매니코어 환경에서 BTRFS의 단일 파일 읽기 성능 분석

Analyzing Single File Read Performance of BTRFS in Manycore Environment

요 약

SSD가 기존의 HDD를 대체하여 주요 저장장치로 자리 잡게 되면서 SSD에 최적화된 파일 시스템이 등장하고 있다. 이러한 신규 파일 시스템의 성능을 다각도로 평가하려는 노력이 이루어지고 있으며, 특히 매니코어 머신 상에서 F2FS의 확장성 분석 및 개선이 활발하게 진행 중이다. 하지만 BTRFS에 대해서는 상대적으로 충분한 검증이 이루어지지 않았으며, 따라서 본 논문에서는 BTRFS의 매니코어 읽기 성능 확장성에 대한 분석을 수행하였다. 실험 결과 BTRFS에서 단일 파일에 대한 임의 읽기를 수행할 때 심각한 확장성 저하가 발생함을 확인하였고, 그 원인으로 다수의 코어가 임의 읽기 요청시 잠금에 머무르는 시간이 길어져 확장성이 떨어지는 현상을 지목하였다.

1. 서 론

HDD를 대체하여 SSD가 주요 저장장치로 자리 잡게 되었다. 그리고 F2FS, BTRFS[1]와 같이 SSD에서 우수한 성능을 보이는 파일시스템이 등장하고 있다. 한편, 기존의 16개 이하의 코어로 이루어진 멀티코어 시스템을 넘어서 100개 이상의 코어로 이루어진 매니코어 시스템이 등장하였다.

따라서 이러한 신규 파일시스템의 성능을 멀티코어 시스템뿐만 아니라 매니코어 환경에서도 다각도로 평가하려는 노력이 이루어지고 있다. 특히 최근에는 수십 개 이상의 CPU 코어를 가진 매니코어 머신에서 F2FS 파일시스템의 성능 확장성을 분석 및 개선하는 연구가 활발히 진행되고 있다[2][3]. 하지만 BTRFS 파일시스템에 대해서는 매니코어 환경에서의 성능 확장성 분석이 충분히 진행되지 않았다.

본 논문에서는 매니코어 환경에서 BTRFS 파일시스템의 단일 파일 임의 읽기 성능을 집중적으로 분석하였다. 매니코어 머신에서 CPU 코어의 개수가 증가함에 따라 BTRFS의 단일 파일 임의 읽기 성능에 대한 확장성이 EXT4 및 F2FS와 비교했을 때 현저히 떨어지는 현상을 확인하였다. 이에 대한 원인을 분석한 결과, BTRFS에서 임의 읽기를 시도하는 코어의 숫자가 증가할수록 extent tree에 접근하여 동일한 extent bit를 참조하려는 요청이 많아진다는 것을 파악하였다. 하지만, 같은 extent bit에 대해 동시 참조는 허락되지 않으므로 lock이 사용되어 성능이 저하됨을 확인하였다.

2. 배경지식

2.1 EXT4

EXT4 파일 시스템은 효율적으로 디스크를 사용하기 위해 저장장치를 논리적인 블록의 집합, 즉 블록 그룹으로 구분한다. 일반적으로 블록의 크기는 4KB이고 시스템의 설정에 따라 달라질 수 있다. 파일 시스템을 구성하는 정보들은 각 블록 그룹에 분산했고, 파일을 저장할 때 특정 부분에 집중적으로 기록하여 하드 디스크의 효율을 높이고자 하는 특성이 있다[4].

2.2 F2FS

F2FS는 모바일 장치부터 서버에 이르는 컴퓨터 시스템에 다양하게 사용되는 NAND 플래시 메모리(SSD) 기반 저장 장치의 특성을 염두하여 개발된 플래시 파일 시스템이다. 기본 쓰기 단위는 4KB이고, 연속된 512개의 블록 집합을 기본 단위로 하는 segment를 받아 순차 쓰기를 한다. 기존에 존재하던 다른 로그 구조 파일 시스템과는 달리 메타 데이터들은 다른 데이터들과 구분하여 관리한다. 또한, F2FS는 wandering tree의 눈덩이 효과라든지 high cleaning 부하와 같은 로그 구조 파일 시스템의 일부 알려진 문제를 해결할 수 있다[1].

2.3 BTRFS

Buffer file system 또는 B-tree file system의 약자로 COW(Copy-On-Write) B-Tree를 데이터 구조로 사용하는 파일 시스템이다. BTRFS에는 B-Tree 기반의 여러 트리가 존재하는데, 파일 디렉토리를 저장하는 sub-volumes, extent를 추적하는 extent allocation tree, 할당된 extent 당 checksum을 관리하는 checksum tree가 대표적이다. 그중 extent allocation tree는 extent item을 관리하며, extent item이란 디스크 상의 연속적인 공간을 의미한다. extent는 스냅샷, 즉 오류 복구 기능을 위해 복제된 파일 데이터가 저장되는 공간이고 기존의 파일 시스템에서 사용되는 페이지 대비 용량이 크다[5].

3. BTRFS 성능 분석

3.1 실험 환경

본 논문에서는 매니코어 환경에서 BTRFS의 단일 파일 읽기 성능을 분석하기 위해 표 1과 같이 실험 환경을 구축하였다. 표 1의 환경에서 FIO 벤치마크[6]의 psync와 libaio 두 가지 I/O 엔진에 대해 임의 읽기 입출력 성능을 측정하였다. 모든 측정치는 같은 조건에서 10회 반복실험 후 중앙값을 취했다.

CPU	Intel Xeon E7 8870 v2 (2.30GHz) * 8
Memory	768GB
OS	Ubuntu 18.04
Kernel	Linux 5.8
NVMe	Samsung SSD 960 PRO 1TB

표 1: 실험 환경

# of jobs	KIOPS	locking ratio(%)
1	10.8	0
5	54.2	0
10	99.1	1.9
20	165	18.8
30	164	56.2
50	82.2	79.8
80	72.4	84.5
120	119	89.2

표 2: BTRFS에서 psync 요청시 job의 개수에 따른 lock의 점유율

3.2 실험 결과

그림 1은 FIO 벤치마크의 psync I/O 엔진, 그림 2는 FIO 벤치마크의 libaio I/O 엔진을 사용하여 job 개수와 파일 시스템에 따른 4k 임의 읽기 성능을 측정한 결과를 보여준다. EXT4와 F2FS의 경우 특정 시점에서 디바이스 최대 성능인 440KIOPS에 근접하는 성능을 보여준다. 즉, EXT4와 F2FS는 동시에 단일 파일 읽기 작업을 수행하는 코어의 숫자가 증가함에 따라 디바이스의 임의 읽기 최대 성능에 도달하기 때문에 충분한 확장성을 보여준다고 할 수 있다. 하지만 BTRFS의 경우에는 읽기 요청 작업의 흐름이 psync I/O 엔진에서는 20개, libaio I/O 엔진에서는 10개를 넘어서는 시점부터 성능 저하가 발생하는 것을 관찰할 수 있다. 이에 대한 원인을 분석하기 위해 리눅스 프로파일링 도구인 perf를 사용하였다.

3.3 실험 결과 분석

표 2는 BTRFS에서 임의 읽기 요청의 개수에 따른 FIO I/O 수행시간에 대한 lock 함수의 실행시간 점유율을 보여주고 있다. BTRFS에서는 특정 page를 읽는 읽기 요청이 왔을 때, 먼저 extent bit를 통해 extent가 할당되었는지 판단해야 한다. 이 상황에서는 두 가지 경우가 발생할 수 있다. 첫 번째로 이미 할당된 extent라면 해당 extent 내부의 page를 읽기 위해 lock이 발생한다. 두 번째는 아직 할당되지 않은 extent라면 extent를 새로 할당하여 extent bit를 부여하고 tree에 삽입하는데, 이때 다른 코어가 동일한 extent를 생성할 수 없도록 lock이 발생한다. 따라서 다수의 코어가 임의 읽기 요청을 진행하면 lock의 발생 확률이 증가하기 때문에 표 2와 같이 코어의 숫자가 많아질수록 lock 점유율이 늘어난다. 그 결과로

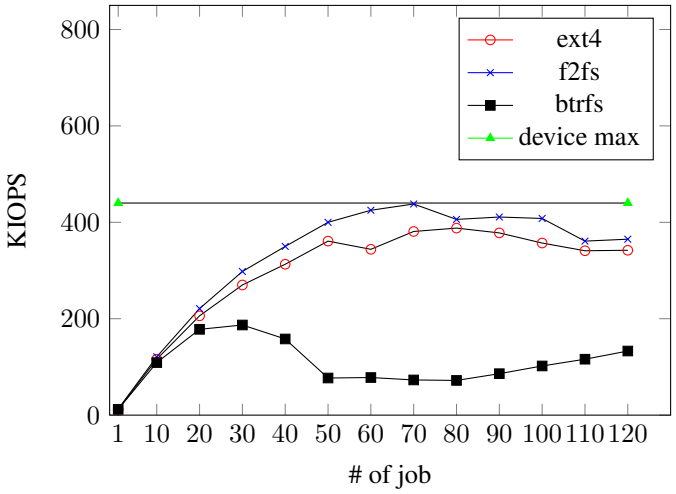

그림 1: Job 개수 증가에 따른 psync 임의 읽기 성능 변화

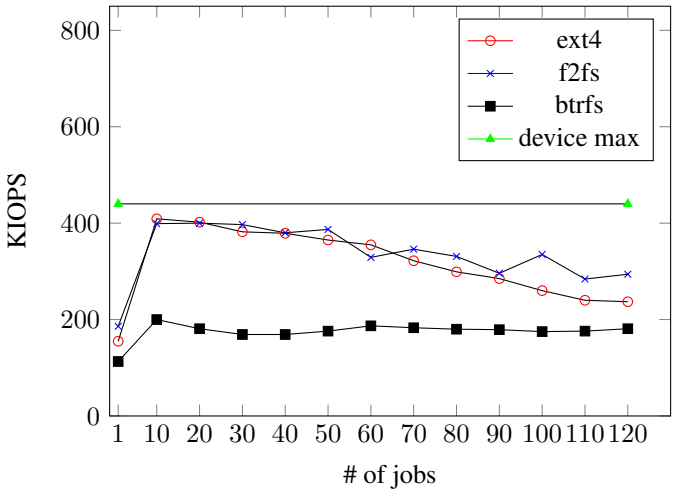

그림 2: Job 개수 증가에 따른 libaio 임의 읽기 성능 변화

그림 1과 같이 성능 저하가 관찰된다.

4. 결론 및 향후 연구

본 논문에서는 매니코어 환경에서 BTRFS의 단일 파일 읽기 성능을 EXT4와 F2FS와 비교하고 코어 수 증가에 따른 성능 저하를 관찰하였다. 또한 그 원인으로 BTRFS가 임의 읽기 요청을 할 때 extent에 접근하는 과정에서 사용하는 lock에서 성능 저하가 발생함을 확인하였다. 향후 연구로 BTRFS의 임의 읽기 작업시 extent에 접근할 때 발생하는 lock 문제를 개선하여 매니코어 확장성을 확보하고자 한다.

참고 문헌

[1] J.-Y. H. Changman Lee, Dongho Sim and S. Cho, “F2FS: A New File System for Flash Storage,” *13th USENIX Confer-*

ence on File and Storage Technologies (FAST'15), pp. 273–286, 2015.

[2] 서동주, 주용수, 임성수, 임은진, "매니코어 환경에서 F2FS 파일시스템의 단일 파일 읽기 성능 개선," *한국정보과학회 학술발표논문집*, pp. 1238–1240, 2020.

[3] 노성현, 이창규, 김영재, "매니코어 환경에서 F2FS 파일시스템의 단일 파일 I/O 병렬화," *한국정보과학회 학술발표논문집*, pp. 1073–1075, 2019.

[4] J. R. S. A. D. Aneesh Kumar K.V, Mingming Cao, "Ext4 block and inode allocator improvements," *Proceedings of the Linux Symoposium*, vol. 1, pp. 263–274, 2008.

[5] Ohad Rodeh, Josef Bacik, Chris Mason, "BTRFS: The Linux B-tree Filesystem," *ACM Transactions on Storage*, vol. 9, no. 3, 2013.

[6] J. Axboe, "Fio-flexible io tester," *URL http://freecode.com/projects/fio*, 2014.