Journal of The Institute of Electronics and Information Engineers Vol.52, NO.3, March 2015 http://dx.doi.org/10.5573/ieie.2015.52.3.116
ISSN 2287-5026(Print) ISSN 2288-159X(Online)

논문 2015-52-3-15

분산 이기종 컴퓨팅 시스템에서 임계노드를 고려한 태스크 스케줄링 알고리즘

(A Novel Task Scheduling Algorithm Based on Critical Nodes for Distributed Heterogeneous Computing System)

김 호 중*, 송 인 성*, 정 용 수*, 최 상 방**

(Hojoong Kim, Inseong Song, Yong Su Jeong, and SangBang Choi©)

요 약

분산 이기종 시스템에서 병렬 응용프로그램의 성능은 태스크를 스케줄링하는 방법에 따라 크게 영향을 받는다. 따라서 병렬 응용프로그램의 성능에 영향을 미치는 요소들을 태스크 스케줄링에 반영함으로써 주어진 환경 내에서 최적의 결과를 도출할 수 있도록 해야 한다. 일반적으로 병렬 응용프로그램의 전체 처리시간에 영향을 미치는 결정적 요소는 입력 그래프의 임계경로이다. 본 논문에서는 임계 경로 상의 임계노드를 고려한 태스크 스케줄링 알고리즘인 CLTS를 제안한다. CLTS는 우선순위 결정 단계에서 계층화를 통해 노드의 병렬처리 효율을 향상시키고 임계노드 처리에 의한 지연시간을 단축시킬 수 있도록 우선순위를 결정한다. 또 프로세서 할당 단계에서는 조건적으로 복제 기반 정책, 혹은 삽입 기반 정책을 사용하여 노드를 프로세서에 할당함으로써 전체 처리시간을 단축시킨다. 제안한 CLTS의 성능 평가를 위해 기존의 리스트 스케줄링 알고리즘인 HCFPD, DCPD와 함께 성능을 비교 평가하였다. 시뮬레이션을 통해 CLTS는 평균 SLR을 기준으로 HCPFD 대비 7.29%, DCPD 대비 8.93% 향상되었고, Speedup을 기준으로 HCPFD 대비 9.21%, DCPD 대비 7.66% 향상된 성능을 보임을 확인하였다.

Abstract

In a distributed heterogeneous computing system, the performance of a parallel application greatly depends on its task scheduling algorithm. Therefore, in order to improve the performance, it is essential to consider some factors that can have effect on the performance of the parallel application in a given environment. One of the most important factors that affects the total execution time is a critical path. In this paper, we propose the CLTS algorithm for a task scheduling. The CLTS sets the priorities of all nodes to improve overall performance by applying leveling method to improve parallelism of task execution and by reducing the delay caused by waiting for execution of critical nodes in priority phase. After that, it conditionally uses insertion based policy or duplication based policy in processor allocation phase to reduce total schedule time. To evaluate the performance of the CLTS, we compared the CLTS with the DCPD and the HCPFD in our simulation. The results of the simulations show that the CLTS is better than the HCPFD by 7.29% and the DCPD by 8.93%. with respect to the average SLR, and also better than the HCPFD by 9.21% and the DCPD by 7.66% with respect to the average speedup.

Keywords : distributed heterogeneous computing system, list scheduling, task scheduling, directed acyclic graph

* 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 15, 2014 Revised ; February 19, 2015 Accepted ; March 4, 2015

Ⅰ. 서 론

최근 디지털 신호처리, 생명공학, 암호학, 수학, 기상 모델링 등 다양한 분야에서 병렬 응용프로그램이 이용되고 있다. 특히 네트워크의 발달로 프로세서 간에 효율적인 통신이 가능해지면서 분산 병렬 처리를 통해 대용량의 프로그램을 빠르게 처리 할 수 있는 분산 이기종 컴퓨팅 시스템이 주목받고 있다[1].

태스크 스케줄링의 목적은 처리 할 병렬 응용프로그램의 전체 처리시간을 태스크의 처리시간, 태스크간의 통신시간, 태스크 간의 선행제약을 고려하여 최소화하는 것이다. 일반적으로 전체 처리시간은 임계경로의 처리시간을 단축함으로써 크게 향상 시킬 수 있다. 분산 이기종 컴퓨팅 시스템 환경에서는 다양한 태스크 스케줄링 방법이 존재하며 그 중 리스트 스케줄링은 일반적으로 낮은 복잡도와 우수한 성능을 나타내는 것으로 알려져 있다.

리스트 스케줄링은 크게 우선순위 결정 단계와 프로세서 할당 단계로 구성된다. 우선순위 결정 단계는 병렬 프로그램 내의 모든 태스크들이 특정 기준 함수에 의해서 처리 순서가 정해지는 단계이며, 프로세서 할당 단계는 주어진 상황에서 해당 태스크를 가장 최적으로 처리 할 수 있는 프로세서 선택하여 태스크를 할당하는 단계이다[2-6].

본 논문에서는 임계노드를 고려한 태스크 스케줄링 알고리즘인 CLTS (Critical node Look-ahead Task Scheduling)을 제안한다. CLTS는 우선순위 결정 단계에서 다음 계층의 임계노드를 미리 고려함으로써 임계노드에 의한 지연을 최소화 시키고 계층화를 통해 병렬성을 향상하여 처리완료시간을 줄일 수 있도록 노드의 우선순위를 결정한다. 프로세서 할당 단계에서는 프로세서에 할당할 노드를 임계노드와의 관계를 고려하여 복제 기반 정책, 혹은 삽입 기반 정책을 조건적으로 사용함으로써 전체 처리시간을 단축시키면서도 태스크의 무분별한 복제를 제한한다.

제안한 알고리즘의 성능을 확인하기 위해 다양한 매개변수에 의해 만들어진 입력 그래프를 사용하여 기존의 리스트 스케줄링 알고리즘 중 복제 기반 정책을 사용하는 알고리즘인 HCPFD[3], DCPD[4]와 성능을 비교 평가하였다. 그 결과 본 논문에서 제안한 CLTS가 다른 알고리즘에 비해 우수한 성능을 보임을 확인하였다.

본 논문은 다음과 같이 이루어져 있다. Ⅱ장에서는 분산 이기종 시스템에서의 태스크 스케줄링을 위한 문제를 정의하고 Ⅲ장에서는 기존의 리스트 스케줄링 알고리즘에 대해 소개한다. Ⅳ장에서는 본 논문에서 제안한 알고리즘을 설명하고, Ⅴ장에서는 시뮬레이션을 통해 제안한 알고리즘의 성능 분석을 기술하며, 마지막으로 Ⅵ장의 결론으로 본 논문을 마무리한다.

Ⅱ. 문제 정의

분산 이기종 시스템에서의 병렬 응용프로그램은 태스크를 나타내는 노드와 태스크간의 상호 제약조건이 존재하는 방향성 비순환 그래프(DAG, Directed Acyclic Graph)로 표현할 수 있으며, 방향성 비순환 그래프 G는 $G=(V,E)$ 로 표현된다. V는 병렬 응용프로그램 내에서의 태스크, 또는 노드, v_i의 집합이고 그래프 내에서 원으로 표현한다. E는 v_i와 v_j간의 에지를 나타내는 $e_{i,j}$의 집합을 나타내며 노드 사이의 간선으로 표현한다. 그림 1은 15개의 노드를 가지는 방향성 비순환 그래프의 예를 보여준다. 노드 간에는 한 노드의 처리가 모두 끝난 뒤에 다른 노드를 처리할 수 있는 상호 제약조건이 존재한다. 노드 v_i와 노드 v_j를 연결하는 에지 $e_{i,j}$가 존재 하면 노드 v_i를 노드 v_j의 부모노드라고 하며, 노드 v_j는 노드 v_i의 자식노드라고 한다. 만약 노드 v_i가

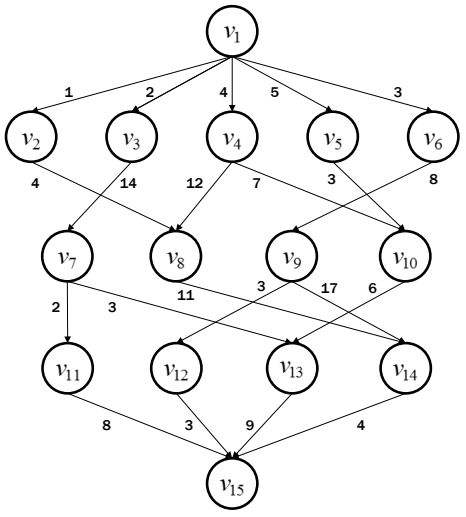

그림 1. 15개의 노드를 가지는 방향성 비순환 그래프의 예
Fig. 1. Example of Direct Acyclic Graph with 15 nodes.

여러 부모노드들과 에지로 연결되어 있다면 노드 v_i의 모든 부모노드들의 집합을 $pred(v_i)$로 정의하며, 노드 v_j의 모든 자식노드들의 집합을 $succ(v_i)$로 정의한다.

평균 처리시간은 모든 프로세서가 노드 v_i를 처리하는데 소요되는 평균 처리시간을 의미한다. 만약 프로세서의 개수가 m개일 때 노드 v_i의 평균 처리시간 $w_{avg(v_i)}$는 다음과 같이 정의한다.

$$w_{avg(v_i)} = \sum_{k=1}^{m} \frac{w_{i,k}}{m} \quad (1)$$

표 1은 노드의 프로세서 별 노드의 처리시간 및 평균 처리시간을 보여준다.

평균 통신시간은 노드 v_i에서 노드 v_j로 데이터를 전달하는데 소요되는 평균 시간이며 $c_{avg(v_i, v_j)}$로 정의한다.

$uprank(v_i)$는 노드 v_i부터 종료 노드까지의 가장 긴 시간을 경로를 나타낸다. 종료 노드부터 상향으로 자식노드의 $uprank(v_i)$를 재귀적으로 호출하고 자식노드와의 평균 통신시간과 노드 v_i의 평균 처리시간을 더하여 계산한다. 노드 v_i가 여러 자식노드들을 가지고 있다면 그 중 가장 큰 $uprank(v_i)$를 노드 v_i의 $uprank(v_i)$로 정의한다. $uprank(v_i)$는 다음과 같이 식으로 정의한다.

$$uprank(v_i) = w_{avg(v_i)} + \max_{v_j \in succ(v_i)} \{c_{avg(v_i, v_j)} + uprank(v_j)\} \quad (2)$$

$uprank(v_i)$와 비슷한 방법으로 노드 v_i의 $downrank(v_i)$를 계산 할 수 있다. 시작 노드부터 하향으로 자식노드의 $downrank(v_i)$를 재귀적으로 호출하고 부모노드와의 평균 통신시간과 노드 v_i의 평균 처리시간을 더하여 $downrank(v_i)$를 계산한다. 노드 v_i가 여러 부모노드들을 가지고 있다면 부모노드와 $downrank(v_i)$를 계산하여 그 중 가장 큰 $downrank(v_i)$를 노드 v_i의 $downrank(v_i)$로 정의한다. $downrank(v_i)$는 다음과 같은 식으로 정의한다.

$$downrank(v_i) = w_{avg(v_i)} + \max_{v_j \in pred(v_i)} \{c_{avg(v_i, v_j)} + downrank(v_j)\} \quad (3)$$

임계경로는 그래프의 시작 노드부터 종료 노드까지의 여러 경로 중 가장 긴 경로를 의미한다. 임계노드는 임계경로를 구성하고 있는 노드이며 노드의 $uprank(v_i)$와 $downrank(v_i)$가 같은 값을 가지는 노드들을 연결하여 구할 수 있다.

최소시작시간 $est(v_i, p_j)$는 노드 v_i의 모든 부모노드로부터 데이터를 전달 받아 프로세서 p_j가 노드 v_i를 처리할 수 있는 가장 빠른 시간을 의미하며 다음과 같은 식으로 정의한다.

$$est(v_i, p_j) = \max\{prt(p_j), \max_{v_j \in pred(v_i)}(aeft(v_j) + c_{avg(v_i, v_j)})\} \quad (4)$$

여기에서 $prt(p_j)$은 프로세서 p_j가 노드를 실행할 수 있는 시간을 의미한다. 또 $aeft(v_j)$는 노드 v_j가 프로세서에 할당되어 처리된 뒤 후의 실제 실행 종료시간을 의미한다. est는 부모노드의 $aeft$와 부모노드로부터의 통신시간을 더한 값을 prt와 비교하여 그 중 큰 값을 선택하여 결정한다. 단 시작 노드의 est는 항상 0이다.

최소종료시간 $eft(v_i, p_j)$는 $est(v_i, p_j)$에 노드 v_i의 프

표 1. 프로세서 별 노드의 처리시간 및 평균 처리시간

Table 1. Computation time and average computation time of each processor for nodes.

$node$	p_1	p_2	p_3	w_{avr}
v_1	8	4	2	4.67
v_2	12	6	3	7.00
v_3	16	8	4	9.33
v_4	8	4	2	4.67
v_5	4	2	1	2.33
v_6	12	6	3	7.00
v_7	8	4	2	4.67
v_8	16	8	4	9.33
v_9	8	4	2	4.67
v_{10}	4	2	1	2.33
v_{11}	4	2	1	2.33
v_{12}	4	2	1	2.33
v_{13}	8	4	2	4.67
v_{14}	12	6	3	7.00
v_{15}	8	4	2	4.67

로세서 p_j에서의 실행시간을 더한 값이며 다음과 같은 식으로 정의한다.

$$eft(v_i, p_j) = est(v_i, p_j) + w_{i,j} \quad (5)$$

Ⅲ. 관련 연구

이번 장에서는 지금까지 연구된 리스트 스케줄링 알고리즘 중 삽입 기반 정책을 사용하는 알고리즘인 HEFT[2]와 PETS[5], 그리고 복제 기반 정책을 사용하는 알고리즘인 HCPFD와 DCPD에 대해 설명한다.

1. HEFT

HEFT (Heterogeneous Earliest Finish Time)는 대표적인 리스트 스케줄링 알고리즘으로 우선순위 결정 단계와 프로세서 할당 단계로 이루어진다. 우선순위 결정 단계에서는 입력 그래프에서 노드로 표현된 태스크들의 우선순위를 정하기 위해 모든 노드의 $uprank(v_i)$를 계산한다. $uprank(v_i)$는 해당 노드의 처리시간과 노드 간 통신시간을 종료 노드부터 시작 노드까지 재귀적인 방법으로 더하여 계산하며, $uprank(v_i)$ 값을 내림차순으로 정렬하여 노드의 우선순위를 정한다. 프로세서 할당 단계에서는 우선순위 결정 단계에서 결정된 노드의 우선순위를 이용하여 노드를 프로세서에 할당한다. 프로세서를 할당할 때에는 한 노드의 각 프로세서에서의 최소종료시간을 계산하고, 이 중 해당 노드의 최소종료시간이 가장 빠른 프로세서를 선택하여 노드를 처리하도록 할당한다. 그 과정에서 HEFT는 선행 제약을 만족하면서, 프로세서에 해당 노드를 처리할 프로세서의 여유 공간이 있고, 다른 프로세서에 비해 최소완료시간이 빠르다면, 삽입 기반 정책을 사용하여 프로세서의 여유 공간에 노드를 삽입하는 삽입 기반 정책을 사용한다.

2. PETS

PETS (Performance Effective Task Scheduling)은 태스크의 계층화를 이용하는 리스트 스케줄링 알고리즘이다. 태스크의 우선순위를 결정하기 전에 입력 그래프 내의 독립적인 노드들을 병렬로 처리할 수 있도록 태스크를 계층별로 분류한다. PETS의 우선순위 결정 과정에서는 부모노드 중 가장 큰 $rank(v_i)$를 가지는 부모노드의 $rank(v_i)$와, 해당 부모노드로부터 현재 노드로의 통신시간, 현재 노드부터 자식노드로의 통신시간의 합, 그리고 현재 노드의 평균 처리 시간을 모두 더하여 현재 노드의 $rank(v_i)$를 계산한다. 다음으로 계산된 $rank(v_i)$를 내림차순으로 정렬하여 노드의 우선순위를 정한다. 프로세서 할당 단계에서는 HEFT와 마찬가지로 한 노드에 대해 최소종료시간이 가장 빠른 프로세서를 선택하여 노드를 할당하며 이때 삽입 정책을 고려한다.

3. HCPFD

HCPFD (Heterogeneous Critical Parents with Fast Duplicator)는 우선순위 결정 단계와 프로세서 할당 단계로 이루어진다. 우선순위 결정 단계에서는 HEFT와 같이 모든 노드에 대해 $uprank(v_i)$를 계산한다. 그 후 임계노드들 중 $uprank(v_i)$가 낮은 순서대로 스택 S에 넣는다. 모든 임계노드들을 S에 넣었다면 S의 최상위 노드부터 빼내어 비어있는 큐 L에 넣는다. 이때 해당 임계노드의 부모노드가 모두 L에 존재하는지 확인하고, 모든 부모노드가 존재한다면 해당 임계노드를 L에 넣고, 만약 L에 해당 임계노드의 부모노드가 하나라도 존재하지 않는다면 부모노드들을 먼저 L에 넣은 후 임계노드를 넣게 된다. 만약 넣어야 할 부모노드가 2개 이상이라면 $uprank(v_i)$를 비교하여 $uprank(v_i)$가 높은 노드부터 순서대로 넣는다. 이러한 방법으로 S의 모든 임계노드들을 꺼내어 S가 빌 때까지 수행하게 된다. 이렇게 완성된 L이 노드의 우선순위 리스트이다. 프로세서 할당 단계에서는 프로세서 중 최소종료시간이 가장 빠른 프로세서를 선택하여 노드를 할당한다. 이때 노드에 데이터를 가장 늦게 전달하는 부모노드를 찾은 뒤, 부모노드를 해당 노드와 같은 프로세서로 복제하여 해당 노드의 실행시간을 줄이는 복제 기반 정책을 고려한다. 부모노드의 복제를 수행하여 해당 노드의 실행시간을 줄일 수 있다면 복제를 수행하며, 그렇지 않을 경우 복제를 수행하지 않는 방법 내에서 가장 빠른 실행시간을 갖는 프로세서에 노드를 할당한다.

4. DCPD

DCPD (Dynamic Critical Path Duplication)는 복제 기반 정책을 사용하는 리스트 스케줄링 알고리즘이다.

DCPD는 우선순위 결정 단계에서는 종료 노드부터 상향으로 $uprank(v_i)$를 계산하고, 시작 태스크부터 하향으로 $downrank(v_i)$를 계산한다. 그 후 부모노드가 모두 처리된 노드를 찾아 준비세트에 삽입하고, $uprank(v_i)$, $downrank(v_i)$, 노드의 평균 처리시간을 고려하여 노드의 우선순위를 결정한다. 프로세서 할당 단계에서는 노드의 복제를 고려하여 프로세서 중 최소종료시간이 가장 빠른 프로세서에 노드를 할당한다.

Ⅳ. 제안하는 스케줄링 알고리즘

이 장에서는 제안하는 스케줄링 알고리즘인 CLTS에 대해 설명한다. CLTS는 우선순위 결정 단계와 프로세서 할당 단계의 두 단계로 이루어진다.

1. 우선순위 결정 단계

CLTS의 우선순위 결정 단계에서는 먼저 입력 그래프 내의 모든 노드들이 계층별로 분류된다. 계층 정렬을 위해 너비 우선 검색을 이용하여 입력 그래프 상의 독립적인 노드를 계층별로 분류한다. 같은 계층에 속한 노드들은 모두 독립이므로 동시에 실행이 가능하다. 이러한 과정을 통해 생성된 각 계층을 $level_l$ 이라고 부른다. 노드들을 계층별로 분류한 후에는 모든 노드의 평균 처리시간과 각 노드 간 평균 통신시간을 계산한 뒤 이를 이용하여 종료 노드로부터 상향으로 모든 노드에 대한 $uprank(v_i)$를 계산한다. $uprank(v_i)$ 계산 후에는 시작 노드가 속한 계층을 시작으로 처리해야 할 계층을 한 단계씩 이동시키며 다음 계층의 임계노드를 미리 고려하면서 노드의 우선순위를 결정한다. 우선순위를 결정하기 위해서 리스트 큐 L을 사용하며, L에 들어간 노드의 순서가 노드의 우선순위가 된다.

계층의 처리 순서는 $level_1$부터 시작한다. $level_l$에서의 임계노드를 CN_l라 할 때, 처리 계층이 $level_1$인 경우에는 $level_1$의 임계노드인 CN_1을 L에 넣는다. 그 후

표 2. 우선순위 결정 의사코드
Table 2. Pseudo code of priority algorithm.

```
CN_i : A node on a critical path at level_i
L : A list queue

for all nodes do
Group nodes that are independent from each nodes
Set w_avg(i), c_avg(v_i,v_j) and compute the uprank(v_i)
Set c_avg(v_i,pred(v_i)) and w_avg(max pred(v_i))
endfor

Priority Algorithm
for each level except the last level do
    if the level_i is the first level then
        enqueue the CN_i to L
    endif
    if the pred(CN_i+1) exist in L then
        enqueue the CN_i+1 to L
        enqueue the other nodes at level_i in descending order of uprank(v_i)
    else
        enqueue the pred(CN_i+1) which are not in L in descending order of uprank(v_i)
        enqueue the CN_i+1 to L
        enqueue the other nodes at level_i to L in descending order of uprank(v_i)
    endif
endfor
```

다음 계층의 임계노드인 CN_2의 부모노드가 L에 존재하는지 확인한다. CN_2의 부모노드인 CN_1은 L에 존재하므로 CN_2를 L에 넣고 $level_1$의 처리를 마친다.

처리 계층이 $level_2$인 경우에는 다음 계층의 임계노드인 CN_3의 모든 부모노드가 L에 존재하는지 확인한다. 만약 CN_3의 모든 부모노드가 L에 존재한다면 CN_3을 L에 넣고, CN_3의 부모노드 중 L에 존재하지 않는 부모노드가 있다면 $uprank(v_i)$의 내림차순으로 부모노드를 L에 넣은 후에 CN_3을 L에 넣는다. 그 후 $level_2$에 존재하는 노드 중 L에 넣지 않은 노드들을 오름차순으로 L에 넣고 $level_2$의 처리를 마친다.

같은 방법으로 단계별로 처리해야 할 계층을 이동시키며 CN_{l+1}의 부모노드가 L에 존재하는지 확인 후, 모든 부모노드가 L에 존재한다면 L에 CN_{l+1}을 넣고, 부모노드가 하나라도 L에 존재하지 않는다면 $uprank(v_i)$의 내림차순으로 부모노드를 L에 넣은 후 CN_{l+1}을 L에 넣는다. 그 후 $level_l$에 존재하는 노드 중 L에 넣지 않은 노드들을 오름차순으로 L에 넣는다. 만약 마지막 계층을 제외한 모든 계층에 대하여 노드 우선순위 결정을 완료하였다면 우선순위 결정 단계를 종료한다.

이와 같은 우선순위 결정방법은 다음 계층의 임계노드에 높은 우선순위를 부여함으로써 각 계층의 임계노드에 의한 지연을 최소화 할 수 있다. 표 2는 CLTS의 우선순위 결정 단계를 의사코드로 나타낸 것이다.

2. 프로세서 할당 단계

(1) 삽입 기반 정책

CLTS는 삽입 기반 정책(insertion based policy)을 고려하여 노드를 프로세서에 할당한다. 삽입 기반 정책은 해당 노드의 부모노드가 프로세서 내에 이미 할당되어 있고, 프로세서 내에 이미 할당된 노드 사이의 유휴공간이 해당 노드의 처리시간보다 크며, 선행 제약을 위반하지 않는다면 현재 노드를 유휴공간에 할당하는 방법이다. 삽입 기반 정책은 비삽입 기반 정책에 비해 빠른 최소종료시간을 제공하여 전체 성능향상에 큰 도움을 주지만, 노드의 삽입을 위해 유휴공간을 탐색해야 하기 때문에 알고리즘의 복잡도는 증가한다.

(2) 복제 기반 정책

CLTS는 복제 기반 정책(duplication based policy)을 사용하여 노드를 프로세서에 할당한다. 복제 기반 정책이란 이미 다른 프로세서에 할당 된 해당 노드의 부모노드를 현재 노드와 같은 프로세서에 복사하여 할당하는 방법으로, 다른 프로세서로부터의 데이터 전송을 기다리는 시간을 크게 줄일 수 있다는 장점을 갖는다. 일반적으로 복제 기반 정책은 전체적인 성능 향상에 큰 도움이 되지만, 알고리즘의 복잡도가 크게 증가한다[7~9].

표 3. 프로세서 할당 단계 의사코드
Table 3. Pseudo code of processor allocation algorithm.

```
Processor Allocation Algorithm
while L is not empty do
    select the first node v_head in L
    for each processor p_j ∈ P do
        compute eft(v_head, p_j) considering insertion based policy.
        if the v_head is CN_l or the predecessor of CN_l then
            if the insertion is impossible and c_avg(v_i, pred(v_i)) > w_avg(max(pred(v_i))) then
                compute eft(v_head, p_j) considering duplication based policy.
            endif
        endif
        allocate the v_head to the p_j that provides the earliest finish time
        dequeue the v_head in L
    endfor
endwhile
```

(3) 삽입 및 복제를 모두 고려한 프로세서 할당

CLTS는 프로세서 할당 단계에서 우선순위 결정 단계에서 생성된 L에 포함된 노드를 순서대로 꺼내어 가장 빠른 최소종료시간을 갖는 프로세서에 노드를 할당한다. 이때 최소종료시간을 계산하기 위해서 삽입 기반 정책과 복제 기반 정책을 모두 사용한다.

먼저 L에서 꺼낸 노드를 삽입 기반 정책을 고려하여 프로세서 별 최소종료시간을 계산한다. 삽입 기반 정책이 사용 가능한 경우에는 최소종료시간이 가장 빠른 프로세서에게 할당하고, 삽입 기반 정책을 사용하지 않은 경우에는 다음 단계로 복제 기반 정책을 고려한다. 이때 복제의 대상이 되는 노드가 임계노드 또는 임계노드의 부모노드인지 확인하고, 대상 노드가 임계노드 또는 임계노드의 부모노드인 경우에만 노드를 복제한다. 또한 부모노드 중 가장 큰 평균 처리 시간을 갖는 노드의 평균 처리 시간과 부모노드로의 평균 통신 시간의 평균값을 비교하여 평균 통신 시간이 부모노드 중 가장 큰 평균 처리 시간보다 큰 경우에만 복제를 고려한다.

이를 통하여 임계노드를 포함한 모든 노드에 대해 노드의 삽입을 고려하여 전체 처리시간을 단축시킬 수 있다. 또한 임계노드와 임계노드의 부모노드를 복제함으로써 전체 실행 시간에 큰 영향을 주는 임계경로의 실행 시간을 크게 줄일 수 있고, 노드의 복제를 임계노드와 임계노드의 부모노드로 제한함으로써 다른 노드의 삽입 혹은 복제 기회를 늘릴 수 있다. 표 3은 CLTS의 프로세서 할당 단계 의사코드를 보여준다.

3. 입력 그래프를 이용한 CLTS의 결과의 예

본 항에서는 CLTS의 수행과정의 이해를 돕기 위해 그림 1의 입력 그래프를 이용한 예를 제시한다. CLTS의 우선순위 결정 단계에서 결정된 노드의 처리순서는 $\{v_1, v_4, v_2, v_8, v_6, v_3, v_5, v_9, v_{14}, v_{10}, v_7, v_{13}, v_{11}, v_{12}, v_{15}\}$ 이다.

표 4는 CLTS를 수행하며 계산해야 할 노드별 est, eft와, 노드가 할당되는 프로세서, 삽입 기반 정책과 복제 기반 정책 사용 여부를 보여준다. CLTS는 입력 그래프를 처리하기 위해 2번의 삽입 기반 정책과 1번의 복제 기반 정책을 사용하였고, 그 결과 전체 처리시간 27이 소요되었다. 그림 2는 CLTS의 최종 스케줄링 결과를 보여준다.

표 4. 노드별 est, eft, 할당 프로세서, 삽입 기반 정책과 복제 기반 정책 여부
Table 4. est, eft, allocated processor, and status of insertion based policy and duplication based policy of each node.

v_i	p_1		p_2		p_3		Allocated processor	Status
	est	eft	est	eft	est	eft		
v_1	0	8	0	4	**0**	**2**	p_3	*nop*
v_2	3	15	3	9	**4**	**7**	p_3	*nop*
v_3	4	20	11	19	**11**	**15**	p_3	*nop*
v_4	6	14	6	10	**2**	**4**	p_3	*nop*
v_5	**7**	**11**	11	13	15	16	p_1	*nop*
v_6	5	17	**5**	**11**	11	14	p_2	*nop*
v_7	17	25	17	21	**15**	**17**	p_3	*Inserted*
v_8	16	32	16	24	**7**	**11**	p_3	*nop*
v_9	19	27	**11**	**15**	19	21	p_2	*nop*
v_{10}	**11**	**15**	15	17	23	24	p_1	*nop*
v_{11}	19	23	19	21	**17**	**18**	p_3	*Inserted*
v_{12}	18	22	**15**	**17**	25	26	p_2	*nop*
v_{13}	20	28	21	25	**23**	**25**	p_3	*nop*
v_{14}	32	44	22	28	**20**	**23**	p_3	v_9 *Dupl.*
v_{15}	34	42	34	38	**25**	**27**	p_3	*nop*

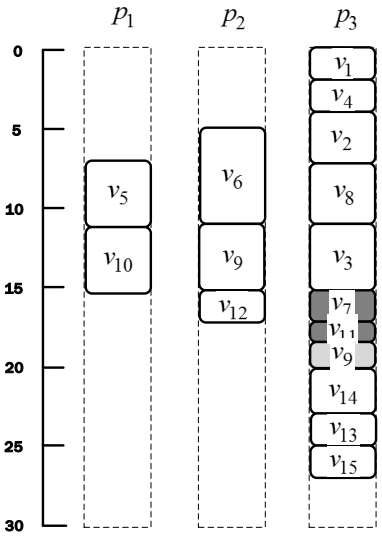

그림 2. CLTS의 최종 스케줄링 결과
Fig. 2. The final scheduling result of the CLTS.

V. 성능 분석 및 평가

본 논문에서 제안한 CLTS 알고리즘의 성능분석을 위해 기존 리스트 스케줄링 알고리즘 중 복제 기반 정책을 사용하는 DCPD, HCPFD를 사용하였다. 시뮬레이션을 위한 입력 그래프는 STDGP (STanDard task Graph Project)에서 제공하는 방향성 비순환 그래프를 노드의 수 50, 100, 300, 500, 700 개, 프로세서 이질성도 (β) 0.1, 0.25, 0.5, 0.75, 1.0, 프로세서의 수 2, 4, 8, 16, 32, 통신비용 대 처리비용 비율(CCR, Communication to Computation Ratio) 0.1, 0.5, 1.0, 2.5, 5.0으로 변화시켜 생성한, 총 106,250개의 입력 그래프를 사용하였다.

공정한 성능 평가를 위해 전체 처리시간을 실행시간의 하한 값으로 표준화 시킨 값 SLR (Schedule Length Ratio)과 순차실행시간을 병렬 실행 시간으로 나눈 값인 Speedup을 성능 평가의 기준으로 사용하였으며 이는 다음의 식으로 계산할 수 있다.

$$SLR = \frac{makespan}{\sum_{n_i \in CP} \min_{p_j \in P} w_{i,j}} \tag{6}$$

$$Speedup = \frac{\min_{p_j} \sum_{v_i \in N} w_{i,j}}{makespan} \tag{7}$$

여기서 $makespan$은 전체 처리시간을 의미한다.

그림 3은 노드 개수의 변화에 따른 입력 그래프의 평균 SLR, 그림 4는 평균 Speedup을 나타낸다. 평균 SLR은 CLTS의 경우 {2.97, 4.18, 8.3, 11.79, 15.58}, HCPFD의 경우 {3.12, 4.45, 8.94, 12.76, 16.92}, DCPD의 경우 {3.06, 4.41, 9.03, 13.03, 17.49} 이다. 평균 Speedup은 CLTS의 경우 {3.92, 4.82, 6.34, 7.26, 7.87}, HCPFD의 경우 {3.73, 4.51, 5.81, 6.57, 7.04}, DCPD의 경우 {3.81, 4.59, 5.93, 6.65, 7.08} 이다. 이를 통해 CLTS가 모든 노드 개수의 경우에 대하여 타 알고리즘 보다 우수한 성능을 보임을 확인 할 수 있고, 특히 노드의 개수가 많을수록 CLTS가 우수한 성능을 보이는 것을 확인

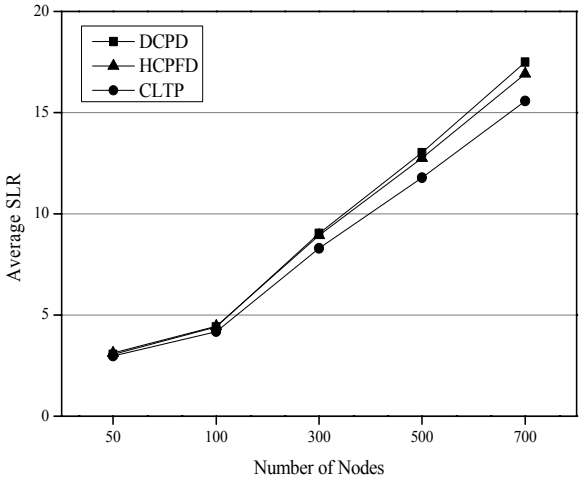

그림 3. 노드의 수 변화에 따른 평균 SLR
Fig. 3. An average SLR with respect to the number of nodes.

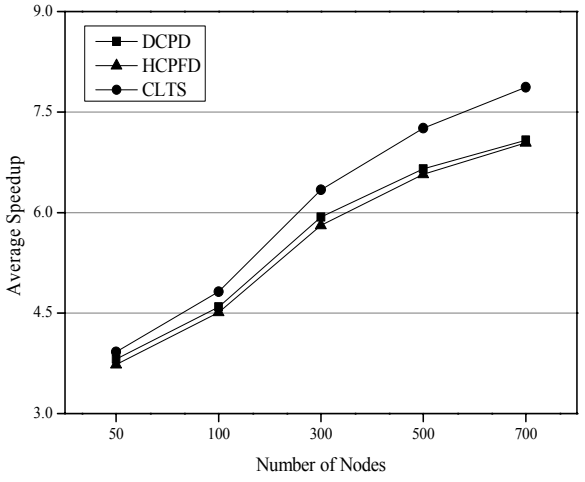

그림 4. 노드의 수 변화에 따른 평균 Speedup
Fig. 4. An average speedup with respect to the number of nodes.

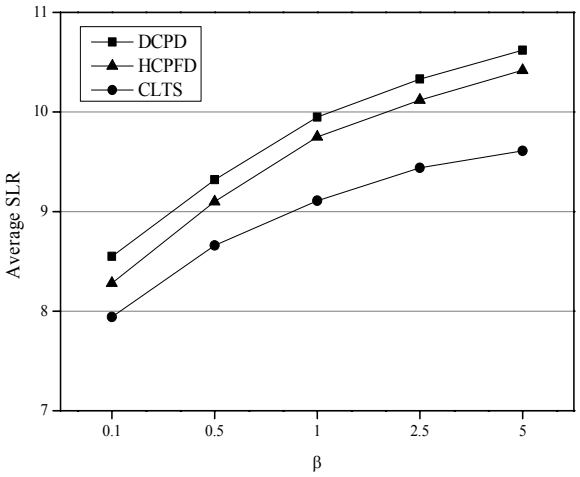

그림 5. β 변화에 따른 평균 SLR
Fig. 5. An average SLR with respect to β.

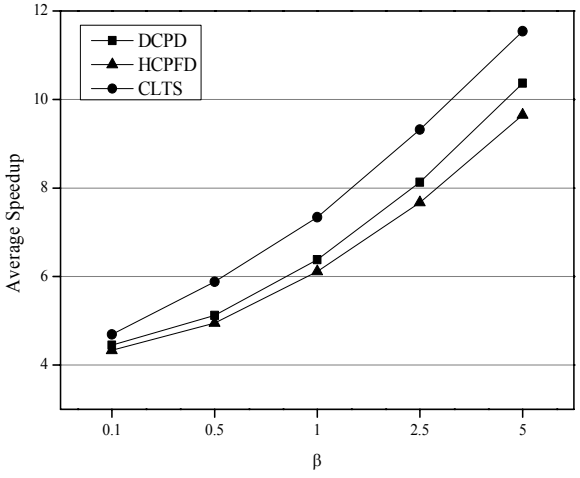

그림 6. β 변화에 따른 평균 Speedup
Fig. 6. An average speedup with respect to β.

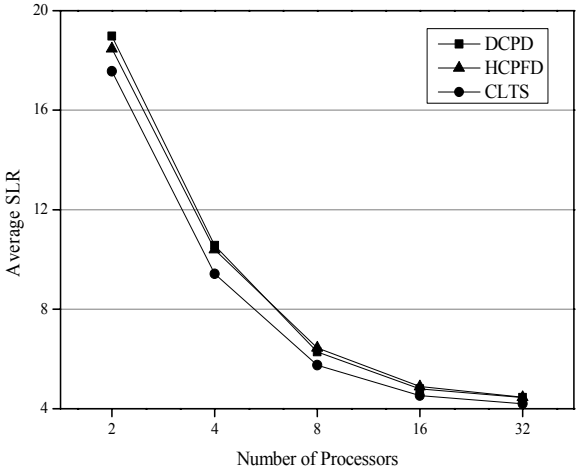

그림 7. 프로세서 수의 변화에 따른 평균 SLR
Fig. 7. An average SLR with respect to the number of processors.

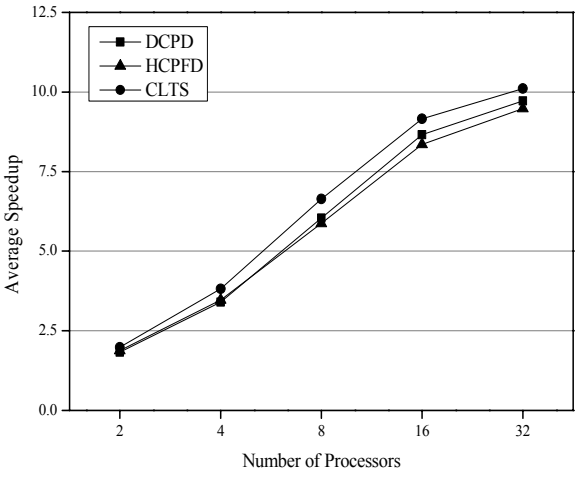

그림 8. 프로세서 수의 변화에 따른 평균 Speedup
Fig. 8. An average speedup with respect to the number of processors.

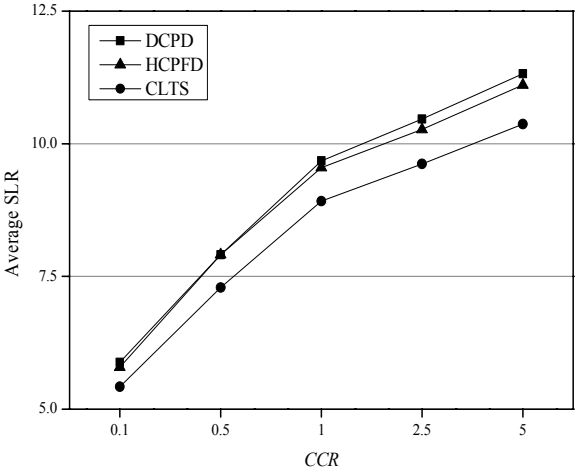

그림 9. CCR의 변화에 따른 평균 SLR
Fig. 9. An average SLR with respect CCR.

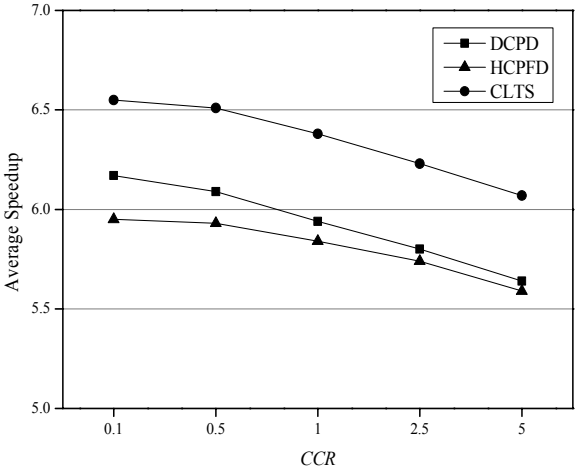

그림 10. CCR의 변화에 따른 평균 Speedup
Fig. 10. An average Speedup with respect to CCR.

할 수 있다.

그림 5와 그림 6은 β의 변화에 따른 평균 SLR과 평균 Speedup을 나타낸다. 평균 SLR은 HCPFD 대비 6.11%, DCPD 대비 8.22%, 향상되었고 평균 Speedup은 HCPFD 대비 7.21%, DCPD 대비 9.66% 향상되었음을 확인하였다.

그림 7과 그림 8은 프로세서 수의 변화에 따른 평균 SLR과 평균 Speedup을 나타낸다. 평균 SLR은 HCPFD 대비 7.14%, DCPD 대비 8.01%, 향상되었고 평균 Speedup은 HCPFD 대비 9.15%, DCPD 대비 6.94% 향상되었음을 확인하였다.

그림 9과 그림 10은 *CCR*의 변화에 따른 평균 SLR과 평균 Speedup을 나타낸다. 평균 SLR은 HCPFD 대비 6.74%, DCPD 대비 8.04%, 향상되었고 평균 Speedup은 HCPFD 대비 8.88%, DCPD 대비 6.71% 향상되었음을 확인하였다. 특히 CLTS는 노드의 복제를 임계노드와 임계노드의 부모노드로 제한함에도 불구하고 타 알고리즘보다 우수한 성능을 보인다. 위 결과를 통해 본 논문에 제안한 CLTS이 모든 경우에 대해 타 알고리즘 보다 우수한 성능을 보이며 평균 SLR은 HCPFD 대비 7.29%, DCPD 대비 8.93%, 향상되었고 평균 Speedup은 HCPFD 대비 9.21%, DCPD 대비 7.66% 향상됨을 확인하였다.

Ⅵ. 결 론

본 논문에서는 임계노드를 고려한 리스트 스케줄링 알고리즘인 CLTS를 제안하였다. CLTS는 우선순위 결정 과정에서 계층화를 통해 태스크 처리의 병렬성을 증가시키고, 다음 계층에 있는 임계노드를 고려하여 임계노드 계산지연에 의한 전체 처리시간의 지연을 감소시킬 수 있도록 노드의 처리 우선순서를 결정하였다. 또 프로세서 할당 단계에서는 임계노드와의 관계를 고려하여 복제 기반 정책과 삽입 기반 정책을 조건적으로 수행함으로써 전체 처리 시간을 단축시키면서도 노드의 무분별한 복제 기반 정책의 고려를 제한할 수 있다.

제안한 알고리즘의 성능평가를 위해 다양한 입력 그래프와 매개변수를 이용하여 시뮬레이션을 진행하였다. 공정한 성능 평가를 위해 표준 입력 그래프를 프로세서의 이질성도 척도, CCR, 노드의 수 등의 매개변수를 조합하여 총 106,250 개의 입력 그래프를 생성한 뒤 사용하였다. 시뮬레이션을 통해 CLTS가 기존 알고리즘인 HCPFD와 DCPD 대비 우수한 성능을 보임을 확인하였다.

REFERENCES

[1] M. Maheswaran, T. D. Braun, and H. J. Siegel, “Heterogeneous distributed computing,” *Encyclopedia of Electrical and Electronics Engineering*, Wiley, New York, Dec. 1999.

[2] H. Togcuglous, S. Hariri, and M. Y. Wu, “Performance Effective and Low-Complexity Task Scheduling for Heterogeneous Computing,” *IEEE Trans. On Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, Mar. 2002.

[3] T. Hagras and J. Janecek, “A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems,” *Journal of Parallel Computing*, vol. 31, no. 7, pp. 653-670, Jul. 2005.

[4] C. Liu and C. Li, “Dynamic Critical Path Duplication Task Scheduling Algorithm for Distributed Heterogeneous Computing Systems,” *Proceedings of the 12th International Conference on Parallel and Distributed Systems (ICPADS'06)*, pp. 28-38, Minnesota, USA, Jul. 2005.

[5] E. Ilavarasan, P. Thambidurai, and R. Mahilmannan, “Performance Effective Task Scheduling Algorithm for heterogeneous Computing System,” *IEEE Proceedings of the 4th International Symposium on Parallel and Distributed Computing (ISPDC'05)*, pp. 28-38, Jul. 2005.

[6] W. Yoon, J. Yoon, J. Yoon, and S. Choi, “A Novel High Performance List Scheduling Algorithm for Distributed Heterogeneous Computing Systems Computing Systems,” *Journal of the Institute of Electronics and Information Engineers*, vol. 47, no. 1, pp. 135-145, Jan. 2010.

[7] M. Kafil and I. Ahmed, “Optimal task assignment in heteroheneous distributed computing systems,” *IEEE Concurrency*, vol. 6, no. 3, pp. 42-51, Jul. 1998.

[8] A. Ranaweera and D. P. Agrawal, “A task duplication based scheduling for heterogeneous system,” *Proc. IPDPS*, pp. 445-450, May. 2000.

[9] C. Boeres, J. V. Filho, and V. E. F. Rebello, “A

cluster-based strategy for scheduling task on heterogeneous processors," *Proc. 16th Symp. on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 214-221, Oct. 2004.

저 자 소 개

김 호 중(정회원)
2011년 인하대학교 전자공학과 학사 졸업.
2015년 인하대학교 전자공학과 석사 졸업.
2015년~현재 전자부품연구원 연구원.
<주관심분야 : 병렬 및 분산 컴퓨팅, 컴퓨터 구조>

송 인 성(정회원)
2009년 인하대학교 전자공학과 학사 졸업.
2011년 인하대학교 전자공학과 석사 졸업.
2011년~현재 인하대학교 전자공학과 박사과정.
<주관심분야 : 병렬 및 분산처리 시스템, 컴퓨터 구조, ADS-B>

정 용 수(정회원)
2013년 인하대학교 전자공학과 학사 졸업.
2015년~현재 인하대학교 전자공학과 석사과정.
<주관심분야 : 컴퓨터 구조, 메모리 시스템>

최 상 방(평생회원)
1981년 한양대학교 전자공학과 학사 졸업.
1981년~1986년 LG 정보통신(주).
1988년 University of washinton 석사 졸업.
1990년 University of washinton 박사 졸업.
1991년~현재 인하대학교 전자공학과 교수
<주관심분야 : 컴퓨터 구조, 컴퓨터 네트워크, 무선 통신, 병렬 및 분산 처리 시스템>