매니코어 환경에서 자동 NUMA 밸런싱 효과 분석

An Analysis of Automatic NUMA Balancing Effect on Many-core CPUs

저자
(Authors)

경주현, 임성수
Joohyun Kyong, Sung-Soo Lim

출처
(Source)

한국정보과학회 학술발표논문집 , 2015.06, 1621-1623 (3 pages)

발행처
(Publisher)

한국정보과학회
KOREA INFORMATION SCIENCE SOCIETY

URL

http://www.dbpia.co.kr/Article/NODE06394496

APA Style

경주현, 임성수 (2015). 매니코어 환경에서 자동 NUMA 밸런싱 효과 분석. 한국정보과학회 학술발표논문집, 1621-1623.

매니코어 환경에서 자동 NUMA 밸런싱 효과 분석

경주현 ○ 임성수

국민대학교 컴퓨터공학부

An Analysis of Automatic NUMA Balancing Effect on Many-core CPUs

Joohyun Kyong ○ Sung-Soo Lim

School of Computer Science Kookmin University

요 약

본 논문은 매니코어 환경에서 최신 리눅스 커널의 자동 NUMA 밸런싱에 대해서 설명하였고, 그 효과에 대해서 분석하였다. 이를 위해 리눅스 커널 진영의 연구 동향에 대해서 분석하였으며, 8노드를 가지는 120코어 기반의 서버를 대상으로 자동 NUMA 밸런싱의 효과에 대해서 실험 및 분석하였다. 또한 NUMA 노드를 기반으로 사용자가 최적화된 파티션을 수행한 기법과 자동 NUMA 밸런싱과의 성능을 SPECjbb2013 벤치마크를 사용하여 비교 분석하였다. 이를 통해 앞으로 NUMA 노드 수가 증가되는 매니코어 환경에서의 자동 NUMA 밸런싱에 대한 연구의 필요성과 해결해야 할 과제를 제시하였다.

1. 서 론

최근 다수의 코어를 필요로 하는 매니코어 시스템에서는 하드웨어가 NUMA 구조로 시스템을 개발되고 있다. 이러한 NUMA 구조의 특징은 메모리 대역폭을 유지하여, 코어가 늘어남에 따라 발생하는 메모리 병목현상을 제거하는 장점을 가진다[1]. NUMA 구조의 단점은 다른 노드의 메모리에 접근할 경우, 더 많은 시간과 비용이 필요하다. 따라서 그에 적합한 메모리 관리와 스케줄링 기법이 필요하게 된다.

그 동안 리눅스 커널에서는 응용프로그램이 자신에게 가까운 NUMA 노드에 할당되어 메모리 할당과 수행되도록 커널 인터페이스를 제공하였다. 또한 이러한 커널 인터페이스를 이용하여, NUMA에 최적화된 방법을 사용하고 있었다. 따라서 그 동안 응용프로그램에서 NUMA API를 이용하여 최적화 하는 방법과 프로세스를 동적으로 설정하는 최적화된 데몬 프로그램을 만드는 방법이 가장 효율적인 방법 중 하나이다.

하지만 NUMA API를 사용하여 최적화하는 방법은 단일 응용프로그램만 동작하는 시스템에서 유용하다. 하지만 리눅스와 같이 다양한 응용프로그램이 동시에 실행되는 환경에서는 최적화된 결과를 얻기 힘들다. 따라서 커널 레벨에서 동적으로 NUMA에 최적화 할 수 있는 자동 NUMA 밸런싱(Automatic NUMA Balancing)[2]을 개발 되었다. 하지만 현재까지 자동 NUMA 밸런싱의 기능에 대한 설명과 효과에 대한 분석이 부족한 것이 문제이다.

* 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.B0101-15-0644, 매니코어 기반 초고성능 스케이러블 OS 기초연구)

본 논문은 이러한 문제점을 해결하기 위해, 자동 NUMA 밸런싱에 대한 리눅스 커널 내부의 개발 동향과 그 내부의 알고리즘에 대해서 소개를 하였다. 또한 8노드를 가지는 NUMA 기반의 서버를 대상으로 자동 NUMA 밸런싱의 효과를 분석하였고, 앞으로 연구의 필요성과 현재의 120코어를 가지는 8노드의 매니코어 환경에서 자동 NUMA 밸런싱의 문제점을 발견하였다.

본 논문의 구조는 다음과 같다. 2장에서 관련 연구에 대하여 설명을 하며 3장에서는 자동 NUMA 밸런싱의 개발 동향과 내부 구조에 대해서 설명한다. 4장에서는 자동 NUMA 밸런싱의 효과와 문제점에 대해서 설명한다. 마지막으로 결론 및 향후 연구에 대해서 설명한다.

2. 관련 연구

관련 연구는 리눅스 커널의 NUMA 알고리즘을 수정하는 연구[3][4]와 기존 알고리즘을 NUMA에 최적화 시키는 연구[5] 그리고 NUMA를 위한 툴에 관현 연구[6]가 있다. 리눅스 커널의 NUMA 알고리즘을 수정하는 연구는 다음과 같다. 과도한 로컬 노드에 할당이 메모리 병목 현상을 발생시켜 성능에 오버헤드가 발생하는 문제를 여러 노드로 분산시켜 해결한 방법[3]이 있다. 다음으로 라지 페이지를 사용함에 발생하는 문제점을 동적으로 라지 페이지를 사용하도록 수정하여 성능을 개선한 연구[4]가 있다. 또한 기존 알고리즘을 NUMA에 최적화 시키는 연구는 그래프 알고리즘을 NUMA 시스템에 최적화 시키는 방법 등[5]이 있으며, 마지막으로 툴에 관련한 연구는 쓰레드간의 메모리

할당 및 흐름을 분석하여 응용프로그램의 정보를 알려주는 연구[6]가 있다.

3. 자동 NUMA 밸런싱

기존 NUMA 시스템의 메모리 할당은 정해진 메모리 영역에 할당되면, 해당 NUMA 노드에서 고정되어 프로그램이 동작하는 방법을 이용해왔다. 또한, 필요에 따라 사용자가 직접 관여하여, 프로그램의 NUMA 노드를 설정하는 방법을 사용하였다. 하지만 코어 수가 점점 증가됨과 서버에 동작중인 프로세스의 복잡도가 증가됨에 따라, 사용자가 직접 NUMA 노드를 설정하는 것은 힘들게 되었다. 이를 위해서 그동안 리눅스 커널에서는 레드헷에서 개발한 AutoNUMA[7]와 향상된 NUMA 스케줄링 기법[8]과 자동 NUMA 밸런싱 기술이 개발되어 왔다. 이중 자동 NUMA 밸런싱은 그림 1과 같이 리눅스 커널 3.8 버전부터 메인 라인 리눅스에 적용되었다. 이처럼 앞으로 리눅스 커널 진영에서의 NUMA에 대한 연구는 자동 NUMA 밸런싱에 기능을 확장하는 방향으로 진행되고 있다.

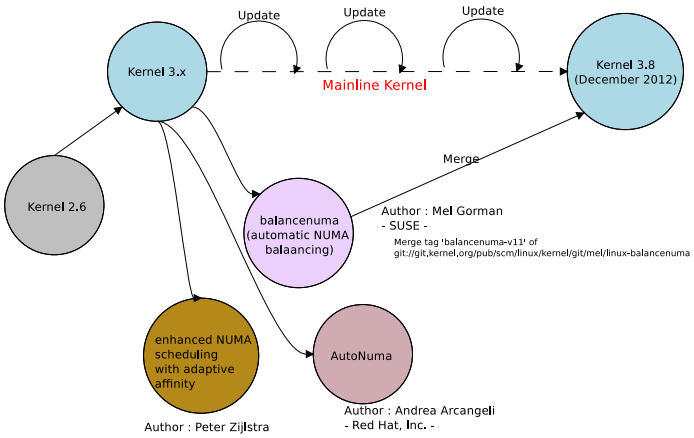

그림 1 리눅스 커널 NUMA 개발 동향

자동 NUMA 밸런싱의 내부 기능을 설명하면 그림 2와 같다. 먼저 힌팅 페이지 폴트 (hinting page faults) 기술은 주기적으로 각 태스크의 일정 부분의 메모리 공간을 언맵(unmapped)으로 설정하여, 리눅스의 페이지 폴트가 발생할 때, NUMA 지역성(Locality)을 측정 하는 기술이다. 이렇게 저장된 NUMA 지역성에 대한 정보는 통계를 위한 폴트 통계(Fault statistics) 기술을 수행 하여, 차후 페이지 이주(page migration)에 사용된다. 다음으로, 페이지 이주 기술은 기존 리눅스에 존재하는 태스크 이주(task migration)의 방법을 메모리로 확장한 개념이다. 태스크 이주 방법보다 오버헤드는 심하나, 태스크가 심하게 다른 NUMA 노드에 할당되었을 때 유용하게 이용된다. 태스크 그룹핑과 배치(Task grouping & placement)는 의존성 있는 태스크를 그룹핑 시킴으로써, 원격 메모리 접근 횟수를 줄이기 위한 방법이다. 마지막으로, 의사 간삽법(Pseudo-interleaving) 기술은 NUMA 노드의 메모리 대역폭(bandwidth)을 고려하여, 한쪽 노드에 쏠리지 않고 분산 시키는 방법을 말한다. 이외에도 최근에는 NUMA 노드의 거리에 비례한 알고리즘을 개발하고 있다.

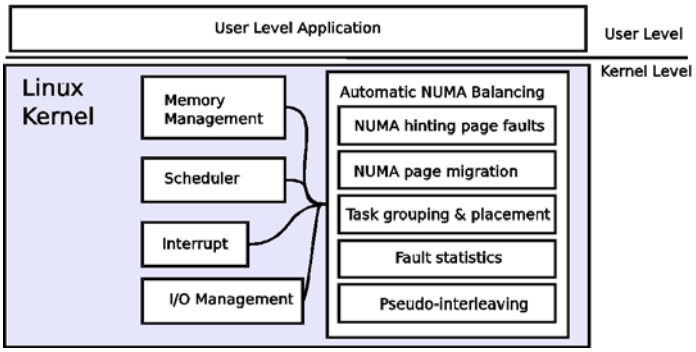

그림 2 자동 NUMA 밸런싱 구조

4. 매니코어 환경에서 자동 NUMA 밸런싱 효과 실험

4.1 실험 환경

실험 환경은 표 1과 같다. 모든 실험의 벤치마크는 SPECjbb2013[9]을 사용하였다. 모든 실험은 물리적인 코어만 실험하기 위해 인텔 하이퍼스레딩 기능을 제거하고, 리눅스 커널의 주파수를 최대로 설정하여 실험을 수행하였다.

표 1 실험 환경

CPU	Intel Xeon E7-4800/8800 v2
코어 수	120코어
메모리	755GB
NUMA 노드 수	8 노드
운영체제	리눅스 커널 3.19.0-rc4

4.2 실험 결과

4.2.1 멀티 그룹

멀티 그룹 실험은 SPECjbb2013 벤치마크의 JVM 수를 노드 수와 동일하게 설정하였고, 메모리 용량을 시스템 최대 가용 용량인 한 노드당 85GB로 설정하여 실험한 방법이다. 그림 3의 가로축은 코어 수를 의미하며, 세로축은 성능을 나타내는 jOPS를 의미한다. 실험은 다음과 같이 3가지 설정을 하여 수행하였다. 자동 NUMA 밸런싱 기능을 사용한 방법(multi-auto)과 사용하지 않은 방법(multi-base) 그리고 그룹을 노드에 최적화한 방법(multi-pin)을 사용하여 실험하였다.

대부분 응용프로그램에서 최적화된 파티션으로 설정한 방법이 가장 좋은 성능을 보였고, 다음으로 자동 NUMA 밸런싱이 좋은 성능을 나타낸다. 마지막으로 자동 NUMA

밸런싱을 사용하지 않으면 최악의 성능을 보였다.

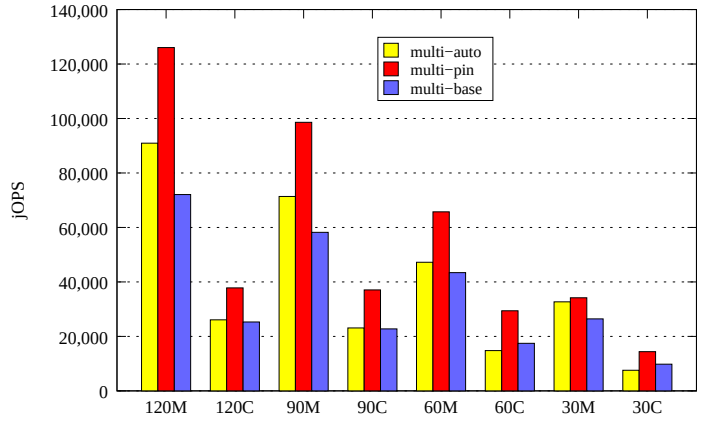

그림 3 멀티 그룹 실험 결과

4.2.2 싱글 그룹

다음 실험으로는 SPECjbb2013의 벤치마크 설정을 그룹별로 하지 않고, 싱글 그룹으로 설정하여 메모리 사이즈를 SPECjbb2013의 디폴트로 설정하여 실험을 하였다. 이는 실제 산업에 적용되어 사용되는 자바 응용프로그램이 싱글 JVM 위에 동작하는 경우도 많고, 자바 응용프로그램이 시스템의 가용한 모든 메모리를 사용하는 경우는 드물다. 따라서 디폴트 메모리 사이즈인 24GB도 충분히 큰 메모리 사이즈라 판단하여 해당 실험을 하였다. 비교는 그룹이 1개이기 때문에 NUMA 파티션을 최적화하여 설정하는 방법은 수행하지 않고, 자동 NUMA 밸런싱과 설정하지 않는 방법만 비교하여 수행하였다. 그림 4는 이러한 설정의 실험 결과를 보여준다.

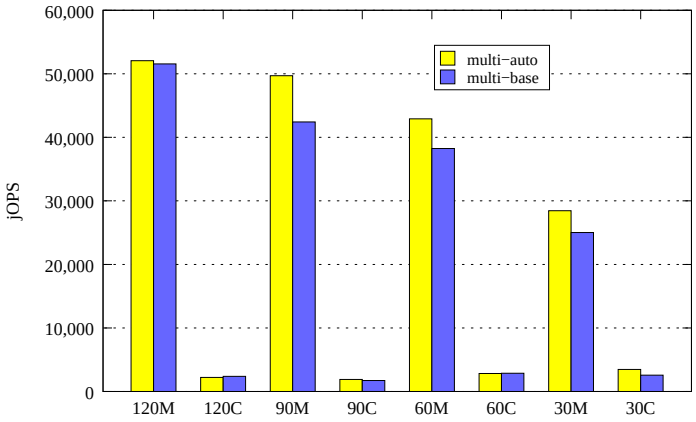

그림 4 싱글 그룹 실험 결과

실험 결과 멀티 그룹과는 다르게 120코어에서 상대적으로 자동 NUMA 밸런싱의 효과가 없다는 것을 볼 수 있다. 이러한 결과는 싱글 그룹과 적은 메모리 상황에서는 자동 NUMA 밸런싱이 문제가 있다는 것을 보여준다. 따라서 본 실험을 통해 나온 결과는 향후 해결해야 할 과제로 제시한다. 또 다른 문제로는 코어 수가 증가 함에도 상대적으로 확장성이 떨어진다.

5. 결론 및 향후 연구

본 논문은 리눅스 커널의 자동 NUMA 밸런싱에 대해서 소개하였고, 8 소켓의 NUMA 노드를 가지는 120코어를 가진 매니코어 시스템을 대상으로 리눅스의 자동 NUMA 밸런싱에 대한 효과를 분석하였다. 또한 적은 메모리를 사용하는 싱글 그룹에서의 자동 NUMA 밸런싱의 문제점을 제시하였다. 향후 연구로 발견된 적은 메모리를 가지는 싱글 그룹 환경에서 최적화된 자동 NUMA 밸런싱을 연구하고자 한다.

6. 참고 문헌

[1] A. Kleen. A NUMA API for LINUX. Technical Report Novell-4621437, Novell, April 2005.

[2] https://lwn.net/Articles/523065/

[3] M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quema, and M. Roth, “Traffic management: A holistic approach to memory placement on numa systems,” in Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). ACM, 2013, pp. 381-394.

[4] Gaud, Fabien, et al. "Large pages may be harmful on numa systems." USENIX ATC. 2014.

[5] Lachaize, Renaud, Baptiste Lepers, and Vivien Quéma. "MemProf: A Memory Profiler for NUMA Multicore Systems." USENIX Annual Technical Conference. 2012.

[6] Zhang, Kaiyuan, Rong Chen, and Haibo Chen. "NUMA-Aware Graph-Structured Analytics." Proc. PPoPP. 2015.

[7] https://lwn.net/Articles/488709/

[8] https://lwn.net/Articles/524535/

[9] SPECjbb2013, https://www.spec.org/jbb2013