클러스터 환경에서의 부하 분산 시스템 설계 및 구현

박은지° 최민 맹승렬
한국과학기술원 전산학과
{pakej°, mchoi , maeng}@camars.kaist.ac.kr

Design and implementation of cluster load balancing system

Eun-Ji Pak° Min Choi Seung-Ryoul Maeng
Division of Computer Science, Department of EECS, KAIST

요 약

클러스터는 두 대 이상의 컴퓨터를 마치 하나의 시스템처럼 행동하도록 연결하여, 막대한 양의 계산이 필요하거나, 중단 없는 서비스를 수행하고자 할 때 사용하는 시스템이다. 이런 클러스터 시스템에서는 동일한 시간에 한 노드에서 많은 작업이 수행되는 경우 한 노드에만 부하가 몰리게 되고, 이런 부하집중 현상은 전체 시스템의 성능을 저하시킬 수 있다. 따라서 시스템의 상태를 파악하여 부하를 적절히 각 노드에 분배해주어야 할 필요가 있는데, 사용자가 처음 작업을 시작하고자 할 때 가장 적합한 노드를 골라서 수행시켜 주거나, 필요한 경우 로컬 노드에 있는 작업을 다른 노드로 옮겨서 수행하는 방식으로 부하를 분산할 수 있다. 이를 위해 부하 분산 시스템을 설계, 구현하여 부하를 각 노드에 균등하게 분배함으로써 전체 시스템의 성능을 향상시키는 방향으로 작업을 수행하도록 하였다.

1. 서론

클러스터 시스템에서는 높은 성능을 얻고, 전체 자원을 효과적으로 사용하기 위해서는 각 노드의 자원 상태를 파악하고, 현재 수행중인 작업을 적절하게 분배할 필요가 있다. 각 노드의 자원활용 상태를 파악하고, 작업을 잘 분배하는 것은 부하 분산의 기본 기능이며 이를 위해 부하 분산 시스템에서는 전체 클러스터 시스템의 자원 정보를 관리하고, 현재 수행중인 작업들에 대한 정보를 관리함으로써 가장 적합한 노드에서 작업을 수행할 수 있게 함으로써 전체 클러스터 시스템의 성능을 향상시키는 것을 목표로 하고 있다. [1]

시스템의 자원을 항상 감시하여 가장 최근의 자원 정보를 기반으로 사용자가 수행하고자 하는 작업이나, 현재 각 노드에서 수행중인 작업의 정보를 살피고, 적절히 작업을 노드에 분배해주고, 필요한 경우 작업들을 노드 사이에서 이동시켜 수행할 필요가 있다. 이를 위해 각 노드의 자원 정보를 가지고 작업을 분배하게 되는 부하 분산 시스템을 설계 및 구현하였다.

본 논문의 구성은 다음과 같다. 2장에서는 부하 분산 시스템의 기본적인 개요와 다른 부하 분산 시스템들이 갖는 특성에 대해 설명하고, 3장에서는 부하 분산 시스템의 전체적인 구조에 대해 설명하고, 4장에서는 부하분산 시스템의 구현에 대하여 자세하게 설명한다. 5장에서는 벤치마크를 사용한 성능 분석 결과를 살펴보고, 6장에서 결론을 맺는다.

2. 관련 연구

클러스터 시스템에서 부하 분산 기능을 제공하는 시스템은 그 방식에 따라 크게 정적 부하 분산과 동적 부하 분산으로 나눌 수 있다. 정적 부하 분산은 과거의 작업 정보와 자원 정보를 사용하여 부하 분산을 수행하고, 동적 부하 분산은 최근의 작업 정보와 자원 정보를 사용하는 것이다. 정적 부하 분산은 작업의 잘못된 분배로 성능의 저하를 가져올 수 있으므로, 자원이나 작업 정보를 업데이트 하는데 쓰이는 오버헤드를 감안하고도 동적 부하 분산을 많이 사용한다.

동적 부하 분산 시스템은 부하 정보를 얻어오는 방식, 작업을 분배하는 방식, 작업분배 시 사용하는 부하 정보 등 구현하는 데 있어서 여러 다른 정책들이 존재한다. [2]

클러스터 환경 하에서 이러한 부하 분산을 지원하는 시스템은 Condor, PBS, Load Leveler, CODINE등이 있으며 서로 다른 방식의 부하 분산을 제공한다. [3,4,5]

Condor는 사용자가 작성한 프로그램을 수정할 필요가 없고, 실행중인 노드가 fail되었을 때 재 수행이 가능하지만 이를 위한 디스크의 오버헤드가 존재한다. [6]

PBS는 작업을 쉽게 관리할 수 있는 명령어들을 충분히 제공하고 있고, GUI를 제공하여 명령어에 쉽게 접근할 수 있지만, 사용자 레벨에서의 체크포인팅 라이브러리나 마이그레이션이 제공되지는 않는다.

Load Leveler는 사용자 편의에 중심을 둔 설계로써 문서가 읽기 쉽고 화면 구성이 잘되어 있으나 자원 사용에 관한 정보 묘사가 다소 제한되어 있다. 또한 uniform한 유저 공간과 AFS[7]나 NFS[8]을 요구하며, 부하 분산에 있어서 메모리 사용량을 고려하지 않기 때문에 성능 저하를 야기할 수 있다. [9]

CODINE은 GUI를 사용한 쉬운 접근이 가능하고, 관리자 기능을 제거하여 사용자가 더 친근하게 느낄 수 있도록 한다. 마이그레이션을 통해 동적 부하 분산을 수행하지만, 체크포인팅을 제공하지는 않는다.

3. 부하 분산 시스템의 기본 구조

본 논문에서 구현한 부하 분산 시스템은 중앙 관리자(central manager), 스케줄러(scheduler), 모니터(monitor)로 구분된다. 부하 분산 시스템을 사용하고자 하는 클러스터 중에서 한 노드가 서버가 되고, 나머지 노드들이 클라이언트가 된다. 중앙 관리자와 스케줄러가 서버에 데몬으로 실행되고, 모니터가 클라이언트 각각에 데몬으로 실행된다. [10]

중앙 관리자는 전체 클러스터에서 수행되는 작업 정보와 클러스터 전체의 자원 정보를 관리하며, 스케줄러는 사용자가 작업을 수행하고자 할 때, 시스템의 부하를 가장 잘 분산시키는 쪽으로 작업을 분배하고, 작업 수행 중에 부하가 집중되는 노드가 있으면 작업을 다른 노드로 마이그레이션(migration)시켜줌으로써 부하를 분산시킨다. 모니터는 현재 클라이언트 노드의 자

원 정보를 모아서 중앙 관리자에게 보내주는 일을 하고, 중앙 관리자에서 작업 수행을 요청하면, 그 작업을 노드에서 수행하는 역할을 한다.

4. 부하 분산 시스템 세부 구현

4.1. 중앙 관리자

중앙 관리자는 서버 쪽 노드에서 수행된다. 중앙 관리자는 크게 자원 정보를 관리하는 부분, 작업을 수행하는 부분, 웹과의 인터페이스를 위한 부분으로 나뉜다.

중앙 관리자는 적절한 작업 분배를 위해서 현재 부하 분산 시스템에 연결되어 있는 클라이언트들의 자원 정보를 알고 있어야 한다. 이를 위해 부하 분산 시스템의 클라이언트에 수행되고 있는 모니터가 현재의 자원 정보를 얻어서 5초마다 한번씩 주기적으로 중앙 관리자에게 소켓(TCP)을 통해 정보를 전송하도록 되어있다. 중앙 관리자는 이 정보를 받아서 현재 부하 분산 시스템에 연결되어 있는 클라이언트들의 접속 여부를 체크하고 그 클라이언트 노드 각각의 부하 정보들을 자료 구조를 작성하여 저장하게 된다. 이때, 클라이언트에서 보내주는 노드의 자원 정보는 CPU, memory, 그리고 load average값이 된다. [11, 12]

또, 중앙 관리자는 사용자가 수행하고자 하는 작업을 적합한 노드에서 수행시켜 줄 수 있어야 한다. 중앙 관리자는 사용자가 작업을 수행하고자 할 때, 사용자가 작성한 작업정보 파일(job description file)을 받아서 이 안에 있는 정보들을 자료 구조로 저장한다. 그리고 나서 이 작업 정보를 스케줄러에게 보내면 스케줄러 쪽에서는 가지고 있는 노드들의 자원 정보와 작업 정보를 사용하여 작업을 수행하기에 적절한 노드를 결정하여 다시 중앙 관리자에게 알려준다. 이렇게 해서 받은 정보를 사용하여 모니터와의 통신을 통해 작업을 수행할 수 있도록 한다.

마지막으로 웹 인터페이스를 제공하는 부분이 필요한데, 현재 작성한 부하 분산 시스템은 웹을 통한 작업 관리 기능(job control), 마이그레이션 기능(migration), 부하정보 모니터링 기능(load information monitoring)을 제공한다. 이 시스템과의 인터페이스를 위한 부분으로서 웹에서 소켓을 사용하여 오는 요구들을 적절히 상황에 맞게 판단하여 정보를 웹을 통해 보여주거나, 혹은 작업을 수행하는 일을 담당하는 부분이다.

4.2. 스케줄러

스케줄러는 중앙 관리자와 함께 서버 측 노드에서 수행된다. 스케줄러는 사용자가 작업을 처음 수행하고자 할 때, 어느 노드에서 수행하는 것이 가장 적절한지 판단하는 부분(default 스케줄러) 과 작업을 수행하는 중에 한 노드에 부하가 집중되는 경우 그 노드에서 다른 노드로 작업을 옮기도록 해주는 부분으로 나누어진다.

기본(default) 스케줄러는 중앙 관리자로부터 받은 작업 정보와 현재 수집되어 있는 자원 정보를 이용하여 현재 사용자가 수행하고자 하는 작업이 어느 노드에서 수행될 때 가장 좋은 성능 향상을 가져올지를 결정한다. 중앙 관리자로부터 받은 작업 정보를 사용하여 현재 사용자가 수행하고자 하는 작업이 serial 작업인지, 아니면 동시에 여러 프로세스를 필요로 하는 parallel 작업인지를 판단한다. 그리고 나서 현재 자원 정보를 이용하여 부하 척도(load metric)을 계산하게 되는데 현재 시스템에서는 세 가지 부하 척도를 제공한다. 이 부하 척도를 이용하여 현재 작업을 수행하기에 적합하다고 판단되는 노드를 결정한다. 이렇게 결정한 것들을 자료 구조로 만들어서 저장하고 작업을 결정한 노드에서 수행할 수 있도록 이를 다시 중앙 관리자에게 보내준다.

나머지 부분이 부하가 한 노드에 몰리는 현상을 방지하기 위한 부분으로 주기적으로 현재의 자원 정보를 검사한다. 한 노드에 부하가 몰렸다는 것이 감지되면 그 노드에서 현재 수행중인 작 업 중에서 적절한 작업을 골라서 다른 노드에서 수행될 수 있도록 하기 위해 자료 구조로 노드 정보와 작업 정보를 중앙 관리자에게 넘겨준다. 그러면 중앙 관리자는 이 정보를 받아서 수행중인 작업을 다른 노드로 옮기게 된다.

4.3. 모니터

모니터는 클라이언트 노드에서 수행되며 자원 정보를 수집하여 중앙 관리자에게 보내는 부분, 중앙 관리자의 요청을 받아서 현재 노드에서 작업을 수행하는 부분, 작업을 한 노드에서 다른 노드로 옮길 필요가 있을 때 마이그레이션을 수행하는 부분으로 나누어져 있다.

자원 정보를 수집하는 부분에서는 현재의 노드 정보를 수집한다. /proc 디렉토리 밑에 있는 loadavg 파일, meminfo파일을 읽어서 현재의 loadavg값과 memory 정보를 수집하고, 각 pid에 해당하는 /proc/[pid]/stat 파일을 읽어서 현재 노드에서 수행중인 프로세스 정보를 함께 수집한다. 이렇게 얻은 노드 정보를 5초마다 한번씩 중앙 관리자에게 보낸다.

작업을 수행하는 부분에서는 사용자가 수행하고자 하는 작업을 중앙 관리자에게 받아서 수행하는 역할을 한다. 작업을 수행하기 시작하면서 그 작업을 수행하는 프로세스의 아이디 정보를 중앙 관리자에게 알려주며, 작업이 끝나면 작업을 수행하는데 사용된 자원 정보를 중앙 관리자에게 알려준다.

마이그레이션을 수행하는 부분에서는 필요에 의해 현재 수행중인 작업을 한 노드에서 다른 노드로 옮길 때, 두 노드간의 통신을 이용해서 작업을 옮기는 일을 수행한다. 이 부분은 작업을 원활하게 한 노드에서 멈추고, 다른 노드에서 재 시작할 수 있도록 구현된 부분으로서 작업을 새로 시작하고자 하는 노드에서 수행된다.

4.4. Load metric

스케줄러는 현재의 부하 정보를 가지고 metric을 계산한다. 이때 metric은 작업을 어느 노드에서 수행할지 결정하는 데 가장 중요한 역할을 하며, 부하 분산 시스템에서는 전체 성능에 영향을 줄 수 있는 값이다.

현재 부하 분산 시스템에서는 metric을 세가지 종류로 구현하였는데, loadavg, CPU length, Mem+CPU로 구현하였다.

Loadavg는 load average로서 클라이언트에서 5초마다 주기적으로 보내오는 정보에 포함되어 있는 값으로 최근 1분 동안 CPU를 사용하는 평균 작업의 개수다. CPU length 는 현재 수행중인 프로세스의 개수로 클라이언트가 보내주는 부하 정보에서 프로세스의 개수를 계산하여 얻는다. Mem+CPU는 메모리와 CPU를 둘 다 고려하여 계산한 metric 인데, 현재 사용 가능한 자원들에 대해 최대한 고려하는 방식으로 접근한 것이라고 할 수 있는데 현재의 load average 값에 memory rate값을 곱한 것이다. Memory rate값은 전체 유저에게 할당된 메모리 값을 현재 사용 가능한 메모리 값으로 나눈 것으로 메모리가 현재 어느 정도의 비율로 사용되고 있는지를 판단할 수 있는 값이다. [13]

5. 부하 분산 시스템의 성능 측정

5.1. 성능 측정 환경

성능 측정을 위해서 8개의 노드로 구성된 클러스터에서 부하 분산 시스템에 대한 성능을 측정하였다.

성능 측정에는 SCALARPACK 벤치마크를 사용하였으며, 이 중에서 사용된 LINPACK은 주로 선형 방정식과 선형 최소 제곱법 문제를 푸는 포트란 서브루틴들로 구성된 수치 해석 패키지의 하나로서 컴퓨터의 연산속도를 측정하는 벤치마크 프로그램

으로 이용되기도 한다. [14, 15]

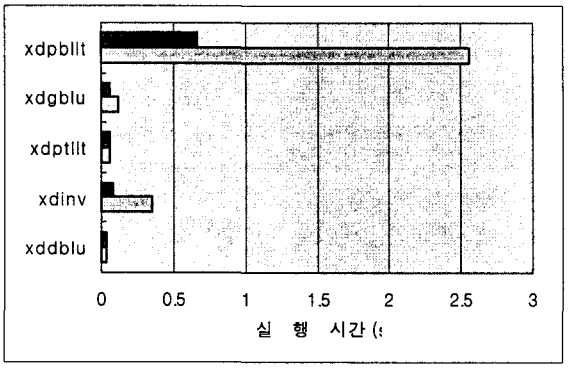
5.2. 성능 측정 결과

성능 측정을 위하여 앞서 2장에서 설명한 PBS라는 프로그램을 설치하여 수행하였다. 8노드에 OpenPBS(PBS의 공개된 버전이다) 을 설치하여 벤치마크 프로그램을 수행시키고, 또 구현한 부하 분산 시스템에서 벤치마크 프로그램을 수행시켜 수행시간을 계산하였다. 부하 분산의 성능측정을 위해서 클라이언트 노드 7개중 두 곳에 프로그램을 수행시켜 loadavg값을 임의로 4로 조정해 둔 상태에서 성능 측정을 하였다.

아래 [표 1]은 벤치마크의 수행 결과 OpenPBS에서와 부하 분산 시스템에서의 수행 시간을 표로 나타낸 것이고, [그림 1]은 이 표를 그래프로 나타낸 것이다.

[표 1] 부하 분산 시스템에서의 LINPACK벤치마크 수행시간

프로그램	OpenPBS에서의 실행시간(sec)	부하 분산 시스템에의 실행시간(sec)
xdlu	0	0
xddblu	0.0383	0.0383
xdls	0	0
xdllt	0	0
xdqr	0	0
xdinv	0.35	0.08
xdptllt	0.0562	0.0552
xdgblu	0.1214	0.0607
xdpbllt	2.56	0.6609



[그림 1] 부하 분산 시스템에서의 LINPACK벤치마크 수행시간

전체 9개의 프로그램을 수행하였을 때 OpenPBS에서는 평균 약0.347초가, 부하 분산 시스템에서는 평균 약0.099초가 걸렸음을 알 수 있고, 1.8배의 성능 향상을 가져온 것을 알 수 있다.

9개의 프로그램을 수행하였을 때 그 중 6개의 프로그램에서는 수행하는 데 같은 시간이 걸린 반면 3개의 프로그램에서는 부하 분산 시스템이 좋은 성능을 나타내었다.

MPI작업을 수행하는 데 있어서 각 노드에 걸리는 부하가 다른 환경 하에서 작업을 수행할 때, 현재 부하가 많은 노드와 부하가 적은 노드 사이에서 프로세스가 통신하면서 작업을 수행해야 하기 때문에 더 많은 성능의 저하를 가져왔다.

6. 결론

클러스터에서 부하의 불균형 현상은 전체 시스템의 성능에 좋지 않은 영향을 줄 수 있기 때문에 부하를 적절히 분배하는 방식으로 작업을 실행시키는 것이 중요하다.

이를 위해 중앙 관리자를 클러스터 시스템에 하나 두고, 작업을 수행시킬 각 노드에 모니터를 하나씩 두어 부하 분산 문제를 해결하였다. 중앙 관리자는 최근의 노드의 부하 정보를 모니터에게 받아서 저장하고 관리하며, 이 정보를 바탕으로 사용자가 작업을 수행하고자 할 때 적절한 노드에서 수행될 수 있도록 한다. 서버에 있는 스케줄러가 이런 역할을 하며, 주기적으로 노드의 상태를 확인해서 필요한 경우 부하가 높은 노드에서 낮은 노드로 작업을 이동하기도 한다.

본 논문에서 구현한 부하 분산 시스템을 OpenPBS와 비교하기 위해 SCALARPACK 벤치마크를 사용하여 성능을 측정한 결과 평균 1.8배의 수행시간 향상을 보였다.

7. 참고 문헌

[1] Buyya, R., ed. 1999. High Performance Cluster Computing: Architectures and Systems. Vol. 1

[2] Narendran Ganapathy and Sunil Kumar, Process migration and Load Balancing.Term paper. Dept of Computer Science, Univ of New Hampshire

[3] Mary Papakhian, 1998. COMPARING JOB-MANAGEMENT SYSTEMS: THE USER' S PERSPECTIVE. IEEE COMPUTATIONAL SCIENCE & ENGINEERING

[4] Jean Suplick and Richardson, January 1994. An Analysis of Load Balancing Technology

[5] S. Zhou, " A trace-driven simulation study of dynamic load balancing," Rept. No. UCB/CSD 86/, University of California, Berkeley, June 1986.

[6] Litzkow, M., Livny, M. and Mutka, M. ' ' Condor □ A Hunter of Idle Workstations' ' . Proceedings of the 8th International Conference on Distributed Computing Systems. San Jose, Calif. June 1988

[7] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, and R.N. Sidebotham, Scale and performance in a distributed file system, ACM Transactions on Computer Systems, 6(1):55-81, February 1988.

[8] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Serverless Network File Systems. In Proceedings of the 15th Symposium on Operating System Principles. ACM

[9] IBM LoadLeveler for AIX, Using and Administering, Version 2 Release 2

[10] Chin Lu, John C.S. Lui, Peter W.K. Lie, M.K. Tang, S.Y. Lau, H.K. Li, Distributed Scheduling Framework A Load Distribution Facility on Mach

[11] Sau-Ming LAU, Qin LU, Kwong-Sak LEUNG, Dynamic Load Distribution Using Anti-Tasks and Load State Vectors

[12] Songnian Zhou, Jingwen Wang, Xiaohu Zheng, and Pierre Delisle, UTOPIA: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. 1992. Software - Practice and Experience

[13] X. Zhang, Y. Qu, and L. Xiao, Improving distributed workload performance by sharing both CPU and memory resources, Proceedings of 20th International Conference on Distributed Computing Systems, (ICDCS'2000), Taipei, Taiwan, April 10-13, 2000.

[14] J.Dongarra.The LINPACK Benchmark: An Explanation. Supercomputing, Spring 1998.

[15] Jack Dongarra, Jim Bunch, Cleve Moler and Pete Stewart, "LINPACK", http://www.netlib.org/linpack/