Paper Review

재귀적 컴팩션 기법

컴퓨터소프트웨어학부

김현정 박정호

논문 – 김종빈, 손서희, 조현수, 정형수. LSM-tree 기반 Key-value 데이터베이스의 재귀적 컴팩션 기법.

CONTENTS

Background

LSM Tree 기반 시스템의 문제

LSM Tree 기반 시스템의 문제

LSM Tree 기반 시스템은 Compaction을 통해서 그 구조를 유지한다. 하지만 Compaction 연산에 쓰이는 파일의 개수에 따라서 쓰기 증폭 or 연산 부하 증가라는 문제가 발생한다.

우선 Compaction 에 쓰이는 파일 개수가 적다면, 다음 레벨로 내려가는 데이터량 자체가 적기 때문에, Compaction이 그 레벨에서 중단될 가능성이 높다. 이는 C_k를 C_{k+1}로 Compaction하는 과정에서 C_{k+1}의 Key-Value를 다시 C_{k+1}에 적게 된다는 말이다. 즉, 같은 데이터에 대해 여러 번 Disk Write가 발생하게 되며, 이는 쓰기 증폭 현상을 야기한다.

그렇다고 Compaction 에 쓰이는 파일 개수가 많아지면, Compaction 자체에 걸리는 시간이 오래 걸리기 때문에 추가적인 쓰기 연산이 지연될 수 있다.

구현

컴팩션

쓰기/읽기

Garbage Collection

컴팩션 연산

재귀적 컴팩션 연산의 기본 컨셉은 다음과 같다.

1. 상황에 따라 Compaction Granularity를 조정한다.
2. 멀티스레딩으로 Compaction을 수행한다.
3. 같은 레벨이 복수로 존재할 수 있다.

컴팩션의 대상이 된 레벨을 컴팩션 스레드가 통째로 올려서 정렬한 다음, 부모 노드의 컴팩션 플래그를 1 올린다. 이 값이 자식의 개수와 같아졌을 때 (즉 C_k의 경우 자식이 모두 K번 compaction이 발생했다면) 부모 노드가 모든 자식 노드의 데이터를 갖게 된다. (이는, Compaction 이전에는 부모 노드는 단순히 internal node이며, Key Value 데이터를 갖지 않는다는 것을 의미한다.) 이 과정은 가장 마지막으로 자식을 컴팩션한 스레드가 맡게 된다.

스레드의 개수는 parameter로 조절하고, 주어진 연산을 끝낸 스레드는 다시 스레드 풀에서 대기한다.

쓰기/읽기 연산

기본적으로 LSM Tree 기반 시스템의 쓰기 연산과 동일하다. 다만, 반복적인 쓰기 요청으로 인해 Compaction 과정이 빈번히 발생하고, 이로 인해 메모리 버퍼 및 C_0이 가득 차게 되더라도, 쓰기 지연이 발생하지 않는다. 이는 새로운 C_0을 두어서 이 파일에 쓰기 연산이 일어나기 때문에 가능하다.

즉, 이 시스템은 같은 레벨이 복수로 존재하는 것을 허용한다. 물론 이는 쓰기 연산이 빈번하게 발생할 경우 디스크의 부담을 키울 수 있으나, 이는 복수로 존재할 수 있는 레벨의 개수 제한을 두는 것으로 제어할 수 있다.

읽기 연산은 트리 구조로 형성된 레벨들을 순회하는 것으로 수행된다. 이때, 부모 노드가 컴팩션이 되어 있다면 자식 노드로 내려가지 않는다. 이미 자식 노드의 데이터를 모두 부모 노드가 가지고 있을 것이기 때문이다.

복수의 같은 레벨들을 관리하는 것은 읽기 연산의 부담을 키울 수 있다. 다만, 쓰기 연산이 그만큼 잦다는 것은 읽기 연산이 상대적으로 적다는 것이기에 성능 감소보다 쓰기 연산의 성능 상승이 더 클 것이라고 볼 수 있다.

Garbage Collection

컴팩션 연산이 발생하면 자식 노드의 정보를 모두 부모 노드가 가지기 때문에, 자식 노드가 가지는 파일은 더 이상 의미가 없다. 즉 Garbage Collection의 대상이 된다.

하지만 컴팩션 연산 도중 해당 자식 노드에 읽기 요청이 들어오는 경우가 있으므로, 컴팩션 연산이 종료됨과 동시에 그 파일을 삭제하는 것은 좋은 선택이 아니다. (LSM Tree에서 컴팩션 연산의 재료가 되는 자식 노드는 데이터의 변경이 발생할 수 없으므로, 읽기 연산이 동시에 이루어질 수 있다.)

따라서 각 파일에 Reference Count를 두어서 이 카운터가 0이 되었을 때 Garbage Collection의 대상이 되도록 한다.

향후 계획

읽어볼 논문

다음 단계

읽어볼 논문

1. 성균관대의 LSM Tree 관련 논문
   “BoLT: Barrier-optimized LSM-Tree”
   LSM Tree 의 Compaction에 대한 논문인 듯

2. Thomas Lively의 LSM Tree 관련 논문
   “Splaying Log-Structured Merge-Trees”
   LSM Tree 기반 시스템의 읽기 성능에 대한 논문인 듯

다음 단계

RocksDB 동작 분석
GDB, cscope를 통해 실제 function flow 확인하기

perf 등을 통해 성능 병목 확인하기

참고할 만한 슬라이드
https://www.slideshare.net/meeeejin/rocksdb-detail

https://www.slideshare.net/meeeejin/rocksdb-compaction

https://www.slideshare.net/HiveData/tech-talk-rocksdb-slides-by-dhruba-borthakur-haobo-xu-of-facebook

THANK YOU!

감사합니다!