리눅스 커널 블록 계층의 매니코어 확장성 분석

김성곤 주용수 임성수 임은진

국민대학교 소프트웨어융합대학

sunggonkim@kookmin.ac.kr ysjoo@kookmin.ac.kr sslim@kookmin.ac.kr ejim@kookmin.ac.kr

Analysis of Linux Kernel Block Layer Scalability on Manycore Server

Sung-Gon Kim Yongsoo Joo Sung-Soo Lim Eun-Jin Im

School of Computer Science Kookmin University

요 약

리눅스 커널의 블록 계층은 단일 CPU와 HDD 저장 장치에 최적화된 구조를 유지해 왔다. 하지만 최근 들어 플래시 기반의 SSD 저장 장치가 HDD를 대체하고 있으며, 또한 NVMe와 같은 새로운 저장 장치 인터페이스도 도입되고 있다. 리눅스 커널 블록 계층에서도 이러한 변화에 대응하여 기존의 단일 큐 구조를 대체하는 다중 큐 구조를 도입하였으나 매니코어 시스템에서의 입출력 성능 확장성에 대한 분석 및 최적화는 충분히 이루어지지 않았다. 본 논문에서는 매니코어 시스템에서 하드웨어 큐 개수가 소프트웨어 큐에 비해 충분하지 않을 경우 및 블록 계층의 다중 큐 내부에 할당되는 I/O request 개수가 충분하지 않을 경우에 매니코어 확장성이 저하되는 문제를 확인하고, 하드웨어 큐 개수 및 I/O request 개수와 매니코어 확장성 사이의 상관관계에 대한 분석을 수행한다.

1. 서 론

HDD를 보조 기억 장치로 주로 사용하는 기존 환경에서는 HDD의 회전 장치의 특성을 고려하는 구조로 리눅스 커널 블록 계층이 설계되었다. 회전 장치는 물리적 특성에 따라 임의 접근과 순차 접근에 따라 속도 차이가 발생한다. 이러한 접근 방식을 고려하여 블록 계층에서는 디스크에 접근하는 대기 시간에 병렬성을 높이기 위해 주 기억 장치에서 플러깅, 정렬, 병합 등의 기술을 활용하였다. 하지만 보조 기억 장치가 SSD로 발달함에 따라 디스크 내부 데이터 병렬 처리를 통하여 접근 및 저장 시간이 비교적 짧아졌고 임의 접근과 순차 접근에 대한 속도 차이가 줄어들었으며 전자 신호 처리 방식을 이용하면서 회전 장치의 특성을 고려할 필요가 사라졌다.

또한 CPU에서는 쿨러의 발열 문제 처리의 한계로 인해 CPU 클럭을 높이는 scale up에서 CPU의 코어 수를 늘리는 scale out방식으로 발달하였다. 이에 따라 여러 개의 코어가 제한된 자원을 이용하면서 생기는 lock에 대해 리눅스 커널 블록 계층에서 충분한 고려가 이루어지지 않았다.

블록 계층은 이러한 비효율성 문제에 대응하기 위하여 기존의 단일 큐 대신 다중 큐를 도입하게 되었다[1]. 다중 큐는 소프트웨어 큐와 하드웨어 큐의 2단계로 구성되며 소프트웨어 큐 개수를 CPU 코어 수에 맞게 할당함으로써 다중 코어와 블록 계층 제한된 자원인 큐 간 경합을 제거하였다. 또한 기존의 단일 큐에서는 HDD에서 seek time을 줄이기 위한 I/O 정렬 기법이 사용되었는데, SSD에서는 그 효과가 크지 않은 반면 I/O 정렬 기법 수행을 위해 큐에 lock을 사용함으로써 다른 I/O가 처리되지 못하는 문제를 야기하였다. 다중 큐에서는 소프트웨어 큐에서만 설정에 따라 I/O 정렬을 지원하고 하드웨어 큐에서는 이를 배제함으로써 lock으로 인한 성능 지연을 개선하였다. 하드웨어 큐의 개수는 제한적인 디바이스의 I/O 큐 개수에 맞게 생성된다.이에 따라 한 개 이상의 소프트웨어 큐와 연결되며 디바이스의 submission rate를 조절해 디바이스 버퍼 과잉을 방지하였다.

하지만 이러한 블록 계층 다중 큐와 매니코어 확장성에 대해서는 충분한 검토가 이루어지지 않았다. 코어 수와 동일하게 만들어지는 소프트웨어 큐 개수는 매니코어 확장성에 영향을 끼치지 않으나 디바이스의 I/O 큐 개수와 동일하게 만들어지는 하드웨어 큐 개수는 한정 자원 제약 상황이 발생한다. 이러한 하드웨어 큐 개수가 소프트웨어 큐 개수 에 비해 일정 개수 이하로 떨어질 때 확장성 저하 문제가 발생한다. 또한 블록 계층과 디바이스 드라이버에서 처리하는 데이터 단위인 request의 최대 개수에 따라서도 매니코어의 확장성에 문제가 생긴다.

따라서 본 논문에서는 다중 큐의 하드웨어 큐 개수와 블록 계층 및 디바이스 드라이버에서 데이터를 처리하는 request의 최대 개수를 통하여 매니코어 확장성의 상관관계에 대해 분석한다.

2. 블록 계층 다중 큐

2-1. 블록 계층 I/O 경로

* 이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(No.2014-3-00035, 매니코어 기반 초고성능 스케일러블 OS 기초연구)

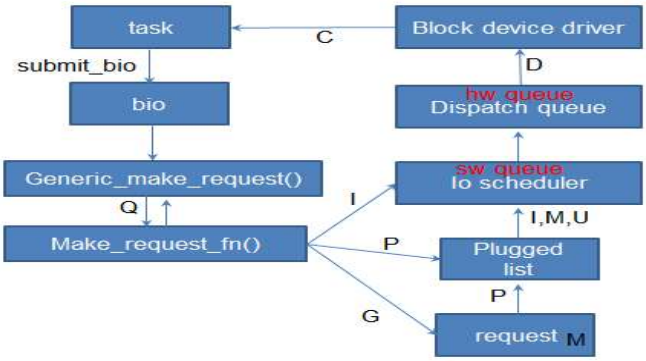

그림 2. Block I/O Path

본 연구에서는 블록 계층을 추적하기 위한 도구로 blktrace[2]를 사용하는데 그림1에서 블록 계층의 각 구간을 나타내는 플래그에 대한 처리 과정을 시각화하였다. Q, G, P, I, M, U, D, C는 각각 queue, get_request, plug, insert, merge, unplug, dispatch, complete를 나타내며 블록 I/O의 시작은 task로부터 시작된다. 이 task는 리눅스에서 프로세스와 쓰레드에서 사용하는 구조체의 기본 단위이다. task에서 submit_bio()라는 함수를 통해 블록 계층에 I/O를 전송한다. 블록 계층에서는 bio 구조체를 할당하고 generic_make_request() 함수를 호출한다. 이후 callback 함수로 디바이스 드라이버에서 bio를 request로 만들기 위해 make_request_fn()을 호출한다. request를 할당하여 bio를 변환시키고 이후 제한된 상황에서만 플러깅, 병합, 정렬, 등을 거쳐 소프트웨어 큐, 하드웨어 큐를 거쳐 다시 디바이스 드라이버에서 I/O를 완료하고 task에 알리는 구조의 반복이다.

2-2. Waitqueue와 nr_requests

디바이스 드라이버에서 설정하는 하드웨어 큐 깊이를 통해 해당 디바이스가 이용하는 블록 계층의 nr_requests 개수를 초기화한다. 이 nr_requests는 블록 계층에서 다중 큐가 사용하는 request 개수로 블록 계층의 디스크 혼잡도 threshold로 작용한다. 따라서 I/O가 발생하여 make_request_fn 함수가 불렸을 때 nr_requests의 개수를 초과하는 경우에는 waitqueue에 task가 잠들게 된다. 이후 request가 처리되고 나면 다시 잠든 task를 깨운다. 이를 통해 nr_requests는 블록 계층의 매니코어 확장성에 중요한 요소라는 것을 추정할 수 있다. 따라서 본 연구에서는 nr_requests를 변경하면서 블록 계층 다중 큐의 매니코어 확장성을 분석하였다.

3. 실험 결과

본 실험은 NUMA(Non Uniform Memory Access) 구조를 이용하는 다중 코어 시스템에서 매니코어 확장성을 확인하기 위해 실제 I/O를 발생시키지 않고 가장 빠른 입출력을 할 수 있는 null_blk 드라이버를 이용한다. 또 block I/O의 성능 측정을 위해 FIO Benchmark[3]를 이용하여 매니코어 시스템에서 nr_requests에 대해 분석하며 실험 환경은 표 1과 같다.

null_blk 드라이버에서 nr_requests의 기본 설정값이 64개로 할당되어 있다. HDD나 SSD 같은 경우에는 nr_requests의 기본 설정값이 보통 128개 또는 256개로 되어 있으며 nvme의 경우에는 1024개로 되어 있다.

Server : IBM x3950 x6	
OS	Ubuntu 16.04
CPU	Intel Xeon E7 8870 v2 (2.30GHz)
CPU Socket	8
Core per CPU	15
L1 Cache	32KB per core
L2 Cache	256KB per core
L3 Cache(LLC)	30MB per socket
Memory	768GB

표 1. IBM x3950 x6 사양

소프트웨어 큐 개수는 코어 수와 동일하게 120개로 생성된다. 하지만 하드웨어 큐 개수는 디바이스의 I/O 큐 개수와 동일하게 생성되나 실제 I/O가 발생하지 않고 물리적 디바이스가 연결되지 않는 null_blk 디바이스에서는 임의로 하드웨어 큐 개수를 조절할 수 있다. 그림 3은 null_blk에서 nr_requests가 기본 설정값인 64개로

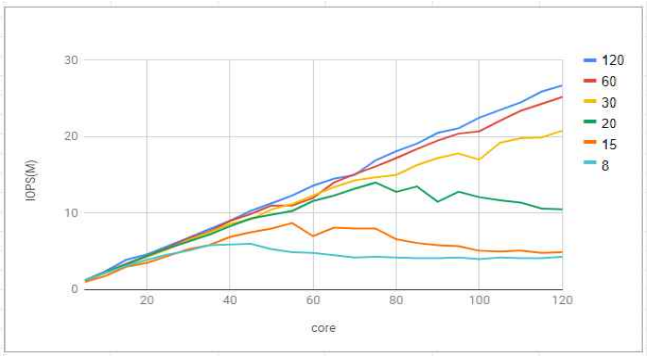

그림 3. 하드웨어 큐 개수와 코어 수에 따른
FIO Benchmark I/O 처리량(nr_requests = 64)

실험을 한 그래프이다. 가로축은 core 수를 나타내며 세로축은 IOPS(M)를 나타낸다. 또 각 그래프 선은 하드웨어 큐의 개수를 나타낸다. 하드웨어 큐가 30개 까지는 확장성을 보이나 20개일 때부터 확장성이 120 코어까지 유지되지 않으며 코어 수가 증가함에 따라 심지어 성능 저하 현상을 보인다.

하드웨어 큐 개수가 부족하여 블록 계층의 매니코어 확장성이 저하될 때의 실험 결과를 blktrace에서 측정한 후 btt(blktrace timeline)로 각 코어별 데이터를 flag의 구간 별로 합친 결과를 다음 그림 4에서 볼 수 있다.

```
==================== All Devices ====================

            ALL           MIN           AVG           MAX           N

Q2Q               0.000000001   0.000000185   0.000840532      312127
Q2G               0.000000379   0.000002068   0.013889409    37456920
D2C               0.000000156   0.000000635   0.016018396      312103
Q2C               0.000001271   0.000003512   0.016020349      312103

==================== Device Overhead ====================

       DEV |       Q2G       G2I       Q2M       I2D       D2C
---------- | --------- --------- --------- --------- ---------
 (251,111) | 7068.0984%   0.0000%   0.0000%   0.0000%  18.0704%
---------- | --------- --------- --------- --------- ---------
   Overall | 7068.0984%   0.0000%   0.0000%   0.0000%  18.0704%
```

그림 4. btt로 확인하는 구간별 오버헤드

그림 3에서 Q2G의 구간에 가장 큰 오버헤드가 발생함을 확인할 수 있다. G 플래그는 get_request()함수가 불리는 구간으로 실제 request가 할당되는 부분을 의미한다. nr_requests 변수에 의해 request 개수가 제한되고 이 개수를 넘어가면 waitqueue에 프로세스가 잠들기 때문에 오버헤드가 발생한다.

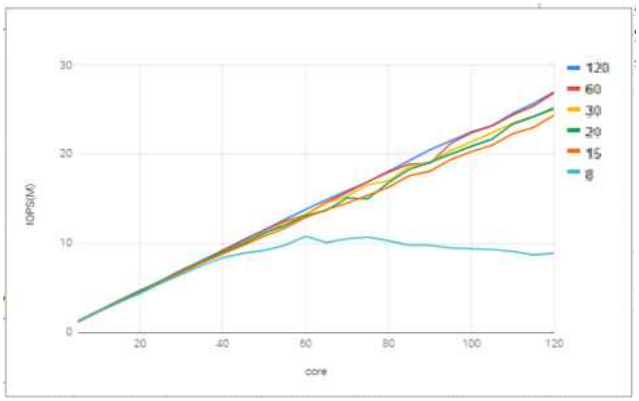

그림 5. 하드웨어 큐 개수와 코어 수에 따른
FIO Benchmark I/O 처리량(nr_requests = 64)

그림 5는 그림 2와 동일한 조건에서 nr_requests를 1024개로 늘렸을 때 확장성을 유지하는 모습이다. 하드웨어 큐 개수가 15개일 때까지 확장성이 유지되며 하드웨어 큐 개수가 8개일 때 확장성이 사라진다.

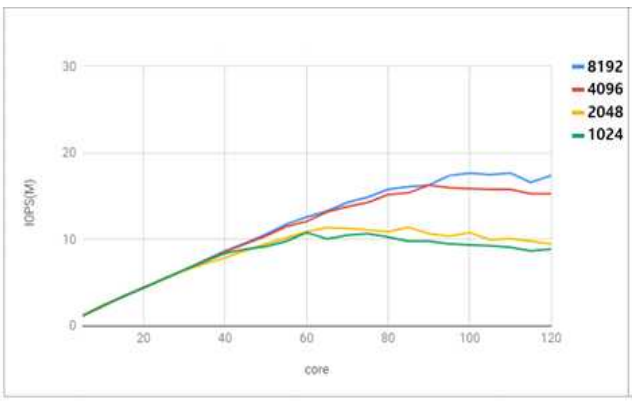

그림 6. nr_requests와 코어 수에 따른
FIO Benchmark I/O 처리량(하드웨어 큐 8개)

그림 6은 하드웨어 큐 개수가 8개일 때 nr_requests를 증가시켜도 확장성이 유지되지 않는 모습을 나타낸 그래프이다. nr_requests가 8192개 또는 4096개일 때가 2048개 또는 1024개일 때 보다 비교적 확장성이 좋으며 약 80코어까지 확장성이 보장된다. nr_requests의 최대값은 커널 설정 상 10240개가 최대이며 커널에서 개수 제한을 풀어 65536개까지 늘려 보았으나 확장성이 유지되지 않았다.

4. 결론 및 향후 연구방향

블록 계층 다중 큐의 request의 threshold 역할을 하는 nr_requests가 블록 계층의 확장성에 큰 영향을 끼치며 request가 해당 변수를 초과할 때 쓰레드는 잠들어 버린다. 대부분의 경우 nr_requests가 2048개 정도에서 최대 성능을 보이며 해당 값을 더 높여도 큰 변화가 없다. 이후 실험을 통하여 하드웨어 큐 개수가 충분히 많을 경우 nr_requests값이 낮을 경우에도 120코어까지의 확장성을 확인하였으며 120코어 이상의 머신에 대한 확장성에 대해 추가 분석이 필요하다. 또한 하드웨어 큐 개수가 적을 때 성능이 다소 불안정한 양상을 보이는데 이에 대한 추가 연구도 수행할 예정이다.

참 고 문 헌

[1]M. Bjorling, J. Axboe, D. Nellans, and P. Bonnet. Linux block IO: Introducing multi-queue SSD access on multi-core systems. In Proceedings of the 6th International Systems and Storage Conference, SYSTOR '13, pages 22:1-22:10, 2013
[2]Jens Axboe, Alan D. Brunelle, and Nathan Scott. blktrace(8) - Linux man page. http://linux.die.net/man/8/blktrace, 2006
[3]Jens Axboe, "fio - Flexible I/O Tester Synthetic Benchmark." http://freecode.com/projects/fio.