법의 의사코드
Fig. 2. A pseudocode for the group task duplication method.



v_i	$s_{1,1}$	$s_{2,1}$
v_1	4	2
v_2	2	4
v_3	3	7
v_4	4	2
v_5	2	4

그림 3. 태스크 할당 과정을 설명하기 위한 예제 그래프와 표
Fig. 3. A graph and a computation cost table to show a task allocation process.

그림 2는 묶음 태스크 복제 기법의 의사 코드를 보여준다. 묶음 태스크 복제 기법은 현재 태스크 v_i의 실행을 가장 빨리 완료할 수 있는 가상 머신 인스턴스 $s_{j,k}$를 찾고 v_i의 실행 완료 시각을 $EFT1$로 저장한다. 다음으로, v_i의 임계 부모 태스크 v_{cp}를 찾아 태스크 묶음 v_{cp+i}를 생성한다. 그리고 v_{cp+i}의 실행을 가장 빨리 완료할 수 있는 가상 머신 인스턴스 $s_{m,n}$을 찾고 v_{cp+i}의 실행 완료 시각을 $EFT2$로 저장한다. 이 때 선행 제약을 만족하기 위해 $s_{m,n}$에 v_{cp}를 할당했다고 가정했을 때 v_{cp}의 실행 완료 시각과 v_i의 $s_{m,n}$에서의 실행 시작 가능 시각을 비교한다. 만약 v_i의 실행 시작 가능 시각이 v_{cp}의 실행 완료 시각보다 크다면 v_i의 실행 완료 시각과 $EFT2$를 v_i의 실행 시작 가능 시각을 기준으로 조정한다. 마지막으로 $EFT1$과 $EFT2$를 비교했을 때 $EFT1$이 $EFT2$보다 작다면 v_i를 $s_{j,k}$에 할당한다. 하지만 $EFT1$이 $EFT2$보다 크다면 즉, v_{cp}를 복제하는 것이 v_i의 실행 완료 시각을 줄인다면 v_{cp+i}를 $s_{m,n}$에 할당한다.

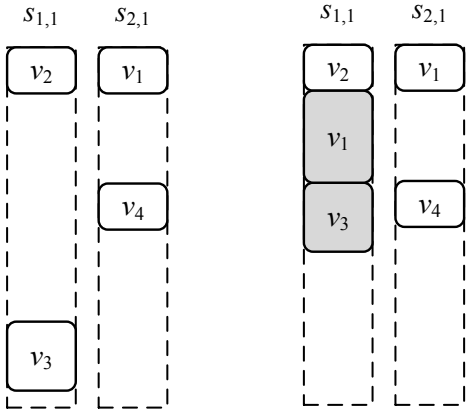


그림 4. 묶음 태스크 복제 기법을 이용한 태스크 할당 예
Fig. 4. An example of task allocation with the group task duplication method.

그림 4는 그림 3의 예제 그래프와 계산비용 표를 이용하여 묶음 태스크 복제 기법을 사용하지 않았을 때와 묶음 태스크 복제 기법을 사용했을 때의 태스크 할당 예를 보여준다. 우선, 태스크 v_3의 가장 빠른 실행 완료 시각을 계산한 뒤 이 시각을 $EFT1$로 저장한다. 구체적으로 v_3의 부모 태스크인 태스크 v_1과 태스크 v_2의 실행 완료 시각 $EFT(v_1, s_{2,1}) = 2$와 $EFT(v_2, s_{1,1}) = 2$에 통신 비용 $c(v_1, v_3) = 10$과 $c(v_2, v_3) = 8$을 더하고, 계산비용 $w(v_3, s_{1,1}) = 3$, $w(v_3, s_{2,1}) = 7$까지 더하면 $EFT(v_3, s_{1,1}) = 15$, $EFT(v_3, s_{2,1}) = 17$이다. 따라서 둘 중 작은 값인 15를 $EFT1$로 저장한다. 그리고 v_3의 임계 부모 태스크 v_1과 v_3을 묶어 태스크 묶음 v_{1+3}을 생성한다. 다음으로 가상 머신 인스턴스 $s_{1,1}$에서 v_{1+3}의 가장 빠른 실행 완료 시각을 계산하면 $IAT(s_{1,1}) = 2$에 $w(v_{1+3}, s_{1,1}) = 7$을 더해 $EFT(v_{1+3}, s_{1,1}) = 9$이다. 이때 v_3의 부모 태스크인 v_1과 v_2가 모두 같은 가상 머신 인스턴스에 존재하므로 선행 제약은 고려하지 않아도 된다. 가상 머신 인스턴스 $s_{2,1}$에서 v_{1+3}의 가장 빠른 실행 완료 시각을 계산하면 $IAT(s_{2,1}) = 2$에 $w(v_{1+3}, s_{2,1}) = 9$를 더해 $EFT(v_{1+3}, s_{2,1}) = 11$이다. 하지만 이때 부모 태스크 v_2와의 선행 제약 위반이 발생하므로 v_3의 할당 시간을 조정해야 한다. 통신비용을 고려하여 조정한 $EFT(v_{1+3}, s_{2,1}) = 19$이므로, 두 실행 완료 시각 중 작은 값인 9를 $EFT2$로 저장한다. 결과적으로 $EFT2$가 작으므로 태스크 묶음 v_{1+3}을 $s_{1,1}$에 할당하며, 묶음 태스크 복제 기법을 사용하여 15의 실행 완료 시각을 9로, 6만큼 줄일 수 있다.

다. 자원 할당

HAGD 알고리즘은 자원 할당 단계에서 *CCR*과 이질성도를 계산한 후, 우선순위 리스트 L에 포함된 태스크를 차례대로 가상 머신 인스턴스에 할당한다.

우선, *CCR*을 계산하기 위해 DAG에 포함된 모든 에지의 평균 통신비용을 더하고, 다음으로 모든 태스크의 평균 계산비용을 더한다. 그리고 식 (6)과 같이 평균 통신비용의 합을 평균 계산비용의 합으로 나눠 *CCR*을 계산한다.

다음으로, 이질성도를 계산하기 위해 DAG에 포함된 모든 태스크에 식 (7)을 적용하고 식을 만족하는 가장 작은 $hf(v_i)$를 태스크 v_i의 이질성도로 정한다.

Algorithm 2. HAGD algorithm

```
1:  // Task prioritizing phase
2:  Calculate w̄(v_i) for all tasks
3:  Calculate BL(v_i) for all tasks
4:  Sort the tasks in decreasing order of BL(v_i)
5:  Create L and insert sorted tasks
6:  // VM instance allocation phase
7:  Calculate CCR of G
8:  Calculate hf(v_i) for all tasks
9:  Allocate v_start on s_j,k that provides minimum
    EFT(v_start, s_j,k)
10: Remove v_start from L
11: for the first task v_i in L do
12:     Find cparent(v_i)
13:     if CCR > 1 + hf(cparent(v_i)) / 2 then
14:         Allocate v_i with group task
            duplication method on Algorithm 1
15:     else
16:         Allocate v_i with
            task insertion method
17:     endif
18:     Remove v_i from L
19: endfor
```

그림 5. HAGD 알고리즘의 의사코드
Fig. 5. A pseudocode for the HAGD algorithm.

위의 사전 계산 과정을 완료한 뒤 HAGD 알고리즘은 태스크를 가상 머신 인스턴스에 실제 할당하는 과정에 진입한다. 먼저 HAGD 알고리즘은 우선순위 리스트 L의 첫 번째 태스크 즉, DAG의 시작 태스크 v_{start}를 가장 빨리 끝낼 수 있는 가상 머신 인스턴스에 할당하고 L에서 v_{start}를 제거한다.

다음으로, HAA 알고리즘은 우선순위 리스트 L에 포함된 첫 번째 태스크 v_i를 가상 머신 인스턴스에 할당한다. 이때 특성 확인 기준을 임계 부모 태스크 *cparent*(v_i)에 적용하여 v_i를 가상 머신 인스턴스에 할당하기 위한 할당 방법을 결정한다.

특성 확인 기준의 부등식이 참이면 HAGD 알고리즘은 임계 부모 태스크 *cparent*(v_i)와 태스크 v_i 간의 예상 평균 통신비용이 *cparent*(v_i)가 가질 수 있는 최대 계산비용보다 크다고 간주하며 가상 머신 인스턴스에 *cparent*(v_i)를 복제할 수 있는 유휴 시간이 있다고 판단한다. 그리고 묶음 태스크 복제 기법을 이용하여 태스크를 가상 머신 인스턴스에 할당한다. 묶음 태스크 복제 기법은 *cparent*(v_i)를 복제했을 때와 복제하지 않았을 때 v_i의 실행 완료 시각을 비교하여 *cparent*(v_i), v_i의 묶음이나 v_i를 가상 머신 인스턴스에 할당한다. 마지막으로 우선순위 리스트 L에서 v_i를 제거한다.

반대로 특성 확인 기준의 부등식이 거짓이면 HAGD 알고리즘은 임계 부모 태스크 *cparent*(v_i)와 태스크 v_i 간의 예상 평균 통신비용이 *cparent*(v_i)가 가질 수 있는 최대 계산비용보다 작다고 간주하며 *cparent*(v_i)를 복제할 가능성이 희박하다고 판단한다. 그리고 태스크 삽입 기법을 이용하여 태스크를 가상 머신 인스턴스에 할당한다. 태스크 삽입 기법은 v_i를 성능 향상 기법 없이 할당했을 때와, v_i를 삽입하여 할당했을 때 v_i의 실행 완료 시각을 비교하여 v_i를 가상 머신 인스턴스에 할당한다. 마지막으로 L에서 v_i를 제거한다.

태스크를 가상 머신 인스턴스에 할당하는 과정은, 우선순위 리스트 L에 포함된 모든 태스크를 가상 머신 인스턴스에 할당할 때까지, L에 포함된 태스크 순서대로 반복된다.

그림 5는 HAGD 알고리즘의 의사 코드를 보여준다.

라. HAGD 알고리즘의 동작 예

HAGD 알고리즘의 동작 예를 설명하기 위해 그림 3의 예제 그래프를 그림 3의 예제 표와 같은 가상 머신 인스턴스를 갖는 환경에서 실행한다고 가정한다. 또한, 예제 그래프는 계산비용과 통신비용이 0인 가상의 시작 태스크와 종료 태스크에 연결되어 있다고 가정한다.

HAGD 알고리즘은 먼저 DAG에 포함된 태스크의 평균 계산비용을 계산한다. 예로 태스크 v_2의 평균 계산비용 $w(v_2) = (2 + 4) / 2 = 3$이다. 다음으로 DAG에 포함된 태스크의 $BL(v_i)$를 계산한다. 예로 v_2의 $BL(v_2) = 3 + 4 + 3 + 4 + 3 = 17$이다. 그리고 $BL(v_i)$의 내림차순으로 정렬한 뒤 순서대로 우선순위 리스트 L에 넣는다.

표 2. HAGD 알고리즘을 이용하여 태스크의 우선순위를 결정하는 과정
Table2. A task prioritizing process with the HAGD algorithm.

v_i	$\overline{w}(v_i)$	$BL(v_i)$	Priority
v_1	3	18	1
v_2	3	17	2
v_3	5	5	4
v_4	3	10	3
v_5	3	3	5

표 2는 HAGD 알고리즘의 태스크 우선순위 결정 단계를 따라 계산한 태스크의 평균 계산비용, $BL(v_i)$, 우선순위를 보여준다. 태스크 우선순위 결정 단계의 결과로 $L = \{v_1, v_2, v_4, v_3, v_5\}$이 된다.

HAGD 알고리즘은 태스크 우선순위 리스트 L을 만든 뒤 DAG의 CCR을 계산한다. 예제 그래프는 $CCR = (10 + 8 + 4 + 4) / (3 + 3 + 5 + 3 + 3) = 1.529$를 갖는다. 다음으로 DAG에 포함된 태스크의 $hf(v_i)$를 계산한다. 예로 태스크 v_2의 $hf(v_2)$는 $w(v_2, s_{1,1}) = 2$, $w(v_2, s_{2,1}) = 4$, $w(v_2) = 3$을 이용해서 계산하면 $hf(v_2) = 0.667$이다.

태스크를 가상 머신 인스턴스에 할당하는 순서는 시작 태스크 v_{start}부터이지만 예제 그래프는 가상의 v_{start}를 가지므로 L의 첫 번째 태스크인 태스크 v_1부터 가상 머신 인스턴스에 할당한다. v_1을 할당하기 위해 $cparent(v_1)$인 v_{start}에 특성 확인 기준을 적용한다. 이 때 v_{start}의 계산비용은 모든 가상 머신 인스턴스에서 0이므로 $hf(v_{start}) = 0$이다. 따라서 $1.529 > 1$로 특성 확인 기준이 참이며, HAGD 알고리즘은 묶음 태스크 복제 기법을 이용하여 v_1을 가상 머신 인스턴스에 할당한다. $EFT(v_1, s_{1,1}) = 4$, $EFT(v_1, s_{2,1}) = 2$이며, v_{start}의 계산비용이 0이므로 $EFT(v_{start+1}, s_{1,1}) = 4$, $EFT(v_{start+1}, s_{2,1}) = 2$로 같다. 따라서 HAGD 알고리즘은 v_1을 가상 머신 인스턴스 $s_{2,1}$에 할당하고 v_1을 L에서 삭제한다.

다른 예로 태스크 v_3을 할당하기 위해 $cparent(v_3)$인 v_1에 특성 확인 기준을 적용한다. $hf(v_1)$은 $w(v_1, s_{1,1}) = 4$, $w(v_1, s_{2,1}) = 2$, $\overline{w}(v_1) = 3$을 이용해서 계산하면 $hf(v_1) = 0.667$이다. 따라서 $1.529 > 1.333$으로 특성 확인 기준이 참이며, HAGD 알고리즘은 묶음 태스크 복제 기법을 이용하여 v_3을 가상 머신 인스턴스에 할당한다. v_3을 묶음 태스크 복제 기법을 사용하여 가상 머신 인스턴스에 할당하는 과정은 4.2장 나절에 있는 예와 같다.

표 3. HAGD 알고리즘을 이용하여 태스크를 자원에 할당하는 과정
Table3. A resource allocation process with the HAGD algorithm.

v_i	$hf(v_i)$	$cparent(v_i)$	Criterion	Method
v_1	0.667	v_{start}	True	Gr.Dup.
v_2	0.667	v_{start}	True	Gr.Dup.
v_3	0.8	v_1	True	Gr.Dup.
v_4	0.667	v_2	True	Gr.Dup.
v_5	0.667	v_4	True	Gr.Dup.

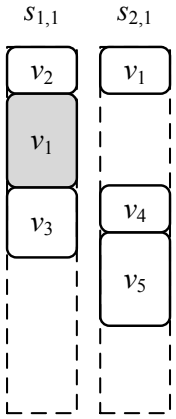


그림 6. HAGD 알고리즘을 이용하여 그림 3의 그래프를 스케줄한 결과
Fig. 6. A scheduling result of a graph in the Fig. 5 with the HAGD algorithm.

HAGD 알고리즘은 위와 같은 과정을 우선순위 리스트 L에 있는 모든 태스크가 가상 머신 인스턴스에 할당될 때까지 반복한다. 표 3은 HAGD 알고리즘의 자원 할당 단계를 따라 계산한 $hf(v_i)$, $hf(v_i)$를 적용하기 위한 $cparent(v_i)$, 특성 확인 기준을 이용한 부등식 비교 결과, 선택한 태스크 할당 방법을 보여주며, 그림 6은 최종 할당 결과를 보여준다. 그림에서 음영으로 처리한 태스크 v_1은 v_1이 복제되었다는 것을 의미한다.

V. 실험 및 성능 평가

본 논문에서 제안하는 HAGD 알고리즘의 성능 평가는 Microsoft Visual Studio 2015를 이용하여 직접 제작한 시뮬레이터를 통해 진행하였다.

표 4. 실험에 사용한 매개변수
Table4. Parameters used in experiments.

매개변수	값				
$\#v$	50,	100,	300,	500,	750
CCR	0.1,	0.5,	1.0,	2.5,	5.0
ß	0.1,	0.5,	1.0,	1.5,	1.9
$\#vm$	2,	4,	8,	16,	32

본 논문에서는 공정한 성능 평가를 위하여 표준 태스크 그래프 모음에서 제공하는 DAG[13]와 실제 응용 프로그램을 모델링한 DAG[14~17]를 사용하였다. 표준 태스크 그래프 모음은 다양한 방식으로 임의 생성한 DAG를 포함하고 있으며, 실제 응용프로그램을 모델링한 DAG로는 Robot control[14], Sparse matrix solver[15], Gaussian elimination[16], Fast Fourier Transform[17] 등을 사용하였다.

또한, 본 논문에서는 다양한 태스크 스케줄링 환경을 가정하고 성능을 평가할 수 있도록 하나의 DAG에 여러 매개변수를 적용하여 실험을 진행하였다. 실험에 사용한 매개변수는 표 4와 같다.

DAG에 포함된 태스크의 개수($\#v$)는 하나의 DAG가 갖는 태스크의 총 개수를 의미한다.

통신비용 대 계산비용의 비율(CCR)은 DAG에 포함된 모든 에지의 평균 통신비용의 합과 모든 태스크의 평균 계산비용의 합의 비율을 의미한다. 0에 가까운 CCR은 DAG가 계산 집약적이라는 것을, 5에 가까운 CCR은 DAG가 통신 집약적이라는 것을 의미한다.

가상 머신 인스턴스의 이질성도(ß)는 한 태스크가 서로 다른 성능을 제공하는 가상 머신 인스턴스에서 갖는 계산비용 편차의 최댓값을 의미한다. 0에 가까운 가상 머신 인스턴스의 이질성도는 컴퓨팅 환경이 같거나 비슷한 유형의 가상 머신 인스턴스로 이루어졌다는 것을 의미한다. 반대로 2.0에 가까운 가상 머신 인스턴스의 이질성도는 컴퓨팅 환경이 다른 유형의 가상 머신 인스턴스로 이루어졌다는 것을 의미한다.

가상 머신 인스턴스의 개수($\#vm$)는 DAG를 실행하는 고성능 컴퓨팅 시스템이 갖는 가상 머신 인스턴스의 개수를 의미한다.

제안하는 알고리즘의 성능 평가는 표준화한 전체 실행 시간(Schedule Length Ratio, SLR)을 기준으로 하였다. 본 논문에서는 DAG에 다양한 매개변수를 적용하여 시뮬레이션을 진행하므로 순수한 알고리즘의 성능을 평가하기 위해 표준화한 전체 실행 시간을 성능 평가 기준으로 사용하였다. SLR은 식 (9)와 같이 전체 실행 시간을 임계 경로를 구성하는 태스크의 최소 계산비용을 합한 값으로 나눠 계산할 수 있다. 따라서 작은 SLR을 제공할수록 알고리즘의 성능이 우수하다고 할 수 있다.

$$SLR = \frac{Makespan}{\sum_{v_i \in CP(G)} \min_{s_{j,k} \in S_{pi}} \{w(v_i, s_{j,k})\}} \tag{9}$$

표 5. 그래프 내의 태스크 개수를 바꿔가며 계산한 Makespan
Table5. Makespans with respect to the number of tasks in a graph.

	50	100	300	500	750
HCPFD	373.2	622.2	1551.2	2452.3	3586.6
HEFT	376.4	602.9	1416.4	2196.3	3188.5
PETS	381.0	613.6	1441.9	2232.9	3236.7
HAGD	358.6	580.9	1386.1	2156.5	3133.0

제안하는 HAGD 알고리즘은 IaaS를 기반으로 하는 고성능 컴퓨팅 시스템에서 대용량 병렬 프로그램을 빠르게 실행하는 것을 목표로 한다. 따라서 같은 목표를 갖는 알고리즘인 HCPFD, HEFT, PETS 알고리즘을 비교 대상 알고리즘으로 선택하였다. 하지만 HCPFD, HEFT, PETS 알고리즘은 IaaS를 기반으로 하는 고성능 컴퓨팅 시스템이 등장하기 이전에 발표된 알고리즘이므로 이를 일부 수정하여 적용하였다.

그림 7은 표준 태스크 그래프를 이용하여 HAGD 알고리즘을 실험한 결과이다. 그림 7.(a)는 DAG 내의 태스크 개수를 바꿔가며 SLR을 계산한 결과이다. 그림에서 볼 수 있듯이 HAGD 알고리즘이 DAG 내의 태스크 개수와 관계없이 HCPFD 알고리즘에 비해 3.8%~16.2%, HEFT 알고리즘에 비해 3.1%~7.3%, PETS 알고리즘에 비해 5.0%~8.7% 우수한 성능을 제공하고 있다.

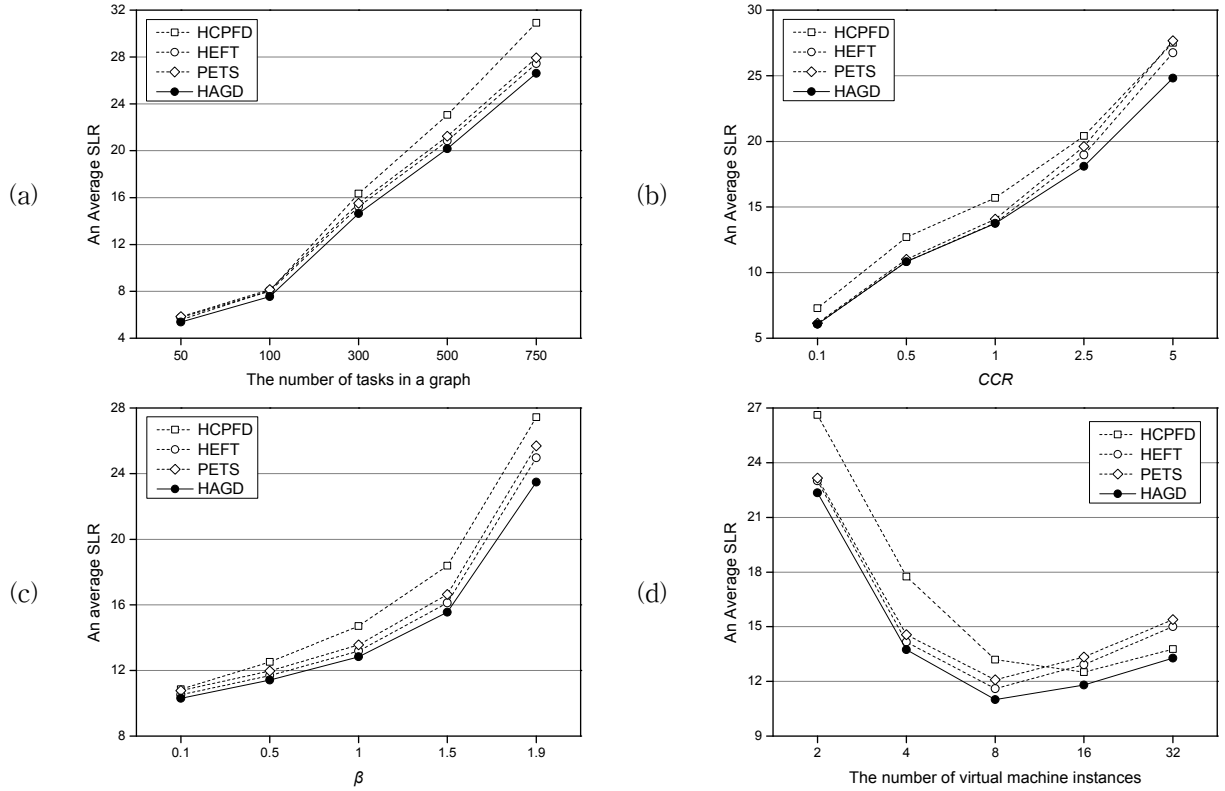


그림 7. 표준 태스크 그래프를 이용하여 HAGD 알고리즘을 실험한 결과
Fig. 7. Results of evaluating the HAGD algorithm with standard task graphs.

표 6. *CCR*을 바꿔가며 확인한 평균 태스크 삽입 및 복제 횟수
Table6. The number of average task insertions and duplications with respect to *CCR*.

		0.1	0.5	1	2.5	5
삽입	HCPFD	0	0	0	0	0
	HEFT	32.3	35.8	39.7	45.5	68.5
	PETS	32.5	36.1	40.6	43.9	62.6
	HAGD	32.3	35.7	39.7	0.1	0
복제	HCPFD	12.5	23.0	31.4	47.7	73.2
	HEFT	0	0	0	0	0
	PETS	0	0	0	0	0
	HAGD	0	0	0	42.3	64.7

표 5는 DAG 내의 태스크 개수를 바꿔가며 *Makespan*을 계산한 결과이다. HAGD 알고리즘은 DAG 내의 태스크 개수와 관계없이 기존 알고리즘에 비해 우수한 성능을 제공하지만, 태스크 개수가 늘어남에 따라 태스크 복제 기법을 사용하는 HEFT, PETS 알고리즘과는 성능 차이가 좁아지고 태스크 삽입 기법을 사용하는 HCPFD 알고리즘과는 성능 차이가 벌어진다. 그림 7.(a)와 표 5의 결과로 봤을 때 DAG 내의 태스크 개수가 많은 경우 태스크 삽입 기법을 사용하는 것이 유리하며, HAGD 알고리즘은 태스크 삽입 기법과 묶음 태스크 복제 기법을 모두 사용하기 때문에 항상 우수한 성능을 제공하는 것으로 볼 수 있다.

그림 7.(b)는 DAG의 *CCR*을 바꿔가며 SLR을 계산한 결과이다. 그림에서 볼 수 있듯이 HAGD 알고리즘이 *CCR*과 관계없이 HCPFD 알고리즘에 비해 10.9%~20.0%, HEFT 알고리즘에 비해 0.1%~7.8%, PETS 알고리즘에 비해 1.2%~11.5% 우수한 성능을 제공하고 있다.

표 6은 DAG의 *CCR*을 바꿔가며 *SLR*을 계산하는 동안 확인한 평균 태스크 삽입 및 복제 횟수이다. 표에서 볼 수 있듯이 HAGD 알고리즘은 태스크 할당 과정에서 확인한 환경 특성에 맞는 성능 향상 기법을 사용하고 있다. 그림 7.(b)와 표 6의 결과로 봤을 때 HAGD 알고리즘은 *CCR*이 낮은 환경에서 태스크 삽입 기법을 적극 사용하여, *CCR*이 높은 환경에서 묶음 태스크 복제 기법을 적극 사용하여 우수한 성능을 제공한다. 또한 *CCR*이 5일 때 HAGD 알고리즘과 HCPFD 알고리즘 모두 태스크 복제 기법만을 사용했음에도 HAGD 알고리즘이 HCPFD 알고리즘에 비해 우수한 성능을 제공하는 것으로 미루어보아 묶음 태스크 복제 기법이 간단한

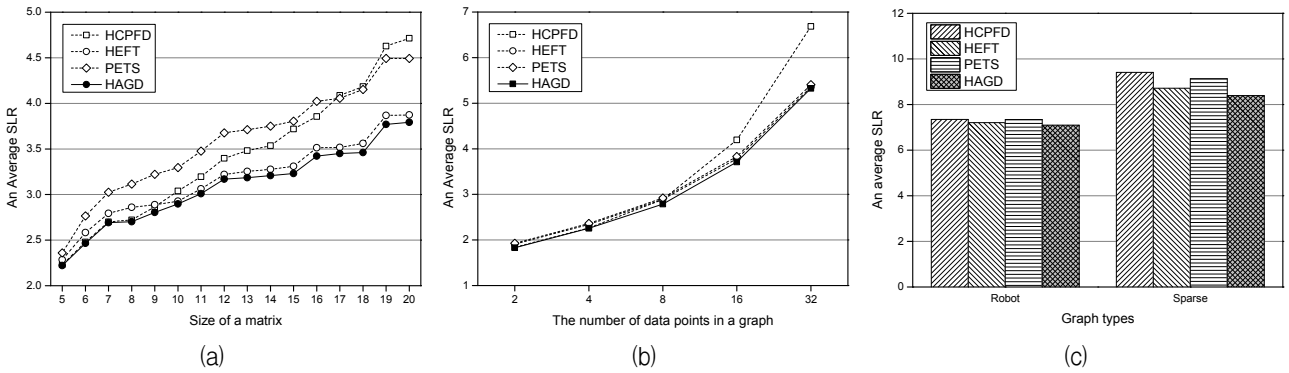


그림 8. 실제 응용 프로그램을 모델링한 태스크 그래프를 이용하여 HAGD 알고리즘을 실험한 결과
Fig 8. Results of evaluating the HAGD algorithm with real application model task graphs.

복제 과정을 가졌음에도 우수한 성능을 제공하는 것으로 볼 수 있다.

그림 7.(c)는 컴퓨팅 환경의 이질성도를 바꿔가며 *SLR*을 계산한 결과이다. 그림에서 볼 수 있듯이 HAGD 알고리즘이 이질성도와 관계없이 HCPFD 알고리즘에 비해 5.3%~16.8%, HEFT 알고리즘에 비해 2.0%~6.3%, PETS 알고리즘에 비해 4.6%~9.4% 우수한 성능을 제공하고 있다.

그림 7.(d)는 컴퓨팅 환경이 포함하는 가상 머신 인스턴스의 개수를 바꿔가며 *SLR*을 계산한 결과이다. 그림에서 볼 수 있듯이 HAGD 알고리즘이 가상 머신 인스턴스의 개수와 관계없이 HCPFD 알고리즘에 비해 3.7%~19.1%, HEFT 알고리즘에 비해 2.9%~13.0%, PETS 알고리즘에 비해 3.6%~15.9% 우수한 성능을 제공하고 있다. 태스크를 충분히 복제할 수 없는 가상 머신 인스턴스가 적은 환경에서는 HAGD 알고리즘이 태스크 삽입 기법을 사용하는 HEFT, PETS 알고리즘에 비해 소폭 우세한 성능을, 태스크 복제 기법을 사용하는 HCPFD 알고리즘에 비해 대폭 우세한 성능을 제공하는 것을 확인할 수 있다. 반대로 태스크를 충분히 복제할 수 있는 가상 머신 인스턴스가 많은 환경에서는 HAGD 알고리즘이 태스크 삽입 기법을 사용하는 HEFT, PETS 알고리즘에 비해 대폭 우세한 성능을, 태스크 복제 기법을 사용하는 HCPFD 알고리즘에 비해 소폭 우세한 성능을 제공하는 것을 확인할 수 있다. 결과적으로 HAGD 알고리즘은 태스크 삽입 기법과 묶음 태스크 복제 기법을 모두 사용하기 때문에 가상 머신 인스턴스의 개수와 관계없이 우수한 성능을 제공한다고 할 수 있다.

그림 8은 실제 응용 프로그램을 모델링한 태스크 그래프를 이용하여 HAGD 알고리즘을 실험한 결과를 보여준다. 그림 8.(a)는 GE 태스크 그래프의 매트릭스 크기를 바꿔가며 SLR을 계산한 결과이다. 그림에서 볼 수 있듯이 HAGD 알고리즘이 매트릭스 크기와 관계없이 HCPFD 알고리즘에 비해 0.5%~26.5%, HEFT 알고리즘에 비해 0.8%~3.5%, PETS 알고리즘에 비해 6.9%~17.6% 우수한 성능을 제공하고 있다. 특히, 매트릭스의 크기가 늘어나면서 태스크 복제 기법만을 사용하는 HCPFD 알고리즘과의 성능 차이가 벌어지는 것을 확인할 수 있다.

그림 8.(b)는 FFT 태스크 그래프의 데이터 포인트 개수를 바꿔가며 *SLR*을 계산한 결과이다. 그림에서 볼 수 있듯이 HAGD 알고리즘이 데이터 포인트 개수와 관계없이 HCPFD 알고리즘에 비해 0.3%~20.5%, HEFT 알고리즘에 비해 0.2%~3.9%, PETS 알고리즘에 비해 0.8%~5.0% 우수한 성능을 제공하고 있다. 특히, 데이터 포인트 개수가 늘어나면서 태스크 복제 기법만을 사용하는 HCPFD 알고리즘과의 성능 차이가 벌어지는 것을 확인할 수 있다.

그림 8.(c)는 Robot control 응용 프로그램과 Sparse matrix solver 응용 프로그램을 이용하여 *SLR*을 계산한 결과이다. 그림에서 볼 수 있듯이 HAGD 알고리즘이 Robot control 응용 프로그램에서 1.6%~3.6%, Sparse matrix solver 응용 프로그램에서 3.9%~12.2% 우수한 성능을 제공하고 있다.

VI. 결 론

본 논문에서는 IaaS를 이용하여 구축한 고성능 컴퓨팅 시스템에서 사용자 비용 등의 고려 없이 대용량 병렬 프로그램을 빠르게 실행하기 위한 HAGD 알고리즘과 묶음 복제 기법을 제안하였다. HAGD 알고리즘은

태스크를 할당하기 전에 고성능 컴퓨팅 시스템과 DAG의 특성을 확인하고, 확인한 특성에 맞게 각 태스크를 태스크 삽입 기법 혹은 본 논문에서 제안한 묶음 태스크 복제 기법을 이용하여 가상 머신 인스턴스에 할당한다. HAGD 알고리즘의 성능을 확인하기 위해 다양한 표준 태스크 그래프와 실제 응용 프로그램을 모델링한 그래프를 이용하여 *CCR*, 이질성도, 가상 머신 인스턴스의 개수 등 환경을 변화해가며 실험하였다. 모든 실험을 종합한 결과 HAGD 알고리즘이 고성능 컴퓨팅 시스템과 DAG의 특성에 맞는 성능 향상 기법을 이용하여 기존 알고리즘에 비해 최소 0.1%에서 최대 26.5% 우수한 성능을 제공하는 것을 확인할 수 있었다. 제안한 HAGD 알고리즘은 대규모 과학 연구나 시간에 민감한 기업의 연구 과제 등에서 IaaS를 이용한 고성능 컴퓨팅 시스템을 사용할 때 가능한 빨리 대용량 병렬 프로그램의 실행을 완료하려는 사용자들에게 유용할 것이다. 그리고 *CCR*, 이질성도와 함께 환경 특성을 알 수 있는 추가 요인을 고려하여 HAGD 알고리즘을 수정하면 더욱 우수한 성능의 알고리즘을 얻을 수 있을 것이다.

REFERENCES

[1] V. Mauch, M. Kunze, and M. Hillenbrand, “High performance cloud computing,” Future Generation Computer Systems, vol. 29, no. 6, pp. 1408-1416, Mar 2012.

[2] M. A. Vouk, E. Sills, and P. Dreher, “Integration of High- Performance Computing into Cloud Computing Services,” Handbook of Cloud Computing, Springer, pp. 255-276, 2010.

[3] E. Deelman et al., “Mapping Abstract Complex Workflows onto Grid Environments,” Journal of Grid Computing, vol. 1, no. 1, pp. 25-39, Mar 2003.

[4] H. Topcuoglu, S. Hariri, and Min-You Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260-274, Mar 2002.

[5] E. Ilavarasan, P. Thambidurai, and R. Mahilmannan, “Performance Effective Task Scheduling Algorithm for Heterogeneous Computing System,” The 4th International Symposium on Parallel and Distributed Computing (ISPDC'05), Lille, pp. 28-38, 2005.

[6] T. Hagras and J. Janecek, “A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems,” Parallel Computing, vol. 31, no. 7, pp. 653-670, Oct 2004.

[7] W. Zheng and R. Sakellariou, “Budget-Deadline Constrained Workflow Planning for Admission Control,” Journal of grid computing, vol. 11, no. 4, pp. 633-651, May 2013.

[8] S. Abrishami, M. Naghibzadeha, and D. H. J. Epemab, “Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Cloud,” Future Generation Computer Systems, vol. 29, no. 1, pp. 158-169, Jan 2013.

[9] M. Al-Fares, A. Loukissas, and A. Vahdat, “A Scalable, Commodity Data Center Network Architecture,” Proceedings of the ACM SIGCOMM 2008 conference on Data communication, pp. 63-74, 2008.

[10] D. C. Marinescu, “Cloud Computing: Introduction,” Cloud Computing; Theory and Practice, Morgan Kaufmann, pp. 99-130, 2013.

[11] O. Sinnen, Task Scheduling For Parallel Systems, Wiley, 2007.

[12] E. N Alkhanak, S. P. Lee, and S. U. R. Khan, “Cost-aware challenges for workflow scheduling approaches in cloud computing environments: Taxonomy and opportunities,” Future Generation Computer Systems, vol. 50, pp. 3-21, Feb 2015.

[13] Kasahara Laboratory. Standard Task Graph set [Online]. http://www.kasahara.elec.waseda.ac.jp.

[14] H. Kasahara and S. Narita, “Parallel Processing of Robot-Arm Control Computation on a Multiprocessor System,” IEEE Journal of Robotics and Automation, vol. RA-1, no. 2, pp. 104-113, Jun 1985.

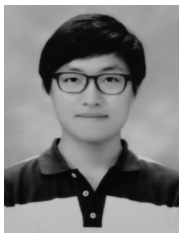
[15] H. Kasahara, H. Honda, and S. Narita, “Parallel processing of near fine grain tasks using static scheduling on OSCAR (optimally scheduled advanced multiprocessor),” Supercomputing '90., Proceedings of, New York, NY, pp. 856-864, 1990.

[16] A. K. Amoura, E. Bampis, and J. C. Konig, “Scheduling algorithms for parallel Gaussian elimination with communication costs,” IEEE Transactions on Parallel and Distributed Systems, vol. 9, no. 7, pp. 679-686, Jul 1998.

[17] E Gelenbe and S. Chabridon, “Characterization, Approximate Analysis and Simulation of Task Graph Models Representing the Dependable Execution of Parallel and Distributed Programs,” QMIPS, 1994.

저 자 소 개

송 인 성(정회원)
2009년 인하대학교 전자공학과 학사 졸업.
2011년 인하대학교 전자공학과 석사 졸업.
2016년 인하대학교 전자공학과 박사 졸업.
<주관심분야 : 병렬 및 분산처리 시스템, 컴퓨터 구조>



박 태 신(정회원)
2016년 인하대학교 전자공학과 학사 졸업.
2016년～현재 인하대학교 전자공학과 석사과정 재학.
<주관심분야 : 무선 센서 네트워크, 병렬 및 분산처리 시스템>

윤 동 성(정회원)
2016년 인하대학교 전자공학과 학사 졸업.
2016년～현재 인하대학교 전자공학과 석사과정 재학.
<주관심분야 : 무선 센서 네트워크, 병렬 및 분산처리 시스템>

최 상 방(평생회원)
1981년 한양대학교 전자공학과 학사 졸업.
1988년 University of Washington 석사 졸업.
1990년 University of Washington 박사 졸업.
1991년～현재 인하대학교 전자공학과 교수
<주관심분야 : 컴퓨터 구조, 컴퓨터 네트워크, 무선 통신, 병렬 및 분산처리 시스템>