워크스테이션 네트웍에서의 휴리스틱 태스크 스케줄링 알고리즘

(A Heuristic Task Scheduling Algorithm in Workstation Networks)

강 오 한 †

(Oh-Han Kang)

요 약 본 논문에서는 워크스테이션 네트웍 (Network of Workstations) 환경에서 태스크 스케줄링 문제를 해결하기 위하여 태스크 중복을 기반으로 하는 휴리스틱 스케줄링 알고리즘을 제안한다. 제안된 알고리즘에서는 NoW에서 통신할 때 발생되는 충돌을 방지하기 위하여 네트웍 통신 자원을 우선 할당하고, 스케줄링 길이를 단축하고 병렬처리 시간을 줄이기 위한 중복 태스크를 선택할 때 휴리스틱을 사용한다. 제안된 알고리즘은 태스크 그래프를 입력으로 받아 NoW 환경의 워크스테이션으로 스케줄링하며, 태스크 그래프에서 노드수가 V일 때 최악의 경우 알고리즘의 시간 복잡도는 $O(V^2)$이다. 제안한 알고리즘을 실제 응용 프로그램의 태스크 그래프에 적용하였다. 시뮬레이션을 통하여 제안된 알고리즘이 스케줄링 길이와 알고리즘에서 요구하는 워크스테이션의 수 관점에서 성능이 향상되었음을 보여준다.

Abstract In this paper, a task duplication based heuristic scheduling algorithm is proposed to solve the problem of task scheduling on network of workstations (NoW). The proposed algorithm pre-allocates network resources so as to avoid potential communication conflict, and the algorithm uses heuristics to select duplication tasks so as to reduce schedule length and parallel processing time. The algorithm schedules the task of a task graph on to the workstations of a NoW, and worst case time complexity is $O(V^2)$, where V is the number of nodes in a task graph. The proposed algorithm has been applied to some practical application DAGs. Simulation results show that the proposed algorithm achieves performance improvement in respect of schedule length and number of workstations required by the algorithm.

1. 서 론

병렬 분산 환경에서 시스템의 성능을 향상시키기 위하여 효과적인 태스크 분할과 스케줄링 알고리즘의 개발이 중요한 연구 분야가 되어왔다. 태스크 분할 알고리즘은 응용 프로그램을 태스크로 분할하고 태스크들을 태스크 그래프 형태로 나타낸다. 태스크 스케줄링 알고리즘은 입력된 태스크 중에서 다음에 처리할 태스크를 선택하는 방법이며, 태스크를 프로세서로 할당한다. 병렬 프로그램의 태스크를 효과적으로 스케줄링 함으로서 시스템 자원의 활용도를 높이고 프로그램의 병렬처리 시간을 단축할 수 있다 [1, 2]. 다중 프로세서(multiprocessor) 시스템에서 태스크 스케줄링은 NP-complete 문제여서 [3] 현재까지 휴리스틱을 기반으로 다양한 알고리즘들이 발표되었다 [4-7]. 이들과는 다른 접근 방법으로 다중 프로세서 시스템에서 태스크 중복(duplication)을 기반으로 다양한 스케줄링 알고리즘이 최근에 제안되었다 [8-12]. 태스크 중복을 기반으로 하는 스케줄링 알고리즘은 태스크를 다수의 클러스터(cluster)로 나누고 각 클러스터를 프로세서에 할당한다. 또한 각 클러스터에 태스크를 중복하여 할당함으로써 통신을 위한 비용을 절감할 수 있다. 현재까지 제안된 대부분의 스케줄링 알고리즘들은 완전연결(fully-connected) 네트웍을 가정하였으며, 이러한 환경에서 각

· 본 연구는 한국과학재단 목적기초연구(2000-1-30300-004-3) 지원으로 수행되었음

† 종신회원 : 안동대학교 컴퓨터공학교육과 교수
ohkang@andong.ac.kr

논문접수 : 1999년 10월 8일
심사완료 : 2000년 8월 28일

프로세서는 다른 프로세서들과 직접 통신을 할 수 있다.

하드웨어 기술 발전에 따른 고성능 저가의 워크스테이션이 개발되고 통신 기술의 발전에 따른 네트웍의 보급이 확산되면서 워크스테이션 네트웍 (NoW, network of workstations)에 관한 관심이 높아지고 있다. NoW 환경은 경제성과 신축성(flexibility) 측면에서 다중 프로세서보다 우수하여 병렬 연산을 위한 기반으로 개발되고 있다. NoW의 확장성(scalability)은 다중 프로세서 시스템과 비교할 때 가장 중요한 장점이며, 다양한 하드웨어 기술과 프로토콜의 개발로 실용성이 증대되고 있다 [13]. 현재까지 NoW 환경에서 병렬 연산을 위한 다양한 네트웍 유형(topology)과 프로토콜이 개발되고 사용되지만 버스(bus) 기반의 네트웍 유형이 가장 경제적이고 광범위하게 사용되고 있다.

다중 프로세서 시스템과 비교할 때 NoW 환경의 중요한 한계중의 하나가 프로세서 사이의 통신을 위한 비용(cost)이 높다는 것이다. NoW 환경에서 병렬 연산을 위한 효과적인 태스크 분할과 스케줄링 알고리즘을 사용함으로서 이러한 문제점을 줄일 수 있다. 버스 기반의 NoW 환경에서 태스크 스케줄링은 완전연결 네트웍에서의 스케줄링과 차이가 있다. 이 환경에서는 네트웍의 통신 자원을 공유할 수 없으므로 서로 독립된 두 개의 통신 태스크가 동시에 수행될 수 없다. NoW 환경에서 동적 결정 경로(dynamic critical path)를 기반으로 하는 휴리스틱 스케줄링 알고리즘이 [14] 연구에서 제안되었다. 이 알고리즘은 다중 프로세서 환경을 기반으로 제안된 DCP [6] 알고리즘을 NoW 환경에 맞도록 수정한 것으로 스케줄링을 위한 후보 태스크를 선택할 때 동적 결정경로를 기반으로 한다.

NoW 환경에서 평균 50% 이상의 워크스테이션이 가용(idle) 상태에 있으며 충분한 잠재적 연산 능력을 가지고 있다 [15]. 따라서 태스크 중복을 기반으로 하는 스케줄링 알고리즘은 NoW 환경에서 효과적인 기법이 될 수 있다. 본 논문에서는 태스크 중복을 기반으로 버스 기반의 NoW 환경에서 적용할 수 있는 휴리스틱 스케줄링 알고리즘을 제안한다. 제안된 태스크 스케줄링 알고리즘은 스케줄링 길이를 줄이고 병렬 처리 시간을 단축하기 위하여 중복할 태스크를 선택할 때 휴리스틱을 사용한다. 또한 알고리즘에서는 태스크들이 통신할 때 발생되는 네트웍 충돌을 방지하기 위하여 스케줄링 단계에서 네트웍 통신 자원을 우선 할당한다.

본 논문의 나머지 부분의 구성은 다음과 같다. 2장에서는 태스크 그래프 모델과 용어를 정의하고 관련 연구를 소개한다. 3장에서는 본 논문에서 제안하는 알고리즘을 설명하고, 예제 태스크 그래프를 사용하여 알고리즘의 동작을 설명한다. 4장에서는 제안한 알고리즘의 성능을 평가하기 위한 시뮬레이션 결과를 보여준다. 5장에서는 본 논문에 대한 결론을 나타낸다.

2. 문제 정의 및 관련 연구

2.1 문제 정의

다중 프로세서 시스템에서 태스크 스케줄링의 주된 목적은 태스크를 서로 다른 프로세서에 할당함으로써 응용 프로그램의 병렬 실행 시간을 단축할 수 있도록 스케줄링 길이를 줄이는 것이다. 이러한 응용 프로그램은 태스크 스케줄링 알고리즘의 입력으로 사용되는 태스크 그래프로 나타낼 수 있다. 태스크 그래프는 V가 태스크 노드이고, E가 통신 링크일 때 튜플(tuple) (V, E, t, c)로 정의할 수 있다. 여기서 집합 t는 연산비용(computation cost)으로 구성되며, 각각의 태스크 $i \in V$는 $t(i)$로 표시하는 연산비용을 갖는다. c는 통신비용(communication cost)의 집합으로 구성되며, 태스크 i에서 태스크 j로 연결되는 링크 $e_{ij} \in E$는 통신비용 c_{ij}를 갖는다. 태스크 그래프에서 링크는 두 태스크 사이의 선행 관계(precedent relation)를 나타낸다. 두 노드 사이에 링크가 존재하면 두 태스크 사이에 데이타 의존(dependency) 관계가 존재한다. 따라서 하나의 태스크는 모든 선행 태스크의 수행이 완료되어야 실행될 수 있으며, 두 태스크가 서로 다른 프로세서에 할당되면 통신비용이 필요하다. 그림 1은 태스크 그래프의 예를 나타낸 것이다.

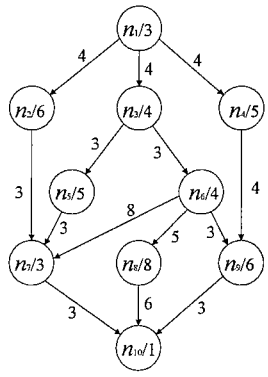


그림 1 태스크 그래프의 예제

태스크 그래프는 하나의 시작(entry) 노드와 하나의 종료(exit) 노드를 갖는다. 하나의 태스크는 독립된 작업

의 단위로 하나의 원으로 나타내며, 원 안에 태스크의 정보를 표시한다. 첫 번째 기호는 노드 번호를 나타내고, 슬래시(/) 문자 다음의 숫자는 태스크의 연산비용을 나타낸다. 태스크 그래프에서 각 링크에는 태스크 사이의 통신비용을 표시한다. 태스크는 하나의 독립된 작업 단위이며 비선점(nonpreemptive)으로 동작한다. 각 프로세서에는 통신 전용 프로세서가 존재하여 연산과 통신을 동시에 처리할 수 있다. 각 태스크가 태스크 그래프의 노드로 나타나므로 본 논문에서는 노드와 태스크를 동일한 의미로 사용한다. 태스크 i에 대한 노드 번호는 n_i로 나타낸다. 아래에서는 스케줄링을 위하여 본 논문에서 사용할 용어들을 정의한다.

정의 1 태스크 i가 실행을 시작할 수 있는 가장 빠른 시간과 실행을 완료할 수 있는 가장 빠른 시간을 각각 $EST(i)$ (Earliest Start Time)과 $ECT(i)$ (Earliest Completion Time)로 정의한다. $ECT(i)$는 $EST(i)$에 태스크 i의 연산비용을 합한 것이다.

정의 2 태스크 i가 지연되어 완료될 수 있는 가장 늦은 시각과 실행을 시작할 수 있는 가장 늦은 시각을 각각 $LCT(i)$ (Latest Completion Time)와 $LST(i)$ (Latest Start Time)로 정의한다. $LST(i)$는 $LCT(i)$에서 태스크 i의 연산비용을 뺀 것이다.

정의 3 스케줄링이 완료되었을 때 태스크 i의 확정된 시작 시간과 종료 시간을 각각 $RST(i)$ (Real Start Time)과 $RCT(i)$ (Real Completion Time)로 정의한다. $RCT(i)$는 $RST(i)$에 태스크 i의 연산비용을 합한 것이다.

정의 4 태스크 그래프에서 모든 태스크의 통신비용의 합한 값을 연산비용의 합한 값으로 나눈 것을 CCR (Communication to Computation Ratio)로 정의한다. 그림 1의 태스크 그래프에 대한 CCR은 56/45 = 1.24가 된다.

정의 5 태스크 i의 모든 부모 노드 j에 대하여 $ECT(j)+c_{ji}$ 값이 최대인 노드를 $CPT(i)$ (Critical Parent)라고 한다. 태스크 i의 $CCPT(i)$ (Critical Parent of CPT)는 $CPT(i)$의 CPT로 정의한다. 그림 1에서 $CPT(7)$은 n_6 노드이며 $CCPT(7)$은 n_3 이다.

정의 6 태스크 그래프에서 노드 i의 레벨(level)은 노드 i에서 종료 노드까지의 연산비용을 합한 값 중에서 최대값으로 정의한다.

정의 7 태스크 그래프에서 결합(join) 노드는 입력 링크가 두 개 이상인 노드이다.

태스크 그래프에서 각 노드에 대한 EST의 계산은 시작 노드부터 종료 노드까지 하향식(top-down) 방식으로 진행된다. 각 노드에 대한 LST의 계산은 종료 노드에서 시작 노드 방향으로 상향식(bottom-up) 방식으로 진행한다. 각 노드에 대한 RST와 RCT는 스케줄링 길이를 계산하는데 사용되며 스케줄링이 완료된 후 각 태스크의 실제 시작 시간과 종료 시간을 나타낸다. CPT는 결합(join) 노드를 스케줄링할 때 결합 노드와 같은 프로세서에 할당할 부모 노드를 선택하기 위하여 사용하는 것으로, 결합 노드와 CPT 노드를 같은 프로세서에 할당함으로써 병렬 시간을 줄이기 위한 것이다. $CCPT$는 현재 스케줄링하고 있는 노드와 CPT 노드의 중복 여부를 결정하기 위한 휴리스틱에서 사용하는 것이다.

2.2 관련 연구

현재까지 다중 프로세서 환경에서 태스크 중복을 기반으로 하는 다양한 스케줄링 알고리즘들이 제안되었다 [8-12]. 이 알고리즘들은 완전 연결 통신망을 기본으로 두 개 이상의 태스크가 동시에 통신할 수 있는 구조를 가정하였다. 따라서 이들 알고리즘들은 네트웍 통신 자원의 충돌로 독립된 두 개 이상의 태스크가 동시에 통신할 수 없는 버스 기반의 NoW 환경에는 적당하지 않다. 그러나 본 논문에서 제안된 알고리즘과 다중 프로세서 시스템을 기반으로 개발된 스케줄링 기법의 차이점을 비교하기 위하여 여기서는 다중 프로세서 시스템에서 적용되는 알고리즘을 간단히 소개한다.

본 절에서는 다중 프로세서 시스템에서 태스크 중복을 기반으로 하는 스케줄링 알고리즘으로 성능이 우수한 STDS [10] 알고리즘을 소개한다. 알고리즘의 첫 번째 단계는 DAG(directed acyclic graph)를 입력으로 받아서 각 태스크의 시작 시간과 완료 시간을 계산한다. 다음 단계에서는 태스크 중복을 기반으로 하는 일반적인 알고리즘과 같이 깊이 우선 탐색을 하여 각 프로세서에 할당할 태스크 클러스터를 생성한다. 태스크 중복을 기반으로 하는 알고리즘들은 결합 노드를 스케줄링하는 방법에서 가장 크게 구분된다. STDS 알고리즘에서는 태스크 클러스터 생성시에 하나의 노드에서 입력 노드까지 경로를 만드는데 반드시 필요한 노드가 아니면 노드를 중복하지 않는다. 또한 이 알고리즘에서는 클러스터 생성시에 "favorate predecessor (*fpred*)"라는 개념을 사용하여 결합 노드와 동일한 프로세서에 할당할 부모 노드를 선택한다. 이 스케줄링 알고리즘에서는 스케줄링 길이를 단축하고 알고리즘의 시간 복잡도를 줄이기 위하여 하나의 클러스터에 할당된 *fpred* 태스크는 클러스터 생성에 결정적(critical)이 아니면 다른 클러스터에 중복하지 않는다.

그림 1에 있는 태스크 그래프에 STDS 알고리즘을 적용하면 그림 2와 같은 스케줄링 결과를 얻을 수 있다. 그림 2는 태스크 클러스터와 각 태스크의 실행 순서를 보여주며, 종료 노드가 완료되는 시각이 26이므로 알고리즘의 스케줄링 길이는 26이 된다.

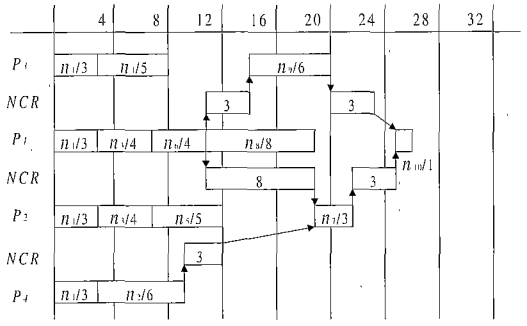


그림 2 STDS 알고리즘을 적용한 스케줄링 결과

그림 2에서 첫 번째 프로세서 P_1에는 태스크 1, 3, 6, 8, 10이 할당되었으며 각각은 3, 4, 4, 8, 1의 연산 비용을 필요로 한 것을 나타낸다. STDS 알고리즘에서는 어떤 노드의 *fpred* 노드가 이미 다른 클러스터에 할당된 경우에는 클러스터 생성을 위한 결정적 노드가 아니면 이를 중복하지 않는다. 예를 들면 그림 1의 태스크 그래프에서 *fpred*(7), *fpred*(8), *fpred*(9)는 n_6이다. 클러스터 생성 과정에서 n_6이 첫 번째 클러스터가 할당된 P_1에 배정되므로 n_7과 n_9이 포함된 클러스터가 각각 할당된 P_2와 P_3에는 n_6을 중복하지 않는다. STDS 알고리즘에서는 '$pred(i)=\{j|e_{ji}\in E\}$'라고 할 때 *fpred*$(i)$를 '$\{j|(ECT(j)+c_{ji}\geq(ECT(k)+c_{ki})\ \forall j\in pred(i)\ k\in pred(i), k\neq j$'로 정의한다. 즉 태스크 그래프에서 결합노드에 해당하는 태스크 i의 *EST*은 *fpred*(i)에 가장 큰 영향을 받는다. 따라서 태스크 i의 *fpred*(i)가 이미 다른 클러스터에 배정되었다면 이것을 중복하지 않음으로써 스케줄링 길이를 단축할 수 있다. 태스크의 중복을 기반으로 하는 스케줄링 알고리즘은 태스크 그래프의 *CCR* 값이 매우 적은 경우에 중복된 태스크로 인하여 스케줄링 길이가 길어지는 현상이 발생된다.

STDS 알고리즘을 적용한 태스크 그래프에서 태스크 사이에 링크가 존재하는 태스크들이 서로 다른 프로세서에 할당되면 통신비용이 필요하며, 그림 2에서 *NCR*로 나타내었다. 그림 2에서 P_1에 할당된 태스크 6이 종료되면 P_2와 P_3에 할당된 태스크 7과 9가 실행될 수 있다. 이를 위하여 P_1과 P_2, P_1과 P_3 사이의 통신이 필요하고 각각의 통신비용은 8과 3이다. 다중 프로세서 시스템에서 각 프로세서 사이의 동시 통신이 가능하므로 그림 2에서와 같이 이들 통신이 중복될 수 있다. 그러나 이더넷(Ethernet)과 같은 버스 기반의 NoW 환경에서는 네트웍 통신 자원의 충돌로 인하여 두 개 이상의 독립된 통신 태스크가 동시에 시작될 수 없다. 따라서 다중 프로세서 시스템과 같이 완전 결합된 네트웍 환경을 가정하고 개발된 스케줄링 알고리즘은 버스 기반의 NoW 환경에서 사용하는 것은 적당하지 않다.

태스크 중복을 기반으로 네트웍 통신 자원이 충돌되지 않도록 NoW 환경에서 적용할 수 있는 스케줄링 알고리즘이 현재까지 제안되지 않았다. 따라서 본 논문에서는 제안할 스케줄링 알고리즘의 성능을 비교하기 위하여 STDS 알고리즘을 NoW 환경에 맞도록 수정하였다. 네트웍 통신 자원이 충돌하지 않도록 STDS 알고리즘을 수정하여 그림 1의 태스크 그래프를 스케줄링하면 그림 3과 같은 결과를 얻을 수 있다. 설명의 편의를 위하여 이 알고리즘을 STDSNoW라고 한다.

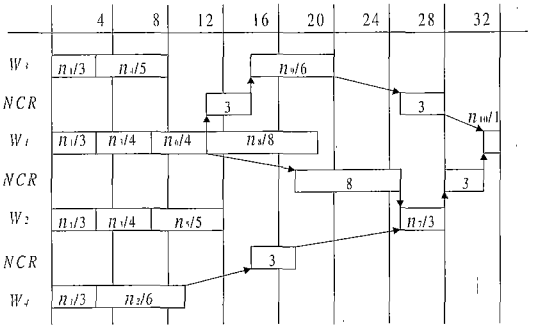


그림 3 STDSNoW 알고리즘을 적용한 스케줄링 결과

STDSNoW 알고리즘에서는 STDS 알고리즘과 같이 단계 1에서 클러스터 생성에 필요한 파라미터를 계산하며, 단계 2에서 큐를 사용하여 태스크가 중복된 클러스터를 생성한다. STDSNoW 알고리즘은 단계 2에서 클러스터를 생성할 때 STDS 알고리즘과 동일한 기준으로 중복할 태스크를 선택한다. 그러나 각 클러스터에 배정된 태스크를 스케줄링 하는 단계에서는 네트웍 통신 자원의 충돌이 발생되지 않도록 한다.

3. 휴리스틱 태스크 스케줄링 알고리즘

이 장에서는 본 논문에서 제안하는 HTSNoW (Heuristic Task Scheduling for NoW) 알고리즘을 소

개한다. 버스 기반의 NoW 환경에서는 네트웍에서의 통신 충돌이 병렬 응용 프로그램의 실행 시간에 매우 큰 영향을 주며, 네트웍 통신 자원의 동시 사용을 위한 공유가 불가능하다. 따라서 제안된 HTSNoW 알고리즘에서는 서로 다른 워크스테이션에 할당된 태스크 사이의 통신을 위하여 네트웍 통신 자원을 우선 배정하여 스케줄링 함으로써 네트웍 충돌을 방지한다. 또한 HTSNoW에서는 다중 프로세서 시스템에서 사용된 알고리즘과 같은 방법으로 클러스터를 생성하지만 중복할 태스크를 선택할 때 다음과 같은 휴리스틱을 사용함으로써 스케줄링 길이를 단축하여 병렬 시간을 줄인다.

(1) *CPT* 노드가 현재 스케줄링하고 있는 노드의 유일한 부모 노드이면 *CPT* 노드를 중복한다.

(2) 스케줄링하고 있는 결합 노드와 동일한 클러스터에 배정할 *CPT* 노드가 이미 다른 클러스터에 배정된 경우에는 결합 노드의 *CCPT* 노드가 결합 노드와 같은 클러스터에 배정될 수 있는 경우에만 *CPT* 노드를 중복하여 배정한다.

(3) 결합 노드에서 모든 부모 노드가 이미 다른 워크스테이션에 배정된 경우에는 *CPT* 노드를 중복한다.

위에서 첫 번째 것은 어떤 노드의 부모 노드가 하나만 존재하는 경우로 부모 노드를 포함하여야 클러스터의 생성이 가능하다. 따라서 이 부모 노드가 이미 다른 클러스터에 포함되었더라고 *CPT* 노드인 이 부모 노드를 현재 스케줄링 하는 노드가 포함된 클러스터에 중복한다.

태스크 그래프에서 결합 노드에 대한 스케줄링이 병렬 시간에 가장 큰 영향을 미친다. 위의 휴리스틱에서 두 번째와 세 번째 내용은 결합 노드와 어떤 부모(parent) 노드를 동일한 워크스테이션에 할당 할 것인지를 결정한다. HTSNoW 알고리즘의 중요한 개념은 결합 노드와 스케줄링 길이에 결정적인 영향을 줄 수 있는 *CPT* 노드를 선택적으로 중복하여 동일한 워크스테이션에 배정함으로써 스케줄링 길이와 병렬 시간을 단축하는 것이다. 태스크 그래프에서 하나의 노드가 두 개 이상의 노드에 대한 *CPT* 노드로 나타날 수 있다. 이때 *CPT* 노드를 계속적으로 중복하여 클러스터에 할당하는 것은 스케줄링 길이를 연장할 수 있는 가능성이 있다. 그 이유는 정의 5의 *CPT(i)* 정의에서 확인할 수 있다. 정의에 따르면 태스크 *i*의 *EST*는 *CPT(i)*에 의해서 결정되므로 *CPT* 노드를 중복함으로써 자식 노드의 *EST*를 지연시켜 스케줄링 길이가 늘어날 수 있다. 따라서 HTSNoW 알고리즘에서는 위의 휴리스틱에서 두 번째 방법을 사용하여 *CPT* 노드를 선택적으로 중복한다. 이 방법은 결합 노드의 *CPT* 노드 중복을 위하여 *CPT* 노드의 *CPT*인 *CCPT* 노드가 동일한 클러스터에 할당 가능한지를 체크한다. 결합 노드와 *CCPT* 노드가 동일할 클러스터에 할당될 수 있으면 결합 노드, *CPT*, *CCPT* 노드를 하나의 클러스터에 할당하여 스케줄링 길이를 단축하는 것이다. 위에서 세 번째 경우는 결합 노드에서 모든 부모 노드가 이미 다른 클러스터에 할당된 상태에서 새로운 클러스터 생성을 위하여 부모 노드 중에서 하나를 선택하는 경우이다. 따라서 결합 노드와 *CPT* 노드를 동일한 클러스터에 할당함으로써 스케줄링 길이를 단축하기 위한 것이다. 위의 휴리스틱을 사용한 스케줄링의 예를 3.2절에서 설명한다.

3.1 알고리즘 구조

HTSNoW 알고리즘은 태스크 그래프를 입력으로 받아서 태스크 클러스터를 생성하는 부분과 생성된 클러스터에 속한 태스크들에 대한 시작 시각과 완료 시각을 확정하는 부분으로 구성된다. 알고리즘의 단계 1에서는 2장에서 나타낸 정의들을 이용하여 알고리즘에서 사용할 필요한 파라미터 값들을 구한다. 이 단계는 다중 프로세서 환경의 스케줄링 알고리즘과 유사하다.

단계 2에서는 단계 1에서 구한 값들을 이용하여 중복된 태스크로 구성된 태스크 클러스터를 생성한다. 각 노드는 스케줄링 되기 전에 태스크 그래프에서의 레벨을 기준으로 오름차순으로 저장한다. 태스크 클러스터는 태스크 그래프에서 레벨이 가장 낮은 노드에서 시작하여 각 노드의 *CPT* 노드를 동일한 클러스터에 할당하면서 진행한다. 각 노드의 중복 여부는 휴리스틱에 의하여 결정되는 것으로, 클러스터 생성을 위해 반드시 필요한 노드와 스케줄링 길이를 줄일 수 있는 노드를 클러스터에 중복한다. 스케줄링하는 현재 노드의 *CPT* 노드가 이미 다른 클러스터에 할당된 경우에는 정의 5의 *CCPT* 개념을 적용하여 *CPT* 노드의 중복을 위한 휴리스틱을 사용한다.

단계 3에서는 단계 2의 결과를 이용하여 태스크의 *RST*와 *RCT*를 확정하고, 병렬 처리 시간을 예측하기 위한 스케줄링 길이를 계산한다. 이 단계에서 알고리즘은 버스 기반의 NoW 특성을 고려하여 클러스터에 배정된 태스크를 네트웍 자원의 충돌이 발생되지 않도록 스케줄링한다. 네트웍 통신 자원을 독립된 두 개의 통신 태스크가 동시에 사용할 때 충돌이 발생되므로 스케줄링 알고리즘에서 이를 방지하도록 통신 자원을 할당한다. 이를 위하여 알고리즘에서는 슬롯(slot) 개념을 사용한다. 두 개의 태스크가 통신하는 시간 동안을 하나의 슬롯으로 보며, 프로그램이 시작되기 전에 모든 슬롯은

비어있다고 가정한다. 알고리즘에서는 태스크가 통신하기 전에 통신비용에 해당하는 길이의 사용하지 않는 첫 번째 슬롯을 태스크에 배정한다.

각 클러스터에 저장된 태스크를 레벨이 낮은 것부터 스케줄링하며, 스케줄링 할 태스크의 데이타 선행 관계가 만족되지 않으면 다음 클러스터의 태스크를 스케줄링한다. 스케줄링 할 노드의 모든 부모 노드의 스케줄링이 완료되어 데이타 선행 관계가 만족되는 경우에는 네트웍 통신 자원의 충돌이 발생되지 않도록 태스크를 스케줄링한다. 다음은 HSTNoW 알고리즘을 나타낸 것이다.

```
단계 1:
// 입력: 태스크 그래프 (V, E, t, c)
   출력: EST, ECT, LST, LCT, level, CPT, CCPT //
compute EST(i), ECT(i), LST(i), LCT(i), level(i), CPT(i),
CCPT(i) for all nodes i∈v.
단계 2:
// 입력: 태스크 그래프 (V, E, t, c), queue (레벨에 따라 오름
   차순으로 정렬된 태스크 큐)
   출력: 중복된 태스크를 갖는 태스크 클러스터들 //
i = 0
n_x = first element of queue
assign n_x to a W_i
while (not all tasks are assigned to a workstation) {
    for (qindex=0; num_of_task - 1; qindex++) dupflag[
    qindex] = 0
    n_y = CPT(x)
    if ( n_y already been assigned to another workstation) {
      n_k = another parent of n_x which has not yet been
      assigned to a workstation
       If ((LST(x) - LCT(y)) ≥ c_xy) then // n_y is not
       critical for n_x //
          { n_y = n_k; dupflag[y] = 1}
       else
        candflag = 0
        for another parent n_z of n_x, z ≠ y
        if ((ECT(x)+ c_yx) = (ECT(z)+ c_zx) &&
          n_z has not yet been assigned to a
          workstation) then
          { n_y = n_z; candflag = 1}
        endif
        if (candflag = 0) { n_y = n_k; dupflag[y] = 1}
       endif
    }
    endif
    assign n_y to W_i
    n_x = n_y
    if n_x is entry node
        assign n_x to W_i
for (qindex=0; num_of_task - 1; qindex++)
  if (dupflag[qindex] = 1 && CPT(qindex) is in W_i)
          assign n_qindex to W_i
    n_x = the next element in queue which has not
    yet been assigned to a workstation
    increment i
    assign n_x to W_i
  endif
}
단계 3:
// 입력: 1) 중복된 태스크로 이루어진 태스크 클러스터들
      2) bus_slot_list: 네트웍 통신 자원에서 사용 가능한
         슬롯을 나타냄
            초기값: [0,∞], 형식: [ s_1, e_1], [s2, e2] ,..., [
                    s_n, ∞] //

let there exit n task clusters on n workstations, i.e. { W_0,
W_1, ..., W_n-1}
while (not all nodes be scheduled) do {
for (i=0; i<0; i++) {
    select the node n_j to be scheduled from W_i
    let there exist p parents of n_j, i.e. { n_j1, n_j2, ..., n_jp}
    if (all n_j's p parents have been scheduled) {
      for (k=1; k≤p; k++) {
        if ( n_jk was scheduled onto W_i)
          if (tmp_rst of n_j < tmp_rct of n_jk) tmp_rst
          of n_j = tmp_rct of n_jk
        else {
        let[ s_i, e_i] be the first slot in bus_slot_list for n_jk
        if ((( e_i - s_i + 1) ≥ c_jkj ) && ( s_i ≥ RCT of
        n_jk)) {
            slot_busy = s_i + c_jkj
            substitute [ s_i, e_i] to [ s_i + c_jkj, e_i]
          }
          else
            if ((RCT of n_jk + c_jkj) ≤ e_i)) {
                slot_busy = RCT of n_jk + c_jkj
                substitute [ s_i, e_i] with [ s_i, RCT of
                n_jk] and [RST of n_jk + c_jkj, e_i]
            }
            if (tmp_rst of n_j < slot_busy) tmp_rst of
            n_j = slot_busy
          }
          if (tmp_rst of n_j < W_i_busy) {
            tmp_rst of n_j = W_i_busy
            tmp_rst of n_j = tmp_rst of n_j + c_jkj
```

```
                W_i_busy = tmp_rct of n_j
            }
        }
    }
    if (RST of n_j is not assigned) {
      RST of n_j = tmp_rst of n_j; RCT of n_j = tmp_rct
        of n_j
    }
    else
        if (tmp_rst of n_j < RST of n_j) {
          RST of n_j = tmp_rst of n_j;  RCT of n_j =
            tmp_rct of n_j
        }
}
}
```

3.2 HTSNoW 알고리즘의 동작 예제

태스크 중복을 기반으로 하는 스케줄링 알고리즘들은 스케줄링 길이를 단축하기 위하여 태스크를 여러 개의 프로세서에 중복하여 배정한다. 이들 알고리즘들은 특히 태스크 그래프에서 결합 노드와 결합 노드의 스케줄링에 가장 영향을 주는 부모 노드를 동일한 프로세서에 할당하며, 노드의 중복이 가능하다. 다중 프로세서 시스템에서는 각 프로세서의 직접적인 통신이 가능하므로 많은 태스크를 중복함으로써 스케줄링 길이를 단축할 수 있다. 그러나 버스 기반의 NoW 환경에서는 독립된 태스크의 통신 자원에 대한 동시 사용이 불가능하여 태스크 중복으로 인한 스케줄링 지연이 발생될 수 있다. 이와 같이 다중 프로세서 시스템을 위한 스케줄링 알고리즘과 버스 기반의 NoW 환경에서 적용될 수 있는 HTSNoW 알고리즘의 차이점을 설명하기 위하여 이 장에서는 [10]의 연구에서 발표된 그림 1의 테스크 그래프를 사용한다. 그림 4는 그림 1의 테스크 그래프를 사용하여 HTSNoW 알고리즘을 적용하여 스케줄링한 결과를 나타낸 것이다.

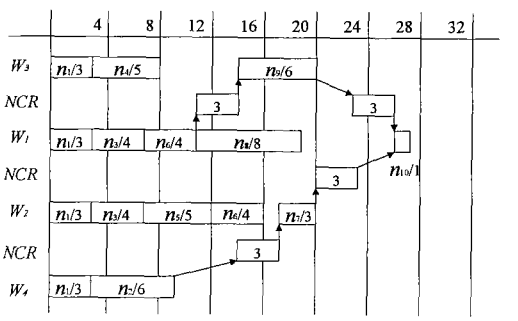


그림 4 HTSNoW 알고리즘을 적용한 스케줄링 결과

그림 4에서 W_i는 워크스테이션을 나타내며 NCR은 네트웍 통신 자원의 사용을 나타낸다. 그림에서 첫 번째 워크스테이션인 W_1에는 n_{10}, n_8, n_6, n_3, n_1 노드들이 포함된 첫 번째 태스크 클러스터가 배정되어 있으며, 각 노드는 1, 8, 4, 4, 3의 연산비용을 갖는 것을 나타낸다. HTSNoW 알고리즘을 적용하면 스케줄링 길이는 27이며, 알고리즘에서 필요로 하는 워크스테이션의 수는 4이다. 그림 4의 스케줄링 결과를 만들기 위한 HTSNoW 알고리즘의 단계별 스케줄링 과정은 다음과 같다.

HTSNoW 알고리즘의 단계 1에서는 2장의 정의들을 사용하여 클러스터 생성에 필요한 값들을 계산한다. 정의 1을 사용하여 n_9의 EST와 ECT를 구하면 다음과 같다. n_9의 부모 노드로 n_4와 n_6이 있으므로 $ECT(4)+c_{49}$와 $ECT(6)$의 최대값은 12이며 $ECT(6)+c_{69}$와 $ECT(4)$의 최대값은 14이다. 12와 14의 최소값은 12이므로 $EST(9)$는 12가 된다. 정의 1에 따라 $ECT(9)$는 18이 된다. 각 노드에 대한 LCT와 LST의 계산은 종료 노드에서 시작 노드 방향으로 상향식(bottom-up) 방식으로 진행한다. 예를 들어 $LCT(6)$값은 n_6이 n_7, n_8, n_9 노드를 자식 노드로 가지므로 정의 2에 의하여 $LST(7)$, $LST(8)$, $LST(9)$ 중에서 최소값인 12가 되며, $LST(6)$은 12에서 $t(4)$를 뺀 8이 된다. CPT 노드는 결합 노드의 스케줄링에 가장 큰 영향을 줄 수 있는 부모 노드를 찾기 위한 것이다. 예를 들어 $CTP(7)$은 n_7의 부모 노드로 n_2, n_5, n_6이 있으므로 세 노드에 대하여 각각 ECT 값과 n_7과의 통신 비용을 합하여 최대값을 갖는 노드가 된다. 각 노드에 대한 이들 값이 각각 (9+3), (12+3), (11+8) 이므로 $CTP(7)$은 n_6이 된다. 정의에따라 각 노드에 대한 파라미터 값을 계산하면 표 1의 결과를 얻을 수 있다.

표 1 그림 1의 태스크 그래프에 대한 파라미터 값

노 드	EST	ECT	LST	LCT	CPT	$CCPT$	$level$
n_1	0	3	0	3	-	-	20
n_2	3	9	6	12	1	-	10
n_3	3	7	3	7	1	-	17
n_4	3	8	3	8	1	-	12
n_5	7	12	7	12	3	1	9
n_6	7	11	8	12	3	1	13
n_7	15	18	15	18	6	3	4
n_8	11	19	13	21	6	3	9
n_9	12	18	12	18	6	3	7
n_{10}	21	22	21	22	8	6	1

알고리즘의 단계 2에서는 일반적인 클러스터 알고리즘과 같이 태스크 클러스터를 생성하며, 각 노드의 *CPT* 노드의 중복을 위해서는 앞에서 설명한 휴리스틱을 사용한다. 클러스터 생성을 위하여 모든 노드를 레벨을 기준으로 오름차순으로 정렬하여 큐(queue)에 저장한다. 표 1에 나타난 레벨을 이용하면 큐에는 n_{10}, n_7, n_9, n_8, n_5, n_2, n_4, n_6, n_3, n_1의 순서로 태스크가 저장된다. 큐의 첫 번째 노드인 n_{10}부터 시작하여 각 노드의 *CPT* 노드를 찾아서 하나의 클러스터를 생성하면 n_{10}, n_8, n_6, n_3, n_1 노드들을 포함하는 하나의 클러스터가 생성된다. 첫 번째 클러스터를 생성하는 과정에서는 중복되는 태스크가 발생되지 않는다.

다음 클러스터의 생성은 큐에서 클러스터에 할당되지 않은 첫 번째 태스크인 n_7부터 시작한다. 결합 노드인 n_7의 *CPT* 노드인 n_6이 첫 번째 클러스터에 포함되어 있으므로 n_7의 다른 부모 노드인 n_5를 사용하여 클러스터를 생성한다. 그 결과 n_7, n_5, n_3, n_1 노드들을 포함하는 두 번째 클러스터가 생성된다. 결합 노드의 *CPT* 노드인 n_6을 이 클러스터에 포함할 것인지는 휴리스틱을 사용하여 중복 여부를 결정한다. 이 경우는 휴리스틱의 두 번째 조건에 해당하는 것으로 *CCPT*(7)인 n_3이 두 번째 클러스터에 포함되므로 n_6을 이 클러스터에 포함하여 첫 번째 클러스터와 중복한다. 두 번째 클러스터에 n_3을 중복한 것은 휴리스틱의 첫 번째 조건에 해당하는 것으로 n_5의 부모 노드로 n_3 만이 존재하기 때문이다.

세 번째 클러스터는 n_9, n_4, n_1로 구성된다. 여기서 n_9의 *CPT* 노드인 n_6을 포함하지 않는 것은 *CCPT*(9)인 n_3이 이 클러스터에 포함되지 않았기 때문이다. 네 번째 클러스터는 n_2, n_1로 구성되며, 모든 노드가 클러스터에 할당되어 단계 2가 종료된다. 알고리즘의 두 번째 단계에서 휴리스틱을 사용하여 결합 노드의 *CPT* 노드를 선택적으로 중복함으로서 스케줄링 길이를 단축할 수 있으며 클러스터의 수를 줄일 수 있다. 그림 4를 보면 알고리즘의 단계 2에서 휴리스틱의 2번째 조건을 적용하여 W_1과 W_2에 n_6을 중복하였으며 W_3에는 중복하지 않음을 알 수 있다. 그 결과 n_7을 스케줄링할 때 W_1의 n_6과 W_2의 n_7이 통신할 필요가 없으며 통신 자원을 할당할 필요가 없으므로 스케줄링 길이가 단축되었다. n_9의 *CPT* 노드인 n_6이 W_3에 중복되어도 태스크의 중복에 의한 스케줄링 길이의 단축 효과가 없기 때문에 단계 2에서 휴리스틱에 의하여 이를 중복하지 않았다. W_3에 n_6이 중복되면 n_3의 실행 결과를 받아야 하고 이로 인한 3 단위의 통신 자원을 사용하게 된다. 따라서 NoW 환경에서 불필요한 태스크의 중복은 네트웍 통신 자원의 사용에 의한 스케줄링 길이가 늘어나는 역효과가 발생한다.

세 번째 단계에서는 두 번째 단계에서 생성한 클러스터를 이용하여 각 태스크의 *RST*과 *RCT*을 확정하고, 응용 프로그램의 스케줄링 길이와 병렬 시간을 계산한다. 이 단계의 스케줄링에서 각 태스크가 실행되기 위해서는 태스크 사이의 데이타 선행 조건을 만족하여야 한다. 따라서 각 클러스터에서 입력 노드부터 스케줄링하며 데이타 선행 조건을 만족하지 못하는 경우에는 다음 클러스터의 태스크를 스케줄링한다. 그림 4에서 첫 번째 클러스터가 할당된 W_1의 n_1부터 스케줄링하며, 이어서 n_3, n_6, n_8을 스케줄링할 수 있다. 이들 태스크는 실행 시각에 부모 노드들이 모두 실행되어 데이타 선행 조건을 만족하므로 스케줄링이 가능하고, 동일한 워크스테이션에 할당되어 통신비용이 소요되지 않는다. n_{10}을 스케줄링하기 위해서는 n_7, n_8, n_9의 스케줄링이 완료되어야 하나 n_7과 n_9의 스케줄링이 완료되지 않아서 다음 클러스터의 태스크를 스케줄링한다.

두 번째 클러스터가 할당된 W_2의 n_1, n_3, n_5, n_6 노드들을 스케줄링하며, n_2가 실행되지 않아서 n_7의 스케줄링이 지연된다. 이어서 W_3에 할당된 n_1, n_4, n_9 노드들을 스케줄링한다. n_4와 n_6의 스케줄링이 완료되어 n_9 스케줄링이 가능하며, n_6가 n_9와 다른 워크스테이션에 할당되어 3 단위의 통신비용이 소요된다. 통신비용 3 단위는 네트웍의 통신 자원을 사용하는 비용으로 n_6이 실제 완료되는 시각 이후에 통신 자원을 사용할 수 있다. n_6 태스크가 완료되는 실제 시각인 *RCT*(6)은 11이다. 따라서 통신은 *RCT*(6)이 완료되는 11 이후에 가능하며 알고리즘에서는 이를 만족하는 최초의 3 단위 가용 슬롯을 찾아서 할당한다. 현재 모든 네트웍 통신 자원의 슬롯이 비어있으므로 11-14까지의 슬롯을 n_4와 n_6의 통신에 할당한다.

이어서 마지막 클러스터가 할당된 W_1의 n_1, n_2가 스케줄링되며, W_1에 할당된 태스크의 스케줄링을 다시 시도한다. W_1의 n_{10}은 n_7이 시작되지 않아서 스케줄링이 지연된다. 이어서 W_2의 n_7이 스케줄링되며 n_2와의 통신을 위하여 3 단위의 네트웍 통신 자원을 사용한다.

n_7은 부모 노드인 n_2, n_5, n_6 노드의 실행이 종료된 시각 후에 스케줄링되며, n_2와 통신을 하기 위해서는 n_6과 n_9의 통신을 위하여 이미 할당된 슬롯과 중복되지 않는 슬롯을 할당하여야 한다. 이러한 특성은 다중 프로세서 시스템에서의 스케줄링 알고리즘과 구분되는 것이며, 이로 인하여 NoW 환경이 다중 프로세서 시스템 보다 스케줄링 길이가 늘어나게 된다. W_3와 W_4에 배정된 모든 태스크의 스케줄링이 완료되었으므로 W_1의 n_{10}의 스케줄링을 시도한다. 모든 부모 노드의 스케줄링이 완료되어 데이타 선행 조건이 만족되므로 네트웍 자원의 중복이 발생되지 않도록 그림 4과 같이 스케줄링 할 수 있다.

그림 1의 태스크 그래프를 HTSNoW 알고리즘으로 스케줄링하면 스케줄링 길이가 27이 되며, [10] 연구에서 보여준 26보다 1 단위가 길다. 이것은 독립된 두 개 이상의 태스크가 네트웍 통신 자원을 공유하지 못함으로써 발생되는 현상으로 버스 기반의 NoW 환경과 다중 프로세서 시스템과의 차이점이다. 그림 1의 태스크 그래프를 HTSNoW 알고리즘과 STDSNoW 알고리즘에 적용하면 스케줄링 길이는 각각 27과 32이며, 각 알고리즘에서 생성한 클러스터의 수는 4이다. 각 클러스터는 하나의 워크스테이션을 필요로 하므로 그림 1의 태스크 그래프를 위해 각 알고리즘에서 요구하는 워크스테이션의 숫자는 4이다. HTSNoW 알고리즘에서는 휴리스틱을 사용하여 *CPT* 노드를 선택적으로 중복함으로써 스케줄링 길이를 단축하고 응용 프로그램의 실행을 위한 워크스테이션의 수를 줄일 수 있다.

3.3 알고리즘의 시간 복잡도

HTSNoW 알고리즘의 첫 번째 단계에서는 스케줄링에 필요한 파라메터의 값들을 구하기 위하여 태스크 그래프의 각 노드를 방문한다. 각 노드의 입출력 링크를 점검하여야 하며 최악의 경우에는 태스크 그래프의 모든 링크를 점점해야 한다. 따라서 E가 태스크 그래프에서 링크의 수라면 최악의 경우 이 단계를 위한 시간 복잡도는 $O(E)$가 된다. 단계 2에서는 클러스터 생성을 위하여 태스크 그래프를 깊이 우선 탐색(depth first search) 방식으로 각 노드를 방문하는 것이므로 V가 태스크 그래프에서 노드 수일 때 이 단계의 시간 복잡도는 $O(V+E)$가 된다. 단계 3에서는 각 워크스테이션에 할당된 클러스터의 모든 태스크들을 스케줄링 해야 하므로 W가 워크스테이션의 수인 경우에 시간 복잡도는 $O(WV)$가 된다. 최악의 경우 워크스테이션의 수는 노드의 수와 같을 수 있으므로 단계 3의 최악의 경우 시간 복잡도는 $O(V^2)$이 된다. 따라서 HTSNoW 알고리즘의 최악의 경우 시간 복잡도는 $O(V^2)$이 된다.

4. 알고리즘의 성능 비교

본 논문에서는 제안한 알고리즘의 성능을 평가하기 위하여 실제 응용 프로그램의 태스크 그래프를 사용하여 시뮬레이션을 하였으며, 스케줄링 길이와 알고리즘에서 요구하는 워크스테이션의 수를 성능 비교를 위한 항목으로 선택하였다. 병렬 응용 프로그램을 NoW 환경에서 실행할 때 가장 심각한 문제중의 하나가 통신 지연에 관한 것이다. 시뮬레이션에서 통신량의 다양한 변화에 따른 알고리즘의 성능을 확인하기 위하여 *CCR*의 값을 1, 2, 5, 10배로 변화시켰으며, 각각에 대하여 HTSNoW 알고리즘과 STDSNoW 알고리즘의 성능을 비교하였다.

시뮬레이션에서는 4개의 실제 응용 프로그램 태스크를 알고리즘에 적용하였으며, *CCR*의 변화에 따른 스케줄링 길이와 알고리즘에서 필요로 하는 워크스테이션의 수를 비교하였다. 시뮬레이션에서는 ALPES 프로젝트 [16]의 일부분인 Bellman-Ford (BF) 알고리즘, Systolic (SY) 알고리즘, Master-Slave (MS) 알고리즘과 병렬 프로그램의 예로 많이 사용되는 Cholesky decomposition (CD) 알고리즘을 사용하였다. 각 응용프로그램의 태스크 그래프에서 태스크의 수는 2,500개 정도이며, 링크의 수는 4,902에서 20,298개로 구성된다. 부모노드와 자식노드의 수는 1에서 140까지이며, 연산비용은 1에서 20,000 시간 단위까지 변한다. 연산비용은 태스크 그래프에서 각 태스크(노드)의 연산에 사용되는 시간을 나타내는 것으로 ALPES 프로젝트에서 사용된 DAG의 연산비용과 동일한 값을 사용하였다. 실험에서 사용한 각 응용 프로그램의 DAG 특성은 표 2와 같다. 표 2에서 *CCR*은 정의 4에 따라 각 응용 프로그램 DAG를 사용하여 계산한 결과이다. 표 2의 '알고리즘에서 요구하는 워크스테이션의 수'는 각 응용 프로그램의 *CCR* 값이 표2와 같을 때 HTSNoW 알고리즘을 사용

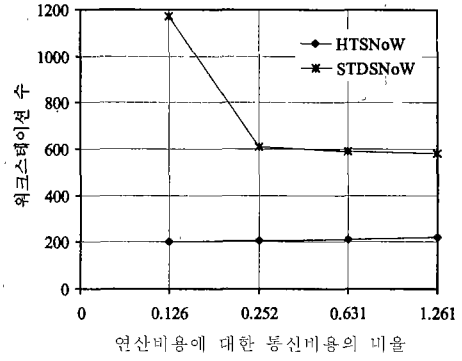
표 2 응용 프로그램의 성능 특성

특성 \ 알고리즘	BF 알고리즘	CD 알고리즘	SY 알고리즘	MS 알고리즘
노드 수	2,291	2,925	2,502	2,262
링크 수	20,298	5,699	4,902	6,711
CCR	0.1261	1.5179	0.0068	0.0068
알고리즘에서 요구하는 워크스테이션의 수	201	75	50	54

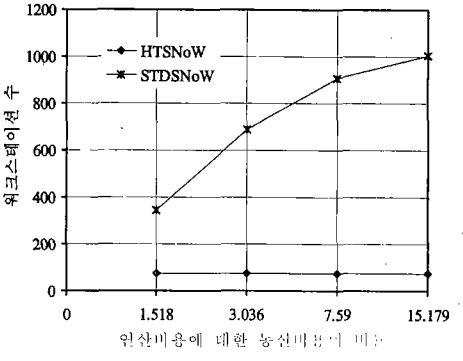
하여 응용 프로그램을 처리하는데 필요로 하는 워크스테이션의 수를 나타낸 것이다.

그림 5는 *CCR* 값을 변화시키면서 응용 프로그램을 실행하기 위하여 알고리즘에서 요구하는 워크스테이션의 수를 비교한 것이다. HTSNoW와 STDSNoW 알고리즘은 모두 태스크의 중복을 기반으로 하는 스케줄링 알고리즘이며, HTSNoW 알고리즘은 휴리스틱을 사용하여 중복할 태스크를 선택한다. 응용 프로그램 MS를 실행하기 위해서 두 개의 알고리즘이 비슷한 수의 워크스테이션을 필요로 하며, 다른 3개 응용 프로그램의 경우에는 HTSNoW 알고리즘에서 요구하는 워크스테이션의 수가 STDSNoW 알고리즘에서 요구하는 워크스테이션의 수보다 매우 적다는 것을 알 수 있다. 특히 CD 응용 프로그램은 *CCR*의 값이 증가함에 따라서 두 알고리즘에서 요구하는 워크스테이션 수의 차이가 더욱 커짐을 알 수 있다.

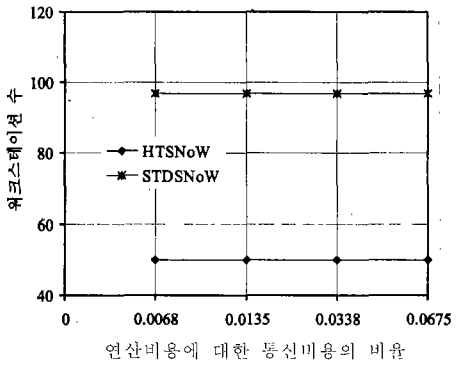
그림 5(a)와 (b)의 HTSNoW 알고리즘에서 *CCR*의 변화에 따른 워크스테이션 수의 변화가 거의 없는 것은



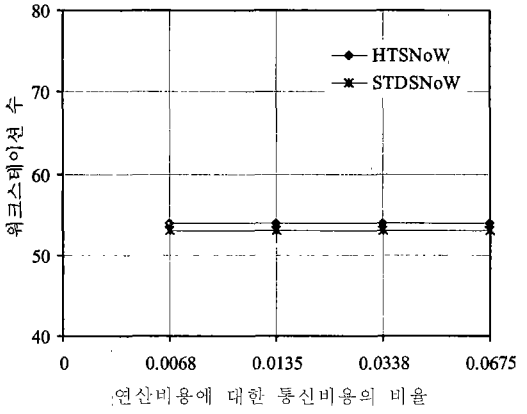
(a) Bellman-Ford algorithm



(b) Cholesky decomposition algorithm



(c) Systolic algorithm

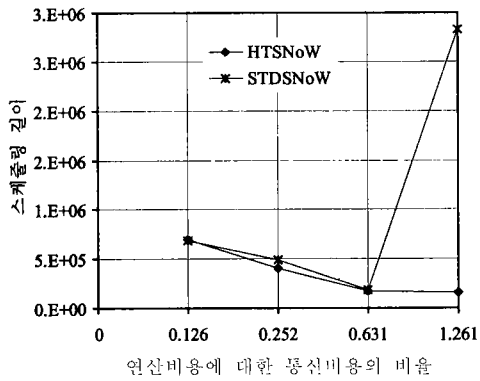


(d) Master-slave algorithm

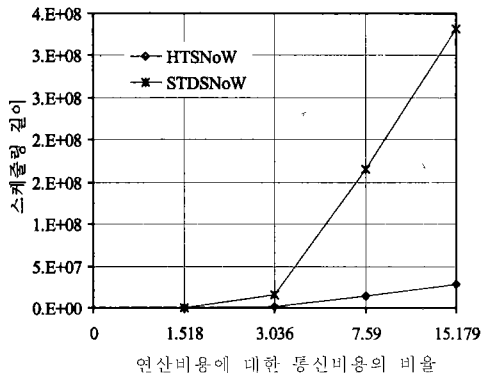
그림 5 *CCR*의 변화에 따른 워크스테이션 수의 비교

3.2절의 예제와 같이 HTSNoW 알고리즘에서 휴리스틱을 사용하여 태스크를 선택적으로 중복한 결과이다. 반면에 그림5(c)와 (d)에서는 두 알고리즘 모두 *CCR*의 변화에 대하여 워크스테이션 수의 변화가 거의 없다. 이와 같은 결과는 응용 프로그램의 특성에 기인한 것으로 표 2에서 원인을 확인할 수 있다. 표 2에서 BF와 CD 알고리즘이 SY와 MS 알고리즘과 비교하여 *CCR* 값이 상대적으로 매우 적음을 확인할 수 있다. *CCR* 값이 매우 적은 응용 프로그램은 통신보다 태스크의 연산에 대부분의 시간이 소요된다. 따라서 이러한 경우 *CCR*이 증가하여도 스케줄링 알고리즘에서 생성하는 클러스터 수에 큰 변화를 주지 않는다. 본 논문에서는 응용 프로그램의 성능 특성을 유지하기 위하여 *CCR* 값의 변화를 주어진 값의 1, 2, 5, 10배까지로 한정하였다.

그림 6은 그림 5에서 보여준 수의 워크스테이션을 사용하는 경우에 알고리즘에서 생성하는 스케줄링 길이를 나타낸 것이다. 각각의 응용 프로그램에서 *CCR*의 변화에 따라 HTSNoW 알고리즘이 생성하는 스케줄링 길이가 STDSNoW 알고리즘이 생성하는 스케줄링 길이보다 매우 적거나 비슷함을 알 수 있다. 또한 이 경우에 BF, CD, SY 프로그램을 수행하기 위해 HTSNoW 알고리즘에서는 상대적으로 매우 적은 수의 워크스테이션을 사용하였음을 그림 5에서 알 수 있다.

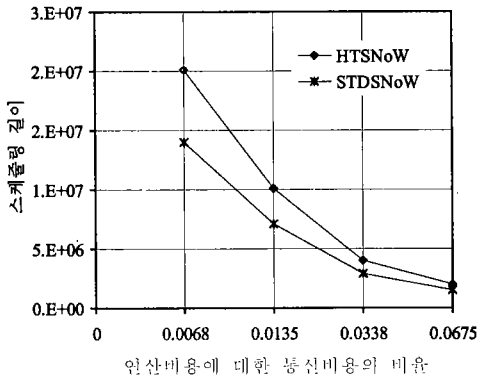


(a) Bellman-Ford algorithm

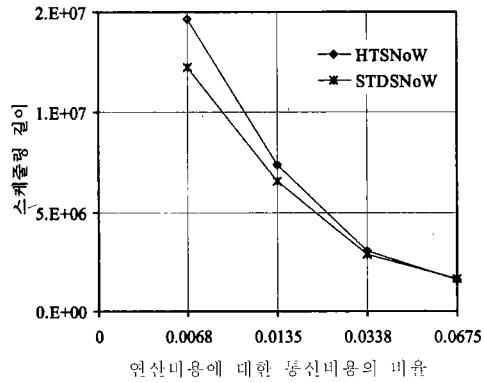


(b) Cholesky decomposition algorithm

BF와 CD 프로그램에서는 HTSNoW 알고리즘의 스케줄링 길이가 STDSNoW 알고리즘의 스케줄링 길이보다 적으며 *CCR*의 값이 증가함에 따라서 그 차이가 더욱 커짐을 볼 수 있다. 이러한 성능의 차이는 HTSNoW 알고리즘에서 결합 노드를 스케줄링 할 때 *CTP* 노드를 선택적으로 중복하는 휴리스틱에 기인한 것이다.



(c) Systolic algorithm



(d) Master-slave algorithm

그림 6 *CCR*의 변화에 따른 스케줄링 길이의 비교

그림 6(c)와 (d)에서 *CCR* 값이 커지면 스케줄링 길이가 줄어드는 것을 볼 수 있다. 이러한 결과는 *CCR* 값이 매우 적은 태스크 그래프에 태스크 중복을 기반으로 하는 태스크 스케줄링 알고리즘을 적용하는 경우에 나타나는 공통된 현상이다. 통신비용과 비교하여 상대적으로 매우 큰 연산비용을 갖는 태스크들을 중복함으로써 중복된 태스크에 의해서 스케줄링 길이가 커지는 현상이 나타난다. 그러나 통신비용을 증가함으로써 중복된 태스크에 의한 영향이 감소하고 스케줄링 길이가 줄어들게 된다.

SY와 MS 프로그램에서는 HTSNoW 알고리즘에서 생성하는 스케줄링 길이가 STDSNoW 알고리즘이 생성하는 스케줄링 길이보다 크거나 비슷한 것을 알 수 있다. 그러나 그림 5에서 SY 프로그램의 실행을 위하여 STDSNoW 알고리즘이 상대적으로 많은 워크스테이션

을 사용하였으므로 두 알고리즘의 성능을 그림 6으로 직접 비교하는 것은 적당하지 않다. 4개의 응용 프로그램이 다양한 서로 다른 특성을 가지고 있으므로 하나의 알고리즘이 모든 응용 프로그램에 대하여 성능이 우수하다고 판단하기는 어렵다. 그림 5와 그림 6에서 BF, CD, SY 응용 프로그램에 대하여 HTSNoW 알고리즘의 성능이 STDSNoW 알고리즘보다 우수함을 알 수 있다. 그러나 MS 응용 프로그램에서는 STDSNoW 알고리즘의 성능이 HTSNoW 알고리즘과 유사하거나 조금 우수한 것을 알 수 있다. 알고리즘의 성능과 더불어, 태스크 중복을 기반으로 NoW 환경에서 통신 자원의 충돌을 고려한 새로운 스케줄링 알고리즘을 제안하였다는 것에 본 논문의 의의가 있다.

5. 결 론

NoW 환경의 가장 심각한 한계중의 하나가 워크스테이션 사이의 통신을 위한 비용이 크다는 것이며 이를 해결하기 위한 하나의 방법이 효과적인 태스크 스케줄링 알고리즘의 사용이다. 현재까지 다중 프로세서 시스템을 기반으로 하는 다양한 스케줄링 알고리즘이 제안되었으나 네트웍 통신 충돌이 프로그램 실행 시간에 심각한 영향을 주는 버스 기반의 NoW 환경에서 그대로 적용할 수가 없다. 본 논문에서는 버스 기반의 NoW 환경에서 적용할 수 있는 태스크 스케줄링 알고리즘을 제안하였다. 제안된 HTSNoW 알고리즘은 태스크의 중복을 기반으로 하며, 스케줄링 길이를 줄이기 위한 휴리스틱을 사용하여 태스크를 선택적으로 중복한다. HTSNoW 알고리즘의 시간 복잡도는 테스크 그래프에서 노드의 수가 V일 때 $O(V^2)$이다. 실제 응용 프로그램 태스크를 적용한 시뮬레이션을 통하여 스케줄링 길이와 알고리즘에서 요구하는 워크스테이션의 수를 기준으로 성능을 비교하였다. 시뮬레이션 결과에서 HTSNoW 알고리즘이 태스크 중복을 기반으로 하는 STDSNoW 알고리즘보다 성능이 우수함을 보여주었다.

참 고 문 헌

[1] H. El-Rewini, H.H. Ali, and T. Lewis, "Task Scheduling in Multiprocessing Systems," *IEEE Computer*, Vol. 28, No. 12, pp. 27-37, 1995

[2] S. Darbha and D. P. Agrawal, "Optimal Scheduling Algorithm for Distributed - Memory Machines," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 1, pp. 87-95, 1998.

[3] M. R. Gray and D. S. Johnson, "Computers and Interactability: A Guide to Theory of NP-Completeness," W.H. Freeman and Company, 1979.

[4] A. Gereasoulis and T. Yang, "A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors," *Journal of Parallel and Distributed Computing*, Vol. 16, pp. 276-291, 1992.

[5] C. L. McCreary, A. A. Khan, J. J. Thompson, and M. E. McArdle, "A Comparison of Heuristics for Scheduling DAGs on Multiprocessor," *Proceedings of Eighth International Conference on parallel Processing*, pp. 461-451, 1994.

[6] Y-K. Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocation Task Graphs to Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 5, pp. 506-520, 1996.

[7] T. Yang and A. Gerasoulis, "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 9, 1994.

[8] I. Ahamd and Y.K. Kwok, "A New Approach to Scheduling Parallel Program using Task Duplication," *Proceedings of International Conference on Parallel Processing*, Vol. II, pp. 46-51, 1994.

[9] H. Chen, B. Shirazi, and J. Marquis, "Performance Evaluation of a Novel Scheduling Method: Linear Clustering with Task Duplication," *Proceedings of International Conference on Parallel and Distributed Systems*, pp. 270-275, 1993.

[10] S. Darbha and D.P. Agrawal, "A Task Duplication Based Scalable Scheduling Algorithm for Distributed Memory Systems," *Journal of Parallel and Distributed Computing*, Vol. 46, pp. 15-26, 1997.

[11] G. L. Park, B. Shirazi, and J. Marquis, "DFRN: A New Approach for Duplication Based Scheduling for Distributed Memory Multiprocessor Systems," *Proceedings of Parallel Processing Symposium*, pp. 157-166, 1997.

[12] B. Shi, H-B. Chen, and J. marquis, "Comparative Study of Task Duplication Static Scheduling versus Clustering and Nonclustering Techniques," *Concurrency: Practice and Experiences*, Vol. 7, No. 5, pp. 371-389, 1995.

[13] D.E. Culler et. al., "Parallel Computing on the Berkeley NOW," *9th Joint Symposium on Parallel Processing*, Japan 1997.

[14] W. M. Lin, Q. Gu, and W. Xie, "DCP-NOW: A DCP-based Task Scheduling Technique for Networks of Workstations," *PDPTA'98 International Conference*, pp. 1034-1040, 1998.

[15] X. Du and X. Zhang, "Coordinating Parallel

Processes on Networks of Workstations," *Journal of Parallel and Distributed Computing*, Vol. 46, pp. 125-135, 1997.

[16] J.P. Kitajma and B. Plateau, "Building Synthetic Parallel Programs: The Project (ALPES)," *Proceedings of the IFIP WG 10.3 Workshop on Programming Environments for Parallel Computing*, pp. 161-170, 1992.

강 오 한

1982년 경북대학교 전자공학과 졸업(학사). 1984년 한국과학기술원 전산학과 졸업(석사). 1992년 한국과학기술원 전산학과 졸업(박사). 1984년 ~ 1994년 (주)큐닉스컴퓨터 연구원. 1994년 ~ 현재 안동대학교 컴퓨터교육과 교수. 관심분야는 병렬처리, 클러스터 시스템, 이동 에이전트 시스템.