Code Review

RockDB 코드 리뷰

컴퓨터소프트웨어학부

김현정 박정호

CONTENTS

실험

실험 세팅

실험 방식

결과 및 분석

실험 세팅

Ubuntu Version : 18.04.2

write_buffer_size : 4MB
max_bytes_for_level_base : 8MB
max_bytes_for_level_multiplier : 2
level0_file_num_compaction_trigger : 2
level0_slowdown_writes_trigger : 3
level0_stop_writes_trigger : 4
num_levels : 5
Key size : 4Byte
Value Size : 1KB

실험 방식

새로운 DB 하나를 열어서, 무작위한 개수(10000 이하)의 랜덤 Key-Value Pair를 insert한다.

이때의 상태가 이후 트랜잭션이 사용할 snapshot이 된다.

이후 snapshot isolation을 사용하도록 지정한 트랜잭션을 시작하고, 일정 주기마다 랜덤한 Key를 가지고 Get 연산을 수행하게 한다. 또한 이 Transaction이 진행됨과 동시에 DB에 새로운 랜덤 Key-Value Pair를 Put하는 것으로, DB에 새로운 버전의 Key-Value Pair가 계속 쌓이도록 한다.

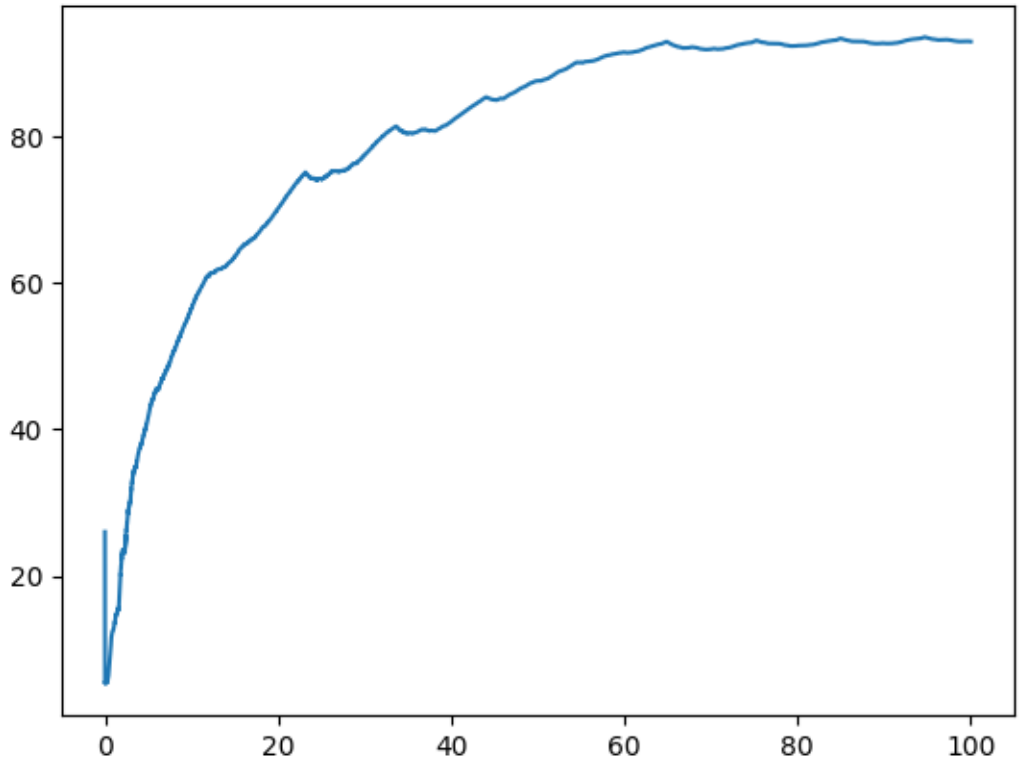
매회 Get 연산이 수행될 때마다 이제까지 수행된 Get 연산의 수행시간의 중앙값을 기록했으며, 이 기록된 값을 가지고 그래프를 그려서 Latency의 변화를 확인한다.

(간혹 Get 연산의 수행시간이 눈에 띄게 커지는 순간이 있는데, 이런 outlier를 배제하기 위해 평균 대신 중앙값을 사용한다. 이러한 현상의 원인은 아마도 Compaction일 것으로 추정한다.)

결과 및 분석

초반 20초 정도까지는 Latency가 가파르게 상승하는 것을 볼 수 있다. 이는 계속된 compaction으로 인해 Binary Search를 수행할 Level의 수가 늘어나기 때문이다.

Level의 개수는 최대 5개로 제한되어 있기 때문에, 5개에 도달한 시점부터는 Latency의 상승폭이 상당히 줄어든 것을 볼 수 있다. 이는 Binary Search의 시간 복잡도가 O(logN)이라 데이터의 증가에 비해 수행시간의 상승 시간이 그리 길지 않기 때문일 것이다.



시행착오

1. 평균을 이용한 관찰에서의 Outlier의 영향
   이유는 명확하지 않으나, 간혹 Get 연산의 결과가 일반적인 속도의 10배 정도로 느리게 동작하는 때가 있었다. 대체로 레벨이 늘어나는 도중에 그런 일이 발생하는 듯 했는데, Compaction 연산이 Read 연산과 겹치면 concurrency control 등의 이유로 Read 연산의 성능이 나빠지는 경우가 있는 듯 하다.

2. 방대한 통계값
   RocksDB가 유지하는 통계는 Perf Context, IO Stats Context, 이외에도 DB에서 자체적으로 유지하는 통계가 있었다. 이 중 Read 연산의 Latency를 나타내는 것이 무엇인지 몰라서 여러 번 실험을 반복했다.

   경우에 따라서는 Latency가 Linear하게 상승하기도 했고, 혹은 아주 크게 진동하는 형태를 띄기도 했다.

   그 외에도 각 Get Operation 단일의 Latency를 측정해야 하는지, 전체 Latency의 대푯값을 사용해야 하는지에 대해서도 확실하지 않아서 실험을 반복했어야 했다.

   현재는 DB를 열 때 사용하는 option의 statistic을 사용하고 있으며, DB_GET의 중앙값을 이용하고 있다.

Get 분석

Logical Flow

Logical Flow

탐색하는 순서는 다음과 같다.

WriteBatch -> Memtable -> SSTFile (레벨 순으로)

여기서 WriteBatch는 실제로 Insert, Write, Delete 등의 Modifying Operation이 적용되는 영역일 것으로 예상된다. Memtable은 외부 슬라이드에서 확인했던 Read Only Memtable일 것이고, 이 둘을 통칭해서 Memtable이라고 할 수 있을 것이다.

SSTFile은 FilePicker를 통해서 순차적으로 접근되고 있었으며, 이때 Memtable과 별개로 Cache가 이용되고 있음을 확인할 수 있었다. 다만, Snapshot을 빠르게 찾기 위한 별개의 처리는 찾을 수 없었고, 그저 레벨 별로 Key에 해당하는 Pair를 찾고, Sequential Number를 가지고 visibility를 확인하는 식으로 진행되고 있는 것을 볼 수 있었다.

향후 계획

이후 단계

이후 단계

1. Put (Insert, Write)의 동작 확인
   실제로 쓰기가 어떻게 수행되고, Sequential Number가 Record에 어떻게 배정되는지 등을 확인한다.
   또한 여기서 WriteBatch와 Memtable의 의미를 확실하게 확인할 수 있을 것이다.

2. 성능 개선을 위한 아이디어 브레인스토밍
   어느정도 내부 구조에 대한 이해를 마친 후, 성능 병목을 해결할 만한 방식을 연구, 논의해보아야 한다.

THANK YOU!

감사합니다!