가변적인 쓰기 패턴을 위한 적응형 SSD 캐시 기법

조정우 ○ 원유집

한양대학교 컴퓨터·소프트웨어학과

jojungwoo@hanyang.ac.kr, yjwon@hanyang.ac.kr

Adaptive SSD Cache Scheme for Variable Write Pattern

Jungwoo Jo ○ Youjip Won

Department of Computer and Software, Hanyang University

요 약

컴퓨터 하드웨어 기술의 고도화로 인한 컴퓨팅의 가속화가 빠르게 이뤄지고 있다. 컴퓨팅의 가속화로 인해 처리되는 데이터의 크기가 커지는 반면 서버 환경에서의 대용량 하드 디스크 처리 속도가 느려 병목 현상이 발생하는 문제가 있다. 해당 병목 현상을 줄이기 위해 최근 하드 디스크 보다는 빠르고 램보다는 용량 대비 저렴한 SSD를 캐시로 사용하는 SSD 캐시 기법을 사용하여 문제를 해결하고 있다. 하지만 서버 환경에서 멀티 프로세스들의 다양한 쓰기 패턴과 지속적인 쓰기 처리를 통해 하드 디스크보다 용량이 작은 SSD 캐시는 빠르게 채워지게 된다. 따라서 본 논문에서는 이러한 SSD 캐시가 빠르게 채워지는 SSD 캐시 full 현상을 지연시키기 위해 가변적인 쓰기 패턴을 식별하여 기록하는 적응형 SSD 캐시 기법을 제안한다.

1. 서 론

최근 하드웨어의 발전으로 기존 CPU만 사용하여 처리하던 작업들을 GPU를 활용하여 처리함으로써 컴퓨팅의 가속화가 진행되고 있다. GPU를 활용한 컴퓨팅의 가속화로 인해 데이터들의 크기도 커지고 있다. 컴퓨팅의 가속화와 빅 데이터들의 처리를 받쳐주기 위해 스토리지 역시 보다 빠른 처리가 필요하다. 일반적인 서버 환경에서의 하드 디스크의 속도는 이러한 처리를 받쳐주기 위한 성능이 부족하다. 따라서 최근 램보다는 저렴하고 하드 디스크 보다는 빠른 성능을 갖는 플래시 메모리 기반의 SSD를 활용한 SSD 캐시 기술이 나타나고 있다[1].

플래시 메모리 기반의 SSD를 활용한 SSD 캐시 기술은 하드 디스크로 데이터를 읽고 쓰는데 소요되는 시간을 개선하기 위해 고안된 기법으로 하드 디스크 보다 상대적으로 빠른 SSD를 디스크 캐시로 설정하여 램과 하드 디스크 간의 속도 차이를 개선한다. SSD 캐시 기술에서 하드 디스크는 가속화된 컴퓨팅의 처리를 위해 1차적 구조로 캐시 역할의 램과 2차적 구조로 캐시 역할을 하는 SSD를 구성하여 계층적인 구조를 사용한다. 계층적 구조를 통해 대용량의 하드 디스크는 더 빠르고 효율적으로 가속화된 컴퓨팅 처리가 가능하게 된다.

기존에 연구되고 있는 SSD 캐시의 대표적인 기술로써 bcache[2], dm-cache[3], flashcache[4] 등이 있다. 이러한 기존 SSD 캐시 연구들은 서버 환경에서 발생하는 지속적인 처리에 대한 캐시의 full 현상에 대해서는 대비하지 못하고 성능이 급격하게 떨어지는 것을 볼 수 있다. 캐시 full 현상에 대해서는 3장 구현 및 실험에서 더 자세히 설명한다.

본 논문에서는 SSD 캐시에 대한 지속적인 처리를 통해 발생하는 캐시 full 현상으로 인한 성능 저하를 해결하기 위해 비교적 빠른 처리가 가능한 순차적인 쓰기는 하드 디스크에서 처리하고 랜덤 쓰기는 SSD 캐시에서 처리하는 방식을 통해 캐시 full 현상을 지연할 수 있다. 본 논문에서는 앞서 설명한 방식으로 SSD와 HDD에 병렬적으로 처리하여 쓰기 성능을 향상시키고 캐시 full 현상을 지연하는 adaptive write 처리를 제안한다.

2. 배 경

해당 단락에서는 대표적인 SSD 캐시 기술에 대한 설명과 본 논문에서 사용한 SSD 캐시 구조를 설명한다.

2.1 SSD 캐시 관련 연구

기존 SSD 캐시 관련 연구 중 대표적인 기술은 bcache, dm-cache, flashcache가 존재한다. bcache는 커널 3.10 버전부터 추가된 리눅스 커널 블록 계층 캐시이다. bcache는 btree 캐시 구조를 사용하고 있다. dm-cache는 device mapper를 기반으로 구현된 SSD cache 기법이며, 커널 3.9 버전에 추가되었다. flashcache는 2010년 4월에 facebook에서 개발한 기법이며, 오픈 소스이다. flashcache는 set associative hash 방식으로 캐시가 구성된다.

앞서 소개한 SSD 캐시 기법들은 캐시 기록 방법으로 write-back, write-through, write-around 모드를 지원한다. write-back 모드는 I/O 요청이 SSD 캐시에 발생하면 SSD 캐싱 이후 하드 디스크에 기록하는 방법이다. write-through 모드는 데이터가 하드 디스크와 SSD 캐시에 모두 기록된다. write-around 모드는 SSD 캐시를 통과하여 하드 디스크에 데이터가 기록이 된다.

bcache와 dm-cache에서는 본 논문과 같이 순차적인 쓰기 동작에 대해서는 하드 디스크에 곧바로 기록하는 동작이 있다. 하지만 이러한 SSD 캐시 기법들을 비롯한 다른 기법들은 지속적인 쓰기 요청이 발생하면 캐시 full 현상이 발생하게 된다.

2.2 SSD 캐시 구성 및 동작

본 논문에서는 오픈 소스로 제공되고 있는 SSD 캐시

기술인 dm-writeboost를 기반으로 실험을 진행하였다. dm-writeboost는 512KB 크기의 데이터 처리 단위로 segment라는 구조를 사용한다. segment는 그림 1에서 나타낸 것과 같이 전체 512KB를 4K 크기의 블록으로 나눌 수 있다. segment의 앞 부분의 4K 블록은 실제 데이터 블록의 위치, 실제 데이터가 저장될 하드 디스크 sector 위치, 현재 segment의 id값 및 유효한 블록의 개수 등 실제 데이터에 정보를 저장하는 블록이다. 그리고 총 512KB에서 맨 앞부분 4K를 제외한 508K 블록들에 실제 4K 블록 단위로 데이터가 저장되어 있다.

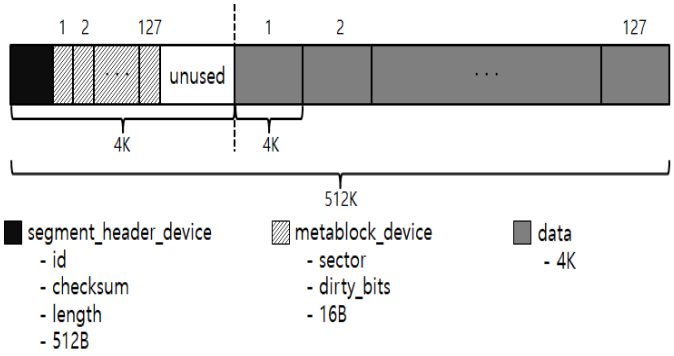

그림 1. Segment 구조

dm-writeboost는 크게 램과 SSD 두 개의 영역를 나눠 계층적으로 segment를 캐싱 하는 구조로 되어있다. 첫 번째 구조는 램에 위치하고 최대 4MB 크기의 rambuffer라는 영역을 갖는다. rambuffer는 내부는 segment가 순차적으로 버퍼링 된다. rambuffer에는 최대 segment 8개까지 버퍼링이 가능하다. 두 번째, SSD 캐시에 segment가 FIFO 방식으로 기록이 된다. 따라서 SSD 캐시에 데이터가 기록이 되고 나서, 가장 맨 처음 들어온 segment부터 하드 디스크에 순차적으로 기록이 된다[5][6].

3. 분석 및 구현

본 논문에서는 기존의 SSD 캐시 기법들에서 SSD 캐시와 하드 디스크에 동시에 기록할 수 없는 문제를 해결하기 위한 Adaptive Write 방식을 제안한다. 먼저 캐시 full 현상 발생 시 성능 저하 분석과 Adaptive Write 방식의 전체 동작 과정을 소개 한 뒤, 순차 쓰기와 랜덤 쓰기를 식별하기 위한 알고리즘을 소개한다.

3.1 캐시 Full 현상

본 논문에서는 캐시 full을 기존 SSD 캐시를 대상으로 연속적인 주기로 쓰기 요청을 하였을 경우 SSD 캐시가 가득 차 성능이 급격하게 저하 되는 현상으로 정의한다. 그림 2는 SSD를 50GB 파티션을 나눈 뒤, 워크로드 A를 dm-cache의 SSD캐시가 full 현상이 발생할 때까지 5초 주기로 실행한 결과이다. dm-cache의 성능이 최대 115K IOPS까지 발생하던 그래프가 SSD의 파티션 최대 크기 50GB에 근접할 경우 20K IOPS까지 급격히 감소하는 것을 볼 수 있다. 이는 dm-cache에서는 SSD 캐시에 기록을 하는 동시에 하드 디스크로 기록을 할 수 없기 때문이다. dm-cache는 SSD 캐시에 쓰기 요청을 모두 처리한 후 하드 디스크에 write-back 처리를 한다. 따라서 5초 주기로 지속적인 쓰기 요청이 발생한다면 그림 2와 같이 SSD 캐시로 설정한 SSD가 가득 차는 경우 성능이 급격히 저하된다. 다른 SSD 캐시 기법들에서도 이와 비슷한 문제를 발생한다.

본 논문에서는 이런 기법들과는 다르게 SSD에 기록하는 daemon과 하드 디스크에 write-back하는 daemon을 따로 두어 동시에 SSD와 하드 디스크에 처리가 가능하다. 따라서 그림 2에서 동시에 기록하기 때문에 dm-cache보다는 약간 성능이 낮게 나오지만 2배 이상 오래 성능을 유지하는 것을 볼 수 있다. 또한 dm-writeboost에서는 캐시 full 이후 SSD 캐시가 빠르게 복구되어 성능 복구가 다른 SSD 캐시 기법보다 빠르다. 따라서 본 논문은 dm-writeboost의 장점을 사용하여 adaptive write 기법을 구현하려 한다.

CPU	Intel(R) core™ i7-6700 CPU@3.40GHz
DRAM	8 GB
운영체제	CentOS 7
하드 디스크	1TB HDD
SSD 캐시	128GB SSD (Samsung 850 pro, 50GB partition)
벤치마크	FIO (3.3-18 version)
워크로드 A	Sequential write, Buffered IO, Block (record) size = 4KB, File size = 10GB, Number of thread = 1

표 1. 실험 환경

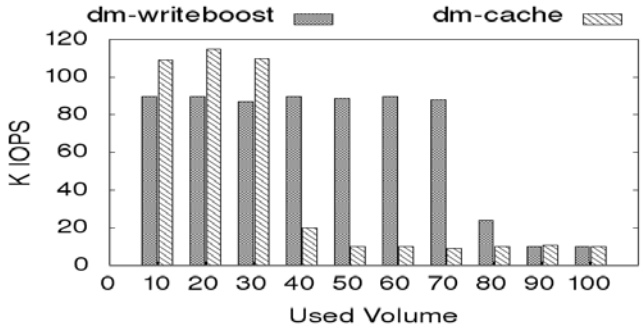

그림 2. dm-cache 캐시 full 현상.

3.2 Adaptive Write

본 논문에서는 dm-writeboost를 기반으로 하여 adaptive write 기법을 적용하였다. 기존의 dm-writeboost 캐시 기법은 모든 쓰기 요청을 램과 SSD 캐시를 거쳐 하드 디스크에 기록하기 때문에 캐시 full 현상이 발생한다. adaptive write에서는 SSD 캐시에 요청받은 데이터를 처리하기 전에 write pattern을 분석하여 순차적인 쓰기 패턴의 segment는 하드 디스크에 랜덤 쓰기 패턴의 segment는 SSD 캐시에 기록한다. 따라서 SSD 캐시와 하드 디스크에 각각 처리하여 캐시 full 현상을 지연시킬 수 있다.

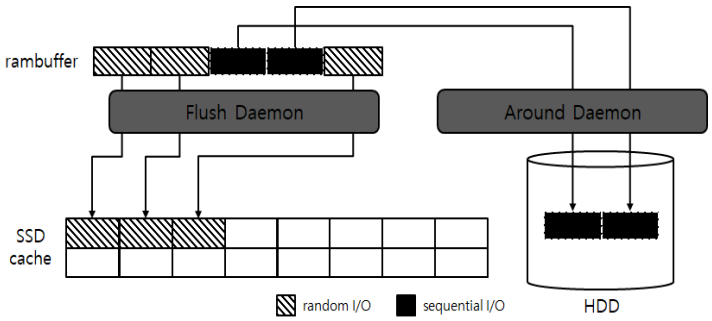

그림 2. Adaptive Write Work

adaptive write 처리 과정을 더 자세하게 본다면 그림 3과 같이 램에 존재하는 rambuffer 공간에 segment가 순차적으로 기록이 된다. 이후 일정 시간이 지나거나 rambuffer(4MB)가 가득 찰 경우 SSD 캐시 또는 하드 디스크에 기록을 하게 된다. 이때 rambuffer의 segment들에 대해 write pattern을 식별하는 동작을 한 뒤, 랜덤 쓰기 segment는 flush daemon에 의해 SSD 캐시에 기록하고 순차적인 쓰기 segment는 around daemon에 의해 하드 디스크에 기록한다. 따라서 쓰기 요청 받은 데이터 패턴에 따라 처리 방법을 병렬화하여 SSD 캐시를 더 유연하게 사용할 수 있다.

3.3 Write Pattern 비교 처리

본 논문에서는 순차적인 쓰기와 랜덤 쓰기를 판별하기 위해 요청 받은 데이터의 sector number를 기준으로 쓰기 패턴을 식별한다. 사용자에 의해 요청 받은 데이터는 램에 위치한 rambuffer 영역에서 SSD 캐시에 기록되기 전에 segment 단위를 기준으로 하여 쓰기 패턴이 식별된다.

```
function check_write_pattern()
  prev_sector ← segment의 첫 번째 Data sector;
  for i ← 1 to size step segment의 크기 do
    current_sector ← i 번째 Data sector;
    if (prev_sector == current_sector - 8)
      sequential_count++;
    prev_sector ← current_sector;
  end for
  if (threshold > sequential_count)
    return 랜덤 쓰기;
  else
    return 순차적인 쓰기;
end function
```

의사 코드 1. check_write_pattern()의 동작

현재 순차적인 쓰기와 랜덤 쓰기에 대한 식별은 segment를 구성하는 4KB 데이터의 sector number가 순차적으로 기록되었을 때마다 sequential-count 값을 증가시킨다. 순차적으로 기록된 것을 확인하는 내용은 의사 코드 1에서와 같이 이전 sector와 현재 sector를 비교하여 sector의 순서가 연속적으로 위치한다면 sequential-count를 증가시킨다. 따라서 sequential-count의 값이 순차적인 쓰기 패턴으로 식별하기 위한 값인 threshold 보다 클 경우 순차적인 쓰기로 식별하고 있다.

4. 결 론

본 논문에서는 SSD 캐시 기술에서 지속적인 처리 중에 발생하는 문제점인 캐시 full 현상에 대하여 분석하고 이에 대한 해결책인 adaptive write 기법에 대해 제안하였다. adaptive write 기법은 실시간으로 스토리지에 대한 요청이 가변적으로 발생할 때, 더 지속적이고 유연하게 SSD 캐시를 사용하기 위해 순차적인 쓰기 요청은 하드 디스크에 바로 처리하고 랜덤 쓰기 요청은 SSD 캐시에 처리한다. 따라서 순차적인 쓰기 요청을 하드 디스크에 처리하기 때문에 지속적인 쓰기에도 캐시 full 현상을 미뤄, SSD 캐시는 오랫동안 성능을 유지하고 빠른 복구한다.

추후 연구 과정으로는 여러 상황에 따른 실험을 통해 다른 SSD 캐시 기법들과 adaptive write 기법을 적용한 SSD 캐시의 실험 결과 비교 및 분석하고 adaptive write 기법 사용시에 쓰기 패턴을 더 확실하게 식별할 수 있는 threshold 값을 구한다.

참 고 문 헌

[1]Hu, Yiming, and Qing Yang. "DCD—disk caching disk: A new approach for boosting I/O performance." ACM SIGARCH Computer Architecture News. Vol. 24. No. 2. ACM, 1996.
[2]Bcache: http://bcache.evilpiepirate.org/
[3]Ahmad Ali, Charles Rose. “bcache and dm-cache: Linux Block Caching Choices in Stable Upstream Kernel.” A Dell Technical White paper, December, 2013.
[4]Flashcache: https://github.com/facebook/flashcache/
[5]Nightingale, Tycho, Yiming Hu, and Qing Yang. "The Design and Implementation of a DCD Device Driver for Unix." USENIX Annual Technical Conference, General Track. 1999.
[6]https://github.com/akiradeveloper/dm-writeboost