유효 작업 수를 이용한 동적 부하분산 시스템 성능개선

최 민[0] 유정록 맹승렬
한국과학기술원 전산학과
(mchoi[0], jlyu, maeng)@calab.kaist.ac.kr

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

Min Choi[0] Jung-Rok Yu Seung-Ryoul Maeng
Division of Computer Science, Department of EECS, KAIST

요 약

부하 분산 시스템의 성능을 향상시키기 위해서는 각 연산 노드들에 대한 부하수준을 잘 파악하여야 한다. 기존의 부하 분산 시스템들은 부하 측정기준(load metric)으로 실행 큐(run queue)에 있는 작업의 수(number of jobs)를 주로 이용한다. 그러나, 여러 프로세스들이 동시에 실행될 때, 각 프로세스의 실행이 서로의 성능에 미치는 정도인 프로세스간 독립수준(interprocess dependence level)을 고려하면 좀 더 정확하게 시스템 부하수준을 측정할 수 있다. 본 연구에서는 시스템 성능에 실제로 영향을 미치는 프로세스들의 수를 의미하는 유효 작업의 수(effective number of jobs)라는 부하 측정기준을 적용하여 성능이 향상된 부하분산 시스템을 설계하고 구현하였다.*

1. 서 론

동적 부하분산 시스템은 원격실행(remote execution) 방식과 선점형 마이그레이션(preemptive migration)의 두 가지 형태로 분류된다. 원격실행 방식은 어떤 새로운 프로세스들을 원격지 호스트에서 실행 시키는 방법이며, 선점형 마이그레이션 방식은 현재 실행중인 프로세스를 중단시키고 이를 다른 원격지 호스트로 이동하여 재실행 시키는 방법이다. 본 연구에서는 선점형 마이그레이션 방식에 기반한 동적 부하분산 시스템을 설계 및 구현하도록 한다.

부하분산 시스템은 전송 정책(transfer policy), 위치선정 정책(location policy), 정보수집 정책(information policy)의 세가지 구성요소로 이루어진다.[1] 전송 정책은 어떠한 호스트가 작업 재배치(task relocation)과정에 참여해야 하는가를 결정하고 재배치 대상이 되는 프로세스를 선정한다. 위치선정 정책에서는 작업 재배치의 적절한 대상이 되는 호스트를 결정한다. 마지막으로, 정보수집 정책은 각 호스트의 부하 수준을 판단하기 위한 정보를 어떻게 수집할 것인지를 지정한다. 동적 부하분산 시스템을 설계하기 위해서는 위의 세가지 정책에 대한 결정이 선행되어야 하며 3절에서 이러한 과정에 기반한 시스템 설계를 소개하도록 한다.

유효 작업 수 부하 측정기준은 프로세스의 독립 개념이 기반이 되어 개발되었다. 따라서, 본문에서도 자주 언급될 프로세스의 독립에 관한 내용을 먼저 정의하고 넘어가도록 한다. "두 프로세스가 독립적(independent)이다." 라는 의미는 두 프로세스가 동일한 호스트에서 동시에 실행되고 있을 때, 그들은 서로의 성능에 아무런 영향을 미치지 않는다는 것을 의미한다. 일반적으로 두개 이상의 프로세스가 동시에 한 노드에서 실행되는 경우에 서로의 성능에 영향을 미치는데 그 정도는 프로세스의 성격에 따라 달라진다. 이렇게, 프로세스의 성격에 따라 두 프로세스 서로의 성능에 영향을 미치는 정도 프로세스간 독립수준(interprocess dependence level)이라 한다.

기존의 부하분산 시스템은 부하 측정기준(load metric)으로 실행 큐(run queue)에 있는 작업의 수(number of tasks)를 주로 이용한다. 그러나, 앞에서 언급한 프로세스간 독립수준을 고려하면 좀 더 정확히 시스템의 부하를 측정 할 수 있다. 이를 위해서, 본 연구에서는 부하 분산 시스템을 위해 사용할 수 있는 새로운 부하 측정기준인 유효 작업 수(effective number of tasks)를 제시한다. 유효 작업 수는 실행시간(runtime)에 측정되는 프로세스간 독립수준에 따라서, 두 프로세스 중 한 프로세스가 다른 한 프로세스에 미치는 영향을 분석하여 시스템 성능에 실제로 영향을 미치는 프로세스들의 수를 나타내는 부하 측정기준이다.

본 연구에서는 유효 작업 수를 사용하는 동적 부하분산 시스템을 설계하고 구현하였다. 그리고, 기존의 실행 큐에 있는 작업의 수를 부하 측정기준으로 사용하는 기존의 시스템과 성능 비교를 수행하였다.

논문의 구성은 다음과 같다. 2절에서는 유효 작업의 수

* 본 연구는 국가지정연구실사업 지원을 받아 수행되었음

부하 측정기준을 소개하고 3절에서는 이를 적용한 동적 부하분산 시스템의 설계 및 구현에 대하여 기술한다. 4절에서는 기존 동적 부하분산 시스템과 성능을 비교 분석하고 5절에서 결론을 맺는다.

2. 유효 작업 수(effective number of tasks)

2.1 프로세스간 독립수준

다음 그래프는 두 가지 종류(CPU bound, I/O bound)의 프로세스를 여러 가지 방법으로 조합하여 레드햇 리눅스, Pentium III 850Mhz CPU, 256M 메모리를 가진 머신에서 실행시간을 측정한 결과이다.

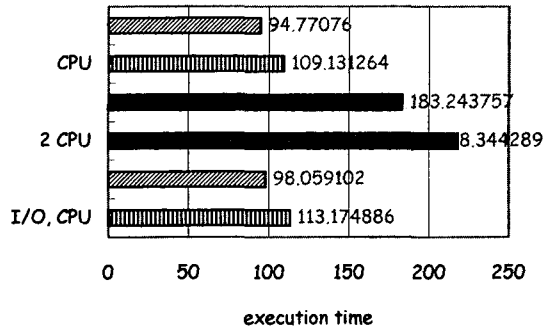


그림 1 프로세스 조합에 따른 실행시간

각 프로세스는 단독으로 실행하는 것에 비해서 동일한 종류의 프로세스 두 개를 동시에 실행하는 경우 그 실행시간이 두 배가 된다(2I/O, 2CPU). 반면, I/O bound 프로세스와 CPU bound 프로세스를 동시에 실행한 결과는 그 실행시간이 원래의 단독으로 실행하는 경우와 비교하여 그다지 큰 차이가 나타나지 않는다. 이것은 라운드 로빈 방식에 기반하는 시분할 방식 스케쥴링을 사용하는 운영체제에서 당연한 결과로 I/O를 위해 block되는 동안 다른 CPU프로세스가 유용한 계산을 할 수 있기 때문이다. 다시말해서, 동시에 두 프로세스가 동일한 호스트에서 실행될 때 서로의 성능에 영향을 미치는 정도에 따라 "독립성이 크다" 혹은 "독립성이 낮다"고 이야기한다. 이러한 독립성의 정도를 유효 작업의 수 계산에 사용하기 위해서는 좀 더 정확하게 수치화할 필요가 있는데 이를 프로세스간 독립수준(ipdl)이라 한다. 원래의 ipdl 개념은 프로세스의 실행으로 인해 나타난 프로그램 실행시간의 증가량에 기반하여 제안되었으나 선점형 마이그레이션 방식을 사용하는 동적 부하분산 시스템에서는 프로세스의 실행이 종료되기 전에 프로세스들간의 독립수준을 구할 수 있어야 한다. 따라서, 실행시간에 ipdl을 구할 수 있도록 하기 위해서 리눅스 프로세스 스케쥴링 과정에서의 정보를 이용하였다

리눅스 운영체제에서의 프로세스 스케쥴링은 CPU 시간을 epoch이라는 작은 단위로 분할하여 진행한다. epoch은 현재 TASK_RUNNING 상태에 있는 모든 프로세스들이 각각 할당받은 time quantum을 모두 소모하는 시점에서 한 epoch이 종료된다. 이 때 스케쥴러는 시스템 내에 존재하는 모든 프로세스에 time quantum을 새로 계산하여 할당하며 새로운 epoch이 시작된다. 아래 그림2가 이를 잘 보여준다.

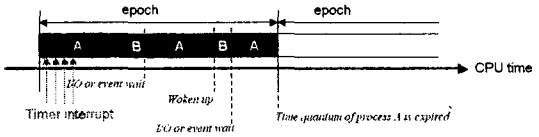


그림 2 리눅스 프로세스 스케쥴링과 독립수준

실제로, 한 epoch이 끝날 때 각 프로세스가 해당 epoch 내에서 CPU를 할당받아 소모한 시간을 프로세스 디스크립터에 저장한다. 프로세스 A는 B와 함께 실행됨으로 인하여 실행시간에 지연(extra delay)을 가져온다. 따라서, ipdl은 지연시간을 경과시간(elapsed time)으로 나누어 구할 수 있다.

$$ipdl = \frac{extra\ delay}{elapsed\ time} \times 2$$

2.2 유효 작업의 수

유효 작업의 수의 계산을 위해서는 우선 몇 가지 함수에 대한 정의가 필요하다. 프로세스의 종류를 나타내기 위해서 $T = \{t_1, t_2, t_3\}$ 를 사용하고 CPU, I/O, MEM 프로세스 타입을 원소로 갖는다. $nP(t_i)$ 는 특정 호스트에서 실행 큐에 존재하는 타입 t_i 인 프로세스의 수를 나타낸다. $ipdl(p_i, p_j)$ 는 그림2에서와 같은 프로세스간 독립수준을 참조하기 위한 함수이다.

$$effective\#\ of\ tasks = \begin{cases} Max(nP(t_i), nP(t_j)) \times (1 + ipdl(p_i, p_j)) & for\ ipdl(p_i, p_j) > 0.5 \\ (nP(t_i) + nP(t_j)) & for\ ipdl(p_i, p_j) \le 0.5 \end{cases}$$

위의 식에서 프로세스간 독립수준이 0.5보다 큰 경우(두 프로세스를 같은 호스트에서 실행하더라도 서로의 성능에 미치는 영향이 작은 경우)는 같은 종류의 프로세스들 수가 많은 쪽에서 시스템 성능에 더 큰 영향을 미친다. 따라서, 수가 많은 종류에 속하는 프로세스들의 수를 제외한 좀더 적은 영향을 미치는 종류의 프로세스들은 소수점 이하 에 프로세스간 독립수준에 상응하는 효과를 반영한다. 반면, 두 프로세스들 사이의 독립수준이 0.5보다 작은 경우(동시에 실행하는 경우 서로의 성능에 미치는 영향이 큰 경우)는 두 종류 프로세스들이 시스템 성능에 모두 함께 큰 영향을 미치기 때문에 이들 프로세스들의 수를 단순히 더해준다.

기존의 동적 부하 분산 시스템은 해당 호스트의 부하를 실행중인 작업의 수를 이용하여 측정하며, 최대 작업의 수가 같게 나오는 여러 개의 호스트가 존재하는 경우 작업을 재배치 대상 호스트를 결정하기 위해서 다시 그 호스트들만을 대상으로 또 다른 나름대로의 선정과정을 거쳐야 한다. 그러나, 유효 작업의 수 측정기준은 실제로 시

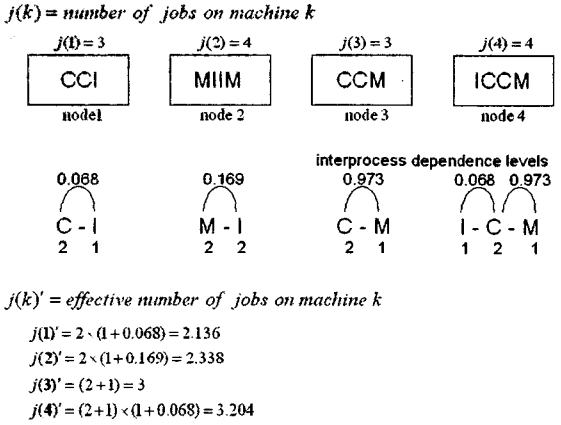


그림 3 유효 작업 수 계산

스템 성능에 영향을 미치는 작업의 개수만을 카운트하므로 기존 부하분산 시스템에서 수행하는 부가적인 재배치 대상 호스트 선정과정이 필요 없게 된다. 위 그림 3은 유효 작업 수를 계산하는 과정을 보여준다.

3. 동적 부하분산 시스템의 설계 및 구현

3.1 시스템 설계

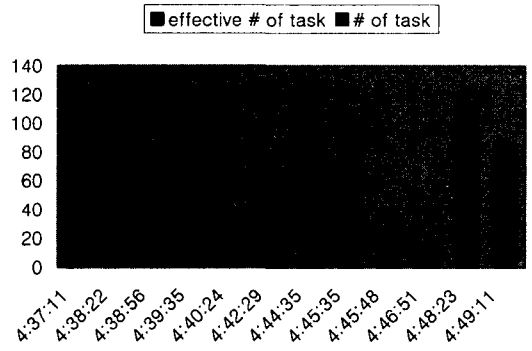
유효 작업 수를 측정기준으로 사용하는 동적 부하분산 시스템의 설계하기 위해서는 앞에서 언급한 세가지 정책에 대한 결정이 선행되어야 한다. 첫번째, 전송 정책으로는 유효 작업 수를 시스템의 부하 수준으로 사용하여 가장 큰 값, $Max(L_k)$,을 가지는 호스트 및 호스트 그룹을 작업 재배치 대상 호스트로 결정한다. 재배치 후보(candidate)는 가장 큰 프로세스간 독립수준을 갖는 프로세스, $Max(ipdl(t_1,t_2))$,중 하나를 선택한다. 두번째, 위치선정 정책에서는 해당 호스트의 부하정도가 시스템 전체 평균 부하수준보다 작으면서 현재 이동하려는 재배치 후보 작업의 종류와 ipdl값이 작은 노드를 선택한다 $Min(ipdl(t_1,t_2))$. 마지막으로, 정보수집 정책은 시스템 영역 네트워크(SAN)의 높은 신뢰성과 연결접속 및 해제 오버헤드를 줄이기 위해서 UDP 프로토콜을 통한 브로드캐스팅 방식으로 구현한다.

3.2 시스템 구현

선점형 마이그레이션 방식의 부하분산 시스템을 개발하기 위해서 필요한 소켓 마이그레이션(socket migration)[1]기능을 운영체제 커널 수준에서 개발하였고, 현재 실행중인 프로세스에 대한 정보를 커널수준에서 얻고 이를 활용하기 위해서 SGI(Silicon Graphics, Inc.,)의 CSA(Comprehensive System Accounting) 프로젝트 에서 제공하는 오픈소스 커널 패치를 적용하고 실행시간(runtime)에 운영체제가 어카운팅 정보를 사용할 수 있도록 수정하였다. 리눅스에서 loadavg 계산은 1분, 5분, 15분의 감속시간(decay time)을 이용하는데 이 과정에서 실행중인 작업의 수를 반환하는 count_active_tasks()함수를 시작으로 유효 작업 수를 반환하도록 구현하였다.

4. 성능 분석

본 연구에서 사용한 기반 환경은 Pentium III 850Mhz CPU, 256M 메모리, Linux 2.4.10 커널의 레드햇 리눅스 머신 3대를 Fast Ethernet으로 연결되어 있다.



위 그림 4는 12가지 프로그램(I/O, CPU, MEM 3개씩)을 실행한 결과를 보인 것이다. 계열1은 실행중인 작업 수를 측정기준으로 사용하는 부하분산 시스템이며 계열2는 유효 작업 수를 사용하는 부하분산 시스템이다. 간혹 실행시간이 더 커지는 작업들은 프로세스 마이그레이션으로 인해 실행시간이 길어지게 된 것이다.

5. 결론

본 논문에서는 유효 작업 수를 이용하는 동적 부하분산 시스템을 설계 및 구현하였다. 시스템 성능에 실제로 영향을 미치는 프로세스들의 수를 의미하는 유효 작업 수를 시스템 부하 측정기준으로 사용하여 최대 4.89%의 성능 향상을 보였다. 앞으로, 소켓 마이그레이션이 수행된 작업에 대해 부가적인 실행시간이 걸리는 문제를 해결하기 위한 연구를 진행해야 할 것이다.

참고 문헌

[1] Sau-Ming LAU, Qin LU, Kwong-Sak LEUNG, Dynamic Load Distribution Using Anti-Tasks and Load State Vectors, Proceedings of the 18th International Conference on Distributed Computing Systems, 26-29 May, 1998

[2] Mor Harchor-Balter, Allen B. DowNEY, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing", ACM Transaction on Computer Systems, Vol. 15, No. 3, August 1997

[1] socket migration = process migration + TCP Handoff