매니코어 환경에서 로그 기반 동시적 업데이트 기법을 활용한 리눅스 커널 확장성 개선

경주현 ○ 윤성민 임성수

국민대학교 컴퓨터공학부

Joohyun0115@gmail.com, sungmin7465@gmail.com, sslim@kookmin.ac.kr

A Concurrent Update Method for Many-Core Linux Scalability

Joohyun Kyong○ Sungmin Yun Sung-Soo Lim

School of Computer Science Kookmin University

요 약

본 논문은 매니코어 환경에서 리눅스 가상 메모리 관리의 확장성에 대한 문제점을 해결 하였고, 이를 위해 로그 기반의 동시적 업데이트 기법을 개발하였다. 본 연구를 통해 개발 된 기법을 리눅스 운영체제의 확장성 문제를 야기 하는 두 가지 역 매핑에 적용하였으며, 120코어를 가진 매니코어 시스템을 대상으로 실험을 하였다. 실험 결과 120코어 상에서 멀티 프로세스 기반의 벤치마크에서 1.5배의 성능 향상을 보았다. 본 기법은 앞으로 매니코어 환경에서 리눅스 메모리 관련 확장성을 향상시키기 위해 유용하게 사용될 수 있다.

1. 서 론

최근 서버의 코어 수가 증가되고 있다. 따라서 서버에서 사용되는 CPU들이 멀티코어 시스템에서 매니코어 환경으로 변화되고 있다. 이와 같이 매니코어 환경에서 많이 사용되는 범용 운영체제 중 하나인 리눅스의 확장성에 대한 연구 필요성이 증가되고 있다. 하지만 매니코어 환경에서 리눅스 운영체제에 대해서는 확장성에 대해 개선해야 부분이 존재한다[1][2]. 그 중 리눅스 메모리 관리에 대해서 해결해야 할 사항들이 있다.

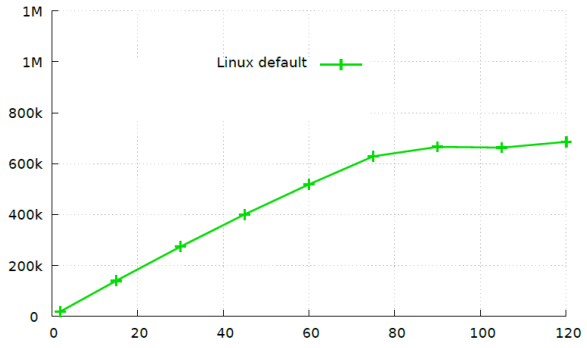

그림 1 AIM7 확장성

그림1는 리눅스 환경에서 확장성 문제의 한 예를 보여준다. 60코어 까지는 성능에 대한 확장성이 있으나 60코어 이후에는 성능 증가 폭이 줄어들었으며, 그 이후 성능이 증가가 없는 문제점이 있다[3]. 여기서 확장성의 원인은 리눅스 커널의 2가지 업데이트 락 때문에 발생하는 문제이다. 결국 업데이트 명령어는 락 때문에 한가지 쓰레드만 진입이 가능하므로 결국 확장성의 근본적인 이유는 업데이트 직렬화에 있다. 이처럼 업데이트 직렬화 문제를 해결하기 위해 본 논문은 새로운 로그기반 동시적 업데이트 기법을 개발하여 리눅스 커널의 역매핑 부분에 적용하였다. 결국 이를 통해 리눅스 커널의 확장성을 개선하였다. 실험 결과 AIM7벤치마크를 대상으로 120코어에서 1.5배의 성능 향상을 보았다. 본 논문의 구조는2장에서는 연구 동향에 대해서 설명하며, 3장에서는 로그 기반 알고리즘에 대해서 설명한다. 4장에서는 리눅스 커널에 적용한 내용에 대해서 설명하며, 5장에서는 실험 결과를 보여주며, 마지막으로 5장에서는 결론 및 향후 연구에 대해서 설명한다.

2. 연구 동향

높은 업데이트 비율 때문에 발생하는 상황의 업데이트 직렬화에 문제에 대한 해결 방법들은 존재한다. 해결 방법은 Non-blocking 알고리즘을 사용하는 방법[4]과 로그기반 알고리즘[5]을 사용하는 방법이 있다. Non-blocking 알고리즘들은 하드웨어 동기화 원자(synchronized atomic) 연산들을 활용하여 동시적 업데이트를 수행하는 방법이다. 하지만 이러한 방법은 공유 메모리를 CAS(Compare And Swap)로 접근하여 병목 현상이 생긴다. 그 이유는 캐시 메모리 커뮤니케이션 오버헤드이다. 최근에는 이러한 공유메모리 시스템에서 캐시 통신 때문에 발생하는 문제를 해결하기 위해 Per-core에 로그를 저장하는 방법이[5] 연구되고 있다.

* 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No. B010-16-0644, 매니코어 기반 초고성능 스케일러블 OS 기초연구)

3. 로그 기반의 동시적 업데이트 기법

업데이트가 많은 자료구제에 대해서 로그기반 방법은 총4가지의 장점을 가진다. 첫째로, 업데이트가 수행하는 시점 즉 로그를 저장하는 순간에는 락이 필요가 없다. 따라서 업데이트를 동시적으로 수행할 수 있을 뿐만 아니라, 락 자체가 가지고 있는 캐시 커뮤니케이션 오버헤드를 줄일 수 있다. 둘째로, 저장된 업데이트를 순차적으로 락과 함께 하나의 코어에서 수행 하 때문에, 캐시 효율성이 높아진다. 셋째로, 큰 수정 없이 기존 여러 데이터(tree, queue) 자료구조에 쉽게 적용할 수 있는 장점이 있다. 마지막으로 저장된 로그를 실제 수행하지 않고, 여러 가지 최적화 방법을 사용하여 적은 명령어로 업데이트를 수행할 수 있다. 우리의 방법도 이러한 로그 기반 방법의 장점을 가진다. 그러므로 앞에서 설명한 로그 기반 방법의 장점을 모두 가짐과 동시에 추가적으로 업데이트 순간 삭제 가능한 로그를 지움으로 성능을 향상시킨다.

우리가 제안하는 방법은 타임스탬프를 사용하지 않고, 개별적인 오브젝트를 대상으로 스왑(swap) 명령어를 사용하여 공유되어 있는 로그를 삭제하는 방법을 사용하였다. 이를 위해, 우리는 모든 오브젝트를 대상으로 삽입(insert)와 삭제(remove)에 대한 마크 필드를 추가해서 업데이트 순간 로그를 지우는 작업을 수행 하였다. 예를 들어 만약 특정 오브젝트를 대상으로 삽입-삭제 명령어 순서가 수행될 경우 처음 삽입 명령어는 삽입 마크 필드에 표시하고 큐에 저장한다. 다음 삭제 명령어부터는 로그를 큐에 저장하지 않고 상태 플래그인 삽입에 표시한 마크 필드에 표시한 값만 원자적으로 지워주는 방식으로 로그를 삭제하였다. 다음으로 우리는 로그를 적용할 때, 큐 안에 로그가 존재하더라도, 마크 필드를 확인하여 표시된 로그만 원래 자료 구조로 적용하는 일을 수행한다. 이것은 스왑이라는 상대적으로 가벼운 연산과 상대적으로 덜 공유하는 개별 오브젝트 안에 있는 마크 필드를 사용해서 명령들을 제거할 뿐만 아니라, 동시에 실제 명령을 수행하지 않고 로그를 지워주는 효과를 가져주므로 성능이 향상된다.

4. 리눅스 커널에 적용

우리는 새롭게 개발한 로그 기반의 알고리즘을 리눅스 커널에서 공유 데이터 때문에 문제가 발생하는 익명(Anonymous) 역 매핑과 파일 역 매핑 코드에 구현하였다. 구현은 업데이트 명령어에 대해서 로그를 저장하는 함수로 대체하고, 락을 제거하였다. 그리고 읽는 명령어 전에 저장된 로그를 반영하는 방법으로 구현하였다. 본 연구에서 개발한 방법을 리눅스 커널 4.5-rc6에 적용하였으며, 리눅스 테스트 프로젝트(LTP)까지 통과하였다.

5. 성능 평가

표1 실험 환경

CPU	Intel Xeon E7-4800/8800 v2
코어 수	120코어
메모리	755GB
NUMA 노드 수	8소켓(소켓당 15코어)
운영체제	리눅스 커널4.5.0-rc6

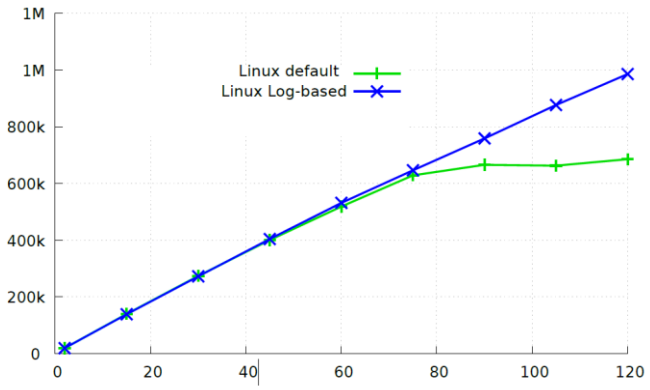

그림 2 AIM7 확장성 - 로그 기반 적용

그림 2는 로그 기반 알고리즘을 사용해서 개선된 성능을 보여준다. 75코어 까지는 두 버전의 리눅스 커널은 비슷한 성능을 보이나. 그 이후에는 디폴트 리눅스 커널 보다 1.5배의 성능 향상을 보인다. 그 이유는 업데이트 명령들을 병렬로 수행되었기 때문이다.

5. 결론 및 향후 연구

본 논문은 120코어 시스템을 대상으로 매니코어를 위한 리눅스의 확장성에 대한 문제점 중 업데이트 직렬화 때문에 발생하는 문제점을 리눅스 커널에 로그 기반 동시적 업데이트 기법을 사용하여 해결하였다. 본 연구의 결과는 아직 리눅스 커널 중 2가지 역매핑에 적용하였다. 차후 업데이트가 많이 발생하는 다른 데이터 구조에 추가적으로 적용할 예정이다.

6. 참고문헌

[1] Kleen, Andi. “Linux multi-core scalability.” Proceedings of Linux Kongress. 2009.

[2] Boyd-Wickizer, Silas, et al. “An Analysis of Linux Scalability to Many Cores.” OSDI. Vol. 10. No. 13. 2010.

[3] 정진환, 김강호, 김진미, 정성인. “Manycore 운영체제 동향.” 전자통신 동향 분석 29 권 5 호, 2014.

[4] T.Harris, "A pragmatic implementation of non-blocking linked-lists", in Proceedings of the 15th International Conference on Distributed Computing, ser. DISC '01, London, UK, UK, 2001, pp. 300—314.

[5] Boyd-Wickizer, Silas, "Optimizing communications bottlenecks in multiprocessor operating systems kernels,"in PhD thesis, Massachusetts Institute ofTechnology, 2013.