Paper Review
LSM Tree

컴퓨터소프트웨어학부

김현정 박정호

논문 – P. O'neil et al., The Log-Structured Merge-Tree (LSM-Tree)

CONTENTS

Background

Five Minute Rule

B Tree의 약점

LSM Tree 개요

Five Minute Rule

“디스크에 저장된 페이지가 캐시에 저장되었을 때, 그 페이지를 5분 안에 다시 참조할 경우 추가적인 disk IO 없이 접근할 수 있다.”

위에 따르면, access frequency가 5분을 넘길 경우 캐시에 디스크 페이지를 저장하는 의미가 사실상 없어진다. 5분이 지나가면 페이지가 이미 evict되었을 것이기 때문이다.

여기서 “5분”이라는 시간은 메모리와 디스크 사이의 가격 차이 등에 영향을 받으며 현재 이 법칙을 검증할 경우 다른 값이 도출될 수 있다.

다만 이 논문에서는 기술적 문제 등으로 인해 이 법칙의 시간을 1분으로 두었다.

B-Tree의 약점

Five Minute Rule에 따랐을 때 disk page의 수가 많아질 경우, cache 의 효과가 약해질 수 있다.

논문의 example 1.2에 따랐을 때 1초에 1000개씩, 하루 8시간 20일동안 row를 쌓았을 경우 약 2.3M 개의 페이지가 생기게 되고(낭비되는 공간이 없다는 가정 하에), 1초에 1000개의 스레드가 매 초마다 무작위의 페이지를 참조한다고 하면 같은 페이지를 다시 참조하는데 평균적으로 2300초가 걸리게 된다. 이는 5분(논문 상에서 1분)을 훌쩍 넘기는 시간으로, 사실상 매초, 모든 스레드가 2번씩 disk IO를 발생시킨다는 것으로, 이론적으론 cache hit rate가 0이 된다.

또한 B Tree Index에서 이루어지는 disk IO는 random access 이므로, 매 IO마다 disk arm의 동작을 수반하기에 성능 저하가 발생한다.

LSM Tree의 개요

메모리와 디스크 양쪽을 모두 이용하는 자료 구조.
적은 disk IO cost를 통한 성능 개선을 위한 자료 구조이다.
기본적으로 disk IO를 늦추고, multi-block IO를 사용하는 식으로 cost를 낮춘다.

기본적으로 메모리 위에 C_0을 두고 모든 modifying operation 을 여기에서 수행한다. 여기서 C_0이 특정 수준 이상으로 커졌을 경우 디스크의 C_1로 flush하게 된다. 이때 C_1과 C_0의 내용물을 합치는 과정이 수행되며 merge sort와 유사한 알고리즘을 사용한다.

flush 까지 어느정도 delay가 있기에 sequential log를 유지한다.

Insert-heavy 한 환경에 적합한 구조이다.

기본 알고리즘

Rolling Merge

Basic Operation

Concurrency Control

Recovery

Rolling Merge

기본적으로 C_0과 C_1이 정렬된 상태이기에 단순히 merge 하는 식으로 진행된다. 또한 비동기화된 과정으로 진행된다. 즉, rolling merge가 수행되는 중간에 다른 TRX가 데이터에 접근할 수 있다.

Emptying block – 기존에 C_1에 있던 multi-page block, merge의 대상이다.

Filling block – 새로 C_1에 적히는 multi-page block, C_0과의 merge 결과물이다.

꽉 찬 filling block은 C_1 파일의 새로운 부분에 적히게 된다.(단, checkpoint 에 적히게 된 multi-page block 을 덮어쓸 수도 있다.) 즉, 기존 부분도 디스크에 남게 된다. 이는 후에 recovery 를 위한 것으로 merge 가 모두 끝나게 될 경우 삭제된다. 또한 디스크에 적힌 filling block은 IO를 줄이기 위해 일정 시간만큼은 buffer 에 머무른다.

또한 꽉 차지 않은 filling block의 경우 디스크에 적히지 않고 buffer 에 남게 되는데, 이때 변경된 directory 정보(parent node)를 저장한 상태로 buffer 에 남게 된다.

Basic Operation - Insertion

C_0의 형태는 굳이 B Tree가 아니어도 된다. C_0 의 노드 구조가 disk로 flush되지 않기 때문에 굳이 노드 사이즈를 disk page 단위로 맞출 필요가 없기 때문이다. 따라서 AVL이나 2-3 Tree 같은 구조를 사용해도 무방하다. 다만 C_1은 B Tree의 형태를 가진다.

기본적인 insertion 은 모두 C_0에서 일어나고, 디스크에 반영되는 것은 rolling merge 에서 수행된다. 모든 disk IO는 multi page block에서 발생하기 때문에 seek time과 rotational latency 가 감소하게 된다.

Directory 정보가 포함된 multi-page block은 아래의 경우에만 disk의 새로운 페이지에 적히게 된다.

1. multi-page block이 다 찼을 때
2. Root node가 split되었을 때
3. Checkpoint가 수행되었을 때

Basic Operation – Delete / Search

Delete

C_0 에 Delete Node Entry를 추가하는 것으로 기본적인 연산을 대체한다. 후에 rolling merge 단계에서 Delete Node Entry가 있을 경우 관련 entry 를 삭제하는 것으로 실제 삭제 연산을 수행한다. 탐색 연산은 Delete Node Entry를 통해서 필터링된다.

Search

기본적으로 C_0 먼저 탐색한 다음 C_1을 탐색하는데, 이 과정에서 기존 B Tree보다 CPU 부담이 클 수 있다.

C_i 에 데이터가 머무르는 최소 시간 T_i 를 두는 것으로 탐색을 최적화할 수 있다. (데이터를 찾기 위해 방문하는 component의 개수를 줄이기 위해)

Concurrency Control

아래의 3가지 원칙을 지킨다.

1. Find 연산은 rolling merge가 수행되고 있는 disk의 노드를 접근해서는 안된다.
2. Insert/Find 연산은 rolling merge가 수행되고 있는 C_0의 노드를 접근해서는 안된다.
3. 상위 노드가 rolling merge가 발생하고 있을 경우, 하위 노드의 rolling merge cursor 가 멈춰야 할 수 있다.

기본적으로 node 단위로 lock 을 건다. Locking protocol은 C_0의 구조에 따라 다르다.

Disk component 간의 rolling merge에서는 각 레벨의 emptying/filling block 총 4개의 페이지에 모두 write lock을 건다.

Recovery

디스크도 이용하는 구조이지만, rolling merge가 되기 전에 record 가 C_0에 남아있을 수 있으므로 sequential log file을 사용한다.

또한, checkpoint에는 C_0을 디스크의 특정 위치에 flush하고, buffer의 모든 dirty page들을 flush한다. 이 때, rolling merge 후 남은 entries(한 페이지에 꽉 차지 않아서 아직 디스크로 flush되지 않은 데이터들)도 같이 flush되는데, 이는 나중에 rolling merge 가 일어나면서 다시 덮어쓰여서 LSM Tree의 구조를 유지한다.

향후 계획

읽어볼 논문

다음 단계

읽어볼 논문

1. FaceBook의 MyRocks 논문
   “MyRocks: LSM-Tree Database Storage Engine Serving Facebook's Social Graph”
   RocksDB의 기본 동작 이해에 필요, LSM Tree가 실제로 어떻게 적용되고 있는지 확인하기

2. HYU-SCSLAB의 rolling merge(compaction) 관련 논문
   “LSM-tree 기반 Key-value 데이터베이스의 재귀적 컴팩션 기법”
   기존 LSM Tree의 성능 문제의 해결책 중 하나, 이를 따라가지는 않아야 하지만, 참고할 가치가 있음

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!