주기적 추진(Periodic Boost)의 바쁜 대기를 줄이기 위한 휴리스틱

정다운[O] 유정록 맹승렬 이준원
한국과학기술원 전자전산학과
{dwjung[O], jlyu, maeng, joon }@camars.kaist.ac.kr

A Heuristic to reduce busy waiting in Periodic Boost

Dawoon Jung[O] Junglok Yu, Seungryoul Maeng, Joonwon Lee
Dept. of EECS, KAIST

요 약

클러스터 시스템은 상대적으로 가격이 싼 컴퓨터를 고성능의 네트워크(Network)로 묶어서 슈퍼컴퓨터와 같은 고성능을 가지도록 만들어진 시스템이다. 이런 클러스터 컴퓨팅 환경에서 효율적인 스케줄링은 그 성능에 직접적인 영향을 주는 요소이다. 이런 시스템에서 완전한 동시 스케줄링(Coscheduling)은 서로 교환해야하는 정보가 많아지기 때문에 그 구현이 어렵다. 이 상황에서 메시지를 기다리는 정보와 메시지의 도착 정보를 이용해서, 즉 단지 그 노드(Node) 자체의 정보만을 이용해 동시 스케줄링의 효과를 구현할 수 있다. 그리고 이것을 이용한 알고리즘 중에 주기적 추진(Periodic Boost(PB))이 있다. 이 논문에서는 주기적 추진에 휴리스틱을 이용하여 더 효과적인 스케줄링을 할 수 있는 알고리즘을 소개한다.

그리고 이 휴리스틱의 효과를 검증하기 위해서 클러스터 노드 2개를 이용해서 실험을 했다. 실험은 계산대 통신 비율(Communication-to-Computation ratio)을 변화시켜가면서 총 수행시간을 측정하고, 서로 통신하는 양이 다른 프로세스를 섞어서 그 성능을 실험한 결과 휴리스틱이 주기적 추진(PB)에서 불필요하게 낭비되는 자원을 효율적으로 사용할 수 있음을 알 수 있었다.

1. 서론

클러스터 시스템은 효율적인 가격으로 고성능의 계산을 할 수 있는 시스템으로 부상했다. 이 시스템은 상대적으로 컴퓨터 노드는 가격이 싸지고, 고성능의 통신망이 개발되면서 더욱 가속화 되었다. 클러스터는 가격이 상대적으로 싼 컴퓨터들을 고성능의 통신장비를 이용해서 묶어, 슈퍼컴퓨터와 같은 고성능의 계산을 처리할 수 있는 시스템이다.

이런 시스템에서 병렬 어플리케이션을 수행시킬 때는 스케줄링 알고리즘이 성능에 많은 영향을 준다. 클러스터에서 스케줄링은 기존에 쓰이던 하나의 작은 시스템에서 쓰이는 스케줄링과는 다르다. 프로세스들이 서로 다른 노드에서 수행되기 때문에 그것을 적절히 스케줄링 해 주는 것이 필요하다. 그리고 이런 시스템에서 병렬 어플리케이션을 수행하면, 서로 다른 노드에서 수행되고 있는 프로세스간 통신 때문에 성능이 저하될 수도 있다.

이렇게 통신에 영향을 많이 받을 수 있는 클러스터 시스템은 노드간 빠른 통신을 위해 고성능의 네트워크를 이용한다. 하지만 네트워크 계층구조상 많은 오버헤드가 있어, 이런 것을 줄이기 위해 사용자 수준(User level) 통신을 이용한다. 커널의 도움을 줄이고, 불필요한 자료의 복사를 방지하고, 네트워크 인터페이스 카드(NIC)에서 어플리케이션으로 바로 통신을 하여 오버헤드(overhead)와 응답시간(latency)을 줄이는 것이다.

고성능의 네트워크 하드웨어를 이용하고, 소프트웨어에서 통신하는데 드는 비용을 많이 줄였지만 여전히 병렬 어플리케이션을 수행하는 데는 문제가 있다. 서로 노드에 분산되어 수행되는 프로세스를 적절히 스케줄링 하기 힘든 것이다. 예를 들어 그림 1의 (a)와 같이, P1과, P2, P3이 있을 때 왼쪽과 같이 서로 다른 노드에서 스케줄링을 하면 서로 통신하면서 기다리는 시간이 많아지기 때문에 수행시간이 더 오래 걸린다. 즉 그만큼 프로세서를 비효율적으로 이용한다. 하지만 그림 1의 (b)와 같이 스케줄링을 하면 서로 통신을 하는데 효율적으로 하여 더욱 좋은 성능을 나타낸다.

하지만 이런 스케줄링을 완벽하게 구현하려면 각 노드끼리 통신을 많이 해야 하고 그에 따른 오버헤드를 무시 할 수 없다. 특히 클러스터 시스템과 같이 각각의 노드가 독립적으로 돌아가는 시스템의 경우에는 하나로 통합되어 있는 병렬 시스템보다 통신하는데 오버헤드가 더욱 크다. 이런 제약조건 속에서 좋은 동시 스케줄링의 효과를 내는 스케줄링 알고리즘이 많이 제안되었다. 그중 대표적인 것이 주기적 추진(PB), 동적인 동시 스케줄링(Dynamic Coscheduling(DCS))등이 있다. 이런 알고리즘들은 자기 노드에 도착하는 메시지 정보와, 프로세스가 기다리는 메시지 정보를 이용해서 스케줄링에 도움을 준다.

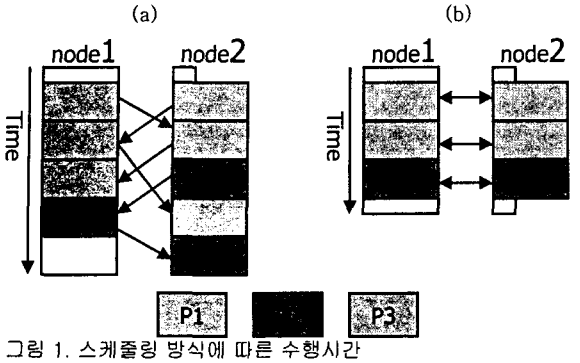

그림 1. 스케줄링 방식에 따른 수행시간

동적인 동시 스케줄링은 메시지가 도착했을 때, 그 메시지를 보낸 프로세스가 현재 스케줄링 되어 있다고 가정하고, 받는 프로세스가 스케줄 되어 있지 않으면, 그것을 바로 스케줄링 하는 것이다. 이럴 때 인터럽트를 이용하는데 메시지가 많아지면 그 인터럽트를 처리하기 위한 오버헤드가 커진다.

주기적 추진(PB)은 동적인 동시 스케줄링과 비슷하지만, 인터럽트를 이용하지 않는 것이 다르다. 이 알고리즘은 주기적으로 커널 쓰레드가 현재 메시지를 확인해서 메시지가 도착한 프로세스를 선택해서 우선순위를 높여 추진(Boost)시켜주는 방식이다. 하지만 메시지를 받기 위해서는 바쁜 대기(Busy Waiting)를 하고 있기 때문에 불필요하게 프로세서가 낭비되는 일이 있다. 이런 것들을 줄이기 위해 기본적인 방법 외에 많은 휴리스틱이 존재한다. 이 논문에서는 주기적 추진(PB)에서 바쁜 대기로 인한 오버헤드를 감소시키기 위한 휴리스틱을 제안한다.

2. 휴리스틱 : 우선순위 낮추기

주기적 추진에서는 메시지를 받을 때 기본적으로 프로세스는 바쁜 대기를 하고 있다. 즉 메시지가 도착할 때까지 프로세서에 스케줄 되어 시간을 쓰고 있다. 하지만 이때 메시지를 받지 못하면 나중에 다시 스케줄 되었을 때도 바쁜 대기를 반복한다. 이렇게 바쁜 대기를 하면 메시지가 도착하자마자 바로 다음 일을 수행할 수 있어 성능상 유리하다. 하지만 불필요하게 프로세서를 잡고 있어서, 다른 프로세스들이 그 시간동안 활용하지 못하여 프로세서 자원은 불필요하게 낭비된다. 만약 이런 상황에서 주기적 추진(PB)이 다른 노드에서 수행되는 프로세스를 추진시켜 빨리 수행되게 하여 메시지를 빨리 받을 수 있으면 성능 향상이 이루어진다. 그러나 프로세스가 많아지고 각각의 프로세스들이 모두 통신을 하고 있으면 결국 바쁜 대기를 하는 프로세스도 많아진다. 이런 상황에서 불필요한 자원 사용을 줄이기 위해서 이 논문에서는 우선순위 낮추기를 제안한다.

바쁜 대기를 하고 있을 때 불필요한 프로세서 사용을 줄이기 위한 방법으로 일시정지(Blocking)와 반복 후 양보(Spin Yield), 반복 후 일시정지(Spin Block) 등이 제안되었다. 일시 정지는 메시지가 올 때까지 정지된 상태로 기다리는 것이다. 그리고 메시지가 도착하면 스케줄 되기를 기다렸다가 수행되는 방법이다. 이 방법은 불필요한 프로세서 시간을 낭비하지 않지만, 메시지가 들어와도 바로 다음 작업을 수행할 수 없기 때문에 동시 스케줄링(Coscheduling)에 불리하다. 반복 후 양보는 어느 정도 바쁜 대기를 하다가 프로세서 자원을 다른 프로세스에게 양보하는 것인데, 이 경우에는 많은 프로세스가 읽기 반복(Spinning)하고 있으면 오히려 문맥교환(*Context Switching*)이 자주 일어나 그 오버헤드가 크고, 결국 읽기 반복하는 프로세스가 많을 경우 불필요하게 프로세서를 잡고 수행되는 시간이 많아진다. 반복 후 일시정지(Spin Block)는 어느 정도 바쁜 대기를 하고 일시정지 상태로 간다. 이 경우 통신이 많을 때 읽기 반복하는 과정에서 메시지를 받을 가능성이 높지만 여전히 일시정지 상태로 간 프로세스는 메시지를 받고나서 다음 작업을 수행하기에 불리하다.

우선순위 낮추기는 주기적 추진(PB)에서 우선순위를 높이는 것과는 반대로 우선순위를 낮추어 자원 사용을 줄이는 것이다. 이렇게 했을 때 장점은 다음과 같다. 첫째, 바쁜 대기를 어느 정도 하고나면 우선순위를 낮추어 나중에 스케줄링 되어 수행되는 시간을 줄이고 자주 스케줄링 되지 않게 하여 문맥교환 오버헤드도 줄일 수 있다. 둘째, 우선순위가 낮아졌지만 어느 정도 읽기 반복은 하고 있어서 일시정지(blocking)와 같이 메시지 수신 후 작업이 늦게 수행되는 것을 방지할 수 있다. 마지막으로 별도의 인터럽트나 커널 쓰레드 같은 것들이 필요가 없고, 메시지를 수신하는 부분에 계수 변수(counter)를 하나 두고 읽기 반복(spin)하는 횟수만 체크하면 되므로, 오버헤드가 거의 없다.

3. 실험 및 결과

우리는 각 노드에 기본적으로 들어 있는 스케줄러(LOCAL)만 이용한 것과, 주기적 추진(PB), 그리고 주기적 추진과 우선순위 낮추기를 적용한 것의 성능을 비교했다. 실험환경은 PIII-500MHz, 512MB RAM, 100Mbps Fast Ethernet을 장착한 노드 2개에서 했다. 운영체제는 Linux 2.4.20 커널을 이용했고, 주기적 추진에서는 20ms마다 메시지를 체크해서 추진(Boost)시키도록 했다.

실험에 이용한 어플리케이션은 실제 어플리케이션 중 모델을 선정해, 비슷하게 모델링 하여 구현한 것을 이용했다. 어플리케이션 모델은 다음과 같다. 1) 프로세스가 다른 프로세스에게 필요한 데이터를 보낸다. 2) 프로세스가 다른 프로세스로부터 필요한 데이터를 받는다. 3)받은 데이터를 바탕으로 계산을 수행한다. 4)위의 1), 2), 3)을 반복한다. 이것은 마치 격자 계산 커널(Grid Solver Kernel)과 같이 처음에 경계면의 데이터를 보내고(1), 그것을 받고(2), 그리고 계산을 하고(3), 이것을 반복하는(4) 구조와 거의 동일하다. 그리고 이런 모델링한 어플리케이션은 2개의 노드에서 4개를 동시에 수행하여 그 결과를 구했다.

첫 번째 실험에서는 계산대 통신 비율(C-to-C ratio)을 변화시켜가면서 총 수행시간을 구했다. 이때 총 계산양은 같고 통신하는 양을 변화시켜서 수행한 것이다. 그림 2에서 보는 것과 같이 전체적인 수행시간은 로컬(LOCAL)이 가장 오래 걸리고, 주기적 추진(PB), 주기적 추진+휴리스틱의 순서로 걸렸다.

특히 통신 양이 많아질수록 전체적인 수행시간은 오래 걸렸다. 그러나 그림 3에서 보면, 주기적 추진(PB)은 그에 비해 상당히 좋은 성능 향상을 가져온다. 특히 통신양이 적을 때는 추진(Boost)의 효과를 볼 수 없지만, 통신이 어느 정도 많아지면 그 효과가 커진다. 하지만 통신 양이 어느 정도 증가하면 그 성능 향상 폭이 다시 감소한다. 주기적 추진(PB)에 휴리스틱을 적용한 것은 주기적 추진보다 더욱 좋은 성능향상을 기록했다. 기본적으로 로컬(LOCAL)보다 50%이상의 성능 향상이 있고, 주기적 추진보다도 최소 9%이상의 성능 향상을 보인다. 특히 통신양이 적을 때도 그 성능향상이 주기적 추진보다 우수하고 통신양이 많아지면서 그 격차가 조금씩 줄어들지만, 통신양이 더욱 많아지면 다시 그 성능 향상 폭이 증가하는 것을 볼 수 있다.

첫 번째 실험에서는 모든 프로세스의 계산대 통신 비율(C-to-

C ratio)이 동일한 경우에 실험한 것이고, 다음의 실험은 작업들을 적절히 섞어 놓은 경우이다. 작업들에서 L은 계산대 통신 비율이 0.1인 작업이고, H는 0.5인 작업이다. 이렇게 4개의 작업을 동시에 수행시켰을 때 수행시간은 그림 4에서 보는 것과 같다. 통신양이 많은 프로세스가 있을 때 성능 향상 폭이 전반적으로 크고, 앞에서의 실험과 비슷하게 휴리스틱을 적용한 경우가 가장 성능이 좋게 나왔다. 이렇게 통신양이 서로 다른 프로세스들이 같이 동작해도 휴리스틱은 잘 동작하고 있음을 알 수 있다.

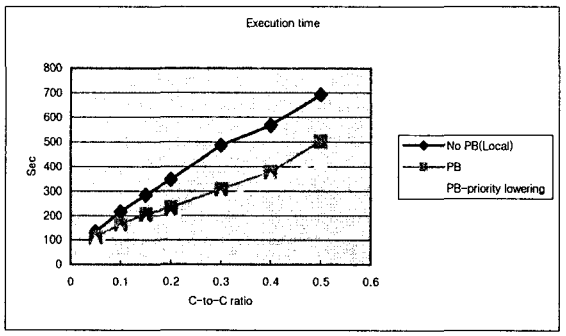

그림 2. 계산대 통신 비율(C-to-C ratio)의 변화에 따른 총 수행시간. 각 노드의 스케줄러를 이용하는 것보다 주기적 추진(PB)을 이용하는 것이 좋은 결과를 보여주고 있고, 주기적 추진에 우리의 휴리스틱을 적용한 것이 더 좋은 결과를 보여준다.

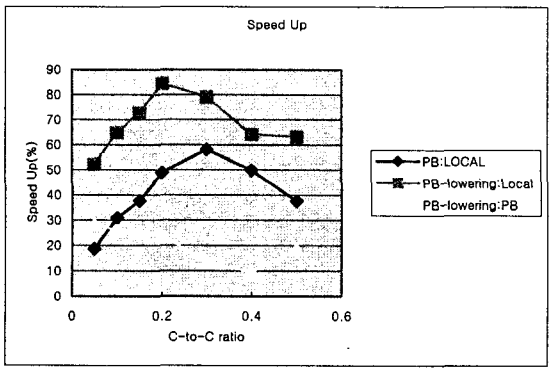

그림 3. 계산대 통신 비율(C-to-C ratio)의 변화에 따른 성능 향상

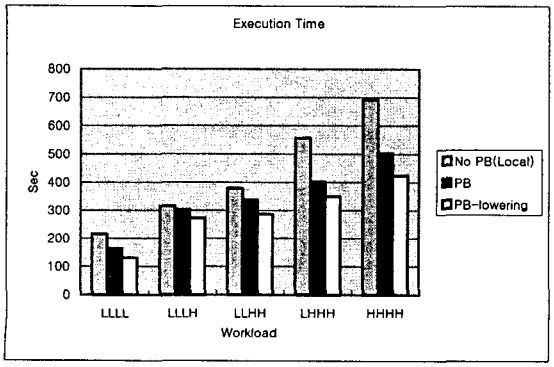

그림 4. 계산대 통신 비율(C-to-C ratio)이 다른 작업들이 동작할 때 총 수행시간.

이렇게 실험결과는 좋게 나왔지만 한 가지 문제점이 있다. 실제 구현한 환경이 이더넷(Ethernet)환경에서 TCP를 쓰는 것이어서 모든 메시지마다 인터럽트가 발생한다. 실제 주기적 추진(PB)의 진정한 장점은 인터럽트를 제거한 것이어서 그 오버헤드가 적다는 것이지만 여기서는 그렇지 않다.

4. 결론 및 향후 연구 과제

클러스터에서 그 성능을 극대화 하는 데에는 스케줄링이 중요하다. 각 노드끼리 정보를 교환하는 오버헤드가 있어 완전하게 효과적인 동시 스케줄링은 힘들다. 하지만 병렬 어플리케이션 특성을 이용해서 메시지를 주고받는 정보만으로 큰 효과를 얻을 수 있다. 특히 주기적 추진(PB)은 자체의 오버헤드도 거의 없이, 효율적인 스케줄링을 한다. 이 논문에서는 주기적 추진에 또 다른 휴리스틱을 써서 통신할 때 불필요한 프로세서 자원 사용을 줄여, 아무것도 하지 않은 스케줄링은 물론 주기적 추진보다 더 좋은 성능을 나타내는 것을 실험을 통해 알았다.

하지만 구현 및 실험 환경의 제약으로 실제 어플리케이션으로 실험하지 못했고, 클러스터 노드도 2개밖에 쓰지 않았다. 그리고 통신 환경으로는 TCP를 이용해서 사용자 수준 통신을 이용한 상황에서의 주기적 추진과는 성능차가 있을 것이다. 앞으로 우리는 여기서 구현한 휴리스틱을 좀더 많은 노드를 쓰는 환경에서 구현하고 인터럽트가 발생하지 않는 통신 환경을 이용해 구현할 것이고, 성능 측정으로 실제 어플리케이션과 벤치마크 툴을 이용해 더욱 정확한 성능 향상을 구할 것이다. 그리고 마지막으로 주기적 추진(PB)의 타이머(timer)와 읽기 반복(Spinning) 횟수를 조절하기 위해 많은 실험을 하고, 최적의 값을 구할 것을 계획하고 있다.

참조

[1] Shailabh Nagar, Ajit Banerjee, and Anand Sivasubramaniam. A Closer Look At Coscheduling Approaches for a Network of Workstations. *Proceedings of 11th ACM Symposium on Parallel Algorithms and Architectures*, June 1999

[2] N.J. Boden et al. Myrinet: A Gigabit-per-second Local Area Network. *IEEE Micro*, Feb 1995

[3] P. G. Sobalvarro. Demand-based Coscheduling of Parallel Jobs on Multiprogrammed Multiprocessors. PhD thesis, Dept. of Electrical Engineering and Computer Science Massachusetts Institute of Technology, Cambridge, MA, January 1997

[4] A. C. Arpaci-Dusseau and D.E. Culler, and A. M. Mainwaring. Scheduling with Implicit Informatin in Distributed Systems. *In proceedings of the ACM SIGMETRICS 1998 Converence on Measurement and Modeling of Computer Systems*, 1998