계산 그리드를 위한 고성능 작업 스케줄링 정책

조지훈[O], 김준상[*], 이원주[**], 전창호[***]
한양대학교 컴퓨터공학과[O *], 두원공과대학 모바일인터넷과[**], 한양대학교 컴퓨터공학과[***]
{jojh[O], kimjs[*]}@cse.hanyang.ac.kr, wonjoo@doowon.ac.kr[**], chjeon@cse.hanyang.ac.kr[***]

A High Performance Job Scheduling Policy for Computational Grid

Jihun Jo[O], Junsang Kim[*], Wonjoo Lee[**], Changho Jeon[***]
Dept. of Computer Science & Engineering, Hanyang University[O],
Dept. of Computer Science & Engineering, Hanyang University[*],
Dept. of Mobile Internet, Doowon Technical College[**],
Dept. of Computer Science & Engineering, Hanyang University[***]

요 약

그리드 컴퓨팅은 방대한 데이터 저장 공간과 고성능 연산능력을 요구하는 작업에 적합하다. 동적계획법(dynamic programming)은 방대한 크기의 동적 테이블(dynamic table)을 구성하여 최적해(optimal solution)을 찾기 때문에 그리드에서 수행하기에 적합한 작업이다. 본 논문에서는 동적 테이블을 구성하고 분산 배치하는 할당 정책을 제안한다. 그리고 동적계획법 기반의 어플리케이션을 그리드에서 효율적으로 수행할 수 있는 그리드 시스템 구조를 제안한다.

1. 서론

대용량의 데이터 저장 능력과 고성능 연산 능력을 가진 그리드 컴퓨팅이 차세대 병렬 컴퓨팅으로 주목 받고 있다[1]. 그리드 환경에서는 이질적인 환경과 성능을 가진 다양한 자원들이 인터넷으로 연결되어 있으며, 고성능의 연산능력을 요구하는 응용프로그램을 실행한다. 고성능 연산능력이 요구되는 문제들은 해결 방법에 따라 추가적인 요구 사항을 갖는다. 문제 해결 과정에서 많은 양의 임시 데이터를 산출하는 작업은 단일 노드에 많은 양의 데이터를 저장하기 힘들기 때문에 그리드 환경에서 처리하는 것이 효율적이다. 이러한 문제를 해결하는 방법에는 동적계획법(dynamic programming) 과 분기한정(branch & bound) 알고리즘 등이 있다.

동적계획법은 최적화 문제(Optimization problems)에서 동적 테이블(dynamic table)을 구성하여 최적해(optimal solution)를 구하는 알고리즘이다. 일반적으로 동적 테이블의 크기는 n^2 이다. 따라서 문제의 크기가 크다면 단일 노드에서 처리하기 힘든 크기의 동적 테이블 구성을 요구한다. 반면에 분기한정 알고리즘은 최적해를 구하기 위해 동적 테이블 구성을 위한 저장 공간이 필요 없다. 동일한 문제에 대해서 분기한정 알고리즘이 지수함수의 시간복잡도를 가진다면 동적계획법에서는 다항식의 시간 복잡도를 가진다. 이러한 시간 복잡도 차이로 인해 동일한 문제를 해결함에 있어서 동적계획법을 이용한다면 좀 더 빠른 시간내에 최적해를 구할 수 있다.

동적계획법을 그리드 환경에서 처리하기 위해서는 먼저 방대한 크기의 동적 테이블을 각 노드에 분산 저장한다. 그리고 각 노드에 작업을 할당하여 수행한다. 이때 그리드 시스템의 성능 향상을 위해 효율적인 그리드 스케줄링 정책은 필수적이다.

따라서 본 논문에서는 동적계획법에 의한 작업을 처리할 수 있는 효율적인 그리드 스케줄링 정책을 제안한다. 이 스케줄링 정책은 동적 테이블을 각 노드에 할당하여 각 노드의 테이블에 접근하기 위해 발생하는 네트워크 오버헤드를 줄인다.

2. 관련 연구

일반적으로 분기한정 알고리즘을 적용하여 검색 문제의 해를 쉽게 찾을 수 있다. 이 알고리즘은 특성상 분기된 하위문제(subproblem)가 다른 하위문제와 독립성을 가지므로 병렬 및 분산 처리가 용이하다. 따라서 분기한정 알고리즘은 병렬 및 분산 컴퓨팅에서 기존의 많은 연구 성과가 있었으며 이를 바탕으로 그리드 환경에 적용하는 연구가 진행중이다[2]. 특히 수천 개의 노드가 동적으로 참여하는 그리드 환경에서 오류 복원을 고려하는 방향으로 연구 되고 있다[3].

동적계획법은 각 문제의 최적해 구조를 분석하여 재

귀 관계식을 정의하고 이를 바탕으로 상향식으로 동적 테이블을 구성하여 최적해를 찾는다. 동적계획법은 분기한정보다 빠른 시간 안에 최적해를 구할 수 있지만 최적화 문제에만 적용 가능하다. 그리고 재귀 관계식에 따라 서로 의존성을 갖는 테이블 값은 병렬 및 분산 처리를 어렵게 하는 요인이다. 이러한 특성 때문에 병렬 처리 분야에서는 파이프 라인이나 네트워크 구조 등을 변형시켜 문제에 접근하거나, 특정 문제에 대하여 동적계획법을 분석하고 병렬처리를 가능하도록 병렬 구조를 설계하는 연구가 진행되어 왔었다[4][5][6]. 반면에 분산 처리 분야에서는 방대한 동적 테이블의 크기와 네트워크 오버헤드 등의 문제로 인해 제한적인 연구가 있었다.

이러한 동적계획법의 문제를 해결하기 위해 본 논문에서는 방대한 데이터 저장 공간과 고성능 연산 능력을 가진 그리드 컴퓨팅을 적용한다.

3. 동적 테이블 구축 방법

동적계획법은 상향식으로 동적 테이블을 구축하기 때문에 분기한정 알고리즘과 같이 하위문제가 명확하게 구분되지 않아 병렬 및 분산처리에 어려움이 있다. 본 연구에서는 특정 문제에 국한되지 않고 대부분의 동적계획법 기반의 문제에 적용하기 위해 하향식으로 접근하여 병렬 및 분산처리한다.

3.1. 상향식 접근 방법

상향식 접근 방법은 동적계획법을 통해 최적해를 구하는 대표적인 문제인 Matrix-chain multiplication를 통하여 자세히 설명할 수 있다. 식 (1)은 동적 테이블을 구성하기 위한 재귀 관계식이다.

$$m[i,j]=\begin{cases}0 & \text{if } i=j \\ \min\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\} & \text{if } i<j\end{cases} \quad \cdots(1)$$

식 (1)의 재귀 관계식은 그림 1과 같은 상향식 접근 방법으로 테이블을 구성하기 위해 순차적으로 계산하기 때문에 병렬 및 분산 처리가 어렵다.

```
for(l=1;l<n;l++){
   for(i=0;i<n-l;i++){
      j=i+l; m[i][j]=-1;
      for(k=i;k<=j-1;k++){
         q=m[i][k]+m[k+1][j]+p[i]*p[k+1]*p[j+1];
         if(m[i][j]==-1 || q<m[i][j])
            m[i][j]=q; s[i][j]=k;
      }
   }
}
```

그림 1. 상향식 Matrix-chain multiplication 코드

3.2. 하향식 접근 방법

하향식 접근방법은 최종적으로 필요한 동적 테이블의 특정 값 계산을 먼저 시도한다. 그리고 이 특정 값을 계산하기 위해 필요한 테이블의 다른 값들을 다시 계산한다. 이러한 과정은 테이블의 다른 값들이 필요하지 않을 때까지 반복하여 계산한다.

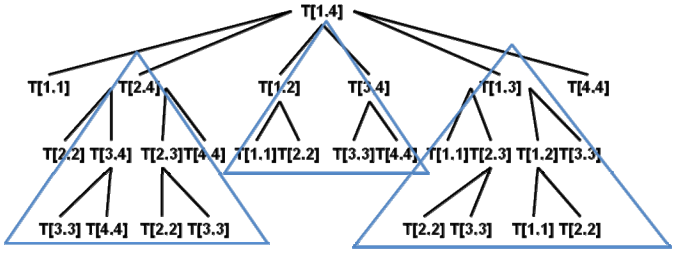

그림 2. 하향식 문제 접근

그림 2는 Matrix-chain multiplication 문제를 하향식 접근방법으로 해결하는 예이다. T[1, 4]는 동적 테이블 1열 4행에 위치한 값을 계산하기 위해 하향식으로 문제를 풀어 가는 과정이다. T[3, 3]에 위치한 값은 동일한 테이블 값을 반복해서 요구한다. 이것을 방지하기 위해 이미 계산 완료된 테이블 값은 동적 테이블에 저장한다. 이러한 중복 계산을 방지함으로써 상향식 동적계획법과 같은 시간복잡도를 갖는다.

그림 1의 상향식 접근방법의 예는 그림 3과 같이 하향식 동적계획법[7]으로 변형할 수 있다.

```
lookup_chain(m, s, p, i, j)
{
   if(m[i][j] != EMPTY) return m[i][j];
   if(i==j){
      m[i][j]=0;
   }else for(int k=i,q;k<=j-1;k++){
      q=lookup_chain(m,s,p,i,k)
        +lookup_chain(m,s,p,k+1,j)
        +p[i]*p[k+1]*p[j+1];
      if(m[i][j]==-1 || q<m[i][j])
         m[i][j]=q, s[i][j]=k;
   }
   return m[i][j];
}
```

그림 3. 하향식 Matrix-chain multiplication 코드

그림 3에서 lookup_chain 함수는 두 번 호출되며 사용하는 인자는 독립적인 값으로서 함수 호출 순서를 바꾸어도 문제가 없다. 그림 2의 삼각형과 같이 재귀적으로 호출하는 lookup_chain 함수들을 하나로 묶는다면 다른 삼각형 안의 함수들은 서로 직접적인 의존성을 가지지 않기 때문에 병렬 및 분산 처리 가능하다.

4. 제안하는 그리드 스케줄링 정책

동적계획법은 동적 테이블을 구성하는 부분이 가장 주된 작업이다. 테이블의 크기는 일반적으로 n^2, $n^2/2$의 크기를 가지므로 MB 단위의 입력을 처리하기 위해서는 GB 단위의 테이블을 구성해야 한다. 단일 노드에 방대한 크기의 테이블이 할당되는 것을 방지하기 위해 테이블을 그리드의 각 노드에 분산 할당한다. 그리고 각 노드는 할당 받은 동적 테이블의 값을 계산하여 전체 동적 테이블을 완성한다.

하지만 이러한 방법은 테이블을 각 노드에 분산 할당하기 때문에 테이블 값을 계산하기 위해 다른 노드의 테이블 값을 요구하면 네트워크 오버헤드가 증가한다.

4.1. 동적 테이블 구성

동적 테이블 구성 과정은 그림 4과 같다.

```
//재귀 관계식 R로부터 T[i, j] 값 계산을 위해 필요한 값 추출
valueList=extract(R, i, j);
foreach value as valueList
{
  // 자신의 테이블일 경우 계산
  if isLocalTableValue(value)
    calculate(value);
  else
  {
    // 해당 값을 가지는 테이블 ID 획득
    table=getTableID(value);
    // 마스터 노드에게 할당된 노드 정보 요청
    location=getTableLocation(table);
    if(location==null)
    {
      // 할당된 노드가 없을 경우 노드 선정 후 테이블 할당
      location=selectNewNode();
      requestAllocateTable(location,table);
    }
    // 해당 노드에게 값 계산 요청
    requestCalculateValue(location,value);
  }
}
// 필요 값들이 계산 완료될 때까지 대기
waitForCalculation(valueList)
calculate(T[i, j])
```

그림 4. 동적 테이블 구성 과정

그림 6에서 하향식으로 동적 테이블을 구성하기 위해 최종적으로 필요한 동적 테이블의 값 T[i, j]을 계산한다. 먼저 extract() 함수는 재귀 관계식 R로부터 T[i, j]를 계산하기 위해 필요한 값들을 추출한다. isLocalTableValue() 함수는 이렇게 추출한 값이 로컬 노드의 동적 테이블에 속하는 값인지를 판별한다. 로컬에 속하는 경우 calculate() 함수를 통해 값을 계산한다. 그렇지 않다면 getTableLocation() 함수를 통해서 그 값을 포함하는 동적 테이블이 어떤 노드에 할당되어 있는지 마스터 노드에 질의한다. selectNewNode() 함수는 아직 해당 동적 테이블이 할당되어 있는 않는 경우, 테이블 할당 정책를 통해 노드를 선정한다. 선정된 노드는 requestAllocateTable() 함수를 이용하여 마스터 노드에게 테이블 할당을 요청한다. 동적 테이블을 할당하고 그 노드 위치를 알면 requestCalculateValue() 함수를 통해 해당 노드에게 계산을 요구한다. 추출한 모든 값들을 계산했다면 이를 바탕으로 계산하고자 했던 T[i, j]를 계산한다.

4.2. 테이블 할당 정책

동적계획법을 위한 재귀 관계식은 해당 동적 테이블 값을 계산하기 위해 인근의 테이블 값을 요구하는 특성을 가진다. Matrix-chain multiplication의 재귀 관계식을 살펴보면 m[i, j]을 구하기 위해서 m[i, i], m[i, i+1], …, m[i, j-1]과 m[i+1, j], m[i+2, j], …, m[j-1, j] 값들이 필요하다. 이러한 값들은 테이블에서 계산하고자 하는 값의 좌측과 하단에 위치한다. 또한 이 각각의 값을 계산하기 위해서는 좌측과 하단의 값을 계산해야 한다. 이와 같이 각 테이블의 값을 계산하기 위해서는 근접한 값을 요구하며 멀리 동떨어진 값을 요구하는 경우는 적다. 즉 테이블의 각 값들은 지역성(locality)을 가지며 동일한 지역성을 가지는 값들이 요구된다. 따라서 테이블 값이 지역성을 유지할 수 있는 고정된 크기 단위로 테이블을 분할하여 각 노드에 할당한다면 분산된 테이블 간의 데이터 요청이 줄어들기 때문에 통신 비용을 줄일 수 있다.

동적 테이블은 그림 5와 같이 여러 개의 분산된 동적 테이블로 나눌 수 있다.

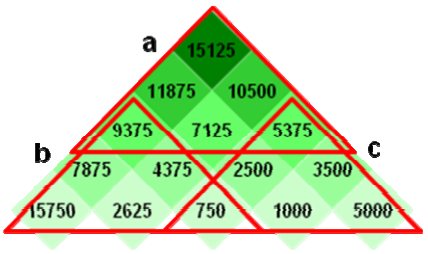

그림 5. Matrix-chain multiplication의 동적 테이블 분할 예

그림 5에서 삼각형으로 표현한 a, b, c는 분할된 테이블을 의미한다. 분할된 테이블에서 좌측과 하단 값들을 요구하여 계산한다.

분산된 테이블 간의 통신비용을 줄일 수 있도록 각 노드에 할당하기 위해서는 두 가지 사항을 고려해야 한다. 첫째, 인터넷 기반의 그리드 환경인 만큼 유동적으로 변하는 네트워크 지연 속도를 고려해야 한다. 둘째, 테이블 할당 정보 관리를 위한 마스터(master) 노드가 필요하다는 것이다. 마스터 노드는 테이블 할당 정보를

관리하는 기능을 한다. 마스터 노드가 하나일 경우 할당 정보에 대한 요구가 단일 노드에 집중되어 성능 저하가 발생할 수 있다. 또한 그리드 환경에서는 동적으로 그리드에 참여하는 노드 변화를 고려하여 다수의 마스터 노드를 두어야 한다. 그리고 결함허용을 고려하여 1~2 개의 마스터 노드에 장애가 발생하면 다른 마스터 노드로 대체할 수 있도록 다수의 마스터 노드에서 할당 정보를 관리해야 한다. 이러한 점을 고려하여 각 테이블을 할당하는 할당 알고리즘은 그림 6와 같다.

단계 1) 새로운 노드가 그리드에 참여 할 때 그리드의 모든 노드와의 통신 지연 속도를 구한다.
단계 2) 그 중 일부 지연 속도가 적은 m개의 노드에 대한 정보를 인접 노드 목록에 보관한다.
단계 3) 일정 시간 간격으로 인접 노드들에 대한 지연 속도를 갱신한다.
단계 4) 이미 할당된 K개의 테이블과 인접하는 새로운 테이블을 할당하는 경우
- 각 할당된 테이블을 가진 K개의 노드에게 인접 노드 목록을 요청한다.
- K개의 인접 노드 목록에서 네트워크 지연 속도가 최소가 되도록 최적의 노드를 선정한다.
- 선정된 노드에 테이블을 할당하도록 마스터 노드에 요청한다.

그림 6. 테이블 할당 알고리즘

4.3. 제안 그리드 시스템

그리드 시스템은 그림 7과 같이 마스터 노드와 슬레이브(slave) 노드로 구성된다.

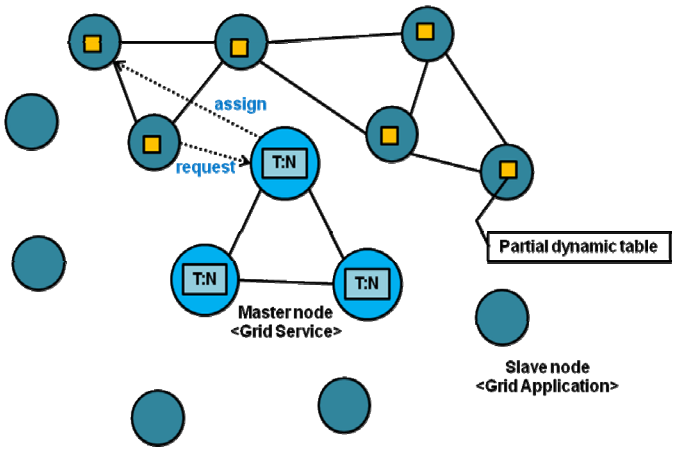

그림 7. 제안 그리드 시스템 구성도

그림 7에서 마스터 노드들은 동적 테이블의 어느 부분이 어떤 노드에 할당되었는지에 대한 정보를 관리하며 슬레이브 노드에게서 테이블 생성에 대한 요청을 받아 테이블을 할당해주는 역할을 한다. 슬레이브 노드는 할당된 동적 테이블의 부분을 보관하고 테이블 값을 계산하는 기능을 한다.

5. 결론 및 향후 연구

본 논문에서는 방대한 크기의 동적 테이블을 구성하여 최적해를 찾는 동적계획법(dynamic programming)을 그리드 환경에 적용하였다. 그리고 병렬 및 분산처리의 한계점과 네트워크 오버헤드 및 지연을 줄일 수 있는 그리드 시스템 구조와 테이블 할당 정책을 제안하였다. 그 결과 동적계획법을 그리드 환경에 적용하면 분기한정 알고리즘에 비해 빠른 시간내에 최적해를 구할 수 있음을 알 수 있었다.

향후 연구 과제는 Globus[8][9] 기반의 그리드 환경을 구축하여 LCS(longest-common-subsequence)[10]와 같은 문제에 제안한 스케줄링 정책을 적용하는 것이다. 그리고 기존의 분기한정 알고리즘과 성능을 비교 평가해 보는 것이다.

6. 참고 문헌

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

[2] Raphael Finkel and Udi Manber "DIB: a distributed implementation of backtracking", ACM Transactions on Programming Languages and Systems, 1987.

[3] 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.

[4] C. Rodriguez, J. Roda, F. Garcia, F. Almeida and D. Gonzalez, "Paradigms for parallel dynamic programming", Proceedings of the 22nd EUROMICRO Conference, 1996.

[5] Canto S.D., de Madrid A.P. and Bencomo S.D., "Parallel dynamic programming on clusters of workstations", Parallel and Distributed Systems, IEEE Transactions , 2005

[6] El Baz and M.E.D., "Load balancing in a parallel dynamic programming multi-method applied to the 0-1 knapsack problem", Parallel, Distributed, and Network-Based Processing, 2006.

[7] Tgomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, "*Introduction to algorithms*", The MIT Press Publishers USA, 1999.

[8] Globus Project, *http://www.globus.org/*

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

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