NUMA 환경에서 리눅스의 스케줄링, 메모리 할당 및 메모리 인터리빙 효과 분석

Analysis of Linux Scheduling Memory Management and Memory Interleaving in NUMA Environments

저자
(Authors)

김해천, 최종무
HaeCheon Kim, Jongmoo Choi

출처
(Source)

한국정보과학회 학술발표논문집 , 2015.12, 66-68 (3 pages)

발행처
(Publisher)

한국정보과학회
KOREA INFORMATION SCIENCE SOCIETY

URL

http://www.dbpia.co.kr/Article/NODE06602898

APA Style

김해천, 최종무 (2015). NUMA 환경에서 리눅스의 스케줄링, 메모리 할당 및 메모리 인터리빙 효과 분석. 한국정보과학회 학술발표논문집, 66-68.

NUMA 환경에서 리눅스의 스케줄링, 메모리 할당 및 메모리 인터리빙 효과 분석

김해천 최종무
단국대학교
haecheon100@gmail.com , choijm@dankook.ac.kr

Analysis of Linux Scheduling, Memory Management and Memory Interleaving in NUMA Environments

HaeCheon Kim, Jongmoo Choi
Department of software Dankook University

요 약

기술의 발전에 힘입어 CPU의 성능과 메모리 용량이 지속해서 향상 되었고, 더 나아가 멀티코어로 발전했다. 하지만 하나의 칩으로 이뤄진 프로세서 안에서 코어의 개수만을 증가시키는 것은 한계가 있었다. 그리하여 프로세서와 메모리의 집합이 여러 개인 컴퓨터 시스템, NUMA로 발전하게 되었다. 하지만 NUMA에서 메모리 접근 시간이 다른 문제가 있었고 이에 따라 프로세스와 스레드의 스케줄링 기법과 메모리 할당 정책에 대한 연구가 지속되어 왔다. 하지만 여러 논문에서 디폴트 리눅스와 성능 차원에서 비교하였지만, NUMA 환경에서 디폴트 리눅스의 메모리 할당과 스케줄링에 대한 분석이 부족하였다. 그래서 본 논문에서는 벤치마크들 수행될 때 프로세스와 스레드의 실제 core의 위치와 메모리 노드의 위치를 일정주기로 샘플링하여 측정하였다. 이를 통해 벤치마크들의 특성에 따라 NUMA에서 스레드와 메모리 배치의 문제점 및 특징을 파악할 수 있었다. 메모리 인텐시브 벤치마크는 한 노드에서 메모리 대역폭이 집중적으로 발생하여 성능 저하로 이어졌다. 이를 위해 메모리의 할당을 노드들에서 균등하게 행하는 인터리빙 정책을 사용함에 따라 메모리 대역폭을 균등하게 나눌 수 있었고 20%의 성능 향상을 보였다.

1. 서 론

컴퓨터 시스템에서 하드웨어의 발전은 지속적으로 진행되어 왔다. 기술발전에 힘입어 CPU의 성능과 메모리 용량이 지속적으로 향상 되어 왔다. 그리고 더 나아가 하나의 프로세서 안에 cpu core가 여러 개 존재하는 멀티코어로 발전했다. 하지만 하나의 칩으로 이뤄진 프로세서 안에서 코어의 개수만을 증가시키는 것은 한계가 있었다. 그래서 그림 1처럼 여러 개의 프로세서를 하나의 시스템으로 구성하는 NUMA 아키텍처가 발전하게 되었다. NUMA(Non-Uniform-Memory-Access)는 임의의 프로세서를 기준으로 메모리들로 접근속도가 차이가 있는 시스템이다. 쉽게 프로세서의 개수를 증가시킬 수 있게 되었고, 메모리 용량 또한 각 프로세서와 그룹 (노드 혹은 소켓)으로 용량이 크게 향상되었다. 최근 AMD Bulldozer machine의 경우 8개의 노드와 각 노드에 8개의 core가 존재하여 64개의 코어가 존재한다. 이렇게 하드웨어는 눈부시게 발전해 왔다. 운영체제도 이러한 하드웨어를 효율적으로 지원하고자 연구가 진행되고 있다. NUMA환경에서의 주된 이슈는 한 프로세서에서 프로그램이 수행될 때 어느 노드의 메모리에 접근하느냐에 따라 접근속도에 차이가 발생하는 것이다. 그래서 프로세서와 메모리의 구조를 고려한 어플리케이션의 배치가 성능에 영향을 미치게 된다. [1]

또한 여러 복잡한 프로그램들이 함께 수행될 때 간단히 접근 속도가 빠른 지역 메모리만 사용하는 것보다 오히려 접근 속도가 더 느린 원격 메모리와 함께 이용하는 것이 성능에 도움이 될 수 있다. [2]

NUMA 시스템이 발전함에 따라 하드웨어 복잡성이 증가했고 노드간 대역폭이 각각 다른 구조에서 운영체제의 스케줄링 관한 연구도 존재했다. [3] 이뿐만 아니라 UMA에서 NUMA로 발전됨에 따라 LLC(Last Level Cache) 및 메모리 컨트롤러의 충돌 문제가 커졌다.[4] 또한 현재의 리눅스는 first-touch

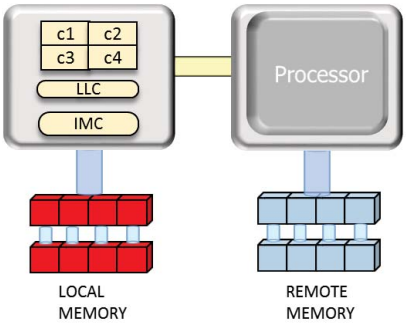

그림 1 NUMA 아키텍처 구조

이 논문은 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2015R1A2A2A01007764)

memory allocation strategy를 사용한다. 이는 프로그램 상에서 malloc함수가 존재하여도 그 즉시 메모리를 할당하는 것이 아니라, 실질적으로 스레드가 page fault를 발생시킬 때 메모리를 할당하고 있다.[5] 그래서 본 논문에서 PARSEC의 벤치마크[6]를 이용하여 실제 실험 결과 스레드가 수행되는 core에 해당하는 메모리 노드에서만 메모리를 할당하게 되어 프로그램이 수행되는 위치에서 메모리를 집중적으로 사용되고, 이는 지역 메모리를 사용하는 것으로 이어진다. 하지만 이는 스레드의 위치와 메모리 대역폭을 적절히 고려하지 못한 것으로 나타났다. 그리하여 벤치마크 특성에 따라 메모리 인터리빙 정책을 적용하여 성능 향상을 이루고자 하였다.

2. NUMA 시스템에서 default linux의 스케줄링과 메모리 할당 분석

NUMA 환경에서 효율적인 스케줄링 기법과 메모리 할당 정책을 연구하기 위해 기존 리눅스의 동작을 분석하고자 하였다. 실험 환경은 IBM System x3650 M2, 이며 NUMA시스템에 사용된 프로세서는 Intel Xeon 5500 processor이고, 총 2개의 노드(소켓)가 있고 각 노드에 4개의 코어가 존재한다. (8개의 코어를 0~7로 표시) 리눅스 커널 버전은 3.6.0에서 실험이 진행되었다. 벤치마크로는 PARSEC이 사용되었고 memory intensive 벤치마크인 streamclsuter와 memory non-intensive 벤치마크인 blackscholes를 이용한 실험을 진행하였다. 프로세스와 스레드가 수행되는 core 및 memory위치는 1초 주기로 샘플링 통해 측정하였다. 그림 2에서 리눅스의 first-touch memory allocation과 프로세스와 스레드의 메모리 공유하는 방식으로 두 벤치마크 모두 한쪽 노드 위주로 메모리를 할당하는 것을 확인할 수 있었다. 특이한 점은 일부 스레드는 원격 노드의 core에서 위치하여 수행되었다. 이는 특히 메모리 인텐시브한 벤치마크에서 불리하게 적용될 수 있음을 알 수 있다. 또한, memory intensive 하지 않은 blackscholes는 스레드의 core이동이 거의 발생하지 않음을 알 수 있었다. 이는 간단히 메모리 인텐시브하지 않다면 NUMA에서 스레드와 메모리 배치에 대한 중요성이 상대적으로 낮음을 알 수 있다. 반면 논문에 추가하지 않았지만 실제로 그림 2에 streamcluster외에 상대적으로 메모리 인텐시브한 벤치마크들은 core간의 문맥교환이 빈번하게 발생함을 알 수 있었다. 이러한 원인이 한쪽 노드 위주로 메모리 할당이 이뤄져 메모리 인텐시브 벤치마크는 메모리로 접근이 상대적으로 매우 빈번하게 발생하게 되어 충돌문제가 발생하기 때문에, 메모리 요청을 하는 중에 문맥교환이 매우 빈번하게 일어나게 되는 것으로 판단할 수 있었다.

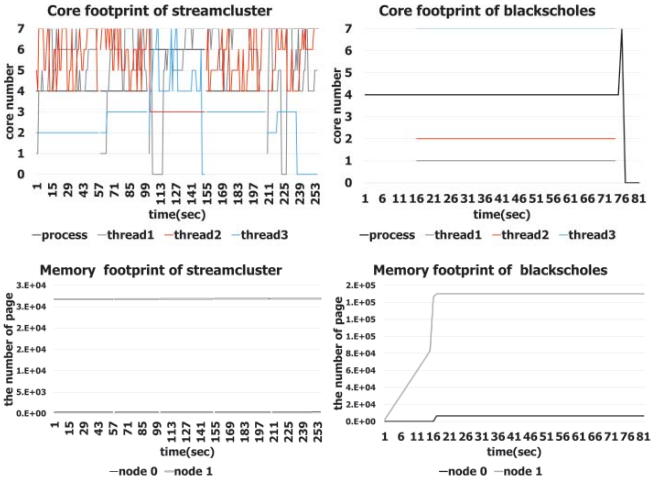

그림 2 Default 리눅스에서 프로세스와 스레드의 core 간 이동 및 메모리 노드 할당

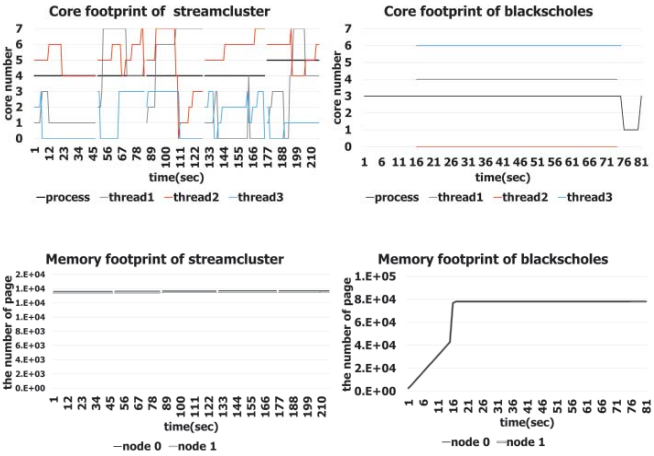

그림 3 메모리 인터리빙 정책 사용 시 모습

3. 메모리 인터리빙 적용

이러한 메모리 접근 특성을 갖는 벤치마크는 두 노드에서 동등하게 메모리를 할당하는 인터리브 정책을 사용했을 때 이득이 있을 것으로 가정하였다. 한쪽 메모리 노드의 대역폭이 부족하게 되는 것이 원격 메모리의 접근보다 더 성능에 악영향을 미칠 것으로 간주한 것이다. 실제 실험 결과는 그림 3에 나타내었다. 그림 2에 비해 streamcluster에서 core간의 문맥교환이 적게 나타났고 메모리가 할당이 두 노드에 분배되어 스레드들의 위치도 두 노드에 고루 퍼져 수행됨을 알 수 있었다. 실제로 메모리 인터리브 정책에서 약 20%의 성능 향상이 있었다. 하지만 memory non-intensive 벤치마크인 blackscholes는 default linux와 인터리브 간에 footprint는 큰 차이가 없었으며 성능도 약 2% 정도의 차이만 발생했다.

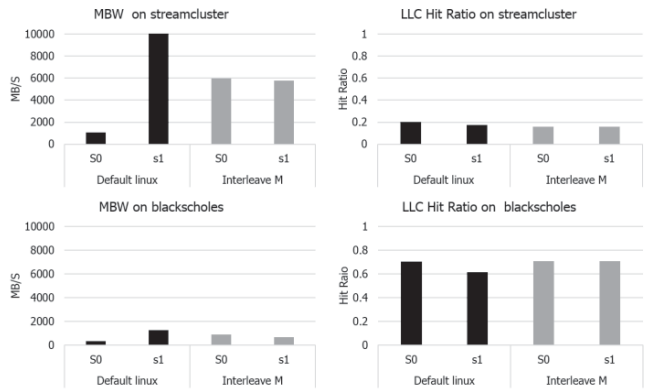

그림 4 메모리 대역폭과 LLC Hit Ratio 비교

4. 메모리 대역폭과 LLC Hit Ratio를 통한 분석

그림 4에는 default linux와 메모리 인터리브 정책에서 노드(소켓)간에 메모리 대역폭(MBW) 및 LLC Hit Ratio을 비교하였다. 앞선 실험에서 streamcluster는 node1의 메모리에서 많은 메모리를 할당한 만큼 default linux에서 노드1의 메모리 대역폭 압도적으로 크게 발생했다. 인터리브 정책에서 메모리 대역폭이 거의 동등하게 발생하는 것으로 나타났다. Memory intensive한 벤치마크의 특성상 LLC Hit Ratio는 두 경우 모두 낮았으며, 상대적인 차이가 크지 않았다. 이러한 결과로 인터리빙 정책은 원격 메모리로 접근이 많아 질 수 있지만 메모리 대역폭을 공평하게 사용할 수 있게 되어 20%성능 향상이 있음을 알 수 있었다. 반면 memory non-intensive 벤치마크는 마찬가지로 대역폭이 분배되긴 했지만 상대적으로 대역폭 자체가 매우 작았고, LLC Hit Ratio의 변화도 작았다. 메모리 대역폭과 지역 메모리 및 원격 메모리에 관한 이슈가 크지 않음을 알 수 있었다.

5. 결론 및 향후 연구

본 논문은 NUMA 시스템에서 default 리눅스의 first-touch memory allocation 방식과 CFS의 스케줄링 방식을 분석하기 위한 연구이다. 일반적으로 프로세스가 수행되는 노드의 지역 메모리에서 메모리 할당이 이뤄진다. NUMA 시스템의 특성상 지역메모리 위주로 메모리 할당하는 것이 성능 향상에 기본이 될 것이다. 하지만 본 논문에서는 한쪽 노드에 메모리 대역폭이 집중되게 되면 문제가 될 것으로 판단하였다. 그래서 두 메모리 노드를 공평하게 사용하고자 인터리빙 정책을 사용하였고, 그 결과 메모리 인텐시브 벤치마크에서 성능이 향상 되었다. 실제로 메모리 대역폭이 두 노드에서 동등하게 발생하는 것으로 나타났다. 하지만 인텐시브 하지 않은 벤치마크에서는 성능의 차이가 미비하게 나타났다. 따라서 NUMA 시스템의 특성상 메모리 인텐시브한 벤치마크가 성능에 크게 영향을 받음을 알 수 있었다. 논문에서 각각 한가지씩 벤치마크를 나타내었지만 여러 벤치마크에서 비슷한 경향을 보여주었다. 향후 연구 계획으로는 여러 개의 프로세스 및 스레드를 복합적으로 함께 수행할 때 default 리눅스가 갖는 특징을 분석하고 이를 위한 스케줄링 기법에 기법과 효율적인 메모리 할당에 관한 연구를 지속할 예정이다.

참 고 문 헌

[1] T. Brecht, “On the importance of parallel application placement in NUMA multiprocessors". In 4th Symp. Experiences with Distributed & Multiprocessor Syst., pp. 1{18, USENIX, Sep 1993.

[2] Z. Majo, et al “Memory system performance in a NUMA multicore multiprocessor.” In SYSTOR’11.

[3] B. Lepers, et al “Thread and Memory Placement on NUMA Systems: Asymmetry Matters. “ USENIX ATC 15.

[4] Hood, Robert, et al. "Performance impact of resource contention in multicore systems." Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on. IEEE, 2010.

[5] S. Kaestle, et al “shoal: smart allocation and replication of memory for parallel Programs” USENIX ATC 15.

[6] PARSEC Benchmark Suite. http://parsec.cs.princeton.edu/