로그 구조 병합 트리 기반의 키-밸류 SSD 기술 연구

서강대학교 ▌민동현·김영재

1. 서 론

RocksDB [1]와 같은 키-밸류 스토어 (KV-stores)는 호스트에서 동작하는 NoSQL 기반 데이터베이스이다. 반면, 키-밸류 기반의 solid-state drives (KVSSDs)는 SSD 안에서 키-밸류 스토어의 스토리지 엔진을 동작시킨다. KVSSD에서는 유저로부터 생성되는 KV 요청이 호스트의 두꺼운 I/O 스택을 거치지 않고 SSD에 직접 전달되어 처리된다. 따라서 높은 I/O 효율성과 쓰기 성능이 보장된다. 최근 로그 구조 병합 트리 (LSM-tree) 기반의 인덱싱 자료구조를 갖는 여러 KVSSD가 제안되어 왔다 [2-4]. LSM-tree는 KV 데이터 저장 시, 덮어쓰기의 형태가 아니라 디스크에 순차적으로 다른 자리 로깅을 수행한다. 따라서, LSM-tree 기반의 KVSSD 기술들은 쓰기 집중적인 워크로드에 특히 최적화가 되어있다. 표 1에 최근 제안된 LSM-tree 기반의 KVSSD의 기술들이 비교 정리되었다. 본 글에서는 LSM-tree 기반의 KVSSD의 대표적인 사례인 iLSM-SSD [2]와 Iso-KVSSD [4] 연구를 중점적으로 다룬다.

표 1 LSM-tree 기반의 KVSSD 기술 비교표

	iLSM-SSD [2]	PinK [3]	Iso-KVSSD [4]
키-밸류 분리 기법 적용 여부 [7]	O	O	O
Tail latency 문제 해결 여부	X	O	X
Multi-tenancy 지원 여부	X	X	O
출판 학회명/ 연도	MASCOTS, 2019	USENIX ATC, 2020	ACM HotStorage, 2021

† SK Hynix의 지원과 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No. 2020-0-00104, 저지연 입출력 집약적 엣지 데이터 처리를 위한 스토리지 모듈 기술 개발)임.

2019년에 iLSM-SSD [2]는 기존의 KV-stores에서 KV 요청을 수행할 때 발생하는 I/O 스택 오버헤드를 분석하여 보고했다. 기존의 널리 쓰이던 키-밸류 스토어인 RocksDB는 데이터 영속성 (persistence)을 위해 파일 시스템의 fsync 혹은 fdatasync와 같은 시스템 콜을 매 KV 데이터 쓰기 시 호출한다. 이러한 잦은 시스템 콜 호출은 전체 키-밸류 스토어의 주요 I/O 병목 요인으로 보고되었다. 이를 개선하기 위해, iLSM-SSD는 SSD 단에서의 키-밸류 스토리지 엔진 설계법을 제안 및 구현, 호스트와의 통신을 위한 API 및 NVMe 커맨드를 새롭게 재정의를 했다. 이에 따라 iLSM-SSD는 호스트의 두꺼운 I/O 스택을 피하고 주요 I/O 병목 요인을 없애서 성능 향상을 이루었다.

그러나 현존하는 모든 LSM-tree 기반 KVSSD 기술들은 multi-tenancy를 고려한 설계가 아니었다. Multi-tenancy는 하나의 서버위에 여러 데이터베이스 인스턴스들을 동시에 호스팅할 수 있는 아키텍쳐를 의미한다. 이러한 멀티테넌트 (multi-tenant) 환경에서 각 사용자들은 독립적인 공간을 제공받아서 보안, 프라이버시, 성능 관점에서 격리되는 것을 요구한다 [5]. 각 사용자의 데이터를 논리적인 묶음으로 엄격히 구분하는 네임스페이스 (namespace) 격리는 사용자에게 이러한 격리 공간을 제공하여 앞서 언급한 요구 조건들을 충족시킬 수 있다. 하지만, 현존하는 LSM-tree 기반의 KVSSD 연구들은 네임스페이스 격리 기법에 대한 설계와 구현 조차 부족하다. 추가적으로 동시적 유저가 존재하는 환경에서 현존하는 LSM-tree 기반의 KVSSD는 스토리지 디바이스가 각 유저에게 제공 가능한 약속된 읽기 처리량을 전달하는데 어려움을 갖는다. 이는 여러 유저의 모든 KV 데이터가 여전히 KVSSD 내부에서 공유되고 있는 하나의 LSM-tree 인덱싱에 의해 관리되기 때문이다.

이러한 문제를 해결하기 위해 2021년에, Iso-KVSSD [4]는 KVSSD 안에서 네임스페이스와 성능 관점에서의 격리 기법을 제안하였다. 네임스페이스 격리를 위해 Iso-KVSSD는 유저의 네임스페이스 정보를 토대로 각

유저의 데이터에 대한 접근 제어 (access control) 를 수행한다. 따라서 각 사용자는 고유의 네임스페이스에 해당되는 데이터만 볼 수 있도록 제어된다. 또한, 성능 격리를 위해 Iso-KVSSD는 각 유저 네임스페이스 당 개별적인 LSM-tree 인덱싱 설계를 지원한다. 기존의 공유되는 하나의 LSM-tree 를 사용하는 대신 각 유저는 고유한 네임스페이스에 따라 독립적인 LSM-tree 인덱싱 구조를 제공받아서 KV 데이터 읽기 시, 다른 유저 데이터와의 간섭으로 인한 성능 저하를 피한다.

본 글에서는 현재까지 제안된 LSM-tree 기반의 KVSSD 기술 중 iLSM-SSD와 Iso-KVSSD의 설계를 중점적으로 소개하여 KVSSD 기술의 발전 과정을 설명한다. 우선 2장에서 LSM-tree의 개념과 매커니즘을 이해한다. 3장과 4장에서 각각 iLSM-SSD와 Iso-KVSSD에서의 이슈와 제안하는 설계법을 살펴보고, 끝으로 5장에서 글을 맺는다.

2. 배경 지식

2.1 로그 구조 병합 트리 (LSM-tree)

LSM-tree [6]는 디스크 중심의, 쓰기 최적화 된 자료구조이다. 개념적으로 LSM-tree는 새로운 쓰기 요청을 모아서 일괄적으로 디스크에 순차적 로깅을 수행하는 방식으로 데이터를 기록한다. 그림1은 LSM-tree를 나타낸다. LSM-tree는 메모리 단의 구성요소인 MemTable과 디스크 단의 구성요소인 SSTable의 계층적 구조를 이룬다. MemTable은 유저로부터 전달된 KV 데이터를 일시적으로 저장하는 자료구조다. MemTable은 일반적으로 skiplist로 구현될 수 있고 데이터의 수정이 가능하다. 반면, SSTable은 메모리단의 MemTable에서 KV 데이터가 디스크로 내려올 때 만들어지는 인덱스 파일이다. SSTable의 데이터는 수정이 불가능하다. 각각의 SSTable은 여러 레벨로 (L0, L1, …, Ln) 나뉘어져 LSM-tree 를 형상화한다. 각 레벨에는 인덱스 정보가 키 기반으로 정렬되어 SSTable에 저장된다.

전통적인 LSM-tree와 달리, WiscKey [7]에서는 키-밸류 분리 설계가 제안되었다. 키-밸류 분리 설계의 경우 LSM-tree의 크기 간소화를 위해 SSTable 인덱스에는 밸류가 저장되지 않는다. 대신, Value log 라는 별도의 디스크 자료구조에서 밸류를 저장시킴으로써 키와 저장되는 장소를 분리시킨다. 키-밸류 분리 설계가 적용된 LSM-tree의 경우에는 각 SSTable은 메타 (키, Value log 주소) 영역과 블룸 필터 (BF) 영역으로 구성된다. 메타 영역에는 KV 데이터의 키, 그리고 밸류가 저장된 Value log의 주소가 저장된다. BF는 확률적 자료구조로써 한 집합에 특정 요소가 있는지 검사하는데 쓰인다. BF 를 활용함으로써, LSM-tree는 찾고자 하는 KV 데이터의 키와 저장되어 있는 모든 데이터의 키를 비교하지 않고도 존재성을 검사할 수 있다. LSM-tree에서 레벨 L_i의 SSTable 크기보다 더 큰 L_{i+1}의 SSTable은 이전 레벨 L_i에 대한 버퍼 역할을 한다. 만약 L_i에서 더 이상 KV 인덱스를 저장하지 못할만큼 크기 임계점에 도달했다면 LSM-tree는 compaction 과정을 수행한다. Compaction은 L_i SSTable 을 두 개 이상

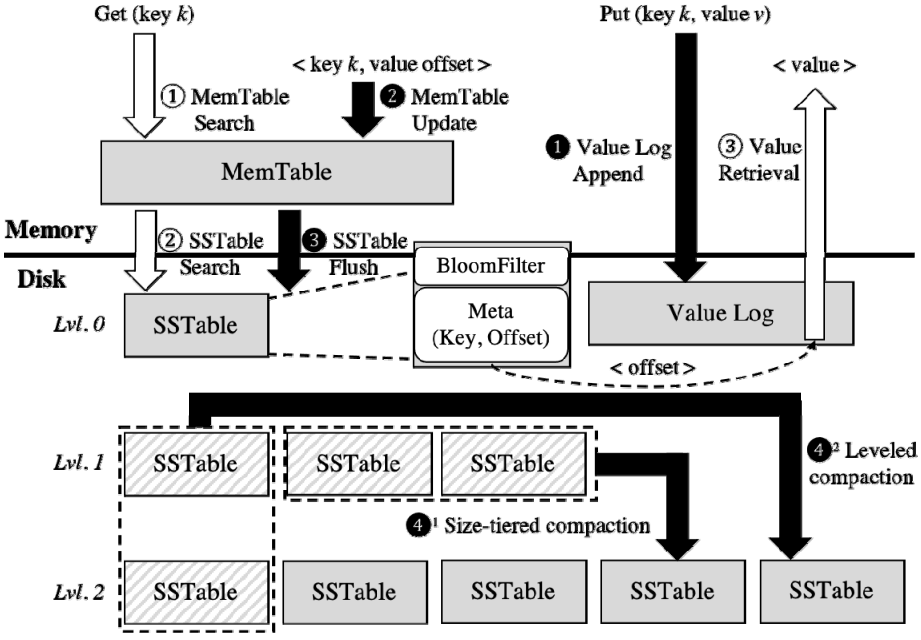

그림 1 LSM-tree에서의 자료구조와 I/O 처리 시 상호작용

선택하고 키 기반의 정렬을 수행하여 다음 레벨인 L_{i+1}의 SSTable로 새롭게 저장하는 과정이다. 이 과정에서 L_i SSTable 내용은 삭제되며 중복된 키를 갖는 인덱스들이 있는 경우 최신 키의 인덱스 정보만 남기고 다른 인덱스 정보는 삭제된다. 키-밸류가 분리 되어 저장된 LSM-tree의 경우, 밸류를 트리에 저장할 필요가 없기 때문에 일반적인 LSM-tree의 경우보다 compaction 오버헤드가 상대적으로 낮다.

2.2 Value log 클리닝

키-밸류 분리가 적용되지 않은 LSM-tree에서는 compaction 과정을 통해 삭제된 키 혹은 중복된 키는 대응되는 밸류와 함께 제거된다. 하지만, 키-밸류 분리가 적용된 LSM-tree의 경우 무효화된 키에 대응되는 밸류를 Value log에서 찾아 삭제해야한다. 이러한 작업을 Value log 클리닝이라 한다. Value log 클리닝 과정은 로그안의 무효화된 밸류를 지우고 동시에 새로운 데이터 로깅을 위해 연속적인 빈 공간을 되찾아야 한다. 이를 위해 Wisckey [7]에서는 밸류 뿐만 아니라 키도 Value log에 함께 저장해두고 head와 tail 포인터로 Value log의 유효 영역을 가리킨다. Value log 클리닝 작업이 불려지면 KVSSD는 우선 tail 포인터로부터 모든 KV 데이터를 읽고 그 후, 키 정보를 기반으로 LSM-tree를 탐색한다. 만약 LSM-tree에서 해당하는 키와 Value log 주소가 있을 때, KV 데이터가 저장된 Value log 주소와 일치한다면 이 KV 데이터는 유효하다고 판단한다. 그렇지 않은 경우는 무효한 데이터라 판단한다. 유효한 데이터의 경우엔 Value log의 head 포인터가 가리키는 새 로그 영역으로 복사되는 방식을 통해 유효한 KV 데이터들이 연속적인 공간에 유지되도록 한다.

3. LSM-tree 기반의 iLSM-SSD 기술

본 장에서는 SSD 내부에서 KV 요청을 처리하기 위한 LSM-tree 인덱싱 기반의 키-밸류 스토리지 엔진을 설계한 iLSM-SSD 를 소개한다. iLSM-SSD [2]는 Wisckey [7]의 키-밸류 분리 설계를 채택한 LSM-tree 기반의 KVSSD 이다. iLSM-SSD 설계에서는 키-밸류 분리를 위해 SSD 내부 DRAM에서 MemTable 이 구현되었고, SSTable과 Value log는 SSD의 NAND 플래시 메모리의 자료구조로써 구현되었다. iLSM-SSD는 유저와의 KV 인터페이스 기반의 통신을 지원하기 위해, 호스트 단에 제공되는 키-밸류 API 라이브러리, 키-밸류 디바이스 드라이버를 구현하였다. iLSM-SSD는 그림 2와 같이 기존 블록 인터페이스 기반의 NVMe 커맨드의 다양한 영역을 KV 데이터를 저장하게끔 용도를 수정하였다. 예를 들어, iLSM-SSD에서는 NVMe 커맨드의 논리 블록 주소 영역에 키를 저장한다. 또한 밸류가 저장된 메모리 주소는 NVMe 커맨드의 페이지 리스트 (Value PRP) 영역에 저장된다. 만들어진 NVMe 커맨드가 후에 iLSM-SSD에 전달되면 DMA를 통해 밸류도 전달이 되고 통신이 완료된다.

SSD 단에 구현된 iLSM-SSD의 compaction 과정에서는 victim L_i SSTable 을 읽기 위한 그리고 새로운 L_{i+1} SSTable의 저장을 위한 NAND 플래시 메모리 읽기 및 쓰기가 수반된다. iLSM-SSD는 compaction 과정을 거쳐서 사라진 키에 대응되는 밸류도 삭제시키기 위해 Value log 클리닝 작업을 별도로 필요로 한다. 하지만 Value log 클리닝 작업 중 발견된 유효한 KV 데이터의 경우 Value log의 head 포인터가 가리키는 영역에 저장되고 새롭게 저장된 Value log 주소를 갖고 LSM-tree의 MemTable에 재삽입된다. 따라서 compaction 오버헤드가 여전히 크게 존재한다. iLSM-SSD는 이러한 오버헤드를 줄이기 위해 Scattered 로깅 기법을 제안하였다. Scattered 로깅 기법은 Value log 주소를 NAND 플래시 메모리의 물리 주소로 사용하는 것이 아니라 SSD 안에서 논리 주소를 사용하고 이 논리 주소와 물리 주소의 관계를 유지시키는 방식이다. 이 방식의 장점은 Value

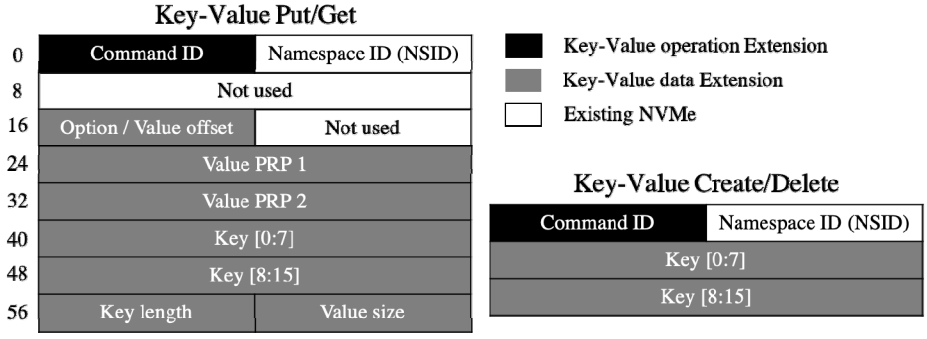

그림 2 KV 인터페이스를 지원하기 위해 확장된 NVMe 커맨드

log 클리닝 과정에서 유효한 로그 엔트리의 물리적 자리가 옮겨진다 하더라도 논리 주소는 변하지 않기 때문에 LSM-tree에 데이터 재삽입 과정이 생략될 수 있다는 점이다.

4. Multi-tenancy 를 위한 최근 KVSSD 기술

Iso-KVSSD [4]는 multi-tenancy를 지원하기 위해 네임스페이스와 성능 격리를 처음으로 시도한 KVSSD이다. Iso-KVSSD는 네임스페이스 격리를 위해, KV 데이터가 MemTable에 저장될 때와 SSTable로 변형되어 저장될 때, 키 뿐만 아니라 유저의 네임스페이스 정보도 함께 저장한다. 유저의 네임스페이스 정보는 그림 2의 Namespace ID (NSID) 영역에 저장된다. 이를 통해 Iso-KVSSD는 서로 다른 유저가 같은 키에 대한 데이터를 접근할 시 저장된 네임스페이스 정보를 기반으로 허가되지 않은 접근을 막는다.

각 유저의 읽기 성능을 격리를 위해, Iso-KVSSD는 각 네임스페이스 당 독립적인 LSM-tree 를 갖는 방식을 제안했다. 제안된 네임스페이스 마다의 독립적인 LSM-tree는 기존의 LSM-tree와 마찬가지로 MemTable과 L_0 SSTable에는 여전히 여러 네임스페이스의 KV 데이터가 혼재되어 있다. 반면 다른 높은 레벨의 (L_1, …, L_n) SSTable에는 각 네임스페이스 별로 KV 데이터가 분리되어 서로 다른 LSM-tree의 서로 다른 SSTable에 인덱싱된다. 이러한 부분적 공유 및 분리 설계는 DRAM 용량이 매우 제한된 SSD 상황에서도 multi-tenancy를 지원할 수 있도록 하기 위함이다. 예를 들어, 만약 각 네임스페이스 마다 MemTable 까지 독립적으로 할당받는 방식 (MemTable provisioning)이라고 가정한 경우, 더 많은 동시적 유저가 존재할수록 추가적인 DRAM 용량이 요구된다. 이는 총소유비용을 증가시키는 방식이다. 각 네임스페이스 별로 서로 다른 LSM-tree 를 구축하는 Iso-KVSSD의 방식은 두 가지 장점이 존재한다. 첫째, KV 데이터가 하나의 LSM-tree 를 공유해서 사용할 때 KV 데이터가 트리의 아래 레벨로 밀려 인덱싱되는 현상을 막는다. 둘째, KV 데이터를 탐색 시, BF 를 NAND 플래시 메모리로부터 여러 번 읽는 것을 줄일 수 있다. 제안된 네임스페이스에 따른 KV 데이터 분리 매커니즘은 기존의 LSM-tree의 L_0 compaction 동작과 같은 시점에 수행된다. L_0 compaction 과정은 victim L_0 SSTable 들을 NAND 플래시 메모리에서부터 읽어들이고, 중복되거나 삭제된 키를 제거, 키 기반으로 데이터 정렬, 그리고 새롭게 만들어진 L_1 SSTable 을 NAND 플래시 메모리에 쓰는 일련의 과정이다. Iso-KVSSD의 KV 데이터 분리 기법은 L_0 compaction 과정에서 이미 읽혀진 SSTable 안에 존재하는 KV 인덱스에 대해서 네임스페이스 별로 분리하는 작업을 수행하기 때문에 추가적인 NAND 플래시 메모리 읽기 과정이 생략된다. L_0 compaction 과정에서 KV 데이터가 네임스페이스에 따라 서로 다른 LSM-tree에 인덱싱 되게끔 분리되므로 그 외의 레벨에서 일어나는 compaction 단계에서는 분리 매커니즘이 비활성화되고 오직 compaction 만 수행된다.

5. 결 론

본 글은 차세대 스토리지 시스템으로 떠오르는 키-밸류 기반의 SSD (KVSSD)의 대표적 연구 사례를 소개하고 각 연구에 대한 핵심적 설계를 살펴보았다. 구체적으로 키-밸류 스토어의 스토리지 엔진을 SSD 내부에 설계하는 법부터 최근 멀티테넌트 (multi-tenant) 환경을 고려한 새로운 KVSSD 설계법에 대해서 논의하였다. 본 글을 통해 KVSSD의 개념적 이해를 높이고 동시에 앞으로 등장할 다양한 KVSSD 관련 연구 사례에 대한 접근성을 높일 수 있는 것으로 사료된다.

참고문헌

[1] Google, RocksDB: A Persistent Key-Value Store for Fast Storage Environment, 2012.

[2] Chang-Gyu Lee, Hyeongu Kang, Donggyu Park, Sungyoug Park, Youngjae Kim, Junhki Noh, Woosuk Chung, and Kyung Park, iLSM-SSD: An Intelligent LSM-Tree Based Key-Value SSD for Data Analytics, In Proceeding of the 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), p. 384-395, 2019.

[3] Junsu Im, Jinwook Bae, Chanwoo Chung, Arvind, Sungjin Lee, PinK: High-speed In-storage Key-value Store with Bounded Tails, In Proceeding of the USENIX Annual Technical Conference (USENIX ATC), p.173-187, 2020.

[4] Donghyun Min, Youngjae Kim, Isolating Namespace and Performance in Key-Value SSDs for Multi-tenant Environments, The 13th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage), 2021.

[5] Ajay Gulati, Arif Merchant, and Peter J Varman, pClock: An Arrival Curve based Approach forl QoS Guarantees

in Shared Storage Systems, ACM SIGMETRICS Performance Evaluation Review 35, 1, p.13-24, 2017.
[6] O'Neil, Patrick and Cheng, Edward and Gawlick, Dieter and O'Neil, Elizabeth, The Log-Structured Merge-Tree (LSM-tree), Acta Informatica, 33, 4, p.381-385, 1996.
[7] Lu Lanyue, Pillai Thanumalayan Sankaranarayana, Gopalakrishnan Hariharan, Arpaci-Dusseau Andrea C, Arpaci-Dusseau Remzi H, In Proceeding of the File and Storage Technologies (USENIX FAST), p.133-148, 2016.

약 력

민 동 현

2017 서강대학교 컴퓨터공학과 졸업(학사)
2019 서강대학교 컴퓨터공학과 졸업(석사)
2019~현재 서강대학교 컴퓨터공학과 박사과정
관심분야 : 시스템 소프트웨어, 스토리지 시스템, 키-밸류 데이터베이스 시스템, 시스템 보안
Email : mdh38112@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 조교수/부교수
관심분야 : 운영체제, 파일 시스템, 스토리지 시스템, 데이터베이스 시스템, 시스템 보안, 분산 시스템
Email : youkim@sogang.ac.kr