이동 에이전트를 이용한 효율적인 PC 클러스터 시스템의 설계 및 구현

최종욱° 김영균 오길호
금오공과대학교 컴퓨터공학부
{jwchoi, ygkim, gilho}@cespc1.kumoh.ac.kr

Design and Implementation of an Efficient PC Cluster Using Mobile Agent

Jong-Wook Choi° Young-Gyun Kim Gil-Ho Oh
School of Computer Engineering, Kumoh National Institute of Technology

요 약

적은 비용으로 고성능 컴퓨팅 시스템을 구축하는 방법중의 하나로 클러스터링이라는 방법이 도입되어 그에 관련된 많은 연구와 방법들이 적용되고 있다. 하지만 베어울프와 같은 대표적인 기존의 고성능 연산 클러스터링의 방법은 주로 동일한 운영체제 환경에서 클러스터링 노드들의 통신에서는 MPI나 PVM등의 병렬처리용 라이브러리에 국한되어 있다. 이러한 방법은 서로 이질적인 네트워크 환경에서 특정 기능의 모듈을 동적으로 확장하거나 적용하는 것에 대해서는 사용자나 관리자의 많은 수동적인 노력을 필요로 하고 있다. 이에 본 논문에서는 네트워크상에서의 자바 기반의 이동 에이전트를 이용하여 서로 다른 이질적인 시스템들에 대한 확장과 이동이 용이하고 실시간 노드들의 정보를 수집하여 제안한 클러스터링 알고리즘을 적용하여 병렬처리 가능한 작업들의 분배 및 처리할 수 있는 이동 에이전트 기반의 클러스터링 시스템을 연구하였다.

1.서론

최근 처리할 데이터 크기의 증가와 더불어 컴퓨터 성능에 관한 관심이 늘어감에 따라 컴퓨터의 속도 개선에 관한 연구가 점차 늘어가고 있는 상황이다. 이러한 컴퓨터 성능을 높이는 방법중의 하나로 여러 대의 컴퓨터를 하나로 묶어 마치 하나의 컴퓨터처럼 동작하도록 서로 연결함으로써, 병렬처리 부하 분산 및 고장 대비 목적에 사용할 수 있는 클러스터링에[1] 관한 연구가 활발히 진행중이다. 클러스터링 시스템은 프로그램을 여러 부분으로 나누어 여러 프로세스가 각 부분을 동시에 수행하여 할당된 작업을 처리할 수 있으며 이러한 프로세스간의 통신을 가능하게 하는 대표적인 라이브러리로 국제적인 표준인 MPI(Message Passing Interface)[2]를 들 수 있다. 그러나 이러한 라이브러리들을 사용하여 클러스터링 환경을 구축하는 기존 방법들의 문제점은 서로 다른 이기종간의 시스템에서는 마스터와 슬레이브 간의 작업 분할과 노드들의 확장이 어렵다는 것이다. 이에 본 논문에서는 지역 네트워크 내의 이질적인 환경에서도 수행이 가능하며 어떠한 노드들에서도 마스터로서의 역할로 수행이 가능한 플랫폼에 독립적인 자바 기반의 이동 에이전트(Mobile Agents)[3]를 이용하여 클러스터링 시스템을 설계,구축하였다.

제안한 시스템은 자바 기반의 이동 에이전트 시스템으로서 실행중에도 작업을 분배하는 역할을 수행하는 마스터 노드와 분배된 작업을 처리할 수 있는 슬레이브 노드들간의 동적인 확장과 이동이 가능하며 사용자 인터페이스를 통한 실시간의 노드정보를 바탕으로 하여 최적화된 클러스터링 알고리즘을 수행할 수 있는 방안을 제시한다. 이와 더불어 이동 에이전트의 특성인 코드 이동성을 이용하여 수행할 코드를 이동시켜 작업을 처리함으로써 각 노드들에서의 사용자 작업 준비시간이 줄어 작업 처리시간을 단축시킴으로써 전체적인 성능을 향상시키면서 좀더 지능적이며 효율적인 방법으로 클러스터링을 구축하여 작업을 처리할 수 있는 방안을 새롭게 연구하였다.

본 논문에서 제안한 방법은 자바 기반의 이동 에이전트를 이용한 동적인 클러스터링 시스템을 구축함으로써 서로 다른 이기종간의 시스템에 대해서도 확장과 이동이 용이하며 이동 에이전트를 이용하여 노드들의 정보를 바탕으로 동적으로 클러스터링 알고리즘을 적용하여 작업을 분배 및 처리할 수 있도록 하였다.

2.이동 에이전트

이동 에이전트란 자율성을 가지고 비동기적으로 수행이 가능한 개체로, 선택적으로 지능을 가지고 특정 작업을 수행하기 위해 네트워크 상의 한 호스트에서 다른 호스트로 이동할 수 있는 개체라고 정의 할 수 있다. 네트워크 상의 호스트는 이러한 이동 에이전트가 실행 할 수 있는 환경을 제공하여 이질적인 하드웨어와 운영체제로 구성된 분산 컴퓨팅 환경에 동적인 이식을 가능케 하여(Portability) 서로 이질적인 분산 환경에서도 에이전트가 실행할 수 있는 환경을 제공하여 주는 에이전트 시스템의 역할을 수행한다. 이러한 에이전트는 특정 작업을 수행하기 위해 자신을 복제(Clone)하여 작업 부하를 분산시킬 수 있으며 자신의 현재 실행 상태를 저장하여 네트워크를 통해 이동하며(Code mobility) 상태를 다시 복원함으로써 해당 작업에 대한 실행을 재개할 수 있어 과부하 된 호스트의 부하를 줄여 수행의 효율성을 높이는 기능을 제공 할 수 있다[4][5].

본 논문에서는 이러한 이동 에이전트의 특성들을 응용하여 지역 네트워크 내의 호스트들에 대한 정보를 실시간으로 감시하여 클러스

터링 알고리즘을 적용하여 작업을 분배, 처리함으로써 좀더 동적으로 확장이 용이한 클러스터링 환경을 설계 및 구축 할 수 있는 장점을 제공한다.

3. 클러스터링 시스템 설계 및 구현

3.1 기본 정책

이동 에이전트를 이용한 동적인 클러스터링 시스템은 노드 그룹핑 알고리즘을 적용하기 위한 인자로서 지역네트워크 내의 노드들의 정보를 CMAM(Clustering with Mobile Agent Manager) 사용자 인터페이스를 통하여 실시간으로 감시할 수 있어야 한다. 또한 등록된 모든 지역 네트워크 내의 Host들에는 플랫폼에 독립적인 자바 기반의 *Agent System*인 *IBM Aglets system*[6]이 설치 되어 있어야 하며 클러스터링 알고리즘을 수행하며 작업을 분배할 수 있는 마스터 에이전트는 어디에서나 실행이 가능하여야 한다. 사용자 인터페이스는 자바의 Swing 라이브러리를 사용하여 그림 1과 같이 구성하였다.

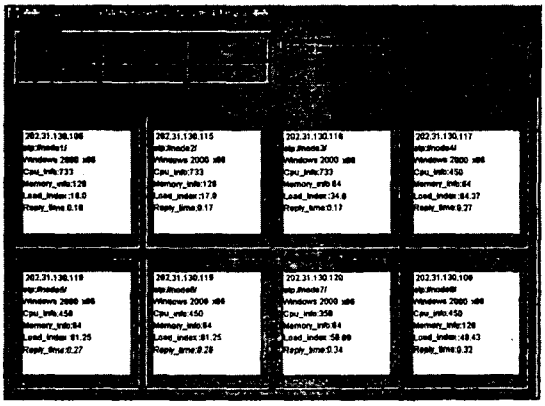
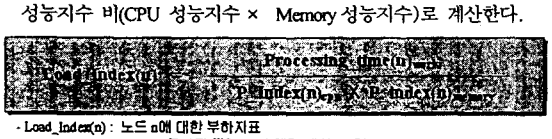


그림 1. Clustering with Mobile Agent Manager GUI

- **Get Node info** : 노드들의 활성화/비활성화 상태정보 및 활성화 노드들의 정보 및 부하지표와 응답시간 획득
- **Apply NGA** : 제안한 노드 그룹핑 알고리즘 적용
- **Select App** : 처리할 작업의 선택
- **Run** : 작업 분배 및 실행

3.2 시스템의 정보 획득 및 노드별 부하지표 계산

- *Get Node Info* 버튼을 통한 동록 노드들의 Active 유무를 확인하여 Master(CMAM)는 현재 Active된 노드들로 NIAMA(Node Information Acquirement Mobile Agent)를 전송하여 NIDB(Node Information DataBase)로부터 노드들의 정보수집과 주어진 작업(행렬계산)에 의해 노드의 현재 부하지표(Load Index)를 계산한다
- 노드들의 현재 부하지표 값은 각각의 노드들의 성능이 다른점을 고려하여 작업(행렬 계산)의 수행시간에 대한 해당 노드들의 성능지수 비(CPU 성능지수 × Memory 성능지수)로 계산한다.



- Load_Index(n) : 노드 n에 대한 부하지표
- Processing_time(n)work : 주어진 작업 처리(행렬 계산) 시간
- P_Index(n)cpu : CPU 성능지수 (0 < P_index(n)memory ≤ 1)
- P_index(n)memory : Memory 성능지수(0 < P_index(n)memory ≤ 1)

- CPU 성능지수는 표준 시스템 성능평가 기관인 SPEC(Standard Performance Evaluation Corporation)[7]의 공인된 정보를 사용하였고 메모리 성능지수는 실험에 참여할 노드중의 최대 메모리 크기의 비율로 실험 및 성능평가를 하였다.

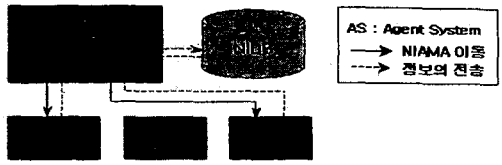


그림 2. 등록 노드들의 정보 획득 과정

3.3 노드들의 정보를 이용한 Node Grouping Algorithm 적용

3.3.1 Half-Division Node Grouping Algorithm

- 지역 네트워크 내의 활성화된(작업 가능한) 노드들의 작업 그룹을 구성하는데 있어서 각기 다른 성능을 가진 노드들로 구성된 각각의 작업 그룹의 성능 차가 최소가 되도록 하여 Master 에서의 작업 분배를 쉽게 할 수 있으며 그 결과 전체 작업 처리 속도와 성능을 향상시킬 수 있도록 하였다.
- ***Parameter***
  - ① 노드별 부하지표 (*Load Index(n)*)
  - ② 노드별 응답시간 (*Reply Time(n)*)
  - Master 노드에서 NIAMA를 각 노드로 전송을 시작할 때의 시간과(Transfer_Start_Time$_{NIAMA(n)}$) NIAMA로부터 각 노드에 대한 정보를 전송 받은 시간 (Response_Receive_Time(n))의 차로한다.

- Reply_time(n) : 노드 n에 대한 응답 시간
- Transfer_Start_Time$_{NIAMA(n)}$: 노드 n에 대한 NIAMA 전송 시작 시간
- Response_Receive_Time(n) : 노드 n 으로부터의 정보 획득 시간

3.3.2 NGPA(Node Grouping Process Agent) 수행

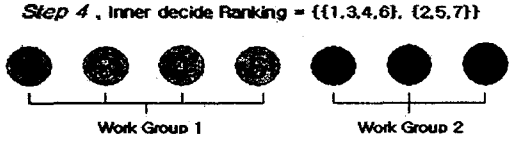
- 적용할 Node Grouping Algorithm을 결정할 Parameter를 입력받아 각각의 작업 그룹에 참여할 노드들을 결정
  - *Parameter 1: Load Index(n),*
  - *Parameter 2 : Reply Time(n)*

- 현재 활성화된(작업을 할 수 있는) 노드들 중에서 Rank Index값이 작은 (노드 부하가 제일 적은) 순으로 정렬
  - *Rank Index = Load Index(n) + Reply Time (n)*
  - 총 활성화된 노드의 수 (*Total Active Node* =7)

Step 1 , Decide Ranking = { 1, 2, 3, 4, 5, 6, 7,}

Step 2 , Grouping1 = Active node number(7) / 2 = 3
= Rank number + 3
= { {1,4}, {2,5}, {3,6}, {7}}

Step 3 , Grouping 2 = { {1,4}, {2,5}, {3,6}, {7}}/2 =2
= Rank Group number + 2
= {{1,4,3,6}, {2,5,7}}



3.4 WDEMA(Work Division & Work Execution Mobile Agent)

작업 분배 및 처리는 분산 스케줄러 방식의 중앙 Manager 노드를 두어 하위 노드에서의 작업 결과 및 처리 시간을 전송 받아 Master에게 전송하는 구조로 성능이 좋은 노드에서의 작업량과 하위 노드에서의 작업량이 비슷하도록 구현하였다.

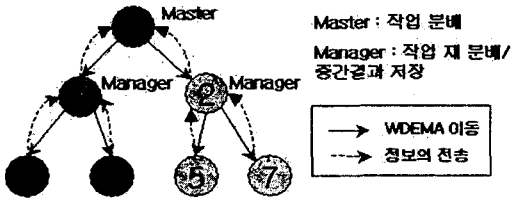


그림 3. WDEMA 수행 과정

4. 결과 및 성능평가

본 논문의 구현에서는 LAN으로 구성된 각기 다른 성능을 가진 8대의 호스트를 대상으로 제안한 알고리즘을 적용하였을 때와 알고리즘을 적용하지 않고 작업을 분배하여 처리하였을 때의 평균 처리 시간과 노드들의 그룹의 수에 따른 작업 처리 시간에 대한 성능평가를 하였다. 처리할 작업은 작업 분할이 가능한 128× 128 Pixel 크기의 Image pixel operating으로 실험 및 성능 평가를 하였다. 구현 환경으로는 에이전트 시스템으로서는 자바 기반의 플랫폼에 독립적이며 자바 2(Swing)를 지원할 수 있는 다중 에이전트 시스템인 IBM의 ASDK(Aglets Software Development Kit) 2.0.2 버전과 Sun Microsystems의 자바 개발 도구 JSDK (Java Standard Development Kit)1.3 버전을 사용하여 구현하였다. 본 논문의 실험에 사용한 노드들의 정보에 따른 성능 지수는 표 1과 같다

표 1. 등록 노드 정보에 따른 성능 지수

1	733	128	19.6(=1.00)	1.00
2	733	128	19.6(=1.00)	1.00
3	733	64	19.6(=1.00)	0.50
4	450	64	12.5(=0.64)	0.50
5	450	64	12.5(=0.64)	0.50
6	450	64	12.5(=0.64)	0.50
7	350	64	11.4(=0.58)	0.50
8	450	128	12.5(=0.64)	1.00

그림 4는 제안한 알고리즘을 적용 시켰을 때와 적용시키지 않았을 때 실험 횟수별 작업 처리 시간의 성능평가 결과의 평균을 나타낸 것이며 그림 5는 그룹의 수에 따른 작업 처리 시간의 성능평가 결과를 나타낸 것이다.

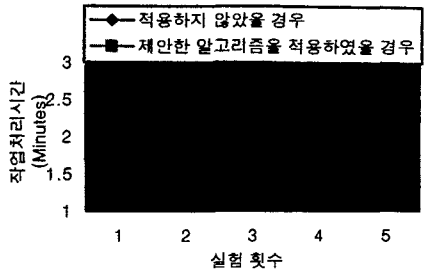


그림 4. 알고리즘 적용 유무에 따른 작업처리시간

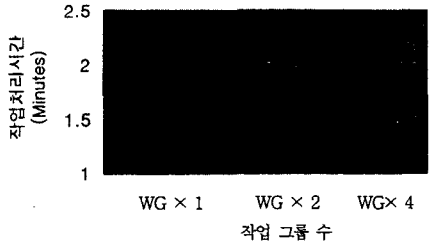


그림 5. 작업 그룹 수에 따른 작업 처리 시간

5. 결론 및 향후 계획

본 논문에서 제안한 시스템은 노드들의 실시간 정보를 수집하여 제안한 노드 그룹핑 알고리즘을 적용, 효율적으로 작업을 배분 및 처리할 수 있도록 하였으며 플랫폼에 독립적인 자바기반의 이동 에이전트를 사용하여 서로 다른 이질적인 시스템에 대해서도 확장이 용이하도록 하였다. 또한 이동 에이전트 특성중의 하나인 코드 이동성을 이용하여 처리할 코드를 이동시켜 처리 함으로써 작업 준비 시간을 단축시켜 전체적인 성능을 높일 수 있도록 설계 및 구현 하였다. 향후 계획으로는 작업 유형에 따른 노드 그룹핑 알고리즘을 각기 다르게 적용시킬 수 있는 이동 에이전트 기반의 클러스터링 시스템을 연구할 계획이다.

참고 문헌

[1] *Rajkumar Buyya* (ed.). High Performance Cluster Computing :Architectures and Systems. Volume 1, 1/e, Prentice Hall PTR,NJ,USA,1999.
http://www.dgs.monash.edu.au/~uajkumar/cluster/index.html

[2] *Peter S.Pacheco*, "Parallel Programming with MPI"

[3] OMG, "Mobile Agent Facility Specification"
http://www.omg.org/, pp12-13, 2000.

[4] *Danny B.Lange , Mitsuru Oshima* , "Programming and Deploying Java Mobile Agents with Aglets"

[5] D. Chess, C. Harrison, A. Kershenbaum. "Mobile agents : Are they a good idea?, In Mobile Object Systems: Towards the Programmable Internet", Vol. 1222 of Lecture Notes in Computer Science, Spring-Verlag, 1997.

[6] Aglets Software Development Kit, http://www.trl.ibm.com/aglets/

[7] The Standard Performance Evaluation Corporation,
http://www.spec.org/