우선 순위 상속 전략을 이용한 내재 스케줄링

*오정섭, *정석용, ***박중기, *김재훈, *최경희, **정기현
*아주대학교 정보 및 컴퓨터 공학부, **아주대학교 전자 전기 공학부, ***한국전자통신연구원

Implicit Coscheduling based on Priority Inheritance

*Jung-Sup Oh, *Suk-Yong Jung, ***Joong-Gi Park, *Jaihoon Kim, *Kyung-Hee Choi, **Gihyun Jung
* Division of Information & Computer Engineering, Ajou University
** Division of Electronical & Electronics Engineering, Ajou University.
*** Electronics and Telecommunications Research Institute

요 약

본 논문에서는 미래형 고성능 컴퓨터인 클러스터링 시스템에 적합한 스케줄링 알고리즘을 제안한다. 클러스터링 시스템은 분산환경과 병렬환경을 같이 갖는 시스템으로써 스케줄링 전략이 굉장히 중요하고 또한 잘 해결되지 않는 부분이기도 하다. 시간 공유(time sharing) 스케줄링 전략으로 접근한 내재 스케줄링(implicit coscheduling)의 단점을 지적하고 이를 보완 하여 우선순위 상속 전략을 이용한 내재 스케줄링 전략을 제안한다

1. 서 론

최근에 일반 사용자들의 컴퓨팅 환경이 급변하고 있다. 기존에는 단순한 텍스트 위주의 정보가 주류를 이루었으나, 점차로 음성, 영상 및 이미지 등이 혼합된 멀티미디어 정보의 활용이 급증하고 있다. 특히, ATM, Fast Ethernet, Myrinet 등의 등장으로 인하여 네트워크의 성능이 급격히 발달됨에 따라 과학 현상에 대한 SOD(Simulation On Demand)나 VOD(Video On Demand)와 같은 사용자 중심의 서비스가 필요하게 되었다. 이에 따라, 고성능 컴퓨터에 대한 필요성이 대두되고 있다. 그러나 고성능 컴퓨터의 가격이 고가로 인하여 일부 전문가에 의해 제한적으로 활용되고 있으며, 시스템 성능 확장에 한계를 보이면서 급증하는 고성능 컴퓨터에 대한 일반 사용자들의 요구를 해소하지 못하고 있는 실정이다

미래형 고성능 컴퓨터인 클러스터링 시스템(clustering system)은 병렬과 분산 컴퓨팅 환경을 모두 갖는다. 분산 환경에서 병렬 응용 프로그램들을 스케줄링하는 문제는 굉장히 중요하고, 또 잘 해결되지 않는 문제들이다.

분산 환경에서의 스케줄링은 시간 공유(time sharing) 스케줄링과 공간 공유(space sharing) 스케줄링으로 크게 나뉘어 진다. 시간 공유 스케줄링의 대표적인 예는 coscheduling[1]을 들 수 있고, 공간 공유 스케줄링의 대표적인 예는 gang scheduling[2]을 들 수 있다. 시간 공유 스케줄링의 경우, 프로세스의 이주(migration)나 프로세스 수행시간의 예측이 필요치 않으므로 다소 효율적이다. 그러나, 좋은 성능을 발휘하기 위해서는 통신하는 프로세스들을 항상 동시에 스케줄링 시켜주어야 하는 문제를 지니고 있다. 많은 연구가 진행 되어 지는 동안에 프로세스의 이주나 수행시간의 예측 보다는 코스케줄링 하는 기술에 많은 연구가 진행되어 졌고, 여러 해결책 들이 제시 되고 있다. 본 논문에서는 시간 공유 스케줄링 전략에 바탕을 둔 스케줄링 전략을 소개할 것이다.

본 논문의 2 장에서는 이미 진행된 연구들에 관하여 알아보고, 3 장에서는 기존 연구에서의 단점을 보완한 새로운 전략을 제안하고, 4 장에서는 결론과 향후과제를 언급하기로 한다.

2. 관련 연구

2.1 요구 기반 코스케줄링 (Demand-based coscheduling)

요구 기반 코스케줄링[3]은 병렬 응용 프로그램을 수행시키는 방법에 있어서, 커널 내부에 일정한 자료 구조를 사용하여 통신을 하는 프로세스들을 알아내어 스케줄링하는 방법이다. 요구 기반 코스케줄링은 다시 크게 2 가지 방법으로 세분할 수 있다. 첫째는 메시지 패싱 멀티프로세서 환경에 적합한 동적 코스케줄링(Dynamic coscheduling) 전략이다. 이것은 서로 통신을 한다거나, 캐쉬 무효화 신호 같은 것들을 검사하여, 현재 수행중인 프로세스들을 알아내어 스케줄링 시키는 전략이다. 둘째는 공유 메모리를 사용하여 구현한 예측 코스케줄링이다(Coscheduling). 예를

요구 기반 코스케줄링은 다음과 같은 이점을 가질 수 있다. 첫째, 커널 내부의 자료 구조를 이용하여 알고리즘을 구현하였으므로, 프로그래머가 특별한 형식으로 프로그램을 작성할 필요가 없다. 둘째, 적은 수의 프로세서에서 많은 수의 프로세스가 수행된다면, 지역적인 통신 패턴을 사용하여 보다 나은 성능을 제공해 준다. 셋째, 프로세스들 사이에 새롭게 시작되는 통신은 요구 기반 스케줄링에 의해서 동기화를 위한 요구로 자동적으로 인식되어 진다. 이러한 것은 OLE와 같은 새로운 패러다임에 매우 적합하다. 넷째, 스케줄링에 관한 판단을 지역적으로 내릴 수 있다. 다섯째, 병렬 작업을 구성하는 프로세스의 수에 제한을 받지 않는다.

그러나, 이러한 요구 기반 스케줄링은 기본적으로 멀티프로세서(multi-processor)환경에 알맞게 설계되어졌다. 기본적으로 커널 내부의 자료구조를 여러 개의 프로세서가 공유할 수가 있어야 하는데, 분산환경에서는 서로의 자료구조를 공유하는 것이 결코 쉬운 문제가 아니다.

여러 가지 장점에도 불구하고 요구기반 스케줄링 알고리즘은 분산환경에서는 적합하지 않은 치명적인 단점을 가지게 된 셈이다.

2.2 내재 스케줄링 (Implicit coscheduling)

NOW(Network Of Workstations) 프로젝트에서 연구 중인 내재 스케줄링[4]은 클러스터링 시스템에 적합한 알고리즘을 가지고 있다. 내재 스케줄링은 메시지를 받기 위해서 블록킹 되려는 프로세스를 일단은 반복 대기(spin wait)를 시킨 후에 일정한 시간이 지난 후에 블록킹 시킨다. 이때 메시지가 도착하면, 메시지를 받은 프로세스의 우선순위를 높여서 곧바로 수행될 수 있게 끔 하는 전략이다.

이 전략에서는 반복 대기를 사용함으로 인하여 불필요한 문맥 교환(context-switching)시간을 절약하는 이점을 보이고 있다. 또한 메시지를 받은 프로세스의 우선순위를 높여서 곧바로 수행하여 줌으로 인하여 현재 통신하는 프로세스들을 동시에 수행하여 주는 것과 같은 효과를 나타낼 수 있다.

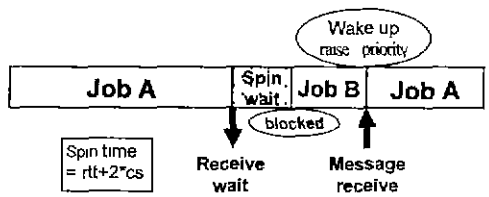

그림 1. Implicit coscheduling mechanism

그림 1 은 내재 스케줄링의 한 예를 보인다. 하나의 프로세서에서 수행중인 Job A 는 다른 프로세서에서 수행중인 Job A'과 메시지를 교환하기 원한다 이때 Job A 는 Job A'으로부터 메시지를 기다리게 된다. 다른 프로세서로부터 메시지가 곧바로 도착하지 못할 경우, Job A 는 일정한 시간 동안 반복 대기를 수행하다가 블록킹 된다. Job A 가 블록킹 되었으므로 프로세서는 Job B 를 수행시킨다. Job B 가 수행하던 중에 Job A' 에서 Job A 로 메시지가 도착하면, 프로세서는 Job A 를 깨워서 우선순위를 높여서 곧바로 수행될 수 있도록 한다.

3. 우선순위 상속 전략을 이용한 내재 스케줄링 (Implicit coscheduling base on priority inheritance)

프로세서는 각각의 task 에 대하여 일정한 양 만큼의 time quantum 을 할당해 주고, 주어진 time quantum 에서는 선점되지 않도록 보장해 주는 FIFO(First-In First-Out)방식의 스케줄링을 수행한다. 따라서 임의의 시점에서 time quantum 을 할당받은 task 가 가장 높은 우선순위를 부여 받은 것이라 볼 수 있다.

그림 1 에서 본 내재 스케줄링은 단순히 메시지를 받는 프로세스의 우선순위를 올려서 메시지를 받는 즉시 프로세스를 수행시켜 줌으로 인하여 우선순위 역전 현상을 보여주게 된다. 다음의 예를 보자.

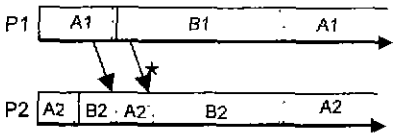

그림 2. Examples of implicit coscheduling

그림 2의 →는 메시지의 흐름을 나타낸다. 초기에 P1과 P2에서 Job A의 subtask인 A1과 A2가, Job B의 subtask인 B1과 B2가 각각 수행이 되었다. A2가 A1으로 부터 메시지를 기다리다가 블록킹되어 지고 B2가 수행된다. B2는 A1으로 부터 메시지가 도착하면, 곧바로 선점되어지고 A2가 다시 수행 된다. 그리고 나서 A1이 A2로 부터 메시지를 기다리다가 블록킹 되어서 B1이 수행이된다. 만일 B1이 B2로 메시지를 보낸다면, A2는 선점되어 지고 B2가 수행된다. A2가 time quantum을 다 소비하지 않았기 때문에 계속 수행이 되어야 함에도 불구하고, A1이 프로세서를 양보하여 줌으로 인하여 A2가 우선순위가 낮은 B2에 의해서 선점되어지는 우선순위 역전현상이 발생하게 된다.

다시 말해서 ★시점에서 A2를 계속 수행 시켜 A1을 깨워주어야 함에도 불구하고, 오히려 B1과 B2가 2개의 프로세서를 모두 차지하는 일종의 우선순위 역전 현상이 발생하게 된다.

물론 그림 2에서도 보이는 바와 같이 P1과 P2에서 같은 Job이 비교적 같은 시간대에 coscheduling 되고 있는 것을 볼 수 있다. 하지만, 우선순위 역전 현상이 생기게 되므로 실시간 성을 요구하는 응용에는 부적합한 결과를 내게 된다.

그림 2에서와 같은 단점을 보완하기 위해서 본 논문에서는 우선순위 상속 전략을 도입하여 문제를 해결해 보았다. 커널 내부의 task 자료구조내에 우선순위 필드를 삽입하여 낮은 숫자가 높은 우선순위를 나타내도록 하였다 그림 3에서 ()안에 있는 숫자는 물리적인 우선순위 값을 나타낸다.

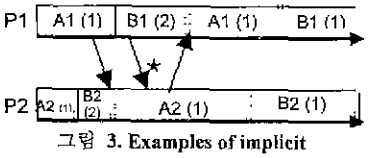

그림 3. Examples of implicit coscheduling with priority

프로세서는 항상 ready state에 있는 프로세스들 중에서 우선순위가 가장 높은 프로세스를 스케줄 한다. 그림 3에서 보면, 초기상태에서 P2에 위치한 A2가 블록되었을 경우, 다음으로 가장 높은 우선순위를 가진 B2를 스케줄 한다. 임의의 메시지가 도착했을 경우, 커널에서 다시 스케줄링 루틴이 호출되게 되는데, 이때는 A2가 우선순위가 가장 높기 때문에, 내재 스케줄링과 마찬가지로 메시지를 받는 즉시 스케줄링 된다. 또한, 그림 3의 ★ 시점에서 B1에서 B2로 향하는 메시지가 도착한다고 하더라고 그 순간의 A2는 우선순위가 B2보다 높으므로 A2가 계속 수행된다. 또한 Job A가 time quantum을 다 소비하여 time out이되면, 가장 낮은 우선순위를 배당받고, 나머지 프로세스들의 우선순위를 1씩 증가시킨다. 또한 Job이 종료되었을 경우, 종료된 job보다 높은 우선순위를 가지고 있는 job들의 우선순위를 낮추어 줌으로써, 우선순위 값이 한없이 커지는 것을 방지하도록 한다.

우선순위 값은 job 단위로 분배가 되어져야 한다. 만일 하나의 동일한 프로세서에 Job A의 subtask인 A1과 A2가 같이 실행이 된다면, A1과 A2는 동일한 우선순위를 갖는다. 이경우 커널 내부의 스케줄링 루틴에 의해서 프로세서를 덜 사용한 프로세스에 우선권을 부여하여 스케줄링을 한다.

4. 결론 및 향후 과제

미래형 고성능 컴퓨터인 클러스터링 환경은 최근에 MOD(Multimedia On Demand)와 같은 시스템에서 많은 필요성이 대두 되고 있다. 따라서 실시간 성을 갖춘 스케줄링의 알고리즘으로서, 본 논문에서 제안한 알고리즘이 좋은 성능을 발휘할 것으로 기대된다.

앞으로 이 알고리즘의 성능을 실험을 통하여 검증을 하고자 한다. 실험은 병렬 응용 프로그램을 수행하여 일반적인 스케줄링 알고리즘과 비교하여, 병렬 응용 프로그램의 수행시간을 비교 하여 볼 것이다.

참고 문헌

[1] J. K. Ousterhout "Scheduling Techniques for Concurrent Systems," *In Third International Conference on Distributed Computing Systems*, pp 22-30, May 1982.

[2] K. Al-Saqabi, et al. "Dynamic Load Distribution in MIST," *In PDPTA 97*, 1997.

[3] Patrick G. Sobalvarro. et al., "Demand-based Coscheduling of Parallel Jobs on Multiprogrammed Multiprocessors," *IPPS '95 Workshop, Santa Barbara, CA, USA, April, 1995.* pp. 106~126

[4] Andrea C. Dusseau, et al., "Effective Distributed Scheduling of Parallel Workloads," *SIGMETRICCS 96*, Philadelphia, PA, USA. 1996.