유효 작업수를 이용한 동적 부하 분산 시스템 성능 개선

최민, 박은지°, 유정록, 맹승렬
한국과학기술원 전자전산학과 전산학전공
{mchoi, pakej°, jlyu, maeng}@camars.kaist.ac.kr

Improving Performance of Dynamic Load Balancing System by Using Number of Effective Tasks

Min Choi, Eun-Ji Pak° , Jung-Lok Yu, Seung-Ryoul Maeng
Div. of Computer Science, Dept. of EECS, KAIST

요 약

클러스터 시스템의 성능 향상을 위해서는 컴퓨팅 자원을 효과적으로 사용하여야 한다. 과거에는 전체 시스템 자원을 효과적으로 사용하기 위해 각 노드들의 부하를 균등하게 하는 방향으로 연구가 진행되어 왔으나, 부하 분산 시스템이 작업의 자원 요구 형태를 고려하여 작업을 배치하는 경우 성능을 더욱 향상시킬 수 있다. 현재까지는 이런 자원 요구 형태에 대한 선행지식을 과거 작업 실행 기록에 기반하여 유추해내는 방법을 많이 사용하였으나 이 방법은 잘못된 예측을 가져와 실행시간을 증가시킬 수 있다. 본 논문에서는 이를 해결하기 위해 유효 작업수라 불리는 새로운 노드의 부하 측정 척도를 제시한다. 유효 작업수를 이용한 부하 분산 시스템은 작업의 자원 요구 사항을 알지 못하더라도 부하 분산 과정에서 작업이 잘못 배치되어 실행시간이 증가하는 경우를 방지한다. 성능분석 결과는 과거 자료에 의한 예측을 사용하는 기존 방법에 비해 전체 실행시간의 감소로 성능이 향상되었음을 보여준다.

1. 서론

최근 네트워크와 마이크로프로세서 기술에 관한 연구가 활발해지면서 독자적인 워크스테이션들을 고속의 네트워크(high speed network)로 연결한 클러스터 시스템이 등장하였다.[1] 이런 클러스터 시스템은 성능 향상을 좀 더 쉽게 얻을 수 있다는 장점이 있는 반면, 부하가 집중되는 경향이 있다.[3] 이런 단점을 해결하기 위해 부하 불균형을 해소하는 부하 분산 시스템(load balancing system)의 개발이 반드시 필요하다.

부하 분산 시스템은 부하 분산 작업에 현재의 시스템 상태를 반영하는지 여부에 따라 정적(static) 부하 분산 시스템과 동적(dynamic) 부하 분산 시스템으로 분류된다. 동적 부하 분산 시스템은 작업을 실행하기 전에 가장 적합한 노드를 찾아 작업을 할당하는 초기작업 할당(initial job placement)방식과 부하 불균형이 발생했을 때 작업을 이동시키는 프로세스 마이그레이션(process migration)방식으로 나뉘는데, 최근에는 이 두 가지 방식을 조합한 하이브리드 접근(hybrid approach)방식을 많이 사용한다.[4]

초기 작업 할당 방식 부하 분산 시스템은 노드별로 작업을 균등하게 배분함으로써 시스템 전체의 자원 활용률을 높이는데, 초기 작업 할당시 작업의 자원 요구사항을 고려하여 작업을 배치하게 되면 시스템의 성능 향상을 높일 수 있다. 작업의 자원 요구 형태를 예측하기 위한 방법에 대한 연구로는 과거 작업을 실행해 본 결과를 바탕으로 예측하는 방법[4], 통계적 접근을 사용하는 방법[5], 작업에 필요한 자원 요구 예측을 사용자가 직접 제공하는 방법[6], 프로세스의 초반 1초 실행 후 요구되는 자원 정보를 바탕으로 한 예측방법[7] 등이 있다. 그러나 이런 방법들은 과거 데이터를 바탕으로 한 것이기 때문에 부정확할 수가 있고, 이를 기반으로 하여 작업을 배치할 경우 성능 저하를 가져올 수 있다.

따라서 본 논문에서는 예측을 통한 방법을 사용하지 않으면서도 성능저하를 방지할 수 있는 방법으로 새로운 부하척도(load metric)인 유효 작업수(effective number of tasks)를 개발하고, 이것을 활용한 동적 부하 분산 시스템을 설계 및 구현, 평가하였다.

2. 관련 연구

부하 분산 시스템에서 초기 작업 할당 시 작업의 자원 요구 사항을 고려하여 작업을 배치하기 위해 작업의 자원 요구 형태를 예측하는 방법에는 여러 가지가 있다.

과거 자료에 기반한 예측 방법(Estimation of future behavior from historical data)은 가장 대표적인 방법으로 예전에 해당 프로세스를 실행할 때 얻은 자원 사용에 관한 자료를 바탕으로 작업을 할당한다.

통계적 접근 방식에서는 작업의 CPU, I/O, memory 활용 정도를 파악하는 데 있어 통계적(statistical)이고 패턴 인식적인(pattern-recognition-based) 방법을 사용한다.

사용자 예측에 의한 방법(user estimation)은 사용자가 실행하고자 하는 작업을 시스템에 제출(submit)할 때 그 작업에 대한 사용자 추측 정보를 함께 제공하는 방법이다.

작업 실행 초반에 나타나는 자원 요구 형태를 이용한 예측 방법은 일단 작업을 실행시키고 처음 1초 동안 수집한 데이터를 바탕으로 작업의 자원 요구 형태를 파악한다.

이러한 여러 가지 방법들은 작업의 자원 요구 형태를 파악하는 데 있어서 가장 보편적인 방법들이지만 예측에 의한 방

법이므로 부정확할 수 있고, 잘못된 예측을 기반으로 작업 특성을 분류하고 이를 부하 분산 작업에 적용하였을 때는 작업 실행시간에 큰 피해를 유발한다. 또한, 각 작업의 자원 요구 형태에 대한 정보를 저장하는 데 드는 시간적 공간적 비용이 크고, 저장된 정보에서 원하는 정보를 검색하는데 있어서 오버헤드 역시 존재한다.

따라서, 본 논문에서는 작업의 자원 요구 형태에 대한 선행 지식을 필요로 하지 않으면서도 초기 작업 할당에 있어 작업이 잘못 배치되는 상황을 방지하는 방법인 유효 작업수를 이용한 동적 부하 분산 시스템을 제안한다.

3. 유효 작업수를 이용한 동적 부하 분산 시스템

3.1. 유효 작업수

유효 작업수란 시스템 성능에 실제로 영향을 미치는 프로세스의 수를 의미하는 것으로 시스템이 부팅된 후부터 종료될 때까지 항상 각 epoch이 끝나는 시점에 계산된다. Epoch은 리눅스 스케쥴링의 기본 단위로 모든 실행 가능한 작업들이 할당 받은 시간 할당량을 모두 소모하기까지의 기간이다. 유효 작업의 계산은 CPU bound한 작업이 한 epoch내에서 사용하는 평균 CPU 사용량에 대해 I/O bound한 작업이 한 epoch 내에서 사용하는 평균 CPU 사용량의 비인 I/O to CPU ratio에 의해 결정된다.

$$IO\ to\ CPU\ ratio = \frac{avg(t_{cpu}(io_i))}{avg(t_{cpu}(cpu_i))}$$

이 I/O to CPU ratio가 큰 경우는 I/O bound 작업이 CPU를 많이 사용하고 있는 경우로 I/O bound 작업과 CPU bound 작업의 overlapping 효과가 크지 않기 때문에 이때는 기존의 작업수와 동일한 값을 유효 작업수로 반환한다.

I/O to CPU ratio가 0.5 보다 작은 경우에는 I/O bound 작업과 CPU bound 작업 중에서 숫자가 많은 쪽의 작업이 시스템 성능에 영향을 미치기 때문에 CPU bound 작업의 수가 더 많은 경우에는 전체 작업이 소모한 CPU시간의 합을 시스템 내의 CPU 작업들이 소모한 평균 CPU시간으로 나누어 구한다. I/O bound작업의 수가 더 많은 경우에 대해서도 마찬가지의 방식으로 구한다.

I/O to CPU ratio < 0.5 이고 n(IO) ≤ n(CPU) 일 때

$$effective\ \#\ of\ tasks = \frac{\sum t_{cpu}(job_i)}{avg(t_{cpu}(cpu_i))} = \frac{\sum t_{cpu}(cpu_i) + \sum t_{cpu}(io_i)}{avg(t_{cpu}(cpu_i))}$$

$$= n(cpu) + \frac{\sum t_{cpu}(io_i)}{avg(t_{cpu}(cpu_i))}$$

I/O to CPU ratio < 0.5 이고 n(IO) > n(CPU) 일 때

$$effective\ \#\ of\ tasks = \frac{\sum t_{io}(job_i)}{avg(t_{io}(io_i))} = \frac{\sum t_{io}(io_i) + \sum t_{io}(cpu_i)}{avg(t_{io}(io_i))}$$

$$= n(io) + \frac{\sum t_{io}(cpu_i)}{avg(t_{io}(io_i))} = n(io) \qquad \because t_{io}(cpu_i) \approx 0$$

3.2. 유효 작업수를 이용한 부하 분산 시스템 설계

본 절에서는 유효 작업수를 부하 척도로 사용하는 하이브리드 방식의 동적 부하 분산 시스템의 설계 및 구현에 대하여 설명한다. 부하 분산 시스템의 전체적인 구조는 다음과 같다.

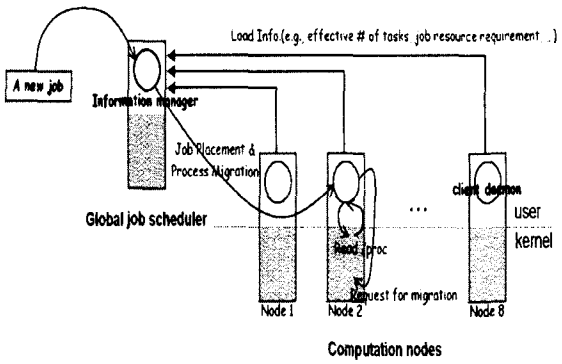

그림 1. 유효 작업수를 이용한 동적 부하 분산 시스템 구조

한대의 전역 작업 스케쥴러(global job scheduler)가 초기 작업 할당 및 프로세스 마이그레이션의 실행을 제어한다. 각 연산 노드(computation node)는 매 순간 계산되는 유효 작업수를 비롯해 현재 노드에서 실행중인 모든 작업에 대한 최근 정보를 항상 유지하고 정보 관리자(information manager)에게 주기적으로 전송한다. 새로운 작업이 시스템에 제출되면 정보 관리자는 유효 작업수가 가장 작은 노드를 그 작업이 실행되기에 가장 적절한 노드로 선택한다.

유효 작업수는 앞 절에서 설명한 수식들을 이용하여 계산되며 linux의 timer를 사용하여 매 epoch마다 주기적으로 계산된다. 이때 I/O bound 작업인지 CPU bound작업인지의 구분은 한 epoch에서 소모하는 시간 할당량과 CPU factor를 이용하였다. 각 연산 노드의 daemon들은 /proc/loadavg 파일을 사용하여 노드의 작업수 정보를 가져오는데 본 연구에서는 이 파일이 현재 노드의 작업수, 유효 작업수, 그리고 실행중인 작업들의 자원 요구 형태 정보를 포함하도록 수정하였다. 이런 정보를 이용하여 전역 작업 스케쥴러가 초기 작업 할당 방식을 주로 사용하여 작업을 할당하도록 하였고, 부하 불균등이 발생하는 경우에 한해 부수적으로 프로세스 마이그레이션을 사용하였다.

4. 성능 평가

본 논문에서 시스템을 구현한 환경은 1대의 global job scheduler와 8대의 computation node이며 이들은 100Mbps Fast Ethernet으로 연결되어 있다. 유효 작업수의 구현은 리눅스 커널 2.4.2에 수정을 가하는 방식으로 구현되었으며, 연산 노드에 있는 client daemon은 리눅스 사용자 수준(user-level)에서 개발되었다. 정보 매니저(information manager)는 MFC 라이브러리를 활용하여 Visual C++로 개발된 window2000용 애플리케이션이다.

성능 비교 대상은 history 기반 예측 방법과 유효 작업수에 기반한 방법의 작업 실행 시간과 전체 실행 시간이다. History에 기반한 예측 방법에서는 명령어의 이름과 작업을 실행할 때 주어진 인자(parameter)의 값이 기록과 같으면 과거에 실행한 바 있는 작업으로 간주한다. [8]

그림 2를 보면 작업 실행 시간을 측정한 결과에서는 전체적으로 유효 작업수 방법을 이용한 경우 조금씩 더 수행시간이 적음을 볼 수 있다. 37번, 45번 작업의 경우 history에 기반한 경우가 잘못된 작업배치가 이루어진 반면에 유효 작업수에 기반한 방법에서는 가장 적절한 노드에 작업을 배치하게 된 경우에 해당한다. 반면, 20번, 36번, 43번과 같은 작업의 경우는

유효 작업수를 이용했을 때 가장 나쁜 경우(worst-case)에 해당하는 작업배치는 방지할 수 있었지만 가장 적절한 노드에 작업을 배치하지는 못한 경우이다.

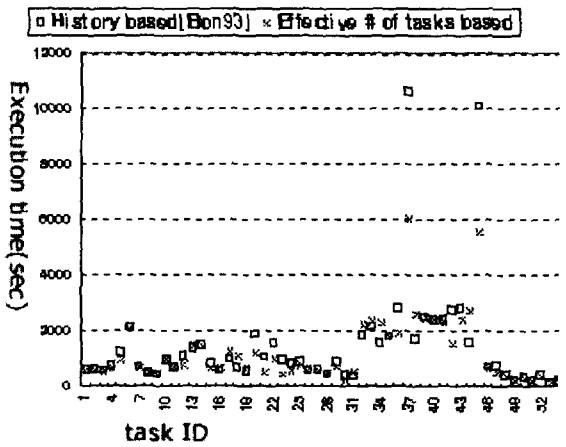

그림 2. 작업 실행 시간
(CPU활용률: CPU job - 99.9%, I/O job-19.6%)

그림 3은 전체 작업 실행 시간을 나타내는 것으로, 결과적으로 유효 작업수를 사용한 동적 부하 분산 시스템은 이상적인 경우에 최대 10256.11초의 전체 실행시간(total execution time)단축을 가져와 약 12%의 성능향상을 보이는 것을 알 수 있다.

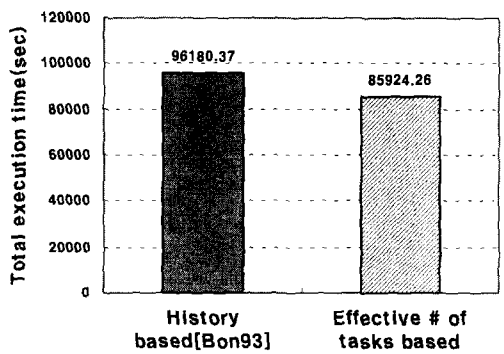

그림 3. 전체 작업 실행 시간
(CPU활용률: CPU job - 99.9%, I/O job-19.6%)

5. 결론 및 향후 연구 과제

클러스터 시스템은 사용자에게 고성능과 확장성을 제공한다. 그러나 부하 불균형 문제의 발생은 전체 시스템의 성능 저하를 가져온다. 이러한 문제를 해결하기 위해 많은 연구가 진행되어 왔으나, 부하 분산 시스템에서 작업의 자원 요구 형태를 history에 기반하여 예측하고 작업을 할당했을 때 여러 가지 문제점들이 발생할 수 있다. 본 논문에서는 예측을 통한 방법을 사용하지 않으면서도 초기 작업 할당에서 성능 저하를 방지할 수 있는 유효 작업수의 개념을 제안했다. 이러한 유효 작업수를 사용하는 동적 부하 분산 시스템을 설계 및 구현하고 성능을 측정해 본 결과 기존의 시스템에 비해 최대12% 의 성능 향상을 얻을 수 있었다.

향후 연구과제로는 짧은 시간 동안에 연속적으로 여러 개의 작업이 시스템에 제공될 때 이들 여러 작업들이 모두 한 노드에 배치되는 현상을 해결하는 것인데, 이를 해결하기 위해 프로세스의 예상 실행시간을 고려한 가중 평균값을 사용하는 방법을 생각하고 있다. 그 밖의 연구과제로는 프로세스 마이그레이션 방식에 기반한 부하 분산 시스템의 위치 정책에서 자원 활용률을 고려하도록 하여 부하 감소의 효과를 좀 더 늘리는 방법을 연구 중이다.

5. 참고 문헌

[1] 유정록, "Myrinet 상에서 적응적 전송 기법을 이용한 효율적인 Virtual Interface Architecture의 구현", Master's thesis, 한국과학기술원, 2001.

[2] G. Pfister, "*In Search of Clusters* (2nd Edition)", Prentice Hall, 1998.

[3] M. Theimer and K. Lantz. "Finding Idle Machines in a Workstation-Based Distributed Systems", *IEEE Transactions on Software Engineering*, Vol. 15, No. 11, November 1989.

[4] B. Folliot, Y. Hajmahmoud, P. Sens, "Quantifying the performance improvement of Migration in load sharing systems," *ACM Transactions on Computer System*, 1992.

[5] K. Bubendorfer, "Resource Based Policies for Load Distribution", Master's thesis, Victoria University of Wellington, 1996.

[6] *P. Kruger and R. Chawla*, "The Stealth Distributed Scheduler", in Proceedings of the 11th International Conference on Distributed Computing Systems, June 1991.

[7] *M. Schaar, K. Eye, L. Delcambre, and L. Bhuyan*, "Load Balancing with Network Cooperation", In Proceedings of the 11th International Conference on Distributed Computing Systems, June 1991.

[8] A. Bond, "Adaptive Task Allocation in a Distributed Workstation Environment", PhD thesis, Victoria University of Wellington, 1993.

[9] D.Feitelson, "Parallel Workload Archive", http://www.cs.huji.ac.il/labs/parallel/workload

[10] J. Ju, G. Xu, K. Yang, "An Intelligent Dynamic Load Balancer for Workstation Clusters," *ACM Operating Systems Review*, Vol. 29, No. 1, January 1995.

[11] D. Feitelson, M. Jette. "Improved utilization and responsiveness with gang scheduling," In Proceedings of IPPS/SPDP '97 Workship. Lecture Notes in Computer Science, Vol. 1291, April 1997

[12] M. Balter, "Task Assignment with Unknown Duration," In Proceedings of the 20th International Conference on Distributed Computing Systems, April 2000

[13] F. Silva, I. Scherson, "Improving Parallel Job Scheduling Using Runtime Measurements," 6th Workshop on Job Scheduling Strategies for Parallel Processing, May 2000

[14] M. Balter, A. Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing," ACM Transactions on Computer Systems, Vol. 15, No. 3, August 1997, Pages 253-285