인터넷 기반 병렬 컴퓨팅에서 중간 관리자의 구성과 결함포용 기법*

김홍수[O] 강인성[†] 최성진[†] 황일선[‡] 황종선[†] 유헌창[∥]
고려대학교 컴퓨터학과 분산시스템 연구실[†], 한국과학기술정보연구원[‡] 고려대학교 컴퓨터교육과[∥]
{hera[O], iskang[†], lotieye[†], hwang[†]}@disys.korea.ac.kr, his@kisti.re.kr[‡], yuhc@comedu.korea.ac.kr[∥]

A Volunteer Manager Organization and Fault-Tolerance Scheme in Internet-Based Parallel Computing

Hong-Su Kim[O], In-Sung Kang, Sung-Jin Choi, Chong-Sun Hwang
Dept. of Computer Science & Engineering, Korea University
il-Sun Hwang
Supercomputing center, KISTI
Hyun-Chang Yoo
Dept. of Computer Science@Education, Korea University

요 약

인터넷 기반 병렬 컴퓨팅은 인터넷에 연결된 수많은 컴퓨팅 자원들을 이용하여 고성능 컴퓨팅 성능을 요구하는 병렬 연산을 수행할 수 있는 컴퓨팅 패러다임이다. 그러나, 자원 제공자에 의해 제공된 자원들의 관리와 작업 할당 및 관리가 모두 중앙 관리 서버에 의해 수행됨으로 인해 서버의 부하가 발생한다. 이러한 문제점을 해결하기 위해 기존 연구들은 복수개의 중간 관리자를 두어 해결하려 했으나 연산에 대한 안정적인 수행을 보장하지 못한다. 중간 관리자들의 선정 및 구성 기법과 중간 관리자의 결함 포용에 대해서는 다루지 않았다. 이에, 본 논문에서는 인터넷 기반 병렬 컴퓨팅 환경에서 중앙 관리 서버의 부하를 줄이고 연산의 안정적 수행을 보장하는 결함 포용적 중간 관리자 구성 기법을 제안하고자 한다.

1. 서론

인터넷 기반 병렬 컴퓨팅은 인터넷에 연결된 개인 사용자들의 컴퓨팅 자원을 이용하여 고성능의 컴퓨팅 성능을 창출할 수 있는 컴퓨팅 패러다임이다. 인터넷 기반 병렬 컴퓨팅은 자원 제공자의 가입과 탈퇴가 자유롭게 발생할 수 있다. 또한, 자원 제공자의 자원은 인터넷 환경에 연결되어 연산이 수행된다. 따라서, 인터넷이라는 동적인 환경에서 안정적인 연산을 수행 할 수 있도록 보장해 주어야한다.

초기에 인터넷 기반 병렬 컴퓨팅 환경에서 자원 제공자의 작업 할당 및 관리는 중앙 관리 서버에 의해 수행되었다. 따라서, 인터넷 환경에서 모든 작업을 중앙 관리 서버가 처리함으로 인해 서버의 부하가 발생한다. 이러한 문제를 해결하기 위해 Bayanihan[1], SETI@home[2], Javelin[4], ATLAS[7]같은 연구들은 복수개의 중간 관리자를 두어 해결하려 했으나 중간 관리자의 결함 포용에 대해서 연산의 지속적인 수행을 보장하지 못하는 문제점이 발생한다. 본 논문에서는 중앙 관리 서버의 부하를 줄이기 위한 중간 관리자 구성 기법과 연산의 지속적인 수행을 보장하는 결함 포용 기법을 제안하고자 한다.

2. 관련연구

Bayanihan[1]은 필리핀의 공동체 정신을 인터넷 기반 병렬 컴퓨팅 분야에 적용시킨 프로젝트이다. Bayanihan은 전체 작업 할당을 위해 "열심인 스케줄링(eager scheduling)"[5]을 사용하여 연산 결과에 대한 정확성을 제공하기 위해 "방해 행위 포용(Sabotage Tolerance)" 기법인 "대다수의 투표(Majority Voting)"와 "결점 미리 검사(Spot-checking)"기법을 사용한다. 중앙 관리 서버의 부하를 줄이기 위해 지역성을 고려한 지역 서버를 사용하지만 지역 서버에 대한 선정 방법과 결함이 발생했을 때 포용할 수 있는 기법은 다루고 있지 않다.

SETI@home[2]은 미국 버클리 대학에서 우주 주파수를 분석해 외계 지성체의 존재를 찾아내려는 프로젝트이다. 인터넷에 연결된 수백만의 자원 제공자들은 클라이언트 프로그램을 다운 받아 서버로부터 일을 받아 처리한 후 그 결과를 데이터 서버로 보내는 방식으로 전체 연산을 수행한다. 만약 결함이 발생한다면 로컬에 저장된 검사점을 이용해 계속적인 연산을 수행하는 방식으로 전체 연산에 대한 신뢰성을 제공한다. 그러나, 이러한 경우에 작업자들의 컴퓨터가 살아날 때까지의 연산 지연이 발생하며 모든 작업 할당 및 작업 관리가 중앙 관리 서버에서 이루어짐으로 인해 부하가 발생한다.

Javelin[4]은 미국 캘리포니아 대학에서 자바를 사용하여 인터넷 기반 병렬 시스템을 개발하는 프로젝트이다. Javelin 역시 인터넷에 연결된 수백만의 컴퓨팅 자원들을 이용하여 고성능을 요구하는 연산을 수행하기 위해 시작되었다. Javelin에서는 작업 할당 및 작업 스케줄링을 수행하는 분산 브로커라는 개념을 도입하였지만 브로커의 선택 방법이 제시되어 있지 않고 브로커가 결함이 발생했을 때 지속적인 연산 수행을 보장하지 못한다.

3. 시스템 모델

본 논문에서 가정하는 시스템 환경은 그림 1과 같은 인터넷 기반 병렬 컴퓨팅 환경으로 중앙 검사점 서버와 중앙 관리 서버, 중간 관리자들 그리고 작업 노드들로 구성된다. 중앙 검사점 서버는 각 작업 노드들의 검사점을 중간 관리자를 통해서 수집하여 저장한다. 중앙 관리 서버는 전체 수행할 작업을 분할하여 각 작업 노드들에게 분배하고 연산 수행 시 필요한 데이터를 저장하고 전체 작업에 대한 작업 트리를 유지한다. 중간 관리자는 중간 검사점 저장소, 중간 데이터 저장소, 작업 관찰자 역할을 수행하고 작업 노드들은 중앙 관리 서버로부터 분배된 작업을 자신의 유휴 상태가 되면 할당된 작업을 수행한 후 결과를 중앙 관리 서버에게 되돌려주는 역할을 한다.

본 논문에서 가정하는 인터넷 기반 병렬 컴퓨팅 환경에서 중간 관리자와 작업 노드는 가입과 탈퇴가 자유롭다. 따라서, 노드들은 고정적인 수가 아닌 유동적인 수로 구성된다. 본 논문의 환경에서는 "적응 병렬성(adaptive parallel)"[3]모델을 기반한 기법 중 "열심인 스케줄링(eager scheduling)"[5] 알고리즘을 이용하여 해당 그룹의 지역 풀을 관리하는 것을 가정한다. 또한, 중간 관리자는 항상 작업노드들의 요청을 받을 수 있어야하므로 언제나 제공되는 자원의 50%를 사용한다고 가정한다.

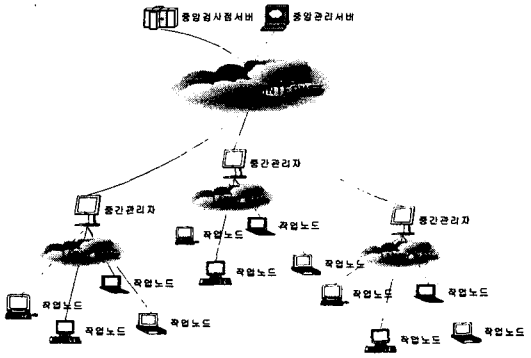

그림 1. 인터넷 기반 병렬 컴퓨팅 환경

* 본 논문은 정보통신부 초고속응용기술지원사업의 일부 연구비 지원 (G-03-GS-02-04X-6)에 의해서 수행되었음

4. 제안 기법

4.1 중간 관리자 선정 및 구성 기법

중앙 관리 서버는 자원 제공자들의 연산 참여시 자원 제공자들의 프로파일 $W_i^{profile} = (w_i^{unique}, w_i^{cpu}, w_i^{mem}, w_i^{disk}, w_i^{type}, w_i^{priority}, w_i^{localcode})$에 따라 중간 관리자 선정 알고리즘을 사용하여 자원 제공자 W_i^{worker}들 중에서 중간 관리자 C_m^{coordi} 들을 선택한다. 중간 관리자 선택은 자원 제공자 중에서 CPU 처리능력, 메모리 용량, 하드디스크 용량을 고려하여 선택하고 나머지는 작업 노드로 한다. 그림 2는 중간 관리자와 작업 노드 선정 알고리즘을 나타낸다. 예를 들어, 중간 관리자의 자격 요건 중 CPU는 자원 제공자들의 프로파일에서 각자의 CPU 양을 모두 더해 자원 제공자수로 나눈 평균값 보다 크거나 같아야 만족된다. 중앙 관리 서버는 주어진 작업을 처리하기 위해 선정된 중간 관리자와 작업 노드들을 그림 3과 같은 작업 트리 G_k^{tree} 형태로 구성한다. 작업 트리는 중앙 관리 서버를 최상위로 하고 중간 노드는 프로파일의 지역 코드와 우선 순위가 높은 것을 바탕으로 중간 관리자를 구성한다. 끝 노드는 중간 노드의 지역 코드를 바탕으로 작업 노드들을 구성한다.

```
n = W_i^num;
W_i^profile = ( w_i^cpu, w_i^mem, w_i^disk, w_i^type, w_i^priority );
w_i^priority - w_i^cpu + w_i^mem + w_i^disk;
if w_i^cpu >= ( Σ_{v=1}^{v=n} w_v^cpu / n ) then
   if w_i^mem >= ( Σ_{v=1}^{v=n} w_v^mem / n ) then
      if w_i^disk >= ( Σ_{v=1}^{v=n} w_v^disk / n ) then
         w_i^type - "coordinator";
      else w_i^type - "worker";
   else w_i^type = "worker";
else w_i^type = "worker";
fi
```

그림 2. 중간 관리자 및 작업 노드 선정 알고리즘

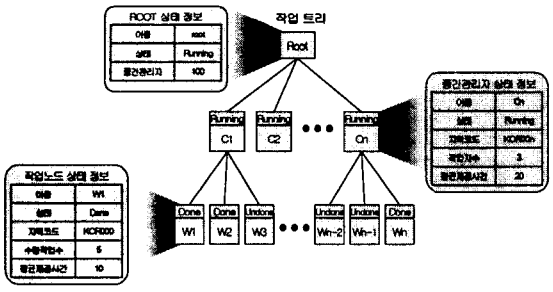

그림 3. 작업 트리

중간 관리자는 다음과 같이 세 가지의 역할을 수행한다.

- 중간 검사점 저장소의 역할
  중간 검사점 저장소에서 그룹의 검사점을 모아 중앙 검사점 서버로 보냄으로써 중앙 검사점 서버로의 병목 현상을 감소시킨다.
- 중간 데이터 저장소의 역할
  중간 데이터 저장소는 지역성을 고려한 중간 데이터 저장소를 둠으로써 서버에서의 전달 지연 시간을 줄인다.
- 작업 관찰자의 역할
  그림 4와 같이 지역성이 높은 중간 작업 관찰자를 둠으로써 지역 풀을 통해 빠른 재 작업 할당으로 전체적인 연산 수행 시간을 줄일 수 있다.

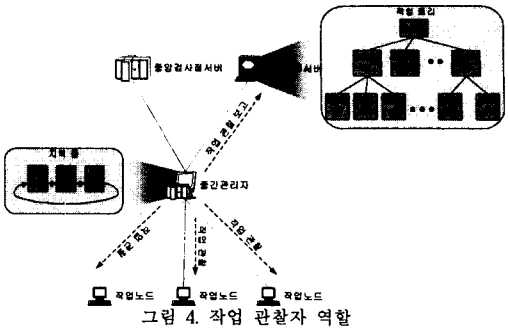

그림 4. 작업 관찰자 역할

4.2 중간 관리자와 작업 노드의 결함 발생 시 해결 알고리즘

중간 관리자의 결함 포용 기법은 그림 4와 같이 서버에서 작업 트리, G_k^{tree}를 구성함으로써 중간 관리자의 결함 발생 시 작업 트리가 지역 풀, C_j^{pool}들의 작업 정보를 유지하고 있다가 새로이 선택된 중간 관리자에게 작업 정보를 주게된다. 다음은 중간 관리자와 작업 노드의 결함 발생 시 해결하는 알고리즘을 설명한다.

- 중간 관리자의 결함 발생 시 알고리즘

첫째로, 그림 5와 같이 중간 관리자의 결함을 서버가 발견했을 때 우선, 서버는 결함이 발생한 중간 관리자의 작업 노드들 결함 발생 메시지를 보낸다. 그런 후, 서버는 연산 능력과 지역 코드를 바탕으로 새로운 중간 관리자를 선정한다. 새로운 중간 관리자에게 그룹의 풀 정보 C_j^{pool}, 작업 데이터 W_i^{data}, 작업자 목록 W_i^{worker}을 준다. 마지막으로 서버는 작업 노드들에게 새로운 중간 관리자를 그룹의 작업 노드들에게 알린다. 둘째로, 그림 6과 같이 중간 관리자의 결함 C_j^{failed}을 작업 노드가 발견했을 때는 일단 서버에게 중간 관리자가 결함이 발생했다고 메시지를 보낸다. 서버는 중간 관리자와 작업 노드 사이의 결함(네트워크 단절, 작업 노드의 에러 등)이면 다른 작업 노드에게 작업을 할당하고 중간 관리자가 결함이라면 그림 5의 알고리즘을 수행한다.

```
if ∃ c_j^failed then
   sendMsgtoWorkers("Coordinator is failed", c_j^failed )
   for( i=0 ; i < W_i^num ; i++ )
         if compareTo( w_i^type , "coordinator" ) then
                  selectNewCoordinator( w_i^unique );
            break;
         fi
   sendPoolInformationtoNewCoordinato1( C_j^pool );
   sendDatatoNewCoordinator( W_i^data );
   sendWorkersListtoNewCoordinator( W_i^worker );
   sendMsgtoWorkers("New Coordinator is running" , c_j^url );
fi
```

그림 5. 서버가 중간 관리자의 결함을 발견했을 때 포용 기법

```
if ∃ c_j^failed then
   sendMsgtoServer("Coordinator is failed", c_j^failed);
   if checkBetweenWorkerandCoordinator() then //작업자의 에러
         sendMsgtoServer("Coordinator is running", c_j^running);
   else                                          //중간 관리자 결함
         sendMsgtoServer("Excute Algorithm 1");
   fi
fi
```

그림 6. 작업 노드가 중간 관리자의 결함을 발견했을 때 포용 기법

● 작업 노드의 결함 발생 시 알고리즘

작업 노드의 결함 w_i^{failed} 이 발생했을 때는 그림 7과 같은 알고리즘을 통해 포용하게 된다. 먼저, 작업 노드의 결함을 중간 관리자가 발견했을 때 새로운 작업을 수행할 수 있는 작업 노드가 지역 풀 $C_j^{pool}(w)$에 있는지 찾는다. 만약 있으면, 중간 관리자는 지역 풀을 갱신하고 서버에서 작업 트리에 재 작업 할당에 대한 정보를 갱신시킨 후 작업을 받아 새로운 작업 노드에게 보낸다. 만약 없다면, 서버에게 결함이 발생한 작업 노드의 유일한 식별자를 서버에게 보내 새로운 작업 노드를 선정하여 최근 검사점을 이주시켜 계속적인 작업을 할 수 있도록 한다.

```
if ∃ w_i^failed then
    if ∃ C_j^pool( w ) then        //만약 지역 풀에 작업자가 있으면,
            w_i^new = selectNewWorkerinPool( C_j^localcode , "done" );
            sendMsgtoServer( "update Migration to new Worker" ,
                    w_i^unique , w_i^new );
            migrationCheckpointtoNewWorker( w_i^point );
    else                           //서버에게 할당 요청
            sendMsgtoServer( "Migrate to New Worker" , w_i^unique );
    fi
fi
```

그림 7. 작업 노드의 결함 발생 시 포용 기법

5. 모의 실험

본 논문에서 제안하는 중간 관리자 구성 기법은 기존에 중앙 관리 서버에서 전체 작업 노드들을 관리하는 것과 비교해 볼 때 지역성을 고려한 중간 관리자를 둠으로써 전달 지연과 서버에 대한 트래픽을 줄일 수 있다. 10~1000Mbps의 대역폭을 가지는 LAN에서의 지연 시간(Latency)은 보통 1~10ms를 가지고 0.101~600Mbps의 대역폭을 갖는 WAN이나 0.010~2Mbps 대역폭을 갖는 인터넷 환경에서의 지연 시간은 100~500ms를 갖는다[8]. 제안하는 시스템의 성능(Performance)은 전달 지연과 서버에 대한 트래픽(Traffic)만을 고려하여 다음과 같이 나타낸다.

● 전달 지연 측정

중앙 관리 서버와 작업 노드만으로 구성된 기존 기법의 전체 전달 지연 시간을 L_{old}라하고 중간 관리자 구성 기법에서의 전체 전달 지연 시간은 L_{my}이다. 전체 작업 노드들의 수를 n, 하나의 그룹에서의 작업 노드의 수를 m, 전체 중간 관리자의 수는 l이다. 그러면, 작업 노드(W)에서 중앙 관리 서버(S)까지의 전달 지연은 L_{WS}이고 작업 노드에서 중간 관리자(C)까지의 전달 지연은 L_{WC}이다. 그리고 중간 관리자로부터 중앙 관리 서버까지의 전달 지연은 L_{CS}이다. 전체 연산에 대한 전달 지연을 수식으로 표현하면 다음과 같다.

$$L_{old} = \sum_{w=1}^{w=n} L_{WS}(w)$$

$$L_{my} = \sum_{w=1}^{w=m} L_{WC}(w) + \sum_{w=1}^{w=l} L_{CS}(w)$$

WS간에는 인터넷 환경이고 WC간에는 LAN 또는 WAN 환경, 그리고, CS간에는 인터넷 환경이기 때문에 그림 8과 같이 노드의 수가 늘어날수록 중간 관리자 구성 기법에서의 전달 지연이 상대적으로 작음을 볼 수 있다.

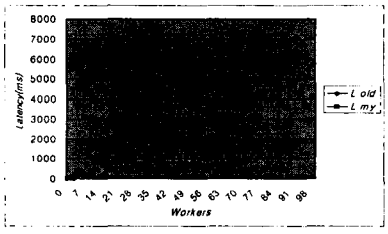

그림 8. 전달 지연 시간 측정

● 트래픽 측정

중앙 관리 서버와 작업 노드만으로 구성된 기존 기법의 전체 트래픽을 T_{old}라하고 중간 관리자 구성 기법에서의 전체 트래픽은 T_{my}이다. 중앙 관리 서버와 작업 노드만으로 구성된 기존 기법의 트래픽을 T_{WS}()라하고 중간 관리자를 구성했을 때 작업 노드에서 중간 관리자 사이의 트래픽을 T_{WC}(), 중간 관리자에서 서버까지의 트래픽은 T_{CS}()이다. 그러면 전체 연산 트래픽을 계산해 보면 다음과 같다.

$$T_{old} = \sum_{w=1}^{w=n} T_{WS}(w)$$

$$T_{my} = \sum_{w=1}^{w=m} T_{WC}(w) + \sum_{w=1}^{w=l} T_{CS}(w)$$

트래픽은 초당 전송되는 비트의 양(bps)으로 나타낼 수 있다. 따라서, 작업 노드가 늘어날수록 P_{old}는 높은 트래픽이 발생하는 반면 P_{my}는 상대적으로 낮은 트래픽을 발생시킨다. 이것을 그래프로 그려보면 그림 9와 같다.

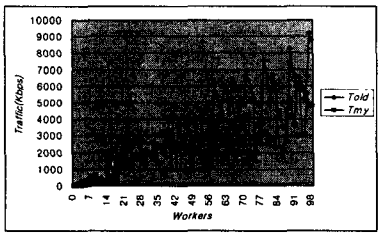

그림 9. 트래픽 측정

6. 결론 및 향후 연구 과제

본 논문에서는 인터넷 기반 병렬 컴퓨팅 환경에서 중앙 관리 서버의 부하를 줄이고 안정된 연산 수행 보장을 위해 중간 관리자가 결함이 발생하였을 경우에 결함 포용 기법을 제안하였다. 제안 기법은 서버의 부하로 인해 발생하는 연산 지연시간을 줄이며 중간 관리자와 작업 노드의 결함을 포용하여 연산의 안정적 수행을 보장한다. 향후에 전체 연산에 대한 수행 시간을 측정하여 비교 분석을 할 계획이다.

앞으로 제안된 기법은 현재 진행중인 KOREA@home[6] 프로젝트에서 구현하고자 한다.

참고 문헌

[1] L. F. G. Sarmenta, S. Hirano. "Bayanihan: Building and Studying Web-Based Volunteer Computing Systems Using Java". Future Generation Computer Systems, Oct. 1999.

[2] SETI@Home homepage. http://setiathome.ssl.berkeley.edu

[3] D. Gelernter, D. Kanminsky, "Supercomputing out of recycled garbage". Preliminart experience with Piranha. Processings of the 1992 ACM International Conference of supercomputing. July 1992

[4] B. O. Christiansen, P. Cappello, M. F. Ionescu, M. O. Neary, K. E. Schauser, D.Wu, "Javelin: Internet-Based Parallel Computing using Java", 1996.

[5] P. Dasgupta, Z. Kedem, and M. Rabin. "Parallel processing on networks of workstations : A fault-tolerant high performance approach" , in *Proc. 15th IEEE International Conference on Distributed Computing Systems*, 1995.

[6] KOREA@Home homepage. http://www.koreaathome.co.kr

[7] J. E. Baldeschwieler, R. D. Blumofe, and E. A. Brewer. "ATLAS: An Infrastructure for Global Computing". In Proceedings of the Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications, 1996.

[8] G. Coulouris, J. Dollimore and T. Kindberg, "Distributed Systems Concepts and Design", Addison-Wesley, third edition, 2001.