계산 그리드를 위한 효율적인 작업 스케줄링 정책

(An Efficient Job Scheduling Strategy for Computational Grid)

조 지 훈 † 이 원 주 ††
(Ji Hun Jo) (Won Joo Lee)

전 창 호 †††
(Chang Ho Jeon)

요 약 본 논문은 그리드 환경에서 동적계획법을 위한 새로운 스케줄링 정책을 제안한다. 이 스케줄링 정책의 특징은 동적 테이블의 지역성을 고려하여 테이블을 분할하고, 네트워크 지연이 최소인 노드에 작업을 할당함으로써 작업의 실행시간을 단축하는 것이다. 또한 최적의 병행성을 얻기 위해 분산 하향식으로 동적 테이블을 구성한다. 시뮬레이션을 통하여 제안하는 그리드 스케줄링 정책이 그리드 환경의 성능 향상 면에서 기존의 분기-한정(branch-bound) 알고리즘에 비해 더 효과적임을 보인다.

키워드 : 그리드, 동적계획법, 동적테이블

Abstract In this paper, we propose a new scheduling strategy for dynamic programming in Grid environment. The key idea of this scheme is to reduce the execution time of a job by dividing the dynamic table based on the locality of table and allocating jobs to nodes which minimize network latency. This scheme obtains optimal concurrency by constructing the dynamic table using a distributed top-down method. Through simulation, we show that the proposed Grid strategy improves the performance of Grid environment compared to previous branch-bound strategies.

Key words : Grid, Dynamic Programming, Dynamic Table

1. 서 론

그리드 컴퓨팅은 지리적으로 분산된 수많은 이기종 컴퓨팅 자원과 대용량 저장장치 등을 인터넷으로 상호 공유할 수 있는 통합 환경을 제공한다[1]. 이러한 환경에서는 분산자원을 이용하여 단백질 분석과 같이 방대한 시간이 소요되는 문제들을 처리할 수 있다. 이러한 문제들 중 일부분은 최적해를 구하는 것이다. 최적해 문제는 다수의 해들 중에 가장 적합한 해를 찾는 것으로 분기 한정(branch-bound), 동적계획법(dynamic programming)과 같은 방법으로 최적해를 찾을 수 있다.

분기-한정 알고리즘은 하나의 문제를 각 하위 문제로 분할하여 해결한다. 각 하위 문제는 서로 독립성을 유지하고, 병행성이 높기 때문에 병렬 및 분산 컴퓨팅뿐만 아니라 그리드 환경에서도 많은 연구를 진행하고 있다. 동적계획법은 재귀 관계식을 정의하고 동적 테이블을 구성하여 최적해를 구하는 것이다. 이 방법은 n2의 동적 테이블 공간이 필요하다. 또한, 대부분의 문제를 다항식 시간 복잡도 내에 해결할 수 있지만 병행성을 확보하기 어려워 병렬 및 분산 처리가 어렵다. 그러나 그리드 환경이 가지는 대용량의 데이타 수용 능력은 동적계획법에서 요구하는 동적 테이블을 구성하는데 적합하다. 또한 그리드 환경은 동적 테이블 구성에 필요한 방대한 연산 능력을 가지고 있기 때문에 동적계획법에 적합하다.

그리드 환경에 동적계획법을 적용할 경우, 다음과 같은 점을 고려해야 한다. 첫째, 동적계획법은 독립성을 가진 하위 문제로 분할하기 어렵기 때문에 병행성이 부족하다는 점이다. 둘째, 그리드는 인터넷 환경이기 때문에 병렬 및 분산 환경 보다 네트워크 지연이 크다는 점이다. 또한 그리드에서는 대용량 데이타를 분산 수용할 수 있기 때문에 분산된 동적 테이블 간의 통신이 필요하다. 이러한 점들을 고려하여 본 논문에서는 동적계획법에 적합한 계산 그리드 작업 스케줄링 정책을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 그리드 환경의 전반적인 개념과 기존의 그리드 스케줄링에 대하여 설명한다. 3장에서는 제안하는 그리드 스케줄링 정책에 대하여 상세하게 설명한다. 4장에서는 제안한 그리드 스케줄링 정책의 우수성을 검증하기 위한 시뮬레이션 환경과 결과를 분석하고, 5장에서 결론을 맺는다.

· 이 논문은 2008 한국컴퓨터종합학술대회에서 '동적계획법을 위한 계산 그리드 작업스케줄링 정책'의 제목으로 발표된 논문을 확장한 것임

† 정 회 원 : 코리아와이즈넛 연구소 연구원
deadwi@wisenut.co.kr
†† 정 회 원 : 인하공업전문대학 컴퓨터정보과 교수
wonjoo2@inhatc.ac.kr
(Corresponding author임)
††† 종신회원 : 한양대학교 컴퓨터공학과 교수
chj5193@hanyang.ac.kr

논문접수 : 2008년 8월 28일
심사완료 : 2008년 10월 20일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제8호(2008.11)

2. 관련 연구

2.1 그리드 작업 스케줄링

그리드 환경에서 자원 할당을 위한 스케줄링 시스템의 구성은 그림 1과 같다[2].

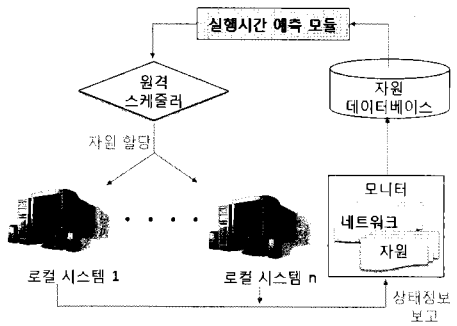


그림 1 그리드 스케줄링 시스템의 구성

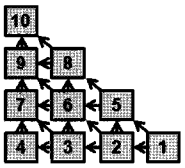
그림 1에서 모니터는 네트워크 모니터와 자원 모니터로 구성된다. 네트워크 모니터는 네트워크의 전송속도 및 사용 상태를 모니터링하고, 자원 모니터는 로컬 시스템의 사용 상태를 모니터링 한다. 원격스케줄러는 그리드 환경의 상태 정보를 바탕으로 현재 그리드에서 수행중인 작업의 실행시간을 예측한다. 그리고 예측한 정보를 바탕으로 대기중인 작업의 실행시간을 최소화할 수 있는 노드에 작업을 할당한다. 그리드 스케줄링 정책은 실시간 작업처리을 필요로 하는 시스템 등에서 주로 사용한다. 작업완료에 대한 실패율은 낮지만 상대적으로 컴퓨팅 자원 비용이 높다는 것이 단점이다. 또한 스케줄러의 성능은 스케줄링 정책의 최소실행시간 예측모듈의 정확성에 따라 결정된다. 하지만 이질적이고 다양한 자원이 분산된 그리드 환경에서는 응용프로그램의 실행시간을 예측하는 것은 매우 어려운 일이다[3].

이러한 그리드 스케줄링 정책 중에는 분기-한정 알고리즘에 특화된 스케줄링 정책이 있다[4]. 이 정책은 태스크 그래프와 같은 정적인 자원을 할당하는 것과 다르게 분기-한정 알고리즘에 기반한 문제에 한정하여 필요한 컴퓨팅 자원을 동적으로 할당할 수 있도록 그리드 환경에 적용한 것이다.

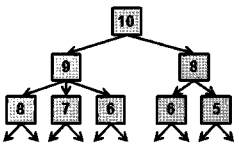
2.2 동적계획법과 동적 테이블 구성 방법

동적계획법은 각 문제의 최적해 구조를 분석하여 재귀 관계식을 정의하고, 상향식으로 동적 테이블을 구성하여 최적해를 찾는다. 동적계획법은 분기-한정 알고리즘 보다 빠른 시간 내에 최적해를 구할 수 있지만 최적성의 원칙을 만족시켜야 한다.

동적계획법은 일반적으로 그림 2(a) 상향식 접근 방법으로 동적 테이블을 구성한다. 하지만 그림 2(b) 하향식 접근 방법으로 변형한다면 독립성을 가진 하위문제



(a) 상향식 접근 방법



(b) 하향식 접근 방법

그림 2 동적 테이블 구성 방법

로 분할 가능하기 때문에 병행성을 확보할 수 있다[5,6]. 하향식 접근 방법은 최종적으로 필요한 동적 테이블의 값 계산을 먼저 시도한다. 그리고 이 값을 계산하기 위해 필요한 동적 테이블의 다른 값들을 재계산한다. 이러한 과정을 반복하여 테이블 값 계산을 위해 다른 값들이 필요 없을 때까지 계산한다. 이 방법은 재귀 알고리즘과 유사하지만 각각 계산된 동적 테이블의 값을 테이블에 저장하는 점이 다르다. 이러한 정보를 바탕으로 중복 계산을 방지함으로써 기존의 상향식 접근 방법과 같은 시간복잡도를 유지할 수 있다.

3. 그리드 작업 스케줄링 정책

제안하는 그리드 작업 스케줄링 정책은 동적 테이블 분산 할당 과정과 동적 테이블 구성 과정으로 나눈다.

동적 테이블 분산 할당 과정은 동적 테이블을 분할하여 특정 노드에 할당하는 과정이다. 이때 각 노드는 마스터 노드와 슬레이브 노드로 분류한다. 마스터 노드는 동적 테이블을 슬레이브 노드에 분산 할당하는 기능을 한다. 동적 테이블 구성 과정은 분산된 동적 테이블을 구성하는 과정이다. 동적 테이블을 할당 받은 각 노드는 자신의 테이블을 구성하는 연산을 수행한다.

3.1 동적 테이블 분산 할당 방법

동적계획법의 재귀 관계식은 표 1과 같이 동적 테이블 값을 계산하기 위해 인접 노드의 테이블 값을 요구한다.

표 1을 살펴보면 (i, j)값을 계산하기 위해서 {(m,

표 1 재귀 관계식에서의 요구 값

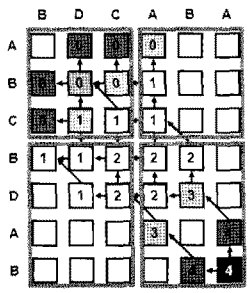
적용 문제	테이블 (i, j)을 구하기 위해 필요 값
LCS	{(i, j-1), (i-1, j)} or (i-1, j-1)
최적 이진 검색 트리	(i, j-1), (i-1, j)
연쇄 행렬 곱셈	{(i, n)| i≤n≤j }, {(m, j) | i≤m≤j }
Needleman-Wunsch	(i, j-1), (i-1, j), (i-1, j-1)

※ LCS(Longest Common Subsequence)

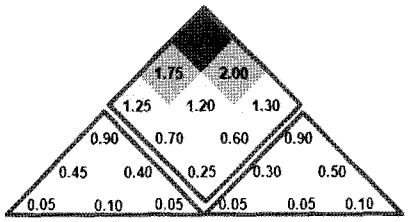
n)|i-1≤ m ≤ j, i-1 ≤ n ≤ j} 값이 필요함을 알 수 있다. 모든 동적계획법으로 해결하는 문제들이 (m, n)값만을 요구하는 것은 아니지만 대부분의 문제들이 동적 테이블의 값을 계산하기 위해 근접한 값을 요구한다. 즉, 테이블의 각 값들은 지역성(locality)을 가진다. 동적 테이블 값이 지역성을 유지할 수 있는 고정된 크기의 단위로 테이블을 분할하여, 각 노드에 할당한다면 분산된 동적 테이블간의 데이타 요청이 감소하기 때문에 통신비용을 줄일 수 있다. 이러한 동적 테이블 분할은 위의 (m, n) 값을 고려하면 간단히 정사각형 모양으로 분할하여 접근할 수 있다.

그림 3은 동적 테이블 분할 예이다. 그림 3의 (a) LCS 문제의 동적 테이블 분할은 사각형 형태로 테이블을 분할한 후, 4개의 분할된 테이블 사이에서 총 9번의 테이블 값 요청이 발생한다. 분할된 테이블 크기가 S라고 하면 한 테이블에서 다른 테이블로 요청하는 값의 수는 sqrt(S)~sqrt(S)*2 정도이다.

그림 3의 (b)는 최적 이진 검색 트리의 동적 테이블 분할 예제이다. 이 예제 또한 사각형 형태로 분할하고 마지막 위치의 값들은 사각형 절반 크기의 삼각형으로 분할된다. 3개의 분할된 테이블 사이에서 총 6번의 테이블 값 요청이 발생하고, 분할된 테이블 크기 S에서 한 테이블에서 다른 테이블로 요청하는 값의 수는 sqrt(S)이다. 이러한 방법으로 테이블을 분할한다면 sqrt(table size)~sqrt(table size)*3 횟수의 참조로 분할된 테이블간의 통신량을 줄일 수 있다. 분할된 동적 테이블은 네트워크 지연을 최소화 할 수 있는 노드에 할당한다.



(a) LCS(Longest Common Subsequence)



(b) 최적 이진 검색 트리

그림 3 동적 테이블 분할 예

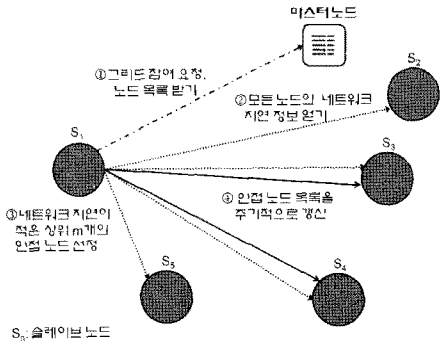


그림 4 그리드 환경의 정보 획득 및 갱신 과정

동적 테이블 할당 과정은 그림 4의 그리드 환경의 정보 획득 및 갱신 과정과 그림 5의 테이블 할당 과정으로 나누어진다.

그림 4에서 마스터 노드는 그리드에 참여하는 노드를 관리하며 동적 테이블을 슬레이브 노드에 분산 할당하고, 그 정보를 관리한다. 이 때 두 가지 사항을 고려해야 한다. 첫째, 인터넷 기반의 그리드 환경이기 때문에 동적으로 변하는 네트워크 지연을 고려해야 한다. 둘째, 분산된 동적 테이블 간의 연관 관계를 고려해야 한다. 또한 마스터 노드가 하나일 경우 동적 테이블 할당 정보에 대한 요청이 집중되어 그리드 환경의 성능이 저하될 수 있다. 따라서 동적으로 그리드에 참여하는 노드 수의 변화를 고려하여 다수의 마스터 노드를 두어야 한다. 그리고 결함허용을 고려하여 마스터 노드에서 장애가 발생하면 다른 마스터 노드로 대체할 수 있도록 다수의 마스터 노드에서 할당 정보를 관리해야 한다. 슬레이브 노드는 마스터 노드를 제외한 그리드에 참여하는 노드로 동적 테이블을 할당 받고, 동적 테이블의 값들을 계산하여 테이블을 구성한다.

그림 4 그리드 환경의 정보 획득 및 갱신 과정은 다음과 같은 과정을 수행한다.

① 슬레이브 노드 S_1은 마스터 노드에 그리드 환경 참여 요청을 하고, 마스터 노드로부터 노드 목록을 받는다.
② 모든 인접 노드의 네트워크 지연 정보를 얻는다.
③ 모든 인접 노드에서 네트워크 지연이 적은 상위 m개의 노드를 인접 노드(S_2~S_5)로 선정한다.
④ 인접 노드(S_2~S_5)의 목록을 주기적으로 갱신한다.

그림 5 테이블 할당 과정은 다음과 같은 과정을 수행한다.

① 연관된 테이블이 할당된 노드 S_1, S_2, S_4에 인접 노드 목록을 요청한다.
② 네트워크 지연이 최소인 인접 노드 S_3를 선정한다.
③ 마스터 노드에 테이블 할당을 요청한다.
④ 해당 노드 S_3에 테이블을 할당한다.

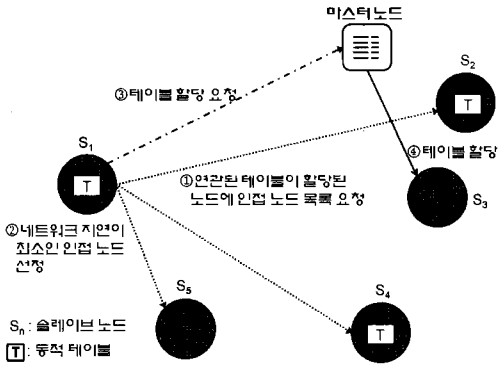


그림 5 테이블 할당 과정

3.2 분산 하향식 동적 테이블 구성

각 노드에 할당된 동적 테이블은 큐와 스택을 이용하여 그림 6의 과정에 따라 동적테이블을 재구성한다.

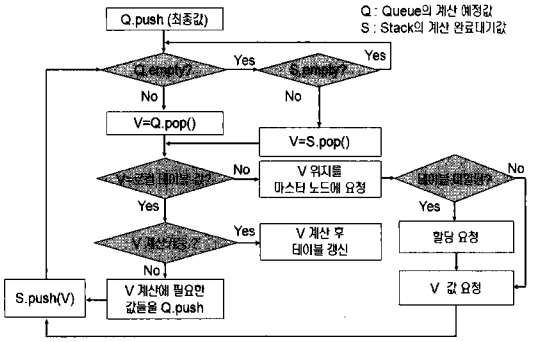


그림 6 분산 하향식 동적 테이블 구성 흐름

그림 6에서 큐(Q)는 동적 테이블의 계산 예정값을 보관하고, 스택(S)은 계산 완료대기값을 보관하면서 동적 테이블 값을 확장한다. Q가 empty 상태이면 스택(S)에 저장된 값들을 사용하여 계산하고, Q가 empty 상태가 아니면 큐(Q)에 저장된 값을 추출한다. 이러한 값들이 로컬 테이블의 값이면 계산한 후 테이블을 갱신하지만 로컬 테이블의 값이 아니면 마스터 노드에 그 값의 위치를 요청한다. 이때 동적 테이블 할당이 가능하면 할당 요청을 하지만 반대의 경우에는 값만 요청한다.

3.3 그리드 시스템 구조

마스터 노드와 슬레이브 노드로 구성된 그리드 시스템 구조에 그리드 스케줄링 정책을 적용할 경우, 각 노드 간의 통신 흐름을 표현하면 그림 7과 같다.

그림 7의 그리드 시스템 구조에서 마스터 노드는 구현된 제안 정책을 그리드 서비스로 제공하고, 슬레이브 노드는 구현된 프로그램을 실행함으로써 그리드에 쉽게 참여할 수 있다[7].

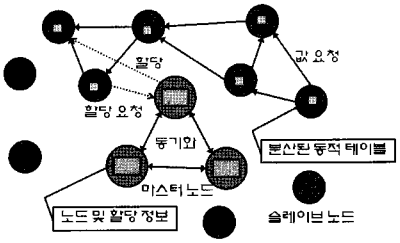


그림 7 그리드 시스템 구조

4. 시뮬레이션

시뮬레이션을 통하여 그리드 환경에서는 기존의 분기-한정 알고리즘에 비해 동적계획법의 성능이 우수함을 검증한다.

4.1 시뮬레이션 환경

시뮬레이션에서는 SIMGRID[8] toolkit을 이용하여 LAN, WAN으로 구성된 그리드 환경을 구축한다. 본 논문에서 제안한 스케줄링 정책은 C++의 STL(Standard Template Library)와 ACE(Adaptive Communication Environment) 미들웨어를 사용하여 구현한다. ACE 미들웨어는 멀티 플랫폼을 지원하는 고성능 객체지향 분산 객체 처리 프레임워크로 C++기반의 객체화된 다양한 네트워크 및 시스템 처리 기능을 제공한다[9].

응용프로그램의 실행시간은 펜티엄III 1Ghz에서 표2의 분기-한정 알고리즘의 하위문제나 동적계획법의 동적테이블 값을 계산하는데 소요되는 시간을 측정하여 구한다. 네트워크 환경은 10Mbps 버스로 연결된 3개의 LAN과 이를 각각 100Mbps로 1:1 연결한 WAN으로 구성된다. 모든 링크는 5.7×10^{-3}s의 네트워크 지연을 가진다.

표 2 동적 테이블 구성 방법

노드	마스터 노드 수	1
	슬레이브 노드 수(N_s)	10~105
LCS 문제 [분기 한정]	문제 크기(n)	30~50K, 100K
	전체 하위문제 수(P_{total}) (i: 하위문제 번호)	$\Sigma 2^i$
	가지치기 비율(r)	0.7
	적용 하위문제 수	$P_{total}*(1-r)$
	하위문제당 계산 소요시간	$2.5*10^{-8}$s
LCS 문제 [동적계획법]	전체 테이블 크기(T_{total})	n^2
	채우기 비율(r')	0.6
	분산 테이블 크기	$T_{total}*r'/N_s$
	한 테이블 값당 계산 시간	$5.0*10^{-7}$s
	노드별 값 요청 횟수	$sqrt(T_{total}/N_s)*2$

4.2 시뮬레이션 결과 분석

LCS 문제를 대상으로 분기-한정 알고리즘과 동적계획법의 문제 크기에 따른 총소요시간을 측정한다.

그림 8은 노드 수가 21 일 때 문제 크기에 따른 총소요시간을 측정한 결과이다. 그림 8을 살펴보면 분기-한정(BB)의 경우 문제 크기가 40 이상에서는 총소요시간이 급격히 증가함을 알 수 있다. 하지만 동적계획법(DP)의 총소요시간은 거의 변함이 없다. 따라서 문제 크기 40 이상에서 최적해를 구하는 것은 동적계획법이 더 유리함을 알 수 있다.

그림 9는 노드 수가 21일 때 문제 크기에 따른 계산 점유율을 측정한 결과이다.

그림 9를 살펴보면 분기-한정(BB) 알고리즘은 문제 크기 45 이상에서는 CPU 자원을 약 100% 쓰고 있지만 동적계획법(DP)은 CPU 자원 점유율이 일정함을 알 수 있다.

문제 크기가 100K일 경우 노드 수에 따른 동적계획법의 총소요시간을 측정한다. 그 결과는 그림 10과 같다.

그림 10을 살펴보면 그리드 환경에 참여하는 노드 수가 증가할수록 총소요시간은 감소한다. 하지만 노드 수 66개 이상에서는 총소요시간이 감소하지 않고 일정 시간으로 수렴함을 알 수 있다. 이것은 노드 수 증가에 따른 분산 테이블의 수 증가로 인해 병행성이 감소하고, 네트워크 오버헤드가 증가하기 때문이다. 노드 수에 따른 분기-한정(BB)의 총소요시간은 동적계획법의 총소요시간과 비교할 수 없을 정도로 편차가 큰 값을 가지기 때문에 그림 10에 표현하지 않았다.

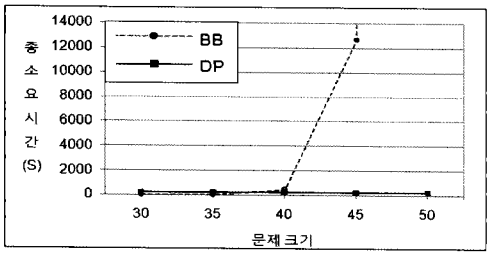


그림 8 문제 크기에 따른 총소요시간

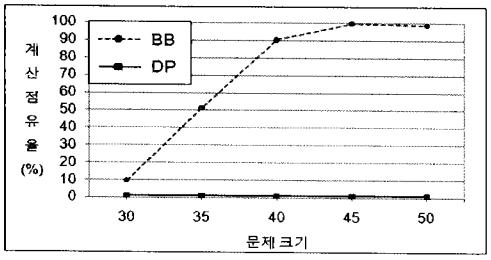


그림 9 문제 크기에 따른 계산 점유율

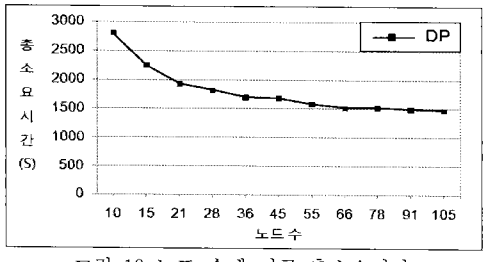


그림 10 노드 수에 따른 총소요시간

5. 결 론

본 논문에서는 동적계획법에 적합한 계산 그리드 작업 스케줄링 정책을 제안하였다. 이 정책은 지역성을 고려하여 동적 테이블을 분할하고, 네트워크 지연을 고려하여 동적 테이블을 할당한다. 시뮬레이션을 통하여 기존의 분기-한정 알고리즘에 비해 본 논문에서 제안하는 작업 스케줄링 정책이 우수함을 검증하였다.

기존의 분기-한정 알고리즘은 문제 크기가 증가하면서 총소요시간과 CPU 점유율이 급격하게 증가함을 알 수 있다. 하지만 본 논문에서 제안한 그리드 작업 스케줄링 정책은 문제 크기가 증가하여도 총소요시간과 CPU 점유율이 크게 변하지 않았다. 따라서 그리드 환경에서는 기존의 분기-한정 알고리즘에 비해 본 논문에서 제안하는 작업 스케줄링 정책이 우수함을 알 수 있었다.

참 고 문 헌

[1] I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal of Supercomputer Applications, 15(3), 2001.

[2] 류경후, 이원주, 전창호, "다중 그리드 사이트에서 어플리케이션 특성을 고려한 동적 작업 재배치 정책", 한국컴퓨터정보학회 논문지, 제13권, 제4호, pp. 31-37, 2008.

[3] Paul Ruth, Junghwan Rhee, Dongyan Xu, Rick Kennell, and Sebastien Goasguen, "Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure," In the 3rd IEEE International Conference on Autonomic Computing (ICAC'06), 2006.

[4] A. Iamnitchi and I.Foster. "A Problem-Specific Fault-Tolerance Mechanism for Asynchronous, Distributed Systems," Proceedings of the 2000 International Conference on Parallel Processing, 2000.

[5] Tgomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, "Introductiontoalgorithms," The MIT Press Publishers USA, 1999.

[6] Bergroth L., Hakonen H. and Raita T. "A survey of longest common subsequence algorithms," String Processing and Information Retrieval, 2000.

[7] I.Foster, H.Kishimoto, A.Savva, D.Berry, "The Open Grid Services Architecture, Version 1.0," Informational Document, Global Grid Forum (GGF), Jan., 29, 2005.

[8] SIMGRID Project, http://simgrid.gforge.inria.fr

[9] The Adaptive Communication Environment(ACE), http://www.cs.wustl.edu/~schmidt/ACE.html