NVMe SSD 환경에서 RocksDB의 읽기 지연에 대한 실증적 평가 및 분석

박현기, 김영재†
서강대학교 컴퓨터공학과
{hyun0712, youkim}@sogang.ac.kr

Empirical Evaluation and Analysis of Read Latency in RocksDB under NVMe SSD Environments

Hyungi Park, Youngjae Kim†
Sogang University

요 약

RocksDB는 순차 쓰기 중심의 구조로 높은 쓰기 효율을 보이지만, 병합(compaction) 과정에서 디스크 I/O 및 스레드 자원을 점유함으로써 읽기 요청이 지연되는 문제(read stall)가 발생한다. 이러한 Read Stall 은 사용자 응답 지연을 유발함에도 불구하고, 기존 연구들은 대부분 Write stall 완화에 집중해 왔다. 본 논문은 NVMe SSD 환경에서 LSM-tree 기반 RocksDB 의 Read stall 현상을 실증적으로 분석하였다. 이를 위해 Block cache, Bloom filter 등 핵심 파라미터를 조정하며 Read Latency의 변화와 부작용(side effect)을 정량적으로 측정하였다. 실험 결과, 블록 캐시 512MB와 10 Bit Bloom 필터 설정이 가장 안정적인 성능을 보였다. 캐시를 더 늘려도 성능 향상은 거의 없었으며, 디스크 쓰기 속도를 제한하거나 compaction 병렬성을 낮추면 데이터 병합이 지연되어 전체 처리량이 감소하는 현상이 나타났다.

1. 서 론[1]

대다수 현대의 대용량 키-값 저장소는 RocksDB같은 LSM(Log-Structured Merge)-tree 기반 엔진을 사용하며, 쓰기 처리를 효율화 하는 데 주안점을 두고 설계되었다 [1]. LSM 구조는 메모리(MemTable)에서 디스크에 SST(Sorted String Table) 파일로 데이터를 순차적으로 내보내고, 이후 내부 연산인 병합(Compaction)을 통해 쓰기 효율과 읽기 성능을 확보한다 [2].

최근의 응용 분야를 보면, 쓰기 성능 뿐만 아니라 읽기 성능 — 즉, 읽기 처리율(read throughput)과 읽기 지연(read latency) — 또한 서비스 품질에 직접적인 영향을 미치는 경우가 많다. 예컨대 실시간 추천 시스템이나 사용자 인터랙티브 애플리케이션과 같은 응용에서는 읽기 지연이 발생할 경우 사용자 경험이 크게 저하될 수 있다.

그럼에도 불구하고 기존 연구들은 주로 쓰기 병목을 완화 [3, 4]하는데 초점을 맞춰 왔다. 이는 RocksDB 내부에서 L0 파일 누적 시 자동으로 Write stall을 유발하는 구조적 특성이 있어, 쓰기 성능 저하가 보다 직접적인 문제 [4]로 인식되어 왔기 때문이다. 반면, 읽기 지연(read stall)에 초점을 맞춘 정량적 분석이나, 읽기 지연을 개선하려는 시도가 시스템 전반에 미치는 영향을 면밀히 살펴본 연구는 상대적으로 부족하다.

본 논문은 NVMe SSD 환경에서 RocksDB의 주요 파라미터가 읽기 지연과 시스템 자원 활용에 미치는 영향을 실증적으로 평가한다. 특히 읽기 성능 향상을 목표로 한 튜닝이 처리량, CPU, I/O 부하, 그리고 compaction 적체 등 전체 시스템 동작에 어떤 부작용을 초래하는지를 정량적으로 분석하였다. 이를 통해 NVMe 기반 RocksDB 운용에서 정적 파라미터 조정이 제공할 수 있는 이익과 한계를 규명하고, 읽기 지연에 민감한 응용을 위한 실질적인 튜닝 가이드라인을 제시한다.

2. 배경 지식 및 연구 동기

2.1. RocksDB

RocksDB는 메모리(MemTable)에 데이터를 우선 저장한 뒤, 주기적으로 이를 디스크의 SST(Sorted String Table) 파일로 플러시(flush)하고, 이후 compaction을 통해 중복 데이터를 제거한다 [2]. 이러한 구조는 순차 쓰기에 유리하지만, flush와 compaction이 동시에 발생하면 CPU와 I/O 자원이 읽기 경로와 경쟁하게 되어 읽기 지연(read latency)을 유발할 수 있다 [5].

RocksDB의 읽기 요청은 *DBImpl::Get()*에서 시작되며, 먼저

1) 본 연구는 정부(과학기술정보통신부)의 재원으로 한국연구재단(RS-2025-00564249) 및 2025년 과학기술정보통신부 및 정보통신기획평가원의 SW중심대학사업(2024-0-00043)의 지원을 받아 수행되었음.

† Corresponding Author

MemTable을 확인하고, 해당 키가 없으면 디스크에 저장된 SST 파일을 탐색한다. 이때 디스크 접근을 최소화하기 위해 다음의 읽기 경로 구성 요소를 사용한다.

TableCache: SST 파일의 메타데이터를 캐싱하여 동일 SST 파일이 반복적으로 접근될 때 디스크의 파일을 다시 열지 않고 메모리에 저장된 정보를 재활용한다.

Bloom Filter: SST 파일 내부의 필터 블록(filter block) 에 저장된 확률적 자료구조로, 특정 키가 파일에 존재하지 않을 가능성을 빠르게 판단한다.

Block Cache: SST 파일의 실제 데이터 블록(key–value pairs)을 캐싱하는 메모리 영역으로, 캐시 미스가 발생한 경우에만 디스크에서 블록을 읽어온다.

이러한 구성 요소들은 "TableCache(메타데이터 조회) → Bloom Filter 검사 → Block Cache 조회 → 필요시 디스크 접근"의 순서로 동작하며, RocksDB의 전체 읽기 경로를 형성한다. RocksDB는 이 전체 경로의 수행 시간을 측정하고 내부 통계를 기반으로 end-to-end latency의 분포를 분석할 수 있다.

특히 Bloom Filter는 존재하지 않는 키에 대한 디스크 접근을 줄여 평균 응답 시간을 크게 줄이는 핵심 구성 요소이며, 본 연구에서는 해당 파라미터와 Block Cache 크기 등을 조정하여, 디스크 접근 회피 효과와 시스템 자원 사용 간의 상관관계를 실증적으로 분석하였다.

2.2. 연구 동기

RocksDB는 수백 개의 파라미터로 세밀한 튜닝이 가능하지만, 이들 간의 상호작용이 복잡해 특정 성능 목표(예: 읽기 지연 최소화)에 최적화된 조합을 찾기 어렵다 [6]. 한편, 기존 연구와 공식 가이드라인은 주로 쓰기 병목(write stall) 완화나 compaction 효율 개선에 초점을 맞춰왔으며, 읽기 지연(read stall)에 대한 체계적 분석은 상대적으로 부족하다.

그 결과, 실무에서는 읽기 지연을 개선하기 위한 명확한 기준과 근거가 부재한 상황이다. 본 연구는 이러한 공백을 보완하기 위해 NVMe SSD 환경에서 읽기 관련 주요 파라미터가 성능에 미치는 영향을 실증적으로 평가하고자 한다.

3. 연구 방법론

3.1. 실험 방안

본 실험은 읽기 지연을 최소화하기 위한 튜닝이 시스템 전반에 미치는 영향을 분석하는 과정에서 처리량, CPU 사용률, NVMe I/O, 그리고 compaction 적체 현상이 어떻게 변화하는지를 관찰하였다. 실서비스의 편향 특성을 반영하기 위해 YCSB Workload A (READ 50% / UPDATE 50%, Zipfian) [7]을 사용하였으며, 각 조건에서 8개 스레드로 100만 트랜잭션을 5회 반복 실행한 평균값을 측정하였다. 데이터는 1KB × 1천만 레코드를 한 번 로드한 뒤, 동일한 데이터베이스에 대해 파라미터를 조정하며 실험을 수행하였다. 비교 구성은 다음과 같다.

① **Baseline:** 블록 캐시 256 MB, bloom_bits=0(비활성).

② **ReadOpt-512MB:** 캐시 512 MB, bloom_bits=10, partitioned filter, 인덱스/필터 캐시가 디스크 접근 회피를 극대화한다.

③ **ReadOpt-1GB:** 위와 동일하되 캐시를 1GB로 설정하여 캐시 증설의 한계를 확인해본다.

④ **Mild-Pressure:** ReadOpt-512MB 구성을 기반으로, 작은 버퍼 크기, 작은 파일 단위, 낮은 compaction 병렬성, 200 MB/s I/O 속도 제한을 동시에 적용하였다. 이는 실제 운영 환경에서 발생할 수 있는 자원 제약(메모리, I/O의 병렬 처리 제한) 상황에서 읽기 최적화 효과가 얼마나 유지되는지를 검증하기 위한 설정이다.

각 실험에서는 처리량과 평균/P99 지연을 측정했으며, 내부 로그를 통해 flush 및 compaction 타임라인, L0 파일 수, pending compaction bytes(적체된 compaction 수), slowdown/stall 메시지를 함께 기록하였다. 또한 *'mpstat/iostat -xm'* 을 사용해 CPU(user/sys/iowait)와 NVMe 활용률을 5초 간격으로 수집하였다.

3.2. 실험 환경

실험은 단일 호스트에서 수행하였다. 시스템은 Intel Core i7-6700(3.40 GHz) 프로세서, 16 GB DRAM, 그리고 SK hynix Platinum P41 NVMe M.2 SSD(1 TB, PCIe 4.0)로 구성되었으며, 운영체제는 Ubuntu 20.04 LTS, RocksDB 버전은 10.8.0을 사용하였다. 데이터셋은 값 크기 1 KB 기준 1천만 레코드로 구성되었으며, 전체 크기는 약 11–12 GB이다.

4. 실험 결과 및 분석

그림 1은 처리량과 평균 지연의 변화를, **그림 2**는 CPU와 NVMe 자원 사용의 변화를 나타낸다.Baseline 대비 ReadOpt-512MB는 블록 캐시를 512MB로 확장하고 10 bit Bloom 필터를 적용한 결과, 처리량이 약 2.45배(16,190→39,680 ops/s) 증가하였다. 평균 READ/UPDATE 지연은 각각 0.283→0.126ms, 0.685→0.255ms로 크게 감소했으며, NVMe 활용률은 32%에서 28%로 낮아졌다. CPU 사용률 중 user 비율은 28.1%에서 40.5%로 증가했으나 sys 비율은 거의 동일했다. *이는 디스크 접근 대신 메모리 기반 캐시 및 Bloom 필터 연산이 늘어난 결과로, 읽기 병목의 주요 원인이 하드웨어보다는 소프트웨어 경로에 있음을 시사한다.*

ReadOpt-512MB에서 1GB로 캐시를 늘리면 성능 변화는 미미했다. 처리량(39,351 ops/s)과 READ 지연(0.125ms)은 사실상 동일했고, NVMe 활용률이 약간 증가(28%→30%)하는 수준이었다. 즉, 캐시 적중률이 이미 포화되어 추가 확장은

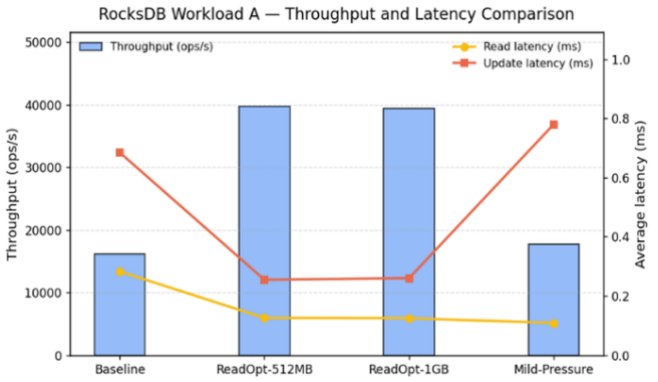

그림 1: Workload별 Read/Write Throughput 및 Latency 측정.

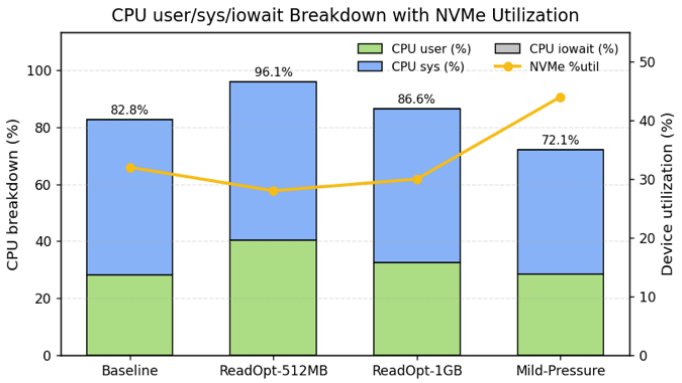

그림 2: Workload별 CPU/IO Utilization 통계.

효과가 제한적이었다.

Mild-Pressure 실험은 작은 버퍼, 낮은 병렬성, I/O 속도 제한 등 자원 제약 환경에서 읽기 최적화의 지속성을 검증하기 위해 수행하였다. 이때 처리량은 17,675 ops/s로 감소하고, UPDATE 지연은 0.78ms로 증가했다. NVMe 활용률은 44%로 상승했으며, 내부 로그에서는 잦은 flush와 약 24GB의 pending compaction bytes 누적이 확인되었다. CPU의 user/sys 비율이 모두 낮아진 것은 I/O 대기로 인한 비활성 시간이 늘어난 결과이다. 한편, READ 지연은 0.109ms로 큰 변화가 없었는데, 이는 Bloom Filter와 Block Cache가 대부분의 읽기 요청을 메모리에서 처리했기 때문이다. 즉, compaction 적체는 주로 쓰기 경로에 영향을 미치며, 캐시 히트율이 유지되는 한 읽기 지연은 안정적으로 유지된다.

5. 결론 및 향후 연구

본 연구는 NVMe SSD 환경에서 RocksDB의 읽기 지연을 실증적으로 분석하고 주요 파라미터의 영향을 평가하였다. 그 결과 ReadOpt-512MB의 실험 조합이 평균/P99 지연과 처리량을 모두 개선하는 가장 효율적인 설정으로 확인되었다. 캐시를 1GB로 확장하더라도 추가 이득은 거의 없었으며, 반대로 디스크 쓰기 제한이나 compaction 병렬성 축소는 compaction 적체로 인해 쓰기 지연과 처리량 저하를 초래했다. 이는 읽기 중심 설정이라도 compaction 처리 여유를 유지해야 전체 성능을 안정적으로 확보할 수 있음을 의미한다. 따라서 실무적으로는 캐시 히트율뿐 아니라 compaction 적체, L0 파일 수, NVMe utilization을 함께 모니터링하는 것이 유효하다.

한편, 실험에서는 NVMe의 높은 병렬 처리 특성으로 인해 명시적인 Write stall은 발생하지 않았으며, 단일 호스트 환경에 국한된다는 제약이 존재한다. 향후 연구에서는 멀티 테넌트 및 다양한 데이터 크기 환경으로 범위를 확장하고, P99 지연, compaction backlog, 장치 활용률 변화를 활용한 조기 경보 지표 모델을 제안할 예정이다.

[참고 문헌]

[1] Facebook, "RocksDB Users and Their Use Cases," https://github.com/facebook/rocksdb/wiki/RocksDB-Users-and-Use-Cases

[2] 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.

[3] Oana Balmau, Florin Dinu, Willy Zwaenepoel, Karan Gupta, Ravishankar Chandhiramoorthi, and Diego Didona, "SILK: Preventing Latency Spikes in LogStructured Merge Key-Value Stores," In USENIX Annual Technical Conference (ATC 19), 2019.

[4] Jinghuan Yu, Sam H Noh, Young-ri Choi, and Chun Jason Xue. 2023. "ADOC : Automatically Harmonizing Dataflow Between Components in Log-Structured Key-Value Stores for Improved Performance," In USENIX (FAST 23), 2023.

[5] Facebook, "RocksDB Tuning Guide," https://github.com/facebook/rocksdb/wiki/RocksDB-Tuning-Guide

[6] Chenxingyu Zhao, Tapan Chugh, Jaehong Min, Ming Liu, Arvind Krishnamurthy, "Dremel: Adaptive Configuration Tuning of RocksDB KV-Store," In ACM, Volume 6, Issue 2, Pages 1 – 30, 2022

[7] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears, "Benchmarking cloud serving systems with YCSB," In ACM symposium on Cloud computing (SOCC 10), 2010.