RocksDB Tiered storage를 이용한 성능 비교

이지환*, 최원기*, 박상현*†
*연세대학교 컴퓨터과학과
{ ji_hwan43 , cwk1412 , sanghyun }@yonsei.ac.kr

Performance comparison of using Tiered storage on RocksDB

Jihwan Lee*, Wongi Choi*, Sanghyun Park*†
*Dept. of Computer Science, Yonsei University

요 약

RocksDB는 데이터를 키-값 쌍으로 저장하는 키-값 데이터베이스 시스템이다. RocksDB는 데이터의 효율적인 저장을 위해 LSM-tree(Log-Structured Merge-tree)를 자료 구조로 사용한다. LSM-tree는 메모리 영역과 디스크 영역으로 나뉘어지며, 디스크 영역에서는 레벨을 나누고 각 레벨에 데이터를 저장하고 있다. RocksDB에서 LSM-tree 디스크 영역의 각 레벨의 데이터를 여러 스토리지 기기에 나누어서 저장함으로써 성능과 비용적 이점을 얻고자 한 기법을 Tiered storage라 한다. 일반적인 Tiered storage의 기기 층 설정은 상위 층에 좋은 성능의 기기를 두고, 하위 층으로 갈수록 비용이 적고 저장공간이 큰 기기를 사용한다.

본 논문에서는 Tiered storage를 사용하여 LSM-tree 디스크 영역의 레벨들을 다중 스토리지 기기 층에 포함시켰을 때의 성능을 비교하는 실험을 진행하였다. 그에 앞서 Tiered storage 성능 평가에 필요한 적정 데이터 크기를 설정하는 실험을 하였고, 이를 바탕으로 상위 레벨의 데이터를 하위 레벨보다 더 좋은 성능의 기기에 포함되도록 설정하고, 좋은 성능의 기기에 포함되는 레벨의 범위를 점진적으로 넓혔을 때 Tiered storage의 성능 비교를 보였다. 성능에 가장 큰 영향을 미치는 요인은 최하위 층의 기기 성능이며, LSM-tree에서 좋은 성능의 기기를 상위 층에 두었을 때 오히려 성능 감소의 결과를 확인했다. 이는 LSM-tree의 레벨마다 쓰기 횟수와 Compaction 횟수가 각각 다른 것에 의한 영향으로 생각된다.

1. 서 론

NoSQL은 정형데이터를 처리할 수 있을 뿐만 아니라 현 시대에 상용화되고 있는 빅데이터와 실시간 웹 애플리케이션에서 발생하는 비정형 데이터 또한 쉽게 담아서 처리할 수 있는 구조를 갖고 있다.

NoSQL의 한 종류인 키-값 저장소는 데이터를 키와 값의 쌍으로 저장한다. 키는 값을 찾기 위한 용도로 사용하고, 값은 어떠한 형태의 데이터도 저장 가능하다. 대표적인 키-값 저장소로는 Redis[9], Memcached[10], Cassandra[11], LevelDB[12], 그리고 RocksDB[1]가 있다.

RocksDB는 Facebook에서 시작된 오픈소스 키-값 데이터베이스 개발 프로젝트이며, Google에서 개발한 LevelDB를 기반으로 했다. RocksDB는 LevelDB보다 실제 저장소로써 사용할 때 높은 성능을 내도록 최적화 되어 있다. RocksDB는 LSM-tree(Log-Structured Merge-tree)[3]를 기본 자료 구조로 한다. 이 자료 구조는 각 제한된 크기의 레벨로 나뉘어져 있고 키-값 데이터를 병합 정렬을 통해 트리를 확장한다. 또한 메모리에 입력된 데이터를 디스크에 반영하기 이전에 먼저 로그를 기록하는 WAL(Write-ahead Logging) 방식을 사용하여, 시스템 복구 상황에서 메모리 속의 데이터를 복구하여 데이터베이스 시스템에 원자성과 내구성을 제공한다.

본 논문에서는 RocksDB의 자료 구조인 LSM-tree의 데이터를 다중 스토리지 기기(HDD, SSD(SATA-SSD), NVMe(NVMe-SSD))에 나누어 저장하는 Tiered storage[2] 방식을 사용 했을 때 기기 층의 용량 설정에 따른 성능을 비교하는 실험을 진행하였다. 그에 앞서 실제 Tiered storage를 적용했을 때의 성능에 영향을 미칠 수 있는 데이터 크기를 찾는 실험을 진행하였다. 이를 바탕으로 Tiered storage 성능에 영향을 주는 요인들을 분석하였다.

LSM-tree의 각 레벨마다 쓰기와 Compaction 횟수가 각각 다르고 이것이 성능 결과에 영향을 준 것으로 보인다. LSM-tree에서 부하가 많은 층은 하위층이기 때문에 최하위층에 설정한 기기의 성능이 성능 결과에 큰 영향을 주었다. 특히 예상과 다르게, 상위 층에 좋은 성능의 기기를 두었을 때 오히려 더 낮은 성능 결과를 보였다.

본 논문의 구성은 다음과 같다. 2장에서는 RocksDB의 자료구조인 LSM-tree의 구조와 동작원리에 대해서 먼저 살펴보고, Tiered storage에 대해 설명한다. 3장에서는 실험 환경과 실험 내용에 대해서 설명하고, 4장에서는 실험 결과를 제시하고 분석한다. 마지막으로 5장에서는 향후 연구 계획과 함께 결론을 맺는다.

* 본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 SW컴퓨팅산업원천기술개발사업(SW스타랩)의 연구결과로 수행되었음 (IITP-2017-0-00477).

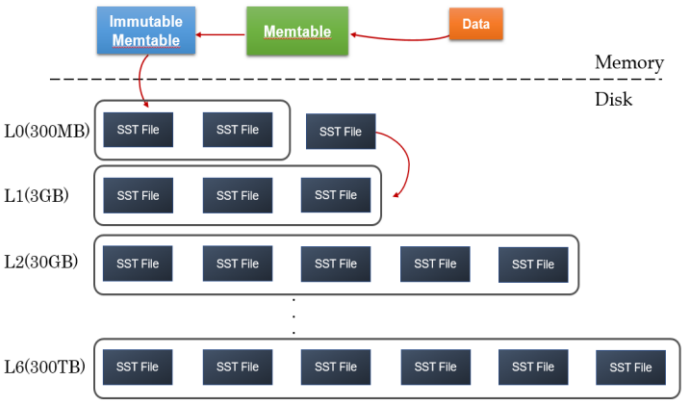
† 교신 저자 : sanghyun@yonsei.ac.kr

2. RocksDB

2. 1 LSM-Tree (Log-Structured Merge-tree)

LSM-tree 는 상위의 메모리에서의 데이터와 그 이하 디스크에서의 다수의 데이터가 레벨 단위로 구성된 트리구조를 갖는다. 데이터의 삽입은 가장 상위의 메모리에서 시작된다. 메모리에 저장할 수 있는 데이터가 꽉 차게 되면 디스크로의 Flush 가 일어난다. 이 때 단순히 데이터를 덧붙이는게 아닌 병합 정렬이 발생되고, 이후의 디스크에서 다음 레벨로의 병합 정렬로 인해 트리구조를 완성시키게 된다.

RocksDB에서 사용하는 LSM-tree구조는 메모리 영역과 디스크 영역으로 크게 나누어 볼 수 있다. 메모리에는 데이터를 입력받는 Memtable이 있다. 디스크에서는 L0(Level 0) 부터 Lk(Level k) 까지 순차적인 층으로 이루어져 있다. 자세한 구조는 (그림 1)과 같다.



(그림1) RocksDB Architecture

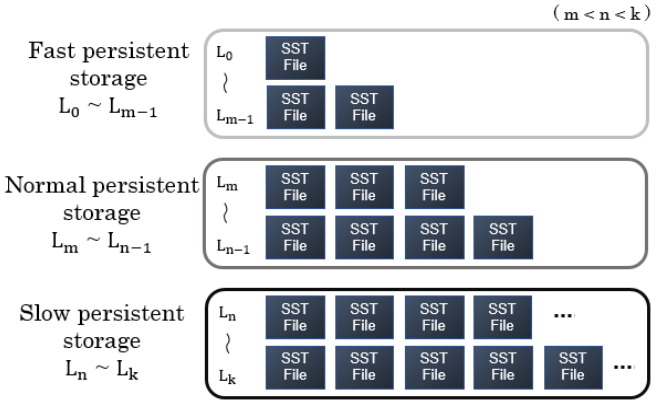
먼저 데이터가 삽입될 때 메모리 내의 Memtable에 데이터 삽입 순서대로 저장된다. 그리고 Memtable에 데이터가 가장 가득 차게 되면, 수정은 불가능하지만 읽기 가능한 Immutable memtable로 변경된다. 이 Immutable memtable은 이후에 디스크 영역의 L0 로 Flush되며, Flush가 될 때 키 순서로 정렬이 되어있는 하나의 SST 파일로 변경된다. 만약 L0 에서 정해진 최대 파일 개수를 넘어서게 되면, L0 의 특정한 SST파일이 선택되고 다음 레벨인 L1 에 있는 파일 중 키 범위가 겹치는 파일들을 선택한다. 선택된 L0 와 L1 의 SST파일들을 병합 정렬하여 하위 레벨에 기록한다. 위 과정을 Compaction 이라고 한다. 이런 방식으로 각 레벨마다 크기를 제한하고 인접한 레벨 간의 Compaction이 일어나면서 Lk까지 트리가 확장된다.

데이터 삽입에 대해서는 Memtable 에서 배치 형식처럼 데이터를 순차적으로 받다가 한 번에 다음 단계로 진행이 되기 때문에 순차적 쓰기가 일어나고, 그렇기 때문에 임의적 쓰기를 하는 B+tree 보다 빠른 삽입이 가능하다. 하지만 Compaction 으로 인해 부가적인 쓰기연산이 발생하는 단점이 있다. 또한 상위의 레벨에서부터 삽입이 되기 때문에 L0 에 가장 최신의 데이터를 갖게 되고 다음 레벨로 갈수록 수정이 오래된 데이터를 갖는다.

데이터 읽기에 대해서는 Memtable, Immutable memtable, L0, L1, … , Lk순서로 진행된다. 그래서 최근에 입력된 데이터에 빠른 접근이 가능하다는 이점이 있지만, 하위 층에 어떤 데이터가 있는지 정확히 알 수 없기 때문에 오래된 데이터나 혹은 존재하지 않는 데이터 읽기에 대해서 좋은 성능을 내지 못한다는 단점이 있다.

2. 2 Tiered storage

Tiered storage는 지속성을 갖는 스토리지를 계층적인 구조로 사용하기 위한 방법이다. RocksDB에서는 LSM-tree를 사용하기 때문에 트리 속의 각 레벨을 계층으로 구분하여 데이터를 스토리지에 저장하는 Tiered storage를 적용할 수 있다. 특히 Tiered storage는 여러 스토리지 기기의 특징들을 고려하여, 가격 대비 성능 및 저장공간의 크기 면에서 효율적인 기기 사용을 하기 위해 고안된 구조이다. Tiered storage에 대한 그림은 (그림 2)에 표현되어 있다.



(그림2) Tiered storage

LSM-tree구조는 각 레벨마다 레벨의 크기, 쓰기 횟수, Compaction횟수가 다르다. 그러므로 기기마다의 특징들을 고려하고 LSM-tree의 레벨의 특징을 고려하여 스토리지를 적절하게 배치했을 때 가장 효율적으로 사용 가능하다.

3. 실험

3. 1 실험 환경

본 논문의 실험에서 지속성 있는 스토리지로 HDD, SSD, NVMe 를 사용하였다. 또한 WAL 을 사용하였는데, WAL 은 실험 성능 평가에 큰 영향을 주는 요소[8]이므로 결과의 일관성을 위해 SSD 를 2 개를 두고 하나의

SSD 는 Tiered storage 만 사용하고 나머지 하나의 SSD 는 WAL 을 위해 사용되었다. 실험환경은 <표 1>과 같다.

<표 1> 실험 환경

OS	CentOS 7.3.1611 (x86_64)
CPU	Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
RAM	64GB
NVME	SAMSUNG 960 PRO 1TB
SSD	SAMSUNG 850 PRO 256GB
SSD2	SAMSUNG 850 PRO 256GB
HDD	WD 1TB Caviar Blue WD10EZEX

3. 2 데이터 크기 변경

LSM-tree 는 병합 정렬을 통해 상위의 레벨로부터 하위의 레벨로 점차 트리가 확장된다. 그러므로 Tiered storage 의 성능을 측정하기 위해서는 설정한 스토리지 층에 적정 크기의 데이터가 들어와야 한다.

본 논문에서는 먼저 데이터 크기를 늘려가면서 실제로 성능 저하되기 시작하는 데이터 크기를 찾는 실험을 하였다. 실험은 NVMe(33G), SSD(200G)로 기기 층을 구성했다. 데이터 크기는 약 14GB, 28GB, 42GB, 그리고 56GB로 구성했다. 이 실험에서 데이터 크기에 따른 성능의 추이를 확인한 후, 특정 데이터 크기를 정하여 여러 기기층의 용량 설정에 대한 성능 비교 실험을 진행하였다.

3. 3 Tiered storage cases

Tiered storage 설정은 기기 층의 용량 설정을 여러가지 경우에 적용하여 실험을 진행하였다. RocksDB 의 LSM-tree 의 기본 L0 크기가 256MB 이고, Multiplier 가 10 이기 때문에 각 레벨의 크기를 고려하여 기기 층의 용량을 설정하였다.

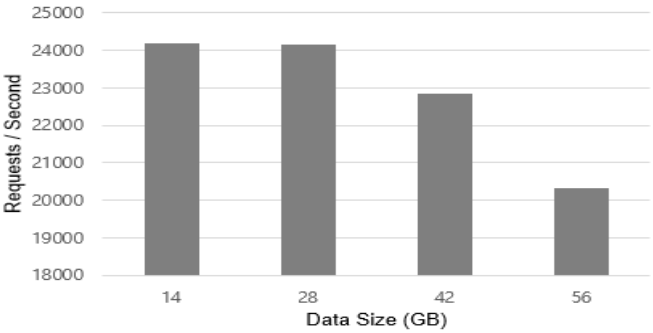
Tiered storage 에 대한 기기 층의 용량 설정은 HDD, SSD, NVMe 단일 층 설정을 기준으로 하여, 2 개의 기기 사이에서 좀 더 좋은 성능의 기기를 LSM-tree 의 상위 레벨에 점진적으로 설정하였다. 이를 통해 한 기기의 성능을 기준으로, 상위 층에 더 좋은 성능의 기기의 용량 설정 비중에 따라서 얼마나 성능에 영향을 줄 수 있는지 실험하였다.

4. 실험 결과 및 분석

모든 실험에 대한 성능 평가는 Linkbench 벤치마크 도구를 이용하였다. Linkbench 는 Facebook 에서 데이터베이스 성능 평가를 위해 만든 벤치마크 도구이다. 실험 결과는 초 당 Request 처리 개수를 나타낸다.

4. 1 데이터 크기 변경 결과 및 분석

실험 결과는 (그림 3),(표 2)와 같다.



(그림3) 데이터 크기에 따른 Linkbench성능

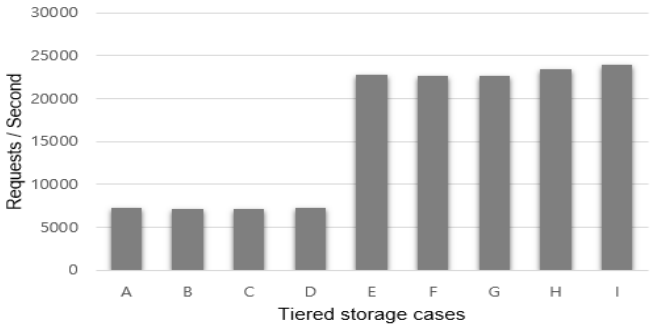
<표 2> 데이터 크기에 따른 Linkbench 성능

Data size (GB)	Requests/second
14	24,179
28	24,147
42	22,843
56	20,330

위 실험에서 약 30GB 까지의 데이터는 성능 차이에 큰 영향을 주지 않았지만, 그 이상의 데이터 크기를 처리할 때는 데이터 크기가 커짐에 따라 성능이 하락했다. 이는 LSM tree 의 하위 레벨이 생겨나고, 그곳에 데이터가 저장되면서 그에 따른 추가적인 쓰기 증폭이 일어나기 때문이다. 그래서 약 42GB 의 데이터 크기를 처리할 때 실제 성능에 영향을 줄 수 있을 것이라 판단하고, 이 데이터 크기를 바탕으로 기기 층의 용량 설정들의 성능 비교를 진행하였다.

4. 2 Tiered storage cases 결과 및 분석

약 42GB 의 데이터 크기로 한 실험 결과는 (그림 4), (표 3)과 같고, 그에 해당하는 Tiered storage 설정은 (표 4)와 같다.



(그림4) Tiered storage cases에 따른 Linkbench성능

<표 3> Tiered storage cases 에 따른 Linkbench 성능

Tiered storage cases	Requests/second
A	7,229
B	7,128
C	7,158
D	7,303
E	22,824
F	22,632
G	22,671
H	23,375
I	23,924

<표 4> Tiered storage cases (A ~ I)

	L_0	L_1	L_2	L_3
A	HDD			
B	SSD	HDD		
C	SSD		HDD	
D	SSD			HDD
E	SSD			
F	NVMe	SSD		
G	NVMe		SSD	
H	NVMe			SSD
I	NVMe			

실험 A ~ E 에서 단일 기기 HDD 설정에서부터 점진적으로 SSD 가 차지하는 LSM-tree 의 레벨을 늘렸을 때 성능 결과이다. 실험 B 와 C 에서 각각 {L0}, {L0, L1}이 SSD 에 포함되게 하고 나머지는 HDD 를 설정 했을 때, 실험 A 에서의 단일 HDD 보다 낮은 성능을 보였다. 실험 D 에서는 {L0, L1, L2}가 SSD 에 포함되게 하고 L3 만 HDD 를 설정 했을 때, 실험 A 의 단일 HDD 보다 더 좋은 성능을 보였다. 실험 D 와 E 의 비교는, 실험 E 에서 L3 를 SSD 에 포함되게 했을 때 실험 D 의 L3 가 HDD 에 포함되게 했을 때보다 훨씬 좋은 성능을 보였다.

실험 E ~ I 에서도 A ~ E 와 마찬가지로, SSD 와 NVMe 기기를 사용하여 NVMe 가 LSM-tree 상위 레벨에서 차지하는 비중을 늘렸을 때 같은 경향을 보였다. {L0} {L0, L1}이 NVMe 에 포함되고 나머지 레벨은 SSD 에 포함될 때 단일 SSD 보다 낮은 성능을 보였고, L2 까지 포함될 경우 단일 SSD 보다 빠른 성능을 보였다. 그리고 L3 를 NVMe 에 포함시킬 때의 성능이 SSD 에 포함시킬 때의 성능보다 좋았다.

RocksDB 의 최적화 관련 논문[4]에 따르면, L2 에서의 쓰기 횟수가 가장 많고 그 다음 L3 에서 쓰기가 많이 발생하며, L0 와 L1 의 쓰기 횟수는 상대적으로 매우 적다. 쓰기와 읽기가 많이 일어나는 Compaction 은 L1->L2 에서 가장 횟수가 많고, 그 다음 L2->L3 에서 횟수가 많다.

위 내용으로 보았을 때, L0 와 L1 의 쓰기 횟수가 상대적으로 매우 적기 때문에, L0 와 L1 이 더 좋은 성능의 기기에 포함되도록 했던 실험 B, C, F, G 는 오히려 다중 기기로의 접근으로부터 발생하는 부하로 인해 단일 기기보다 더 낮은 성능을 보였다. 그리고 L2 에 더 빠른 기기를 두었던 실험 D, H 는 다중 기기로의 접근에도 불구하고, 많은 Compaction 과 쓰기가 일어나는 층에 더 빠른 성능의 기기를 두었기 때문에 단일 기기보다 더 높은 성능을 보였다. 이는 Tiered storage 의 상위에 성능이 좋은 기기를 배치함으로써 성능의 이점을 얻고자 했던 목적과는 다른 경향을 보였다.

또한 실험 D 와 E 를 비교 했을 때, Tiered storage 에서 기기의 성능과의 관계는 하위층인 L3 에 지정된 기기의 성능과 연관되어 있음을 알 수 있다.

5. 결론

본 논문은 RocksDB 에서 Tiered storage 가 성능에 영향을 미치는 데이터 크기를 알아보았고, 이 데이터 크기를 기준으로 하여 단일 기기 층 설정과 Tiered storage 에서 다중 기기 층 설정에 따라서 여러 합리적인 경우로 나누어 실험했을 때의 성능을 비교하고 결과를 분석해보았다.

결론적으로 성능 결과에 가장 큰 영향을 미치는 요인은 가장 하위 층 L3 의 기기 성능이다. 상위 층에 빠른 기기를 지정하여 좋은 성능을 얻고자 했던 Tiered storage 의 목적 측면에서는 오히려 예상과 다른 결과를 보였다.

향후에는 실험에 쓰이는 데이터 크기를 늘려 LSM-tree 의 L4 까지 실험을 적용했을 때 위와 같은 실험 결과가 나오는지 확인하려 한다. 또한 RocksDB 의 LSM-tree 각 레벨 별 Compaction 및 쓰기 양이 실제 성능에 미치는 영향을 분석하고, 실제 Tiered storage 가 효율적으로 사용될 수 있는 최적화된 환경에 대해서 제시하려 한다.

참고문헌

[1] RocksDB, http://rocksdb.org

[2] RocksDB https://github.com/facebook/rocksdb/wiki

[3] Patrick O'Neil, et al. "The Log-Structured Merge-Tree." *Acta Informatica* 33. 1996

[4] Dong S, et al. "Optimizing Space Amplification in RocksDB." *CIDR*. 2017

[5] Lu, Lanyue, et al. "WiscKey: Separating Keys from Values in SSD-Conscious Storage." *FAST*. 2016.

[6] Linkbench, https://github.com/facebookarchive/linkbench

[7] TPC-C, http://tpc.org/tpcc/

[8] 성한승, 이두기, 박상현. (2017). RocksDB WAL Overhead 분석. 한국정보처리학회 학술발표논문집,, 857-860.

[9] Redis, https://redis.io

[10] Memcached, https://memcached.org/

[11] Cassandra, http://cassandra.apache.org/

[12] LevelDB, http://leveldb.org/