블록 탈피 접근법 기반의 대역폭 및 공간 효율적인 KV-SSD 기술 연구

서강대학교 ▌박준혁·김영재

1. 서 론

효율적인 스토리지 시스템을 설계하려면 호스트 메모리에서 스토리지로의 데이터 이동 비용을 줄이는 것이 중요하다. 그러나 기존의 Key-Value Store (KVS)인 RocksDB [1]나 LevelDB [2]는 파일 시스템 위에서 동작하여, 사용자 I/O 요청이 커널의 파일 시스템 및 블록 계층을 거쳐야만 한다. 이로 인해 메모리 복사와 커널 컨텍스트 전환에 따른 오버헤드가 발생하게 된다. 반면, KV-SSD [3-5]는 이러한 계층을 제거함으로써 낮은 지연 시간과 높은 처리량을 제공한다.

KV-SSD의 주요 이점 중 하나는 전통적 블록 기반 SSD와 달리, 사용자의 가변 크기 요청을 정확한 크기로 처리할 수 있다는 것이다. 이를 통해 저장 공간 활용도와 응답 시간을 개선할 수 있는 기회가 생기지만, 상업적으로 출시된 대부분의 KV-SSD [3-5]는 여전히 블록 기반 스토리지 장치를 위해 설계된 NVMe 프로토콜을 사용하고 있다. 이러한 프로토콜은 블록과 키-값 쌍 크기의 차이로 인해 I/O 증폭 문제를 야기한다.

또한, NAND 플래시 메모리는 블록보다 큰 페이지 단위 (보통 16KB)로 쓰기를 수행해야 하기 때문에, NAND 플래시 기반 드라이브에서는 이러한 I/O 증폭 문제가 내재되어 있다. 블록 기반 SSD는 디바이스 내 DRAM을 사용하여 NAND 페이지 버퍼를 구현함으로써 이를 효과적으로 완화할 수 있지만 [6], KV-SSD는 키-값 쌍의 크기가 블록 (보통 4KB)과 다르게 가변적이기 때문에 NAND 페이지 버퍼에서 바이트 단위의 오프셋을 관리해야 하는 추가적인 문제를 안고 있다.

기존 접근법의 한계로는, 첫째로 호스트 측에서 많은 키-값 항목을 배치하여 NAND 페이지 I/O 세맨틱에 맞추는 방식이 있다 [7, 8]. 하지만, 이는 전력 손실 시 데이터 손실 위험이 존재하고, 호스트에서 패킹된 여러 키-값 쌍을 보내는 경우, 이를 KV-SSD 내에서 개별적으로 인덱싱하고 관리하는 오버헤드가 발생한다. 둘째로, 디바이스 내에서 문제를 해결하려는 시도도 존재하는데, 예를 들어 KAML [9] 같은 경우 KV-SSD 내부에서 로그 구조를 형성하여 바이트 단위 오프셋 관리를 완화하려고 한다. 하지만 전송 단위와 키-값 크기의 불일치로 인해 여전히 I/O 증폭 문제는 남아 있다.

이러한 문제를 해결하기 위해 본 특집 원고에서는 BandSlim 기술을 제안한다. BandSlim 기술은 호스트와 KV-SSD 간의 작은 키-값 페이로드 전송에서 PCIe 트래픽 오버헤드를 최소화하고 데이터 전송 효율성을 개선한다. 또한, BandSlim은 NAND 페이지에 가능한 한 많은 작은 키-값 레코드를 밀도 있게 패킹하는 방식을 통해 NAND 페이지 쓰기를 최소화하는 설계를 도입한다. 이러한 설계는 NVMe 표준을 따르면서도, NVMe의 유틸리티를 그대로 유지할 수 있도록 호환 가능한 완전한 SW 수준의 방법론이다.

2. 배경지식

2.1 키-값 솔리드 스테이트 드라이브 (KV-SSD)

KV-SSD는 전통적 블록 단위의 I/O 트랜잭션을 키-값 단위로 바꾸어 스토리지 인터페이스를 혁신하였다. 장치 수준에서 Key-Value Store (KVS) 기능을 제공해 파일 시스템과 블록 계층을 제거함으로써 다층 공간 관리 오버헤드를 크게 줄였다 (그림 1).

최근 KV-SSD 연구에서 주로 사용되는 KVS 설계는 Log-Structured Merge-tree (LSM-tree)와 키-값 분리 방식을 따른다 [4,5,8]. LSM-tree는 RocksDB [1]나 LevelDB [2] 같은 호스트 측 KVS에서 많이 사용되며, 쓰기 집중형 워크로드에서 강점을 가진다. 그러나

* 본 연구는 정부 (과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (RS-2024-00453929, RS-2024-00416666).

LSM-tree의 컴팩션 작업이 쓰기 증폭 문제를 일으켜 성능 저하와 스토리지 수명에 영향을 미치기 때문에, 이를 해결하기 위해 키-값 분리 방식이 제안되었다. 이 방식에서는 값들을 LSM-tree에서 분리하여 Value Log (vLog)에 저장함으로써, 반복적인 값 재기록을 방지하고 쓰기 증폭을 줄인다.

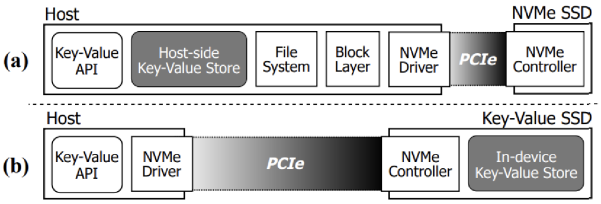

그림 1 (a) 블록 SSD 기반의 호스트 단 KVS와 (b) KV-SSD의 저장 소프트웨어 스택 비교

KV-SSD는 파일 시스템과 블록 계층을 우회할 수 있으며, 사용자 수준의 키-값 API와 NVMe 기반의 장치 드라이버, 그리고 저장소 내의 LSM-tree 기반 KVS로 구성된다. 사용자 API는 PUT, GET, SEEK, NEXT와 같은 요청을 처리하며, 키와 값은 블록 단위가 아닌 가변 길이로 다뤄진다. 키-값 쌍의 주소는 LSM-tree에 저장되며, 값은 vLog에 저장된다. vLog는 논리적 NAND 플래시 주소 공간으로, 이 공간은 여러 논리적 NAND 페이지로 나뉜다. LSM-tree의 엔트리는 vLog 내의 해당 값을 가리키게 된다.

2.2 NVMe 키-값 저장 장치 인터페이스

2.1.1 키-값 쌍 전송 매커니즘

NVMe 프로토콜은 KV-SSD를 위한 키-값 명령 세트를 도입하였다 [10]. NVMe 키-값 인터페이스에서, 키-값 쌍을 쓸 때 NVMe 드라이버는 키와 메타데이터를 NVMe 명령 구조체의 예약된 필드에 저장한다. 값은 PRP (Physical Region Page)를 통해 전송되며, 이는 NVMe의 블록 인터페이스와 동일한 방식이다 [11]. PRP는 호스트 메모리의 물리적 메모리 페이지 주소를 설명하는 연결 리스트로, 값이 저장된 하나 이상의 메모리 페이지가 전송 대상으로 지정된다.

그 후 드라이버는 NVMe 명령을 제출 큐에 삽입하고, 장치에 쓰기 요청을 알리기 위해 초인종을 울린다. NVMe 컨트롤러는 큐에서 명령을 가져와 해석한 후, PRP 리스트에서 지정된 페이지를 찾아 호스트 메모리에서 장치 메모리로 페이지를 복사하는 DMA (Direct Memory Access) 트랜잭션을 시작한다. 컨트롤러는 전달 받은 키를 LSM-tree의 메모리 구성 요소인 MemTable에 삽입하고, 값을 vLog에 기록한다.

2.1.2 NAND 페이지 버퍼 관리 기법

NVMe 블록 SSD와 마찬가지로, NVMe KV-SSD는 SSD의 배터리 또는 캐패시터로 백업된 DRAM 내에 NAND 페이지 버퍼를 포함하고 있다. 이 버퍼의 각 항목은 NAND 페이지를 위한 지속적인 쓰기 버퍼로 작동한다. vLog에 값을 삽입할 때, 각 값의 크기가 NAND 페이지보다 작으면 여러 값들을 하나의 버퍼 항목에 패킹할 수 있다. 버퍼 항목이 더 이상 값을 수용할 수 없게 되면, 해당 내용이 물리적 NAND 페이지에 기록되며, 이 페이지는 FTL (Flash Translation Layer)에 의해 vLog 영역의 현재 논리적 NAND 페이지에 매핑된다.

3. PCIe 트래픽 및 NAND 쓰기 증폭 문제

일반적인 KVS처럼, 키와 값의 크기는 가변적이며 메모리 페이지에 꼭 맞춰져 있지 않다. Meta에 따르면, RocksDB는 실제 환경에서 평균적으로 값의 크기가 100바이트에 미치지 않는 경우가 많다고 한다 [12]. 이는 4KB 메모리 페이지 크기보다 훨씬 작다. 결과적으로, KV-SSD는 이러한 가변 크기의 작은 값

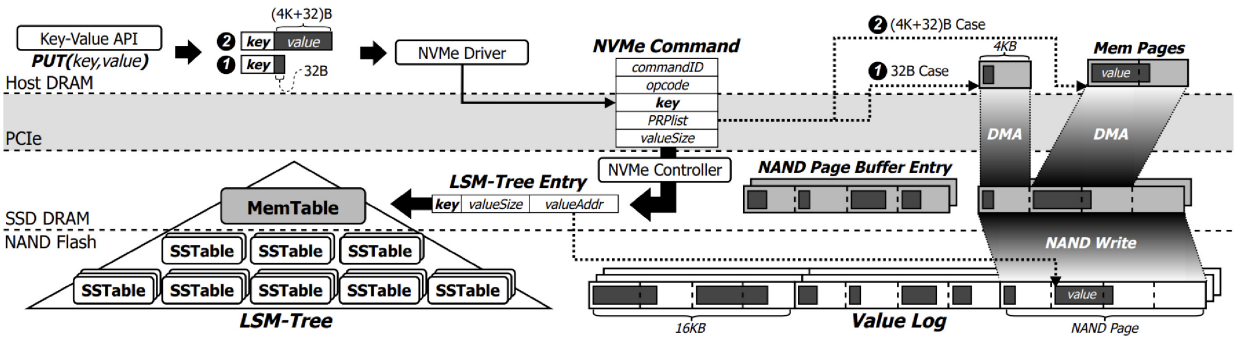

그림 2 LSM-tree 기반 KV-SSD에서 4KB 미만 크기의 페이로드와 4KB 이상 크기의 페이로드에 대한 키-값 쌍 쓰기 요청 처리 시 발생하는 PCIe 인터커넥트 트래픽 증폭 문제 및 NAND 쓰기 I/O 증폭 문제

을 효과적으로 처리할 수 있어야 한다. 그러나 현재와 같이 KV-SSD에 NVMe 표준을 키-값 인터페이스로서 단순히 적용하는 것은 값 전송과 NAND I/O에서 심각한 비효율성을 초래한다.

3.1 PCIe 트래픽 증폭 문제

이 문제는 NVMe 키-값 인터페이스가 원래의 블록 NVMe 프로토콜과 동일한 절차를 따르기 때문에 발생한다. 특히, NVMe의 데이터 전송 방식인 PRP는 DMA 전송을 메모리 페이지 크기인 4KB 단위로 제한한다. 이는 블록 스토리지 스택 진화 과정에서 메모리 페이지 단위와 일치하도록 발전해온 결과이다.

예를 들어, 그림 2의 묘사처럼, 값의 크기가 32바이트일 경우를 고려하면, PRP는 값을 임시로 저장할 4KB 메모리 페이지 하나를 지정하고, 4KB의 DMA 복사가 이루어진다. 반면, 값의 크기가 메모리 페이지 크기를 약간 넘는 경우 (예: 4KB+32바이트)에는 두 개의 메모리 페이지가 필요하며, PRP에 의해 8KB의 데이터가 전송된다. 이처럼 PCIe 트래픽이 불필요하게 커지면 시스템의 에너지 소비와 전력 사용이 크게 증가할 수 있으며, 이는 총 소유 비용을 높이는 요인이 된다. 이러한 이유로 현대 데이터 센터는 데이터 이동 비용 절감을 주요 목표로 삼고 있다 [13].

3.2 NAND 쓰기 I/O 증폭 문제

두 번째 문제는 NVMe SSD 내부에서 수신된 데이터를 NAND 페이지에 패킹할 때도 4KB 메모리 페이지 단위로 이루어진다는 것이다. NVMe SSD의 NAND 페이지 버퍼는 4KB 경계에 맞춰 데이터를 패킹하며, NAND 페이지 크기가 16KB인 경우, 호스트에서 전송된 데이터는 최대 4개의 쓰기 요청 (예: 32바이트 값 크기)을 하나의 NAND 페이지 버퍼 항목에 채울 수 있다. 하지만 값의 크기가 4KB보다 조금 큰 경우 (예: 4KB + 32바이트), 단 두 개의 쓰기 요청만으로 하나의 버퍼 항목을 채울 수 있으며, 그 이상은 채울 수 없게 된다 (그림 2). 이러한 장치 내부의 페이지 단위 패킹은 KV-SSD와 충돌하여 심각한 NAND 쓰기 증폭 문제를 일으킨다.

4. 블록 탈피 접근법 탐구

NVMe 프로토콜은 현재 두 가지 데이터 전송 방식을 제공한다: 전술한 PRP와 Scatter-Gather List (SGL) [14]이다. PRP는 호스트 메모리의 페이로드를 물리적 페이지 리스트로 설명하지만, 바이트 단위의 키-값 쌍을 처리하는 데 한계가 있다. 또한, NVMe 스토리지 스택은 페이로드가 여러 블록으로 구성된다는 가정하에 블록 크기 전송에 최적화되어 있다. 반면, SGL은 다양한 크기의 DMA 전송을 지원하지만, 32KB 이하의 I/O에서는 SGL을 사용하는 비용이 이점을 초과하는 것으로 보고되었다 [14]. 그래서 리눅스 커널은 SGL을 통한 데이터 전송의 최소 임계값을 32KB로 설정하고 있다 [15].

이러한 상황을 고려하여, 우리는 NVMe 명령을 이용해 기존 블록 기반 페이로드 전송 메커니즘에서 벗어나 키-값 전송을 최적화하고자 하였다. 실제 KVS 워크로드에서 값의 크기는 보통 100바이트를 넘지 않으며, 종종 64바이트 이하이다 [12]. 이를 기반으로 64바이트 크기의 NVMe 명령이 실제 워크로드에서 작은 값의 전송을 상당 부분 처리할 수 있다는 점에 주목하였다. 즉, NVMe 명령을 활용하여 세밀하고 대역폭 효율적인 키-값 쌍 전송이 가능하다는 것이다.

또한, 가변 크기 값을 처리할 때 NAND 쓰기 I/O를 증폭시키는 블록 기반 NAND 페이지 버퍼 관리에서 벗어나기 위해, KAML [9]은 여러 값을 배치하여 로그 방식으로 NAND 페이지에 저장하는 방법을 제안하였다. 하지만, 블록 크기보다 작은 값을 효율적으로 패킹하는 설계는 LSM-tree KV-SSD에 적용하기에는 충분히 상세하지 않았다. 게다가, 일부 DMA 엔진은 전송 크기와 대상 주소가 페이지 정렬을 필요로 하며 [16], 이는 드라이버 설계에 영향을 미친다 [17]. 이를 해결하기 위해, 우리는 큰 값은 기존 방법으로 전송하고, 작은 값은 그 사이에 배치하는 전략을 고안했다.

5. 제안 기술: BandSlim

그림 3 (a)는 본 원고의 제안 기술인 BandSlim의 소프트웨어 아키텍처를 보여준다. 검은색으로 표시된 모듈들이 BandSlim의 주요 구성 요소이다. 첫 번째는 BandSlim Key-Value 드라이버로, 이 구성 요소는 값을 NVMe 명령에 실어 전송하며, 요청된 값 크기에 가까운 PCIe 트래픽을 달성한다. 두 번째는 BandSlim Key-Value 컨트롤러로, 이 구성 요소는 수신된 값을 NAND 페이지 버퍼 항목에 세밀하게 패킹한다. PRP 기반 DMA로 전송된 상대적으로 큰 값을 패킹할 때 발생하는 메모리 복사 오버헤드를 해결하기 위해 선택적 패킹과 백필링 정책을 특징으로 한다. 마지막으로 세밀한 값 주소 지정이 가능한 LSM-tree이다. 키-값 분리 방식의 LSM-tree 스토리지 엔진으로, vLog에 세밀한 값 주소 지정을 지원한다.

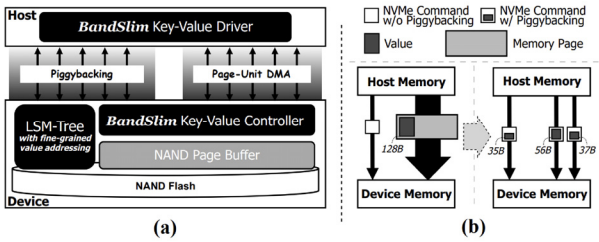

그림 3 (a) BandSlim의 소프트웨어 아키텍쳐, (b) NVMe 명령상 값 페이로드 피기백 (Piggyback) 기법

5.1 NVMe 기반 세밀한 키–값 전송 기법

우리는 NVMe 명령을 통해 세밀한 값 전송을 구현하기 위해, NVMe 명령 구조를 검토하였다. 그림 4처럼, 기존 PRP 기반 전송 방식이 사용되지 않는 경우, dword4-9 (24바이트)와 예약된 dword12-13 (8바이트), dword11의 2바이트와 1바이트를 활용해 최대 35바이트를 값 전송에 사용할 수 있다. BandSlim은 이 필드를 이용해 값을 실시간으로 전송한다.

값이 한 번의 NVMe 명령으로 전송되지 않을 경우를 위해 BandSlim은 Opcode를 분리해 (i) 쓰기 명령과 (ii) 전송 명령을 나누었다. 첫 번째 명령은 키와 메타데이터를 포함하며, 최대 35바이트의 값을 전송한다. 이후 남은 값은 전송 명령을 통해 56바이트씩 추가 전송된다. 예를 들어, 128바이트의 값을 전송하려면 3개의 NVMe 명령이 필요하며, 이는 기존 4KB 전송 방식에 비해 약 78.4%의 트래픽을 줄일 수 있다.

그러나 값의 크기가 커지면, NVMe 명령을 반복 생성하고 처리하는 데 시간이 오래 걸릴 수 있다. 예를 들어, 값의 크기가 (4K+32)바이트인 경우, 혼합 방식이 더 효율적일 수 있다. 첫 4KB는 DMA를 통해 전송하고, 남은 32바이트는 전송 명령으로 처리하는 것이다. BandSlim은 이러한 이슈를 해결하기 위해 값의 크기에 따라 NVMe 명령 기반 전송, PRP 기반 전송, 그리고 혼합 전송 중 가장 적합한 방식을 선택하는 임계값 기반 반응형 방식을 사용한다.

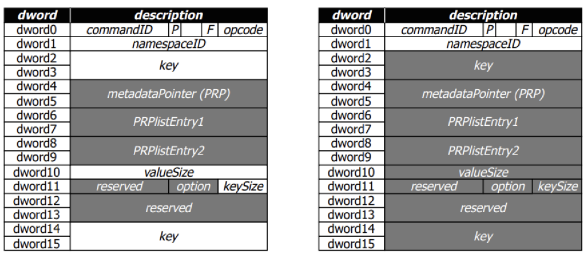

그림 4 BandSlim에서 새로 제안하는 값 페이로드 피기백 향 NVMe 키-값 쓰기 및 전송 명령어

이 결정 과정은 벤치마크 테스트를 통해 사용자 설정에 맞게 조정되며, 두 가지 주요 임계값이 있다: (i) $\alpha\times$ threshold_1은 피기백 전송이 비효율적이 되는 값을, (ii) $\beta\times$ threshold_2는 PRP 기반 전송이 혼합 전송보다 우수해지는 값을 나타낸다. 이 임계값은 사용자 선호에 따라 조정되며, BandSlim은 이를 통해 소형 값부터 대형 값까지 효율적으로 처리할 수 있다.

5.2 KV-SSD 장치 내부 세밀한 값 패킹 기법

5.2.1 All Packing 기법과 그 한계점

KV-SSD의 NAND 페이지 버퍼 관리를 위한 효율적이고 세밀한 패킹 정책을 설계하기 위해, 우리는 KAML [9]에서 언급된 간단한 버퍼링 접근 방식을 검토하였고 이를 “All Packing” 기법이라 부르기로 했다. 우리는 Write Pointer (WP)를 도입하여 NAND 페이지 버퍼 내에서 현재 쓰기 오프셋을 추적하며 KAML의 All Packing 기법을 구현하였다.

NVMe 명령 피기백을 통해 전송된 값의 경우, 컨트롤러는 명령을 가져와 피기백 필드에서 값을 추출한 후, 이 값을 WP가 가리키는 주소에 메모리 복사하고 WP를 업데이트한다. 추가 전송 명령도 같은 방식으로 처리되며, WP를 순차적으로 업데이트하여 버퍼에 값을 패킹한다. 호스트 측에서는 해당 값에 대한 쓰기 명령이 제출된 큐에 전송 명령을 추가하여, 피기백된 값 조각들이 FIFO 순서로 처리되도록 한다.

페이지 단위 DMA 또는 혼합 방식을 통해 전송된 값의 경우, DMA 대상 주소의 페이지 정렬 제한으로 인해 메모리 복사가 필요하다. 컨트롤러가 페이지 단위 DMA를 실행할 때, 대상 주소는 WP 뒤의 가장 가까운 페이지 정렬된 주소로 설정되어야 한다. WP와 대상 주소가 일치할 경우 메모리 복사를 생략할 수 있지만, 그렇지 않으면 피기백된 값과 동일하게 메모리 복사가 수행된다.

5.2.2 선택적 Packing 기법

All Packing 기법은 NAND 페이지 쓰기를 최소화하는 것을 목표로 한다. 하지만 페이지 단위 DMA를 통해 전송된 값은 적응형 방법 때문에 평균 크기를 초과하는 경우가 많아, 과도한 메모리 복사가 오버헤드를 초래할 수 있다. 저장 장치의 자원 제약을 고려하면, 큰 메모리 복사는 운영 속도를 크게 저하시켜 다른 요청을 지연시킬 수 있다.

이 문제를 해결하기 위해 “선택적 Packing” 기법을 제안한다. 이 정책은 페이지 단위 DMA로 전송된 값의 경우, 패킹을 수행하지 않고 WP를 값 끝부분으로

업데이트하는 방식이다. 이는 실제 KVS 워크로드에서 큰 값을 쓰는 요청이 드물다는 가정에 기반하며, 약간의 공간 손실이 있더라도 큰 값 복사로 인한 오버헤드를 줄이는 데 유효하다. 피기백을 통해 전송된 값은 여전히 패킹되며, 즉 선택적 Packing 기법은 피기백된 값만 패킹하는 방식이다.

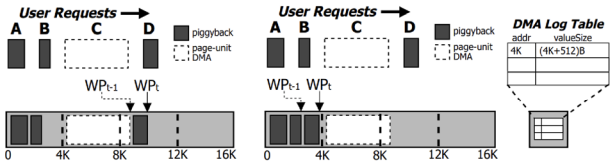

그림 5 (a) 제안된 선택적 Packing 기법과 (b) 백필링 기술이 적용된 최적화 버전에 대한 묘사

그림 5 (a)를 보면, 작은 값과 큰 값이 섞인 네 가지 요청 (*A*, *B*, *C*, *D*)이 있다고 가정할 때, 요청 *A*, *B*, *D*는 피기백 방식으로 값을 전송하고, 요청 *C*는 페이지 단위 DMA를 사용한다. 선택적 Packing에서는 요청 *A*와 *B*의 값이 WP를 따라 압축적으로 패킹되지만, 요청 *C*의 값은 WP에서 가장 가까운 페이지 단위 경계에 위치하게 된다. 그 후 WP는 *C*의 값이 끝나는 지점으로 업데이트되고, *D*의 값은 이 위치에 패킹된다.

5.2.3 선택적 Packing 기법 확장: 백필링 (Backfilling)

작은 값이 주로 사용되는 워크로드에서는 선택적 Packing 기법이 All Packing 기법과 유사한 성능을 보일 수 있다. 그러나 간헐적으로 페이지 단위 DMA로 전송되는 값이 포함될 경우 NAND 페이지 내부 단편화가 발생할 수 있다. 이를 해결하기 위해 우리는 선택적 Packing을 위한 백필링 (Backfilling) 기법을 제안하여 NAND 페이지 내부 단편화 문제와 큰 값 패킹 시 메모리 복사 문제를 모두 해결하고자 한다.

그림 5 (b)를 보면, 요청 *C*까지 처리하는 과정은 이전과 동일하다. 하지만 여기서는 DMA로 전송된 값을 받은 후 WP를 업데이트하지 않고, 요청 *D*의 값이 원래 WP에 패킹되어 빈 공간을 채우는 백필링 방식을 도입한다. 이를 위해 페이지 단위 DMA로 전송된 값을 추적하는 추가 데이터 구조가 필요하며, 이를 DMA Log Table (DLT)이라고 한다. DLT는 NAND 페이지 버퍼와는 별도의 장치 메모리 공간에 저장되며, 각 DMA 작업의 목적지 주소와 값 크기를 기록한다. DLT는 순환 큐로 동작하며, 가장 오래된 미사용 항목을 가리킨다. 컨트롤러는 피기백 방식으로 값을 패킹할 때마다, 현재 WP와 값 크기가 미사용된 DLT 항목의 주소를 초과하는지 확인한다. 초과할 경우, DMA로 전송된 값의 크기를 더한 새로운 주소로 WP를 업데이트하고 과정을 반복한다. 이 참조 과정은 O(1) 시간 복잡도를 갖는다.

DLT는 추가적인 메모리 공간을 필요로 하며, 이는 설계 비용을 증가시킬 수 있다. 이를 절약하기 위해 BandSlim은 전체 주소 대신 논리적 NAND 페이지 번호와 메모리 페이지 오프셋만 기록하여 비트 수를 줄였다. 예를 들어, 1TB의 NAND 공간과 16KB의 페이지 크기를 사용할 경우 40비트 대신 (26+2)비트만 필요하다. 또한, NAND 페이지 버퍼 항목 수가 제한적이므로 최대 항목 수를 512개로 제한하였다. 각 DLT 항목에 값 크기를 지정하는 데 4바이트를 할당하면, DLT의 메모리 공간 상한은 4KB에 불과하다.

5.3 vLog에 대한 세밀한 주소 지정

세밀한 값 패킹은 vLog에서 바이트 단위 주소 지정을 필요로 하며, 이는 LSM-tree에서 주소 필드의 비트 크기를 증가시킨다. SSD는 메모리 용량이 제한적이기 때문에 공간 사용을 최소화하려는 경향이 있어, 주소 필드의 비트 크기가 줄어드는 경우가 많다 [6]. 하지만 본 원고에서는 vLog 주소 필드의 비트 크기 증가, 즉, MemTable의 크기가 약간 증가하더라도, LSM-tree의 플러시 및 초기화로 인해 그 크기는 일정하게 유지된다는 점에 주목한다. 일정량의 메모리 요구 증가에도 불구하고, 세밀한 값 패킹이 제공하는 공간 활용 효율성 향상과 KV-SSD 성능 개선이라는 장점은 충분히 합리적인 타협이라고 주장한다.

6. 제안 기술의 효과

BandSlim이 적용된 KV-SSD는 기존 NVMe 기반 KV-SSD에 비해 상당한 성능 향상을 보였다 [18]. 특히, 세밀한 값 전송을 통해 PCIe 트래픽을 최대 97.9%까지 감소시켰으며, NAND 페이지 쓰기 횟수도 최대 98.1% 줄어들었다. 1KB 이하 작은 크기의 값에 대해선 Piggyback 방식을 적용했을 때 트래픽이 크게 줄이면서 응답 시간도 절반 가까이 단축되었다. 그러나 큰 값의 경우 후속 명령으로 인한 오버헤드로 성능 저하가 발생했으며, 하이브리드 및 적응형 전송 방식은 트래픽 절감과 성능 향상 사이의 균형을 맞추는 데 효과적이었다. 패킹 관련 실험에서는, 작은 값 크기 (4바이트에서 32바이트)에 대해 NAND 쓰기 횟수가 최대 98.1%까지 감소했다. 이는 작은 값에 대한 세밀한 값 패킹이 NAND 페이지 쓰기를 크게 줄였기

때문이다. 백필링 기반 선택적 Packing은 NAND 페이지 내부 단편화를 방지하면서 메모리 복사 오버헤드를 줄이는 데 효과적이었다. 큰 값이 많이 포함된 워크로드에서는 성능이 다소 저하되었지만, 작은 값이 주로 사용되는 실험에서는 All Packing보다 성능이 약 7% 더 향상되었다.

7. 결 론

본 원고에서는 전통적인 블록 기반 스토리지 프로토콜과 KV-SSD의 키-값 인터페이스 간의 비호환성을 지적, 이를 해결하기 위한 솔루션인 BandSlim을 소개하였다. 불일치로 인해 PCIe 트래픽 과부하와 NAND 쓰기 증폭이 발생해 성능 저하를 초래한다. BandSlim은 세밀한 값 전송과 효율적인 장치 내 값 패킹을 통해 이러한 문제를 해결한다.

참고문헌

[1] Facebook. 2014. RocksDB.

[2] Google. 2017. LevelDB.

[3] Samsung. 2017. Samsung Key-Value SSD Enables High Performance Scaling.

[4] C. G. Lee, H. Kang, D. Park, S. Park, Y. Kim, J. Noh, W. Chung, and K. Park, “iLSM-SSD: An Intelligent LSM-Tree Based Key-Value SSD for Data Analytics,” Proc. MASCOTS, 2019.

[5] J. Im, J. Bae, C. Chung, Arvind, and S. Lee, “PinK: High-speed In-storage Key-value Store with Bounded Tails,” Proc. USENIX ATC, 2020.

[6] H. Jo, J.-U. Kang, S.-Y. Park, J.-S. Kim, and J. Lee, “FAB: Flash-aware Buffer Management Policy for Portable Media Players,” IEEE Trans. Consum. Electron., Vol. 22, No. 2, 2006.

[7] C. Duffy, J. Shim, S. H. Kim, and J. S. Kim, “Dotori: A Key-Value SSD Based KV Store,” Proc. VLDB Endow., Vol. 16, 2023.

[8] I. Park, Q. Zheng, D. Manno, S. Yang, J. Lee, D. Bonnie, B. Settlemyer, Y. Kim, W. Chung, and G. Grider, “KV-CSD: A Hardware-Accelerated Key-Value Store for Data-Intensive Applications,” Proc. IEEE CLUSTER, pp. 132-144, 2023.

[9] Y. Jin, H. W. Tseng, Y. Papakonstantinou, and S. Swanson, “KAML: A Flexible, High-Performance Key-Value SSD,” Proc. IEEE HPCA, pp. 373-384, 2017.

[10] NVM Express Inc. 2021. NVM Express Key Value Command Set Specification.

[11] NVM Express Inc. 2011. NVM Express Specification.

[12] H. Cao, S. Dong, S. Vemuri, and D. H. C. Du, “Characterizing, modeling, and benchmarking RocksDB key-value workloads at Facebook,” Proc. USENIX FAST, pp. 209-224, 2020.

[13] SK hynix. 2023. Accelerating Data Analytics Using Object Based Computational Storage in a HPC.

[14] 2017. nvme : add Scatter-Gather List (SGL) support in NVMe driver. [Online]Available:https://lore.kernel.org/all/04aaed5c-1a8a-f601-6c9c-88bf1cf66e8a@mellanox.com/T/

[15] The Linux Kernel source code. 2024. sgl_threshold. [Online]Available:https://github.com/torvalds/linux/blob/master/drivers/nvme/host/pci.c

[16] W.-O. Kwon, S.-W. Sok, C.-H. Park, M.-H. Oh, and S. Hong, “Gen-Z memory pool system implementation and performance measurement,” ETRI J., Vol. 44, No. 3, pp. 450-461, 2022.

[17] The Linux Kernel documentation. 2020. Dynamic DMA mapping Guide.

[18] J. Park, C.-G. Lee, S. Hwang, S. Yang, J. Noh, W. Chung, J. Lee, and Y. Kim, “BandSlim: A Novel Bandwidth and Space-Efficient KV-SSD with an Escape-from-Block Approach,” Proc. ICPP, 2024.

약 력

박 준 혁

2024 서강대학교 컴퓨터공학과 졸업(학사)
2024~현재 서강대학교 컴퓨터공학과 석사과정
관심분야 : 스토리지 시스템, 데이터베이스 시스템, 하드웨어 가속기 기반 이기종 컴퓨팅, 캐시-일관성 인터커넥트 기반 메모리 확장
Email : junttang@sogang.ac.kr

김 영 재

2001 서강대학교 컴퓨터공학과 졸업(학사)
2003 KAIST 전산학과 졸업(석사)
2003~2004 한국전자통신연구원 (ETRI) 연구원
2009 펜실베니아주립대학교 컴퓨터공학과 졸업 (박사)
2009~2015 미국 US Department of Energy’s Oak Ridge National Laboratory (ORNL) Staff Scientist
2016~ 현재 서강대학교 컴퓨터공학과 정교수
관심분야 : 운영체제, 파일 시스템, 스토리지 시스템, 데이터베이스 시스템, 클라우드 컴퓨팅, 머신러닝 시스템 최적화
Email : youkim@sogang.ac.kr