NUMA 환경에서 메모리 친화력을 고려한 부하 균등 모델

윤대석[†O], 박희권[†], 최종무[††]

[†] 단국대학교 컴퓨터학과

woodsmen@dankook.ac.kr, parkhk81@dankook.ac.kr, choijm@dankook.ac.kr

Memory Affinity based Load Balancing Model for NUMA System

Dae Seok Youn[†O], Heekwon Park[†], Jongmoo Choi[††]

[†]School of Computer Science and Engineering, Dankook University

요 약

AMD에서 사용한 HyperTransport 기술 기반 다중 처리기가 좋은 성능을 보이면서 최근 NUMA(Non Uniform Memory Access) 환경에 대한 관심이 증가하고 있다. 본 논문에서는 NUMA 시스템을 위한 부하 균등 모델을 제안한다. 다중 처리기 시스템에서 운영체제는 특정 처리기에 부하가 많아지는 것을 부하가 적은 처리기로 나누어 주기 위해 부하 균등 기법들을 가지고 있다. 이런 부하 균등 기법은 처리기가 가지고 있는 태스크 개수에 의존적인 연구가 많다. 본 연구에서는 NUMA 시스템의 메모리 접근 비용이 위치에 따라 다른 것을 반영한 부하 균등 기법의 모델을 제시한다. 이를 위해 모의 실험 환경을 구축하고 특정 상황들에 대한 실험을 통해 증명한다.

1. 서 론

처리기의 집적도가 증가하면서 최근 출시되는 CPU는 대부분 다중 코어를 채택하고 있다. 인텔은 Dual core, Quad core에 이어 Many core라는 16개~32개의 CPU를 포함한 다중 처리기 출시를 계획하고 있다. 한편, AMD에서는 HyperTransport라는 기술을 갖는 다중 처리기를 출시하여 좋은 성능을 보이고 있다 [7]. 이 기술은 CPU들이 메모리를 서로 다른 접근 속도로 접근하게 되는 NUMA 구조를 기반으로 한다.

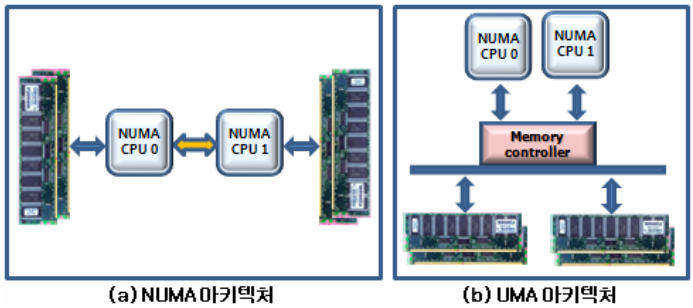

그림 1 메모리 접근 방식에 따른 분류

다중처리기 시스템에서 메모리 접근 방식은 크게 NUMA(Non-Uniform Memory Access)구조와 UMA (Uniform Memory Access)구조로 구분된다. <그림 1>은 NUMA구조와 UMA구조의 차이를 도시하고 있다. NUMA 구조는 메모리의 위치에 따라 접근 비용이 다르게 설계되었다[1]. 반면, UMA구조의 경우는 각 처리기가 동일한 메모리 컨트롤러를 통하여 메모리에 접근하도록 설계되었다. 즉, 모든 메모리는 메모리 컨트롤러를 통하여 접근되며 같은 비용으로 접근 할 수 있다.

NUMA구조의 대표적인 예로는 AMD사의 opteron 처리기가 있다. 이는 그림 1의 (a)와 같은 구조이다. NUMA구조는 메모리 접근 속도에 따라 지역(local) 메모리와 원격(remote)메모리로 구분한다. 이때 각각의 처리기가 지역 메모리를 우선적으로 참조[1]함으로써 메모리 접근 시 발생할 수 있는 버스의 병목현상을 최소화 할 수 있다. 반면 UMA구조의 대표적인 예로는 INTEL사의 XEON 처리기가 있다. 이는 <그림 1>의 (b)와 같은 구조이다. 각각의 처리기가 동일한 버스를 통하여 메모리를 접근함으로 병목현상을 유발 할 가능성이 높다. XEON은 이러한 병목현상을 최소화 하기 위하여 MCH(Memory Controller Hub)와 처리기 사이의 FSB(Front Side Bus)를 병렬화 했다.

[1] 운영체제의 스케줄링 정책에 따라 효율성이 달라 질 수 있다.

기존 NUMA구조는 클러스터 컴퓨터, 대용량 데이터베이스 등 대형 서버를 위한 다중처리기 시스템에서 주로 사용되었다. 하지만 최근 다중처리기와 대용량의 메모리를 탑재한 개인용 컴퓨터가 보편화 되면서, 또한 처리기의 개수가 증가함에 따라 메모리 접근의 병목 지점을 줄이기 위해 NUMA구조의 사용 범위가 확대되고 있다.

NUMA구조의 특성상 데이터 참조의 지역성(locality of reference)은 전체 시스템 성능에 큰 영향을 미친다. 따라서 NUMA구조를 지원하는 운영체제는 데이터 참조의 지역성을 고려하여 메모리를 할당해야 한다. 그리고 부하균등(load balancing)시에도 데이터의 지역성을 고려한 기법을 적용해야 한다. 기존 지역성을 고려한 연구로는 [1][2][6]등이 있다. [1]에서는 메모리 접근 속도가 같은 메모리를 도메인(domain)으로 하여 같은 도메인 안에서의 부하균등을 우선적으로 수행하도록 설계하였다. 이는 데이터 참조의 지역성을 고려한 부하균등기법이라 할 수 있다. 하지만 전체 시스템의 부하균등을 고려하지는 못한다. 현재 리눅스는 [1]에서 제안된 기법을 기반으로 부하균등을 구현하였다.

본 논문에서는 데이터 참조의 지역성을 모델링하고 이에 따른 가중치를 부과함으로써 전체 시스템의 부하균등을 이룰 수 있도록 설계 하였다.

본 논문의 구성은 다음과 같다. 2장에서는 NUMA구조를 지원하기 위한 리눅스의 스케줄링 정책에 대하여 기술한다. 3장에서는 부하균등 시 고려되는 지역성에 대해 측정 가능한 요소로 정량화한 모델을 제안하고 데이터 참조의 지역성을 고려한 부하균등에 대하여 설명한다. 4장에서는 모의 실험을 통한 본 논문의 모델에 대한 검증을 하고, 마지막으로 5장에서는 결론 및 향후 연구 계획에 관하여 기술한다.

2. 관 련 연 구

리눅스의 스케줄러는 2.5 버전 이후로 O(1) 스케줄러라는 이름으로 태스크 개수와 상관없이 상수 시간에 스케줄러의 루틴을 수행 가능하게 만들었다. 현재 최신 Linux(버전 2.6.24) 스케줄러는 SMP와 NUMA 아키텍처를 위해 많은 기법이 사용되어 있다.

기본적으로 Linux 스케줄러는 선점형이며, 동적 우선순위 큐들을 가지고 각 CPU의 태스크 활용에 따라 우선순위를 계산하고 적용한다. 하나의 태스크가 특정 CPU의 실행 큐에 등록이 되고, 부하가 많지 않다면 등록된 CPU에서 수행한다. 이런 것을 CPU 친화력(affinity)이라고 하는데, 태스크가 수행되던 CPU에서 수행되는 것이 그 CPU에서 쓰던 캐쉬 데이터를 쓸 수 있기 때문에 메모리의 참조가 적어져서 상당한 이득을 볼 수 있는 것이다.

Linux 스케줄러는 우선 순위 비트맵을 통해 빠르게 우선순위 태스크 선택을 할 수 있다. 각 실행 큐는 활동(active) 배열과 만료(expired) 배열로 구성된다. 각 CPU의 태스크 스케줄링은 활동배열에 있는 우선순위 비트맵을 참조 하여 우선순위가 가장 높은 태스크를 수행한다. <그림 2>은 실행 큐에서 태스크의 우선순위를 고려하여 수행되는 과정을 도시하였다. 활동 배열에 있는 모든 태스크가 주어진 시간이 만료되었다면 만료 배열과 포인터를 교환하여 만료 배열을 활동 배열로 바꾸고 수행을 계속한다[5].

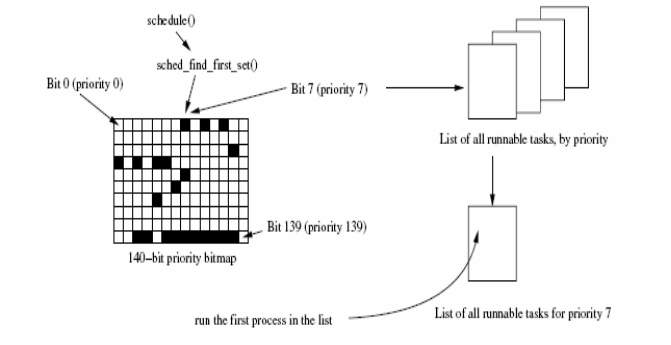

그림 2 Linux 우선순위 스케줄링

태스크가 생성이 될 때, 부모 태스크의 CPU의 친화력, 우선 순위 등 많은 자원을 물려 받고 수행된다. 하나의 CPU에 많은 태스크들이 대기하고 있다면, 다른 CPU와 작업 부하를 공평하게 수행을 하기 위해 태스크를 이동 시킨다.

UMA구조의 다중처리기 시스템인 경우엔, 작업 부하를 나누어 주기 위해 위와 같은 부하 균등을 하더라도 모든 CPU가 메모리 간 접근 속도가 같아서 태스크의 다른 CPU로의 이동은 작업 부하를 공평하게 가질 수 있다. 하지만 NUMA 시스템의 경우엔 메모리 접근 속도가 CPU마다 다르게 적용되기 때문에 CPU 작업 부하를 공평하게 하기 위해 UMA구조와 같이 한다면 원격 메모리의 참조가 많아져 성능 저하의 요인이 될 수 있다. 그래서 Linux에서는 NUMA구조의 다중 처리기를 위해 도메인 개념을 도입하여 메모리 참조 지역성을 고려하였다[6].

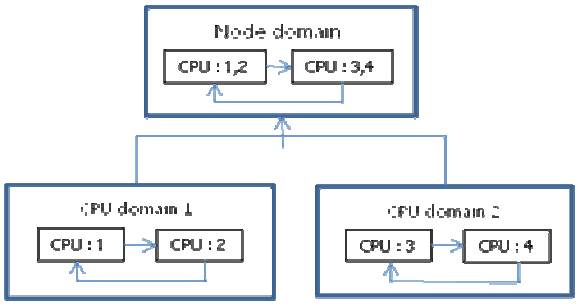

그림 3 *sched domain* 구조

<그림 3>에서 "CPU domain"은 같은 지역 메모리를 공유하는 CPU 그룹이다. 같은 도메인 안에 있는 처리기들의 부하를 우선적으로 고려하며, 전체 처리기들의 부하는 같은 도메인 안의 처리기들의 부하가 균등할 시에 고려된다. 즉 같은 도메인 안에서의 부하 균등은 주기적인 반면, 도메인 간 부하 균등은 비주기적이다. 이는 Linux의 데이터 지역성을 고려한 부하균등 정책이다[8]. 본 논문에서는 같은 도메인에 속한 처리기들의 부하를 균등하게 유지하면서 동시에 도메인 간에 데이터 지역성을 고려할 수 있는 기법을 제안한다.

3. 부하 균등 모델

본 논문에서는 NUMA시스템에서 효과적인 부하균등을 위한 모델을 제시한다. 수식에 의존한 모의 실험을 하였으며, 조건은 다음과 같다.

(1) 태스크(i) 하나의 총 수행 시간은 처리기에서의 수행시간, 메모리 접근 시간 그리고 수행되고 있는 처리기에서 문맥교환의 부하 시간을 합한 것이다.

$$T_i = T_i^{cpu} + T_i^{mem} + T_{cpu_j}^{cs}$$

(2) 각 태스크의 메모리 접근 시간은 지역 메모리 접근 시간과 원격 메모리 접근 시간의 합으로 계산된다.

$$T_i^{mem} = T_i^{Rmem} + T_i^{Lmem}$$

원격 메모리 접근 시간은 지역 메모리 접근 시간에 비해 10%~30%정도 차이가 있다. 실제 Opteron 멀티 프로세서 환경에서는 <그림 4>와 같이 cpu0과 cpu1을 기준으로 1 hop으로의 메모리 접근 속도와 2 hop으로의 메모리 접근 속도는 자신의 지역 메모리 접근 속도에 비해 각각 10%, 30% 증가한다[7].

(3) 지역 메모리 접근 시간과 원격 메모리 접근 시간은 각각 메모리 접근 횟수와 접근 시간의 곱으로 계산된다.

$$T_i^{Lmem} = N_i^{Lmem} * T_i^{Laccess}$$

$$T_i^{Rmem} = N_i^{Rmem} * T_i^{Raccess}$$

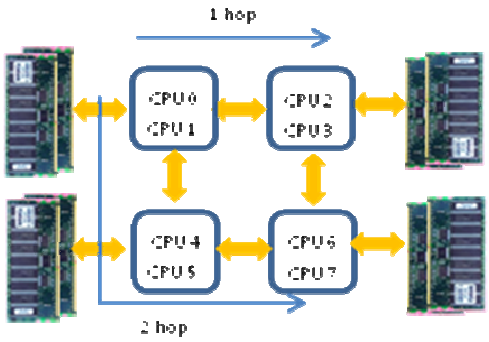

그림 4 AMD Opteron의 메모리 접근

(4) 태스크가 수행되고 있는 처리기의 문맥교환 부하 시간은 태스크에 개수에 의존적이다.

$$T_{cpu_i}^{cs} \propto (N_{tasks}^{cpu_i} * \alpha)$$

(5) 처리기 하나에 수행되고 있는 모든 태스크의 부하 시간을 합하여 총 부하를 계산한다.

$$L_{cpu_j} = \sum_{i=0}^{N_{tasks}^{cpu_j}} T_i$$

(6) 모의 실험에서 모든 태스크의 처리가 끝난 후의 총 시간은 태스크 수행시간 중 최대값으로 한다.

$$Max(T_0, T_1 \ldots, T_n)$$

본 논문에서 제시 하는 모델은 위의 수식을 통해 각 CPU의 총 부하를 계산 하고, 메모리 친화력을 고려하여 태스크 이동을 한다. 태스크 이동은 <그림 5>와 같이 CPU0과 CPU1의 도메인에서 부하 균등을 요청한다면 다른 도메인들은 블랙 박스가 된다. 각각의 블랙 박스에 부하를 구하고 전체 평균 부하보다 높은 박스 안에 메모리 참조가 적은 태스크가 이동함으로써 처리기가 원격메모리 참조를 최대한 적게 하고 처리기의 수행능력을 높여준다. 단, 태스크를 이동하여 옮겨간 도메인의 부하가 더 커지거나 같아진다면 태스크 이동을 자제한다. 그리고 같은 도메인 내에서의 부하 균등은 빈번히 수행되어 부하를 맞춰 주게 된다.

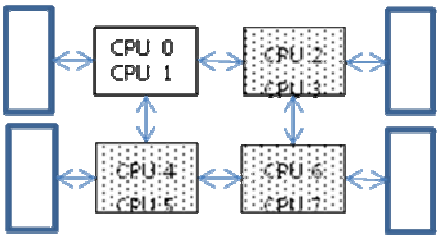

그림 5 부하 균등 모델

4. 실 험 결 과

본 논문에서는 다음 3가지 기법, 구체적으로 처리기에 할당된 태스크의 이동이 없이 수행되는 환경(Fix), 전체 처리기의 태스크 개수를 최대한 동일하게 만들어주는 환경(Forced Fair), 3장에서 설명한 단계별 균등 부하의 환경(Multi-level)으로 실험하였다. 실험은 할당된 태스크가 모두 종료되는 시점이며, 최종 시간은 태스크들 중 수행시간의 최대값으로 한다.

각 처리기에서 태스크 하나가 수행을 할 때마다 처리기의 부하를 계산하고 부하들의 표준 편차를 구하여 처리기들의 부하의 격차를 보여준다.

모의 실험은 가상적으로 만들어진 100개의 태스크가 1~1000ns 사이의 임의의 수행시간을 가지고 만들어진다. 태스크는 요구되는 메모리 공간은 각각 다르게 생성되며, 태스크의 수행시간이 지날 때마다 증가되는 구조이다. 부하균등을 통해 다른 처리기로 옮겨간 태스크는 메모리의 원격 메모리와 지역 메모리를 구분하여 관리한다. 모의 실험에서 부하 균등 기법은 처리기의 태스크 개수에 의존한 것이다.

본 논문은 두 가지의 상황에 고려하여 실험하였다. 메모리 집중적 태스크들로 구성된 모의 실험과 처리기 집중적 태스크들로 구성된 모의 실험이다.

(단위:마이크로초)

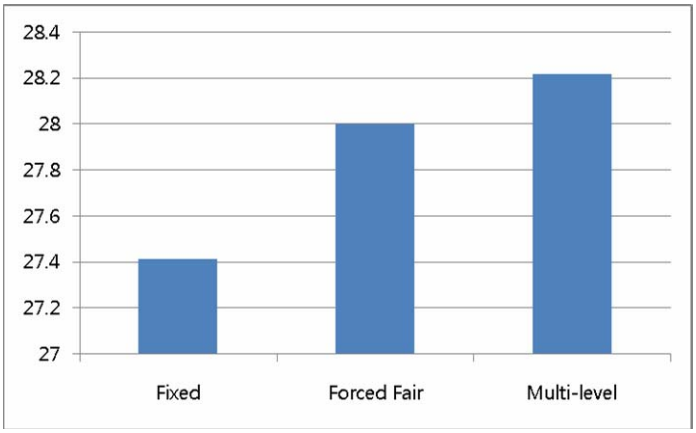

그림 6 처리기 부하 편차(메모리 집중적 태스크)

<그림 6>과 <그림 7>의 모의실험은 메모리 사용량이 많은 태스크들로 구성되어 수행 되었다. 처리기의 수행시간이 적은 반면에 메모리의 사용량이 많아 태스크 이동이 전혀 없는 경우(Fixed)에 부하 편차가 낮고, 수행시간 또한 짧게 나왔다. Multi-level의 경우 Forced Fair 기법보다 부하는 높지만 상대적으로 태스크의 이동이 적어 수행시간은 짧게 나왔다. 이는 메모리 집중적 태스크의 이동은 원격메모리의 접근으로 인한 시스템 성능이 저하 될 수 있다는 것이다.

(단위: 밀리초)

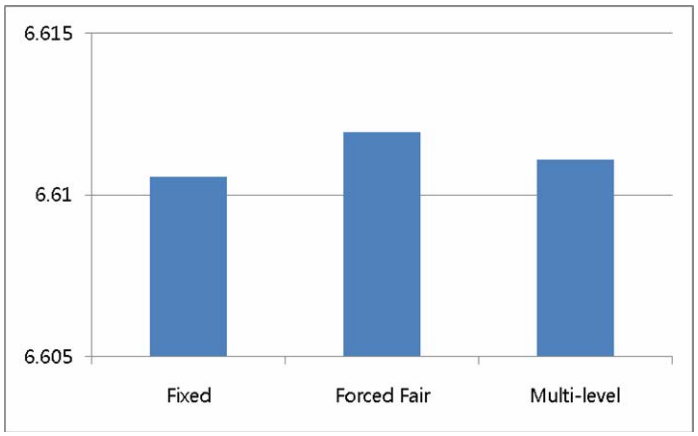

그림 7 모의실험 수행시간(메모리 집중적 태스크)

<그림 8>과 <그림 9>는 처리기 집중적 태스크들로 구성되어 모의 실험 하였다. 메모리 집중적 태스크와는 달리 태스크의 이동으로 이득을 볼 수 있는 상황이다. 실험 결과에서는 Multi-level이 많은 이득을 보았다. 이 결과는 태스크의 이동이 같은 도메인에서 먼저 수행되고, 그 후에 전체 시스템에서의 이동이 가장 적절하다는 것을 보여준다.

(단위: 마이크로초)

그림 8 처리기 부하 편차(처리기 집중적 태스크)

(단위: 밀리초)

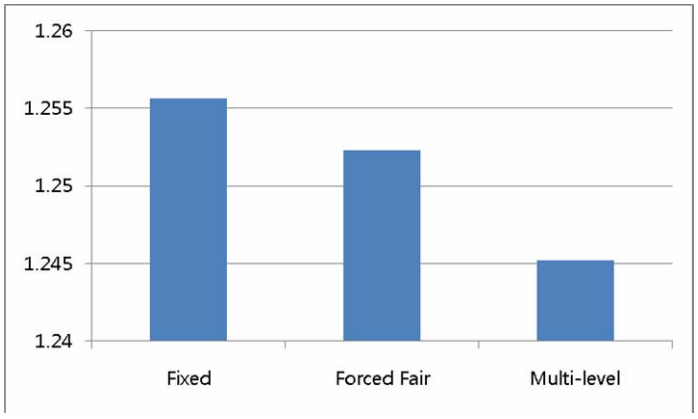

그림 9 모의실험 수행시간(처리기 집중적 태스크)

실험 결과를 보면 NUMA 시스템에서 태스크의 이동은 원격메모리 접근에 대해 고려되어야 한다는 것을 보여준다. 본 논문 3장에서 제시한 부하 균등은 원격 메모리의 접근은 최소화 할 수 있고, 처리기들의 전체 부하를 맞춰 줄 수 있는 기법이라 할 수 있다.

5. 결 론

NUMA 시스템에서의 처리기들 간의 부하 균등 기법은 태스크들의 원격 메모리 접근을 고려하여 시스템의 부하를 줄여줌과 동시에 수행 속도 또한 보장해 주어야 한다. 앞으로 AMD Opteon 시스템에서 본 논문의 부하 균등 기법을 적용할 것이며, 실제 NUMA 시스템에서의 성능을 모의 실험과 비교 및 분석 해 볼 것이다.

참고 문헌

[1] M. Correa, R. Chanin, A. Sales, R. Scheer, and A. F. Zorzo. Multilevel Load Balancing in NUMA Computers. In 20th ACM Symposium on Operating Systems Principles (SOSP 2005), pages 1{9, Brighton, United Kingdom (submitted), October 2005.

[2] M. Correa, A. Zorzo, R. Scheer, Operating system multilevel load balancing. ACM SAC, pp. 1467-1471, 2006.

[3] http://www.digit-life.com/articles2/cpu/rmma-numa.html, Non-Uniform Memory Architecture (NUMA): Dual Processor AMD Opteron Platform Analysis in RightMark Memory Analyzer.

[4] http://www.valueram.com/fb-dimm/default.asp, FB-DIMM.

[5] Robert Love, Linux Kernel development, 2003.

[6] Rafeal Chanin, Monica Correa, Paulo Fernandes, Afonso Sales, Roque Scheer, Avelino F. Zorzo, “Analytical Modeling for Operating System Schedulers on NUMA systems”, Electronic Notes in Theoretical Computer Science, pp. 131-149, 2006.

[7] AMD application note, “Performance Guidelines for AMD Athlon 64 and AMD Opteron ccNUMA Multiprocessor Systems”, June 2006.

[8] Martin J.Bligh, Matt Dobson, Darren Hart, Gerrit Huizenga, “Linux on NUMA Systems”, Proceedings of the Linux Symposium, vol. 1, 2004.

[9] Christoph Lameter, “Local and Remote memory in a Linux/NUMA System”, Silicon Graphics, Inc., 2006.