이질 시스템에서 통신 시간을 고려한 복제 기반 태스크 스케줄링

백정규[0] 정진하 윤완오 신광식 최상방
인하대학교 전자공학과
sangbang@inha.ac.kr

Duplication Based Task Scheduling with Communication Cost in Heterogeneous Systems

Jungkyu Baek[0], Jinha Cheong, Wanoh Yoon, Kwangsik Shin, and Sangbang Choi
Department of Electronic Engineering, Inha University

요 약

병렬 및 분산 컴퓨터 시스템에서 선후 관계의 제약을 갖는 노드들의 스케줄링은 잘 알려진 NP-complete이다. 이러한 노드들의 스케줄링을 효율적으로 수행하기 위해 많은 알고리즘이 부모 노드와 이질 프로세서에 대한 정보를 고려하여 제안되었다. 하지만 여러 개의 부모 노드와 이질 프로세서에 대한 다양한 경우를 충분히 고려 하지 못했다. 본 논문은 부모 노드에 대한 선후 관계와 이질 시스템의 특성을 고려, 이질 수행 시간을 갖는 다중 프로세서를 대상으로 태스크가 가능한 빨리 수행할 수 있는 시간과 태스크가 가능한 빨리 완료될 수 있는 시간을 이용한 복제 기반의 태스크 스케줄링 기법(DTSC)을 제안하였다. 제안된 알고리즘의 성능은 기존 STDS 알고리즘과 대표적인 입력 그래프에 대해 비교하였고, 스케줄링의 성능 향상을 보여 주었다

1. 서 론

태스크의 수행을 빠르게 처리하기 위해 사용되는 병렬 처리 및 분산 시스템에서는 노드에 대한 정보를 교환해야 하므로 시스템간의 통신 시간은 중요한 요소이다. 하지만 시스템 및 프로세스의 성능 향상에 비해 통신 시간의 감소는 이루어지지 못했다. 통신 시간을 감소시키기 위해서 선후 관계를 갖는 노드를 동일 프로세서에 할당 하는 방법과 부모 노드의 복제를 이용한 많은 알고리즘이 제안되었다. 또한 서로 다른 시스템 환경에 인한 작업 수행 시간의 차이 역시 병렬 처리 및 분산 시스템의 중요한 요소이다. 이러한 특성을 가지는 시스템을 이질 시스템이라 하며, 이질 시스템에서는 통신 시간과 작업은 태스크 스케줄링의 중요한 요소이다. 스케줄링의 목적은 입력 그래프에 대한 스케줄링 결과 길이를 최소화하는 것이다. 최적의 스케줄링은 병렬 처리 및 분산 시스템의 성능 향상을 가져다 주지만, 이런 스케줄링 문제는 NP-complete로써 최적의 스케줄링 결과를 얻기 위해서는 모든 경우를 고려해 보아야 한다 [1]

본 논문에서 제안한 알고리즘은 첫 번째 단계에서는 상호 의존성이 있는 노드 간의 통신 시간을 고려하여 개선된 est (earliest start time), ect(earliest completion time)를 바탕으로 클러스터(cluster)를 구성한다. 마지막 단계에서 구성된 클러스터의 전체 수행 시간 합이 최소가 되는 프로세서를 기본으로 각 노드의 ect가 최소가 되는 프로세서를 선택함으로서 스케줄링을 수행한다.

2. 기존 스케줄링 알고리즘

태스크 스케줄링 방법에는 각각의 노드에 우선순위를 부여하고 이를 바탕으로 리스트를 작성하여 스케줄링을 수행하는 리스트 스케줄링과 노드들 간의 상호 의존성에 의한 통신 시간을 감소 시키기 위해 선행 노드의 복제를 이용한 태스크 복제 스케줄링, 마지막으로 주어진 그래프에서 노드 간의 선후관계를 바탕으로 클러스터를 구성 후 스케줄링을 수 행하는 클러스터 스케줄링이 있다. 그 중에서 노드 복제를 통한 클러스터 구성 방식의 대표적인 STDS 알고리즘이 [2] 있다

STDS 알고리즘은 4단계로 이루어져 있다. 첫 번째 단계는 입력 그래프에 대해 est(earliest start time, ect(earliest completion time), fpred(favorite predecessor), fproces (favorite processor), level, queue를 top-down 방식으로 STDS에서 사용한 식을 이용해 계산한다 [2]. 두 번째 단계는 last(latest start time), lact(latest completion time)를 bottom-up 방식으로 계산하고, 세 번째 단계는 앞에서 구한 결과값을 이용해 노드 간의 선후 관계를 바탕으로 클러스터를 구성한다. 구성된 클러스터는 클러스터의 첫 번째 노드의 최적 프로세서에 할당한다. 마지막 단계에서 노드 간의 메시지 스케줄링을 수행하면 스케줄링이 완성된다.

STDS 스케줄링의 est와 ect는 자신의 노드와 부모 노드 사이의 통신 시간에 대한 고려가 부족하다. 또한 여러 개의 부모 노드가 있을 경우 주어신 식으로 정확한 est와 ect를 구해낼 수 없다 [2].

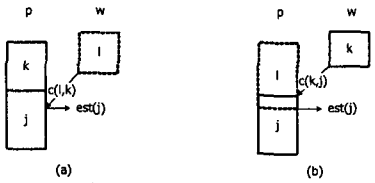


그림 1. 부모 노드의 다양한 경우를 고려하지 못한 STDS 스케줄링의 est(j), (a) fproc(k)=p인 경우, (b) fproc(k)≠p인 경우.

그림 1 (a)에서 볼 수 있듯이 fproc(k)=p를 만족한다고 해서 언제나 est(j/p)가 ect(k)와 동일하지 않는다. 프로세서 w에 있는 다른 부모 노드 l의 ect(l)+c(l,j)가 ect(k)보다 클 수 있기 때문이다. 그림 1(b) 역시 fproc(k)≠p를 만족해도 언제나 est(j/p)

가 ect(k) +c(k,j)와 동일하지 않는다. 프로세서 p에 ect(k)+c(k,j) 보다 ect(l)이 더 큰 부모 노드가 존재 할 수 있기 때문이다. STDS에서 ect(j)는 est(j)+τ (j,fproc(j))와 같이 정의한다. 이질 시스템에서는 노드의 수행 시간이 프로세서 마다 다르다. 따라서 단순히 est에 수행 시간을 더해 ect를 구하는 것은 적절한 계산이 아니다. 이질 시스템이므로 노드 j가 자신의 최적 프로세서(fproc(j))에서 통신 시간을 고려해도 더 빨리 완료 할 수 있기 때문이다. 따라서 그림 2 (c), (d)를 비교해 최소 값을 ect로 선택해야 한다 클러스터 구성 시 최적 부모 노드가 이미 할당 되었다면 선임 노드 중에서 다음 fpred를 선택하는 방법이 명확하지 못하다

마지막으로 구성된 클러스터를 프로세서에 할당방법은 시작 노드의 최적 프로세서(fproc)를 선택하는 것이다. 하지만 시작 노드의 최적 프로세서가 클러스터의 최적 프로세서라는 명확한 근거가 없다. 따라서 이런 점들을 보안하고 부모 노드와의 선후 관계를 충분히 고려하는 새로운 스케줄링 기법이 필요하다

3. 제안된 알고리즘

본 논문에서 제안한 DTSC(Duplication based Task Scheduling with Communication cost) 알고리즘은 4단계로 구성되어 있다. 첫 번째 단계는 top-down 방식으로 est, ect, fpred, fp를 입력 그래프를 바탕으로 계산한다. 두 번째 단계는 구해진 fpred 정보를 고려해 클러스터를 구성한다. 세 번째 단계는 구성된 클러스터를 최적 프로세서에 할당한다. 마지막으로 노드간의 메시지 스케줄링을 수행하면 최종 스케줄링이 완성된다

병렬 처리 및 분산 시스템에서 태스크의 선후 관계는 DAG (Directed Acyclic Graph) [2],[3]로 나타낼 수 있다. DAG는 (V, E, W, CPC, CMC)로 표현되며, V는 모든 노드를 나타내고, E는 프로세서간의 통신을 나타낸다. W는 사용 가능한 프로세서를 나타내고, CPC(Completion cost)는 CPC(j,w)로서 w(w∈W)에서 j(j∈V) 노드의 처리 시간을 나타낸다. 마지막으로 CMC(Communication cost)는 CMC(j,k)로서 노드 j와 k 사이의 통신 시간을 의미한다. (j,k∈V)

3.1 1개의 부모 노드만 있는 경우

그림 2 (a) 처럼 오직 한 개의 부모 노드가 있는 경우는 부모 노드의 프로세서 선택은 ect(k/w)와 ect(k/p)+CMC(k,j)의 두 가지 경우가 있다 이 중 ect가 최소가 되는 경우를 선택한다. 첫 번째 경우라고 가정을 하고 est를 계산한다

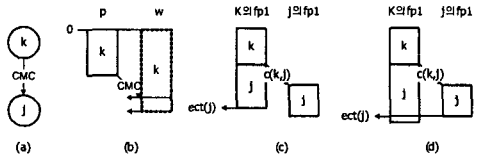


그림 2. (a) 1개의 부모 노드만 있는 경우, (b) 부모 노드 k의 2가지 할당 방법, (c) ect(j/k의fp1) < ect(k/k의fp1)+CMC(k,j)+CPC(j,j의fp1) 일때 ect(j), (d) ect(j/k의fp1) > ect(k/k의fp1)+CMC(k,j)+CPC(j,j의fp1) 일때 ect(j)

오직 1개의 부모 노드만 있는 경우 fpred, est는 부모 노드가 복제되므로 통신 시간(CMC)이 필요하지 않다.

$$fpred(j) = \{Max[ect(k)]\}\text{인 노드} \quad (1)$$

$$est(j) = 2^{nd}\,Max[ect(k)], \quad (2)$$

$2^{nd}\,Max$는 두 번째로 큰 값을 의미한다.

$$est(j)_{Max} = Max[ect(k)] = est(j) \quad (3)$$

그림 2 (c), (d) 경우 ect를 구하기 위해서는 통신 시간을 고려해야 한다 $fpred(j)_1$는 노드 j에 가장 늦게 정보를 보내주는 부모 노드를 뜻한다. 선임 노드에 정보를 늦게 보내는 순서로 $fpred(j)_1$, $fpred(j)_2$, $fpred(j)_3$, ...를 결정하면 된다

est 역시 한 개의 부모 노드만이 존재하므로 부모 노드의 ect가 된다 ect는 부모 노드와 동일 프로세스에 있을 경우, 현재 노드가 자신의 최적 프로세서(fp)에 할당되는 경우를 비교해 최소값을 선택한다. 현재 노드가 최적 프로세서 할당되기 위해서는 통신 시간이 필요하므로 CMC(k,j)가 추가되었다

$$ect(j) = Min[(est(j) + CPC(j,k\text{의}fp1)),\ (est(j)_{Max} + CMC(k,j) + CPC(j,j\text{의}fp1))] \quad (4)$$

이질 시스템이므로 통신 시간(CMC)을 포함하고 현재 노드의 fp1에서의 ect가 더 작은 경우도 있기 때문이다. fp의 순위는 수행 시간(CPC)이 가장 짧은 순서다. 수행 시간이 동일할 경우는 주어진 프로세서의 순서를 따른다.

3.2 2개 이상의 부모 노드가 있는 경우

그림 3처럼 2개 이상의 부모 노드가 있는 경우 부모 노드의 ect와 통신 시간(CMC)를 더한 값 중 두 번째로 큰 값을 현재 노드 j의 est(j)로 선택하고, 최대값을 갖는 노드가 $fpred(j)_1$가 된다. 현재 노드는 부모 노드에 대해 $fpred(j)_1$, $fpred(j)_2$, … 순위를 부여한다

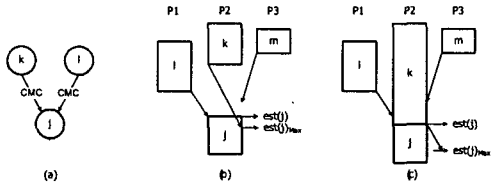


그림 3. 2개 이상의 부모 노드를 갖는 노드 j의 est(j) 계산방법, (a) 2개 이상의 부모 노드를 가진 경우, (b) ect(l)+CMC(l,j) > ect(k) 인 경우, (c) est(l)+CMC(l,j) < ect(k) 인 경우

$$fpred(j) = \{Max[(ect(k) + CMC(k,j)),\ (ect(l) + CMC(l,j)),...]\}\text{인 노드} \quad (5)$$

$$est(j) = Max\{2^{nd}\,Max[(ect(k) + CMC(k,j)),\ (ect(l) + CMC(l,j)),...], ect(fpred(j))\} \quad (6)$$

$$est(j)_{Max} = Max[(ect(k) + CMC(k,j)),\ (ect(l) + CMC(l,j)),...] \quad (7)$$

est(j)에서 마지막 ect(fpred(j))는 그림 3의 (c)경우를 고려한 것이다. ect는 1개의 부모 노드만 있을 때(4)와 동일하지만 부모 노드가 동일 프로세서에 있지 않으므로 통신 시간을 포함한 est와 est_{Max}를 사용하게 된다.

$$ect(j) = Min[(est(j) + CPC(j,k\text{의}fp1)),\ (est(j)_{Max} + CPC(j,j\text{의}fp1)] \quad (8)$$

fp의 순위는 수행 시간(CPC)이 가장 짧은 순서다. 수행 시간이 동일할 경우는 주어진 프로세서의 순서를 따른다

3.3 최적 프로세서 선택 방법

Bottom-up 방식으로 계산된 각 노드의 fpred 정보를 이용해 클러스터를 구성한다. 만약 첫 번째 클러스터 구성시 $fpred(j)_1$을 이미 할당 했다면 두 번째 클러스터는 $fpred(j)_2$를 선택하여 구성하면 된다. 포크 노드(fork node)의 경우는 복제를 통해 각각의 클러스터에 할당될 수 있다 [3-5].

구성된 클러스터를 할당했을 때 가장 짧은 수행 시간을 갖는

프로세서(최적 프로세서)에 할당하는 방법은 두 가지 이다. 첫 번째 방법은 각각의 노드 사이에서 통신 시간(CMC)이 수행 시간(CPC) 보다 큰 경우, 통신 시간에 의한 오버헤드를 제거 하기 위해 같은 프로세서 할당하는데 최적 프로세서를 찾는 방법이고, 두 번째 방법은 각각의 노드 사이에서 통신 시간(CMC)이 수행 시간(CPC) 보다 적어서 최소의 ect를 위해 다음 노드가 자신의 fp1에 할당하는 경우를 고려한 것이다.

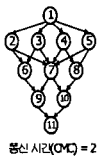
방법1 : 클러스터에 포함된 모든 노드의 CPC 합이 최소가 되는 프로세서를 찾는다 첫 번째 클러스터가 프로세서1을 선택했다면, 두 번째 클러스터는 프로세서1을 제외하고 CPC 합을 계산한다. 만약 CPC의 합이 동일할 경우는 클러스터의 처음 노드의 CPC가 최소가 되는 프로세서를 선택한다 1st 노드가 동일하면 순차적으로 2nd, 3rd 노드 순으로 비교 후 프로세서를 선택한다.

방법2 : 방법1에서 선택된 프로세서를 바탕으로 클러스터를 프로세서에 할당 한다.

- **ect(k)+CPC(j,k의프로세서)≤ ect(k)+CMC(k,j)+CPC(j,j의fp1)**
  j역시 k의 프로세서에 할당한다. 여기서 k의 프로세서는 방법1에서 선택된 프로세서 일수도 있고, k의 fp1일수도 있다
- **ect(k)+CPC(j,k의프로세서)>ect(k)+CMC(k,j)+CPC(j,j의fp1)**
  j는 j의 fp1에 할당한다.

4. 스케줄링 성능 분석

STDS에서 제시한 DAG를 바탕으로 두 스케줄링의 수행 결과를 비교해 보겠다. 그림 4는 STDS [2]에서 사용한 DAG와 수행 시간 표이다. STDS의 스케줄링 기법 [5]의 수행 결과 길이는 4개의 프로세서를 사용하여 26의 결과 길이를 보여준다. 또한 그림 5 (a)에 볼 수 있듯이 결과 길이를 감소 시키기 위해 2개의 여유 프로세서를 이용해 노드 10의 $fpred(10)_1$를 복제함으로써 5개의 프로세서를 이용해 25의 결과 길이를 얻어 내었다



Node	P1	P2	P3	P4	P5	P6
1	3	3	4	2	5	4
2	5	6	6	5	5	4
3	2	3	4	2	4	3
4	5	3	5	3	5	3
5	7	7	6	5	5	6
6	3	4	2	3	2	4
7	6	7	7	5	7	5
8	5	7	5	6	6	5
9	7	7	5	6	6	5
10	2	3	2	2	3	2
11	3	2	3	3	2	3

(a) (b)

node	est	ect	fpred				fp(processor)					
			fpred1	fpred2	fpred3	fpred4	fp1	fp2	fp3	fp4	fp5	fp6
1	0	2	0	0	0	0	P4	P1	P2	P3	P6	P5
2	2	2	1	0	0	0	P6	P1	P4	P5	P2	P3
3	2	2	1	0	0	0	P1	P4	P2	P6	P3	P5
4	2	5	1	0	0	0	P2	P4	P6	P1	P3	P5
5	2	7	1	0	0	0	P4	P5	P3	P6	P1	P2
6	6	10	2	3	0	0	P3	P5	P1	P4	P2	P6
7	9	14	5	2	4	3	P4	P6	P1	P2	P3	P5
8	7	13	5	4	0	0	P1	P3	P6	P4	P5	P2
9	12	18	7	6	0	0	P3	P6	P4	P5	P1	P2
10	15	17	7	8	0	0	P1	P3	P4	P6	P2	P5
11	19	22	9	10	0	0	P2	P5	P1	P3	P4	P6

(c)

그림 4. (a) 입력 DAG, (b) DAG를 위한 노드 수행시간, (c) DAG를 위한 DTSC의 계산값

동일한 DAG를 이용해 본 논문에서 제안한 DTSC 스케줄링 기법을 사용하여 얻어낸 결과 길이는 그림 5 (b)와 같다. 개선된 est, ect, fpred, fp의 계산값은 그림 4 (c)와 같다. 그림 4 (c)를 바탕으로 클러스터를 구성한다. Bottom-up 방식으로 노드 11의 $fpred(11)_1$이 노드 9이므로 노드 9를 선택하고, 노드 9의 $fpred(9)_1$이 노드 7이므로 노드 7을 선택한다. 노드 7의 $fpred(7)_1$이 노드 5이므로 노드 5를 선택, 마지막으로 노드 5의 $fpred(5)_1$은 노드 1이므로 노드 1을 선택하면 첫 번째 클러스터 11→9→7→5→1 구성된다. 두 번째 클러스터는 아직 할당 되지 안은 노드 중에서 가장 마지막 노드인 노드 10을 이용해 구성한다.

노드 10의 $fpred(10)_1$이 노드 7이다. 하지만 노드 7은 이미 할당 되었으므로 $fpred(10)_2$를 선택하여 클러스터를 구성한다. 이과 같은 방법으로 모든 노드를 클러스터로 구성하면 11→9→7→5→1, 10→8→4→1, 6→2→1, 3→1 네 개의 클러스터를 구성 할 수 있다

최적 프로세서를 찾기 위해 방법1을 사용하면 첫 번째 클러스터 는 프로세서 4에서 11→9→7→5→1 의 수행 시간을 순서대로 더하면 2+5+5+6+3=21인 가장 작은 수행 시간 합을 갖는다. 따라서 첫 번째 클러스터는 프로세서 4를 선택한다. 프로세서 4가 첫 번째 클러스터에 사용되었으므로 프로세서 4를 제외한 나머지 프로세서에 대해 두 번째 클러스터의 수행 시간 합을 구한다. 두 번째 클러스터의 수행 시간 합이 최소가 되는 프로세스는 프로세스 6이다. 이와 같이 각각의 클러스터에 대한 최적 프로세서를 구하면, 세 번째 클러스터는 프로세서 1에, 네 번째 클러스터는 프로세서 2에 할당된다. 방법1로 선택된 프로세서를 바탕으로 실제 클러스터의 각 노드를 할당하기 위해서 방법2를 사용한다. 통신 시간(CMC)를 감소시키기 위해 같은 프로세서에 할당하지만 이질 시스템 환경이므로 ect는 그림 2 (d)처럼 감소될 수 있다. 따라서 자신의 fp1에서의 ect를 고려해야 하기 때문에 방법2를 사용한다. 주어진 DAG에서는 방법2에 해당하는 경우가 발생하지 않았다. 메시지 스케줄링을 수행하면 그림 5 (b) 결과를 얻을 수 있다

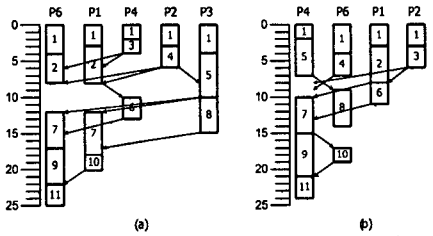


그림 5. 주어진 DAG(그림 4(a))에 대한 스케줄링 결과, (a) STDS 스케줄링 결과 25, (b) DTSC 스케줄링 결과 24.

5. 결론

본 논문은 이질 시스템에서 프로세서간의 통신 시간 및 노드들 간의 선후 관계, 이질 시스템의 특성을 고려한 DTSC 스케줄링을 제안하였다. 제안된 스케줄링 방법은 입력 그래프가 직선인 경우는 클러스터를 프로세서에 할당하는 방법2에 의해 최적의 스케줄링 결과를 보여주었다. 또한 주어진 DAG 상에서도 향상된 스케줄링 결과를 보여주었다.

DTSC 스케줄링 기법은 클러스터 방식의 STDS 스케줄링 기법 [2] 보다 결과 길이를 최소화 하기 위해 부모 노드 복제에 필요한 여분의 프로세서를 사용하지 않고, 즉 동일한 수의 프로세서만을 사용해서 더 짧은 결과 길이를 보여주었다.

참고문헌

[1] T. Cassavant and J.A. Kuhl, " Taxonomy of Scheduling in General Purpose Distributed Memory Systems," IEEE Trans. Software Eng.,vol. 14, no. 2, pp. 141-154, 1988

[2] A. Ranaweera and D.P. Agrawal, " A Scalable Task Duplication Based Scheduling Algorithm for Heterogeneous Systems," Proc. Int' l Conf. Parallel Processing, pp. 383-390, Aug. 2000

[3] Rashmi Bajaj and Dharma P. Agrawal, Fellow, " Improving Scheduling of Tasks in a Heterogeneous Environment," IEEE Trans. Parallel and Distributed Systems, vol. 15, no. 2, pp. 107-118, Feb. 2004.

[4] I. Ahmad and Y.K. Kwok, " On Exploiting Task Duplication in Parallel Program Scheduling," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 9, pp. 872-892, Sept. 1998.

[5] C.I. Park and T.Y. Choe, " An Optimal Scheduling Algorithm Based on Task Duplication," *IEEE Trans. Computers*, vol. 51, no. 4, pp. 444-448, Apr. 2002.