버져닝 지원 해시 인덱스 기반 네트워크 키-값 스토어

박정환[1], 박여현[1], 김경표[2], 박성순[2,3], 김영재[1]
[1]서강대학교 컴퓨터공학과, [2]글루시스, [3]안양대학교 컴퓨터공학과
{junghwan, yeohyeon, youkim}@sogang.ac.kr, {kpkim, sspark}@gluesys.com

A Hash Index-Based Network Key-Value Store Supporting Versioning

Junghwan Park[1], Yeohyeon Park[1], Kyungpyo Kim[2], Sung-Soon Park[2,3], Youngjae Kim[1]
[1]Sogang University, [2]Gluesys, [3]Anyang University

요 약

최근 분리화 스토리지 환경에서 스토리지 노드 기반의 네트워크 키-값 스토어인 NetKV가 제안된 바 있다. NetKV는 경량화된 API를 사용하여 호스트 노드의 커널 I/O 스택 오버헤드를 회피한다. 본 연구에서는 분리화 스토리지 환경에서 데이터 변조 문제 해결 및 사용자 수준의 버전 관리 기능 지원을 위해 NetKV에 노드 체이닝(Node Chaining) 기법을 적용한 버저닝 지원 해시 인덱스 기반 네트워크 키-값 스토어 (VNetKV)를 제안한다. VNetKV는 해시 인덱스 기반의 자료구조를 통해 상이한 버전의 키-값 쌍들을 외부 갱신(Out-of-Place Update) 방식으로 저장하고, 포인트 쿼리(put, get)와 버전 회귀(rollback) 등의 사용자 요청을 처리할 수 있다. 실험 평가를 위해, 두 종류의 SSD를 각각 장착한 환경에서 DB벤치마크의 Fillseq 워크로드에 대한 성능을 쓰기 캐시 활성화 여부와 LBA Space 크기의 변화에 따라서 NetKV와 비교 평가하였다. 그 결과 VNetKV는 LBA 공간이 작을 경우 둘 사이의 성능 차이는 미미했으나, LBA 공간이 커질 수록, 쓰기 캐시가 비활성화된 상태에서 성능 저하가 발생함을 관찰하였다. 또한 낮은 지연시간으로 시스템 회복(recovery)와 이전 버젼 데이터 읽기가 가능함을 보였다.

1 서론

네트워크 키-값 스토어(NetKV)는 분리화 스토리지 환경에서 호스트 노드의 커널 I/O 스택 오버헤드를 회피하여 고성능 I/O를 지원한다 [1]. NetKV는 계산 노드 대신 스토리지 노드에 키-값 스토어를 구동한다. 사용자는 호스트 상의 파일시스템을 우회하는 경량의 키-값 API를 사용하여 네트워크 기반 스토리지에 직접 키-값 요청을 보낸다. 즉 NetKV에서 사용자의 I/O는 호스트의 파일시스템을 거치지 않아 짧은 I/O 지연시간을 가진다. 또한, NetKV는 스토리지 노드에서 사용자 영역 디바이스 드라이버인 Intel SPDK [2] 안에서 키-값 스토어를 구동하여 커널 우회를 통해 서버의 I/O 스택 오버헤드를 최소화하였다.

한편, 스토리지에 저장된 데이터는 랜섬웨어와 같은 악성 소프트웨어나 사용자의 실수로 인해 변조되거나 삭제 될 수가 있다 [3]. 랜섬웨어는 데이터를 암호화 한 뒤 이를 복원하는 방법에 대가를 지불하지 않을 시 데이터를 파괴하여 기관/개인에 막대한 피해를 입힌다. 또한, 사용자의 실수로 데이터가 삭제되거나 갱신되는 일이 빈번히 발생하고, 개발 상 이전 버전으로의 회귀를 원할 수 있다. 이와 같이 데이터 손실의 방지 및 버전 관리를 위해 데이터 백업 또는 버져닝 기법이 요구된다 [3]. 특히, 버저닝 기법은 데이터를 버전 별로 관리하는 기법으로 최신 데이터가 삭제되더라도 이전 버전으로 데이터를 복원을 가능하게 하여, 앞서 언급한 데이터 손실로 인한 피해를 최소화할 수 있다.

본 연구에서는 NetKV에서 데이터 변조 문제를 해결하기 위해 NetKV에 노드 체이닝 기법을 적용한 버져닝 지원 해시 인덱스 기반 네트워크 키-값 스토어(VNetKV)를 제안한다. VNetKV는 해시 인덱스 기반 자료구조를 채택하여 키-값 쌍들을 관리하며, 사용자의 포인트 질의 뿐만 아니라 롤백(Rollback)을 지원한다. 평가를 위해 두 가지 종류의 SSD가 장착된 환경에서 쓰기 캐시 활성화 여부 및 LBA Space의 크기에 따른 DB벤치마크의 'Fillseq' 워크로드 실험을 수행하였다. 다양한 실험을 통해 우리는 LBA Space가 크고 캐시가 활성화 된 경우에는 VNetKV가 NetKV보다 약간의 성능 저하가 발생하지만, 낮은 지연시간으로 시스템 회복(recovery)과 이전 버젼 데이터 읽기가 가능함을 보였다.

이 논문은 2023년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2020-0-00104, 저지연 입출력 집약적 엣지 데이터 처리를 위한 스토리지 모듈 기술 개발)

2 배경 지식

2.1 Storage Performance Development Kit (SPDK)

SPDK [2]는 고성능 스토리지 응용을 위해 인텔에서 구현한 사용자 영역 NVMe 드라이버 및 스토리지 프레임워크이다. SPDK는 다음과 같은 세가지 설계 원칙으로 초저지연 NVMe SSD 환경에서 병목이 되는 소프트웨어 오버헤드를 줄인다.

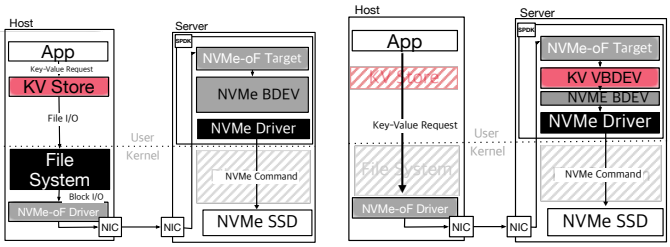
- **무잠금(lock-free) 자료구조:** SPDK가 점유하는 CPU 코어마다 쓰레드가 할당된다. 쓰레드끼리 자료구조를 공유할 경우 락 경쟁(lock contention)이 발생하므로 쓰레드마다 자료구조를 두는 방식으로 락 경쟁을 해소하였다.
- **폴링모드 IO처리:** 하드웨어를 폴링함으로써 IO 완료 여부를 체크한다. 이는 저지연 SSD 환경에서 인터럽트 방식 보다 더 빠르게 완료된 IO를 처리할 수 있다.
- **유저/사용자 영역 디바이스 드라이버:** 디바이스 드라이버를 유저 영역으로 옮김으로써 시스템콜 호출시 발생하는 컨텍스트 스위치, 유저-커널 모드 스위치 등의 비용을 제거한다.

유저는 직접 정의한 BDEV(VBDEV) 를 IO경로 상에 추가함으로써 IO 처리 과정에 중복 제거, 메타데이터 관리 등의 작업을 추가할 수 있다.

2.2 스토리지 노드 기반 키-값 스토어

본 연구에서 기반으로 하는 스토리지 노드 기반 키-값 스토어는 전통적인 계산 노드 기반 키-값 스토어에 비해 경량화된 I/O 경로를 가진다. 그림 1는 전통적인 계산 노드 기반 설계와 스토리지 노드 기반 NVMe-oF 기반 키-값 스토어의 설계 구조를 각각 보여준다.

그림 1(a)은 계산 노드 기반 키-값 스토어에서 사용자의 키-값 요청이 처리되는 과정을 보여준다. ❶ 호스트 노드의 응용은 키-값



(a) 계산 노드 기반 설계 (b) 스토리지 노드 기반 설계

그림 1: 키-값 스토어 구조

쓰기, 읽기 등 I/O 요청을 키-값 스토어에 보낸다. ❷ 키-값 스토어는 I/O 요청에 대한 File I/O를 파일 시스템에 전달한다. ❸ 파일시스템은 File I/O 요청을 블록 I/O 요청으로 변환하여 NVMe-oF 드라이버에 전달한다. ❹ NVMe-oF 드라이버는 이를 NVMe 명령어로 변환하고 NVMe 프로토콜을 이용하여 스토리지 노드로 전송한다. ❺ 스토리지 노드는 SPDK가 제공하는 기능인 NVMe-oF 타겟과 NVMe 드라이버를 이용하여 NVMe 명령어를 처리한다. ❻ NVMe 드라이버는 NVMe 커맨드를 NVMe SSD로 전송한다. NVMe SSD가 I/O처리를 완료하면 콜백을 통해 호스트 노드에 완료를 알린다.

반면, 그림 1(b)는 스토리지 노드 기반 키-값 스토어에서 사용자의 키-값 요청이 처리되는 과정을 보여준다. ❶ 응용은 파일시스템을 거치지 않고 키-값 요청을 NVMe-oF 드라이버로 전달한다. ❷ NVME-oF 드라이버는 키-값 요청을 NVMe 명령어로 변환한 뒤 스토리지 노드로 전송한다. NVMe-oF 타겟은 NVMe 이 명령을 KV VBDEV로 보낸다. ❸ KVBDEV에서는 키-값 쌍에 대한 메타데이터를 키 별 노드에 담아 해시테이블에 저장하고 빈 LBA를 할당받거나(put), 요청받은 키에 대한 LBA를 해시테이블에서 탐색한다(get). 데이터 버퍼와 타겟 LBA 등을 이벤트에 담아서 NVMe BDEV로 보낸다. ❹ NVMe BDEV는 이 요청을 유저 영역 NVMe 드라이버로 보낸다. ❺ NVMe 드라이버는 NVMe 커맨드를 생성해서 NVMe SSD로 보낸다. NVMe SSD가 I/O처리를 완료하면 반대의 경로로 응용에 알림을 보낸다.

3 버져닝 지원 해시 기반 키-값 인덱스

3.1 동시성 지원 (Concurrent) 버져닝 해시 기반 인덱스 구조

그림 2은 버저닝 지원 해시 기반 인덱스의 자료구조의 설계와 동작을 보여준다. 해싱 모듈은 NVMe-oF로 넘어온 키-값 요청에 대하여 키 값으로 해시함수를 돌려 4바이트의 해시값을 생성한다. 해시테이블 자료구조는 두 개 계층의 해시 구조이다. 제 1계층은 해시값의 LSB N비트로 결정하는 2^N개의 서브-해시테이블로 구성된다. 각 서브-해시테이블은 해시값의 다음 M비트로 그 인덱스를 결정하는 2^M개의 버킷을 가진다. 동시성을 가능하게 하기 위해 노드 당 락을 할당하는 Fine-grained 락 기법을 적용했다. 해시테이블에서 각 버킷은 노드들을 연결리스트로 연결하고, 연결리스트를 순회하면서 같은 키 값의 노드가 있는지 탐색할 수 있다.

쓰기 요청의 경우, 현재 입력된 데이터가 기존키-값 쌍의 새로운 버전의 데이터라면 후입 선출(LIFO)의 방식으로 링크를 재조정한다. 포인터를 조정할 때 노드 단위의 락을 앞뒤로 잡고 풂으로써 동시 키-값 요청들에 대해 동시적으로 처리되도록 설계하였다.

읽기 요청의 경우, 함께 들어온 키 값에 대한 노드를 찾기 위해 해시값을 계산한다. 해시값으로써 서브 해시테이블 인덱스와 버킷 인덱스를 결정하고, 해당 버킷에 연결된 연결 리스트를 탐색하여

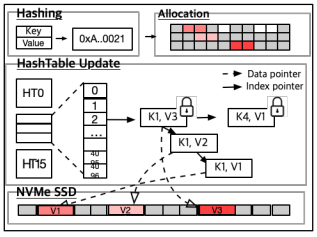


그림 2: VNetKV를 위한 해시기반 인덱스 및 KVBDEV 설계 및 구현

노드를 탐색한다. 목표 노드를 찾은 경우 해당 노드에 기록된 LBA를 블록IO 요청에 담아 NVMe BDEV 에 전달하고, 목표 노드가 해시테이블에 없는 경우 상위 BDEV에 IO 완료 알림 콜백만 보낸다. 이와 같이 동시성 높은 처리와 확장성을 위하여 락을 세밀하게 잡고 임계구역의 최소화 기법 등을 설계 단계에 적용하였다.

3.2 빈공간 관리를 위한 비트맵 구조

새로운 키-값 쌍을 저장할 때 스토리지 디바이스의 빈 주소 관리를 위해 빈공간 주소 할당 모듈이 필요하다. 주소 할당 모듈로 쓰이는 자료구조에는 비트맵이나 연결리스트 등이 있다. 본 연구에서는 비트맵 구조의 주소 할당자를 택했다.

빈 공간 탐색을 위해 디바이스 저장 공간을 4KB의 논리 블록 단위로 나누고, 비트맵 배열을 이용해 각 블록의 할당 여부를 비트 단위로 기록했다. 이로써 1TB 공간의 디바이스를 30MB의 인메모리 자료구조로 관리할 수 있다. 주소할당 모듈은 First fit 알고리즘으로 쓰기 요청한 데이터 크기에 맞는 연속 블록을 할당한다. 쓰기 포인터(Write Pointer/WP)가 할당 가능한 첫 블록 주소를 가리키고, WP부터 원형 탐색을 하여 할당 가능한 블록을 찾은 뒤 WP를 그 다음 블록 주소로 바꾼다. 이 방식은 비트맵의 맨 처음부터 빈 공간을 탐색하지 않아도 돼 로그 기반 구조의 쓰기 특성을 반영한 설계지만, 연속 블록 할당으로 인한 외부 단편화 문제가 남아 있다.

3.3 버져닝 롤백을 지원하는 API 정의

표 1는 VNetKV에서 지원하는 API를 보여준다. 기본적으로 Point Query (put/get)을 지원하며, 해당하는 키에 대한 버젼 목록 보여주기, 특정 버젼의 키-값을 리턴하기, 최신 버젼을 이전 버젼의 키-값으로 되돌리기를 위한 API를 지원한다. 각 API에 대한 NVMe command의 opcode를 새롭게 정의하였다.

4 실험 및 평가

실험 환경 및 설정: 평가를 위해 우리는 10개 코어에 하이퍼쓰레딩이 활성화 된 두개의 동일한 스펙의 서버(계산노드와 스토리지 노드)를 10G 이더넷으로 연결한 클러스터를 구성하였다. 스토리지 노드는 Samsung 970 EVO 500 GB NVMe SSD와 SK Hynix

표 1: VNetKV를 위한 API 정의

API Routine	Description
put(key, value)	$key, value$ 쌍을 저장 장치에 저장함.
get(key)	key에 대응되는 헤더포인터가 가리키는 값을 읽은 후 반환함. 헤더 포인터는 특정 버전 값을 헤더 포인터가 가리키는 버전의 값을 읽어서 반환함. (일반적으로 가장 최신 버전의 값을 가리킴.)
get_version(key, v_id, type)	type=$OFFSET$인 경우, key의 v_id 버전 전의 값을 읽어서 반환함. type=$ABSOLUTE$인 경우, key의 버전v_id 값을 읽어서 반환함.
list_version(key)	key에 대한 버전들의 모든 목록을 반환함.
update_version(key, v_id)	key의 헤드 포인터를 버전v_id의 노드로 바꿈.

CPU	Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz
Memory	32GB DDR4
Network	10 Gbps Ethernet
SSD	250GB Samsung 970 EVO SSD 500GB SKHynix Platinum P41 SSD
OS	Ubuntu 20.04.
SPDK/RocksDB	v.21.10 (스토리지 노드) / v.6.23 (계산 노드)

표 2: 계산 노드와 스토리지 노드의 하드웨어, 소프트웨어 세부 상세 설명

Platinum P41 500 GB SSD를 사용하였다. SPDK의 경우 v.21.10을 사용하였다. 표 2은 실험을 수행한 계산 노드와 스토리지 노드의 상세 세부 스펙을 설명하며, 우리는 다음 두개 시스템에 대한 비교 평가를 수행하였다.

- **NetKV:** 해시 구조 기반 스토리지 엔진을 SPDK에 통합한 스토리지 노드 기반 시스템(버저닝 미지원).
- **VNetKV:** 본 연구에서 제안하는 버져닝 지원 NetKV.

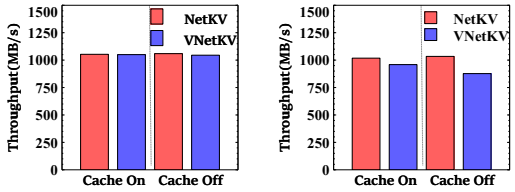
우리는 DB Benchmark의 Fillseq(ReadRandom) 워크로드로 NetKV 대비 VNetKV의 쓰기 및 읽기 성능을 평가하였다. 키와 값 크기는 각각 4B, 8KB이고, 호스트 단에서 각 30만 개의 데이터를 전송하는 40개의 쓰레드를 사용하여 총 1200만 개의 키-값 쌍을 전송하였다.

SSD 쓰기 캐시의 성능 상의 영향을 확인하기 위해 NVMe명령어를 전송하여 SSD에 Write Cache를 활성화 또는 비활성화한 상태로 쓰기 실험을 진행하였다. 또 SSD에서 데이터가 저장되는 실제 LBA 공간을 조절하여 쓰기 캐시의 효과를 확인하였다. 실사용 LBA 공간을 1만 블록(각 4KB)(10k, Small LBA 공간) 또는 5천만 블록(50M, Large LBA 공간)으로 LBA 공간을 조절하였다. 각각의 실험 결과는 실험을 5회씩 반복하여 계산된 평균값이다.

결과 및 분석: 그림 3는 스토리지 노드에서 SK Hynix Platinum P41 SSD를 사용하고 LBA 공간의 범위에 따라 Small (a), Large (b)를 설정하여 수행한 실험 결과를 보여준다. Small LBA 공간에서 NetKV와 VNetKV의 처리율은 3MB/s 차이로 쓰기 캐시 활성화 여부에 상관 없이 두 시스템 성능이 유사하였다. 반면, Large LBA 공간에서는 VNetKV가 NetKV 보다 처리율이 낮았다. 특히, 쓰기 캐시가 활성화된 경우 VNetKV의 처리율이 8.5% 감소한 반면, 비활성화된 경우 처리율은 15%감소했다. LBA 공간이 커지며 캐시에 기록되어 있지 않은 블록의 수가 증가하고(버저닝으로 외부갱신), 이에 따라 VNetKV의 처리율은 캐시 크기에 민감하게 반응하였다.

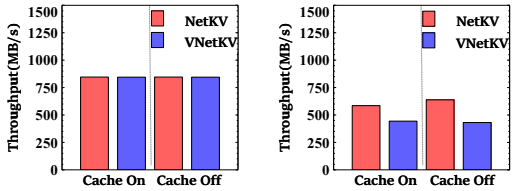
그림 4는 앞의 실험과 동일한 환경에서 Samsung 970 EVO Plus SSD를 사용한 실험 결과이다. Small LBA 공간의 경우 쓰기 캐시 활성화 유무에 상관 없이 NetKV와 VNetKV의 처리율은 각각 846MB/s, 845MB/s로 동일하게 측정되었다. 반면 Small LBA 공간일 때에 비해 Large LBA 공간일 때 두 모델의 처리율은 각각 30.8%, 47.4%씩 감소하였다. 또한 Large LBA 공간일 경우 VNetKV의 처리율은 NetKV 대비 24.1% 감소하였으며, 쓰기 캐시를 비활성화하면 32.4% 감소하여 쓰기 캐시를 비활성화했을 때 큰 폭으로 감소하는 것을 확인할 수 있다.

Large LBA 공간일 경우 VNetKV이 NetKV 보다 성능이 저하되며 특히 쓰기 캐시를 비활성화할 경우 큰 폭의 성능 저하가 발생한다. 반면 Small 공간일 때는 두 모델의 성능 차이가 없었으며, 쓰기 캐시가 성능에 영향을 미치지 않았다. LBA 공간의 크기 차이에 따른 성능 차이는 쓰기 캐시의 적중 비율의 차이로 발생한 결과일 수 있다. NetKV는 이미 저장된 키에 대한 쓰기 요청의 경우 덮어쓰기를 수행하는 반면 VNetKV에서는 모든 키-값 쌍을 외부 갱신(Out-of-Place Update)하여 쓰기 패턴의 차이가 존재한다. 이러한 쓰기 패턴의 차이와 SSD 내부 쓰기 캐시 이용률의 관계에서 VNetKV의 성능 저하의 원인을 일부 찾을 수 있는 것으로 추정된다.



(a) Small LBA 공간 (b) Large LBA 공간

그림 3: SK Hynix Platinum P41 SSD환경에서 쓰기 캐시 활성화 여부와 LBA 공간 크기에 따른 처리율 비교



(a) Small LBA 공간 (b) Large LBA 공간

그림 4: Samsung 970 EVO Plus SSD환경에서 쓰기 캐시 활성화 여부와 LBA 공간 크기에 따른 처리율 비교

한편 DB Benchmark의 ReadRandom 워크로드로 VNetKV의 *get_version(key, v_id, type)*과 NetKV의 *get(key)*의 지연시간을 비교하였다. *get*는 1390.8µs의 지연시간을, *v_id=-3*일 때 *get_version*는 1391.2µs의 지연시간을 가짐으로써 *get_version* API의 성능이 get API와 같음이 확인되었다. 성능 상 차이가 없는 이유는, 데이터를 N버전 되돌리는 것은 연결리스트를 N번 타고 노드를 탐색하는 것과 같은 지연시간을 수반하는데, 연결리스트를 3-4번 탐색하는 소요 시간은 15ns로 매우 경미하기 때문이다.

마지막으로 랜섬웨어에 시스템이 감염된 상황을 상정해 *update_version(key, id)* API를 이용하여 시스템 상 모든 데이터를 회복(recovery)하는 실험을 진행하였다. 50,000개의 고유한 키-값 쌍에 대하여 2버전 전, 1버전 전으로 회귀하는 시스템 회복에는 각각 50ms, 25ms가 소요되었다. 값의 크기를 8KB일 때 이는 400MB 수준의 크기의 데이터이다. 따라서 데이터가 수 TB 수준으로 적재되어도 수 분 이내로 시스템 회복이 가능할 것으로 전망된다.

5 결론

본 연구에서는 분리화 스토리지 환경에서의 데이터 변조 문제를 해결하고 사용자 수준의 버전 관리 기능을 제공하기 위해 NetKV에 노드 체이닝 기법을 적용한 버져닝 지원 해시 인덱스 기반 네트워크 키-값 스토어 (VNetKV)를 제안하였다. 두 개의 NVMe SSD 환경에서 DB Benchmark의 Fillseq 워크로드로써 쓰기 성능을 평가한 결과 LBA Space가 작을 경우 VNetKV와 NetKV와는 성능 차이가 거의 없었지만, LBA Space가 커질 경우 VNetKV가 NetKV 보다 낮은 처리율을 보였다. 또한, SSD의 내부 캐시를 비활성화 할 때는 성능 저하가 더 큼이 확인되었다. 또한 시스템 상 저장된 모든 데이터를 적은 지연시간 안에 복원 처리할 수 있다는 것도 보였다.

참고 문헌

[1] 박여현, 이창규, 김경표, 박성순, and 김영재, "스토리지 분리화 환경에서 스토리지 서버 기반 키-값 스토어 설계 및 구현," in *한국소프트웨어종합학술대회 프로시딩*, KSC '21, 2021.

[2] "SPDK." https://spdk.io/, Accessed 2020-10-10.

[3] X. Wang, Y. Yuan, Y. Zhou, C. C. Coats, and J. Huang, "Project almanac: A time-traveling solid-state drive," in *Proceedings of the 14th EuroSys Conference*, 2019.