제28회 한국정보처리학회 추계학술발표대회 논문집 제14권 제2호 (2007. 11)

병렬 작업 스케줄링에 대한 조사 연구

윤지현, 염헌영
서울대학교 컴퓨터공학부
e-mail : {jhyoon, yeom}@dcslab.snu.ac.kr

Survey of various parallel job scheduling techniques on clusters

Ji Hyun Yoon, Heon Y. Yeom
Dept. of Computer Science & Engineering, Seoul National University

요 약

클러스터를 이용하여 다수의 작업을 실행시키는 경우에 효율적으로 사용자들이 자원을 사용하기 위해서는 작업 스케줄링이 매우 중요하다. 다양한 스케줄링 방법들이 제안되었으며 그 중 효율적으로 병렬 작업을 스케쥴링하기 위해 제안된 방법으로는 backfilling, co-scheduling, gang scheduling 을 들 수 있다. 이러한 연구에서는 이론적인 논의가 많았고, 실제로 구현을 했다고 하더라고 multiprocessor 를 대상으로 backfilling 을 다룬 내용이 대부분이었다. 이 논문은 클러스터상에서의 parallel job scheduling 에 대해 조사하였다.

1. 서론

다양한 기관에 속한 computing resource 들을 결합하여 Grid 를 구성하는 경우, 다수의 사용자들이 공평하고 효율적으로 자원을 사용할 수 있도록 하기 위해서는 job scheduling 이 매우 중요하다.

한 개 이상의 클러스터를 이용하여 다수의 작업을 실행시키는 경우, 가장 단순한 job scheduling 방법은 PBS 와 같은 batch scheduler 를 이용하는 방법이다. 그러나 이러한 경우 한 작업이 끝나야 다음 작업을 시작할 수 있으므로, 긴 작업이 앞에 있고 짧은 작업이 뒤에 있다면 짧은 작업은 긴 작업이 끝날 때까지 긴 시간을 기다려야 하기 때문에 공평하지 않다.

이러한 문제를 해결하기 위해 다양한 스케줄링 방법이 제안되었으며, 그 중 대표적인 방법들이 backfilling, co-scheduling, gang scheduling 등이다.

2. 연구 배경

Backfilling 은 batch scheduling 의 변형으로, 무조건 FCFS 를 지키는 것이 아니라 특정한 조건을 만족할 때는 스케줄링 순서를 바꾸는 것도 허용함으로써 utilization 을 최대화하고자 하는 기법이다. 예를 들어 작업 큐의 맨 앞에 긴 작업 A 가 기다리고 있고, 그 뒤에 짧은 작업 B 가 있다고 할 때, B 를 먼저 실행시켜도 A 의 시작 시간에 영향을 주는 것이 아니라면 작업 순서를 바꾸도록 하는 기법이다. Backfilling 의 대표적인 문제점은 각 작업에 대해 예상 실행 시간을 미리 제시해야 한다는 것이다. 예상 실행 시간을 통해 다른 작업을 지연시키는지 그렇지 않은지를 판단해서 실행 순서를 바꾸게 되는데, 실행 시간을 미리 예측하는 것은 어려울 뿐 아니라, 악의적인 사용자가 고의로 짧은 실행 시간을 제시할 수도 있으므로 문제가 된다.

Backfilling 이 한번에 하나씩 작업을 실행시키는데 반해, co-scheduling 과 gang scheduling 은 동시에 다수의 작업을 실행시킨다. 두 기법의 차이는 local scheduling 을 하느냐 global scheduling 을 하느냐 여부이다.

Co-scheduling 에서는 클러스터에 다수의 작업을 실행시킨 후, local scheduling 에 의존한다. 즉, 각 노드의 OS 가 알아서 작업들을 스케줄링 하게 되므로, 전체적으로 보면 같은 작업이 서로 다른 시간에 실행될 수 있다.

Gang scheduling 은 각 작업에 대해 dedicated machine 과 같은 환경을 만들어 준다. Global scheduler 가 존재해서 같은 작업은 같은 순간에 실행될 수 있도록 한다.

Co-scheduling 이나 gang scheduling 의 경우, 각 작업이 time slice 를 할당 받게 되므로, 기다리는 시간의 편차가 크지 않아서 공평하다. 특히 synchronization 이 중요한 작업의 경우, co-scheduling 에서는 같은 작업이 동시에 실행되는 것이 아니라서 synchronization 을 위해 기다려야 하는 경우가 발생할 수 있지만, gang scheduling 에서는 전혀 기다릴 필요가 없기 때문에 이러한 종류의 작업을 실행시키는 경우에는 매우 효율적이다.

3. 클러스터상에서의 병렬작업 스케쥴링에 대한 연구

Backfilling, co-scheduling, gang scheduling 이 효율적으로 parallel job 을 스케줄링하기 위해 제안된 방법이긴 하지만, 기존 연구에서는 이론적인 논의가 많았고, 실제로 구현을 했다고 하더라고 multiprocessor 를 대상으로 backfilling 을 다룬 내용이 대부분이었다.

클러스터상에서의 parallel job scheduling 에 대한 연구로는 [1]~ [7] 정도를 들 수 있는데, 이 논문 모두 parallel job processing 이 batch processing 보다 성능을 향상시킬 수 있다는 점에서는 동의하고 있지만, 구체적인 scheduling 방법에 대해서는 약간 다른 입장을 취하고 있다. 즉, [1]~ [3]까지는 application 의 특성에 따라 훨씬 효율적인 parallel scheduling 방법이 있다는 입장이고, [4]~ [7]까지는 application 의 특성에 상관없이 co-scheduling 이 항상 가장 좋은 성능을 보여줄 수 있다는 입장이다.

[1]은 preemptive job scheduling 에 대한 이론적인 연구는 많았지만, 실제 상황에 적용해서 성능을 측정한 연구는 거의 없었다는 동기에서 출발했다. 따라서 이 논문에서는 실제 supercomputing center 에서 얻은 job log 를 얻어서 job 을 특성에 따라 구분한 후, simulation 을 통하여 preemptive scheduling 의 성능을 측정했다.

이 논문에서 제안한 preemptive scheduling 방법은 SS(Selective Suspension)인데 idle job 이 running job 보다 suspension priority 가 높으면 running job 을 preemption 하게 되는 방법이다. SS 는 SF(suspension factor)를 지정할 수 있어서, idle job 의 priority 가 running job 의 priority 보다 SF 배 이상이 되는 경우에만 preemption 을 할 수 있게 함으로써 suspension rate 을 조정한다. 실험은 CTC, SDSC, KTH supercomputing center 에서 얻은 로그를 이용해서 작업을 분류했다. 실행 시간에 따라서는 very short, short, long, very long 의 4 가지로 분류하고, 필요로 하는 processor 의 개수에 따라 seq, narrow, wide, very wide 의 4 가지로 분류했다. 실험은 우선 작업 시간에 따라 전체 작업을 4 가지로 분류한 후, 각각에 대해 seq, narrow, wide, very wide 작업을 나누고, suspension 이 없는 경우와 SF=1.5, SF=2, SF=5 에 대해 slowdown 과 turnaround time 을 측정했다. 결과는 짧고 processor 를 많이 필요로 하는 작업일수록 preemption 을 자주 해 주면 성능이 향상된다는 것이다.

[2]는 cluster 에서의 parallel computing 에 대해 전반적으로 정리한 글로서, cluster 의 interconnection technology, programming model, cluster application 등에 대한 내용을 담고 있다. 이 논문에서는 fast Ethernet, gigabit Ethernet, myrinet, SCI 의 4 가지 방법으로 cluster 를 연결한 후, NPB 의 FT 와 LU 를 실행시켰다. 스케줄링은 [1]과 유사하게 각 노드에 작업을 실행시킨 후에는 각 node 의 local scheduling 에 의존했다. 실험 결과는 응용 프로그램의 특성에 따라 성능 향상의 정도가 다를 수 있다는 것이다. 즉, computation-intensive job 인 LU 의 경우 interconnection 에 상관없이 성능이 비슷했지만, communication-intensive job 인 FT 의 경우에는 latency 가 긴 interconnection 에서는 성능이 좋지 않았다.

[3]은 PVM-Linux NOW 에서 explicit co-scheduling 과 implicit co-scheduling 을 구현한 후 성능을 비교한 내용이다. 이 논문에서의 기본 가정은 synchronization 을 위해 기다릴 필요가 없어야 parallel scheduling 을 통해 성능을 향상시킬 수 있다는 것이다. 여기서 다루고 있는 co-scheduling 은 일반적인 co-scheduling 이 아니라 preemptive scheduling 을 의미한다. 즉, explicit co-scheduling 은 gang scheduling 을 의미하고, implicit co-scheduling 은 통신이 필요한 process 들은 동시에 실행될 수 있도록 preemption 을 하는 방법이다. 실험은 PVM- Linux NOW 에서 7 가지의 다른 scheduling 방법으로 NPB 의 mg 와 is 를 실행시켰을 때의 성능을 측정했다. 여기서 scheduling 방법은 PVM(원래의 PVM 스케줄링), SPIN(implicit co-scheduling), MXI, MXISPIN, PRIO(always the highest priority is assigned to distributed tasks), PRIOSPIN, EXPLICIT(explicit co-scheduling)이 되고, PRIO 가 optimal case 가 된다. 실험 결과는 PVM 의 성능이 가장 좋지 않고 EXPLICIT 는 PRIO 에 가까운 성능을 보여준다.

이상의 연구를 보면 작업 특성에 따라 적절한 스케줄링 방법이 존재한다는 결론을 내릴 수 있지만, 다음 논문들에서는 약간 상이한 입장을 취하고 있다.

[4]역시 기존에 heterogeneous cluster 에서 multiple parallel job 을 실행시킬 때의 성능에 대한 연구가 없다는 동기에서 출발했다. 실제의 heterogeneous 클러스터에서 real workload 에 가까운 job 들을 parallel processing 할 때의 성능을 측정하고자 한 것이다. 실험은 대상 클러스터 노드 모두에 openMosix 를 설치하여 openMosix 클러스터를 구성한 후, LAM/MPI 를 설치하고 NPB 등을 사용하여 성능을 측정했다. 우선 global scheduler 가 클러스터에 multiple job 을 할당하면, 각 노드가 자신에게 할당된 job 을 local scheduling 하게 된다. 실험 결과는 프로그램의 특성에 상관없이, 즉 computation-intensive application 이던지 communication-intensive application 에 상관없이 parallelism 을 높이면 거의 동일하게 성능이 좋아진다는 것이다.

이 논문에서의 실험은 openMosix 시스템에 기반을 두고 있으므로 다음과 같은 문제가 있다. 우선 이 논문은 openMosix 시스템의 dynamic load-balancing 을 이용하여 parallel job execution 을 하는데, 이 방법은 우선 모든 parallel process 를 한 노드에서 실행시킨 뒤 그 노드의 load 가 많아지면 load 가 적은 다른 노드를 찾아 process 를 migration 해서 load-balancing 하는 방법을 택하고 있다. 따라서 job execution 의 초기부터

parallelism 을 충분히 이용할 수 없을 뿐 아니라, migration overhead 도 발생하게 된다. 또한 migrated process 의 통신은 모두 starting node 의 MPI daemon 을 거치게 되므로 이 daemon 이 bottleneck 으로 작용하게 된다. 따라서 실제의 클러스터 컴퓨팅 환경을 잘 반영하고 있다고 보기는 어렵다.

[5]는 gang scheduling 의 효과에 대한 의문에서 출발한 논문이다. 일반적으로 distributed memory parallel computer 의 경우 gang scheduling 을 통해 최상의 속도를 얻을 수 있다고 얘기하는데, cluster 상에서도 같은 효과를 나타내는지 알아보기 위해 Beowolf 클러스터를 사용해서 실제 성능을 비교했다. 즉, 클러스터에서 communication-intensive job 을 실행시킬 때 local scheduling 을 사용했을 때와 gang scheduling 을 사용했을 때의 성능을 비교했다.

Local scheduling 을 위해서는 LAM MPI 를 이용하고, gang scheduling 을 위해서는 Score 클러스터 시스템을 사용했으며, 각 방법에 대해 노드수와 parallelism 을 증가시켜가면서 Matrix multiplication 과 Linpack 을 실행시켰을 때의 실행시간을 비교했다. 실험 결과 노드수가 많아지고 parallelism 이 커질수록 local scheduling 의 성능이 좋은 것을 나타났다.

그러나 LAM MPI 의 local scheduling 은 실제로는 implicit co-scheduling 이기 때문에 엄밀한 의미에서 local scheduling 이 gang scheduling 보다 낫다고 말하기는 어렵다.

[6], [7]은 여러 개의 MPI application 을 동시에 실행시킬 때, 어떤 scheduling 방법을 쓰는 것이 전체적으로 성능을 향상시킬 수 있는지에 대한 내용으로, co-scheduling, gang scheduling, hybrid scheduling 의 세가지 방법으로 프로그램을 실행시킨 후 성능을 측정했다. 여기서 hybrid scheduling 은 2-level scheduling 방법으로, 우선 gang scheduling 으로 동시에 실행할 작업의 개수를 제한하고, 동시에 실행되는 프로그램의 스케줄링은 co-scheduling 을 사용하는 방법을 말한다. 즉 12 개의 프로그램이 있을 때 동시에 실행할 수 있는 프로그램의 개수를 4 개로 제한한다면, 프로그램은 3 개의 set 으로 나뉘게 된다. Global scheduler 는 이 3 개의 set 에 대해 global time slice 를 적용해서 stop/restart 를 시키게 되고, 실행되는 각 set 의 프로그램들은 실행 노드의 local scheduling 에 따라 실행된다.

[6], [7]은 [5]의 결과, 즉 application 의 특성에 상관없이 언제나 co-scheduling 이 가장 좋은 parallel job scheduling 라는 사실을 기본 가정으로 하고 있다. 그럼에도 불구하고 hybrid scheduling 을 제안하게 된 이유는 physical memory 의 제한 때문이다. 즉 메모리가 부족하지 않을 상황에서는 co-scheduling 이 최적의 성능을 보여주지만, 프로그램의 개수가 많아져서 메모리가 부족하면 co-scheduling 이 더 이상 최적의 성능을 보여줄 수 없다는 것이다. 따라서 프로그램의 개수가 많아지면 우선 gang scheduling 으로 동시에 실행되는 프로그램의 개수를 제한한 후, co-scheduling 을 한다면 최적의 성능을 보여줄 수 있다고 주장하고 있다. 실제로 hybrid scheduler 의 성능에 대해서도, 프로그램의 개수가 적은 상황에서는 co-scheduler 와 거의 성능이 같고, 프로그램의 개수가 많아지면 co-scheduler 보다 성능이 낫지 때문에 가장 뛰어난 스케줄러라고 말하고 있다.

이 논문은 작업의 특성에 상관없이 co-scheduling 이 gang scheduling 보다 나은 성능을 보여준다는 가정하에 자신들이 제안한 hybrid scheduling 의 성능이 최적이라는 결론을 내리고 있기 때문에, [5]의 결론이 틀린다면 [6], [7]에서 제안한 방법이 최적이라고 주장하기 어렵게 된다.

4. 결론 및 향후과제

기존의 연구에서 부족한 점을 정리하자면, NPB 나 대표적인 벤치마크 프로그램 중 몇 가지를 가지고 성능을 측정했기 때문에

i) real workload 를 충실히 반영한다고 보기 어렵다.

ii) 소수의 benchmark 프로그램이 실제 상황에서 일어날 수 있는 다양한 job class 를 표현할 수 있다고는 보기 어렵다. 따라서 job characteristic 과 scheduling 방법의 상관 관계에 대해 정확히 표현하기 힘들다.

iii) 다양한 parallel job scheduling 방법의 성능을 비교한 예가 없다. 단순히 co-scheduling 을 사용한 예가 대부분이다.

따라서 다음과 같은 분야에 대한 성능 비교가 필요하다.

i) real workload 를 충분히 반영할 수 있는 job class 를 분류한다. 일반적으로 job class 는 작업 시간이나 필요로 하는 resource 양에 따라 나누는 경우가 많은데, computation attribute, communication attribute, memory attribute 등도 고려해서 좀 더 작업의 특성을 잘 나타낼 수 있도록 분류한다.

ii) backfilling, co-scheduling(각 노드의 local scheduling 에 의존), explicit co-scheduling(gang scheduling 과 같다. time slice 가 끝나면 무조건 process 를 preemption 한다), dynamic co-scheduling(message 를 수신할 process 가 실행 중이 아니면 preemption 해서 실행시킨다), implicit co-scheduling(communication 을 기다리는 프로세스의 경우 일정 시간 동안 spin-blocking 을 하며 메시지를 기다릴 수 있게 해서 통신하는 process 들은 같은 순간에 실행될 수 있도록 한다)등 다양한 parallel scheduling 방법에 대해 성능을 측정한다.

iii) 기존 연구의 대부분은 global scheduler 는 노드를 정해서 작업을 뿌리는 역할만 하고, 그 이후에 각 노드에서 어떤 스케줄링을 해야 성능을 향상시킬 수 있을까에 초점을 두었다. 하지만 [6], [7]의 경우처럼 local scheduler 뿐 아니라 global scheduler 의 스케줄링 방법도 변화시켜가면서 다양한 조합에 따라 어느 정도의 차이가 발생하는지 측정한다.

참고문헌

[1] Selective Preemption Strategies for Parallel Job Scheduling, Rajkumar Kettimuthu,
Vijay Subramani, Srividya Srinivasan, Thiagaraja Gopalasamy, ICPP'02
[2] Cluster Computing: High-Performance, High-Availability, and High-Throughpt Processing on a Network of Computers, Chee Shin Yeo, Rajkumar Buyya, Hossein Pourreza, Rasit Eskicioglu, Peter Graham
[3] Co-scheduling Techniques and Monitoring Tools for Non-Dedicated Cluster Computing, Francesc Solsona, Francesc Giné, Porfidio Hernández
[4] Parallel Computing on Clusters and Enterprise Grids: Practice and Experience, Adam K.L. Wong, Andrzej M. Goscinski, Int. J. High Performance Computing and Networking, 2006
[5] Local scheduling out-performs gang scheduling on a Beowulf cluster
[6] Hybrid Preemptive Scheduling of MPI Applications on the Grids, Aurelien Bouteiller, Hinde-Lilia Bouziane, GRID'04
[7] Hybrid Preemptive Scheduling of Message Passing Interface Applications on Grids, Aurélien Bouteiller, International Journal of High Performance Computing Applications, Vol. 20, No. 1, 77-90 (2006)