Performance Analysis
RockDB 분석

컴퓨터소프트웨어학부
김현정 박정호

CONTENTS

Perf

사용한 옵션

분석 결과

사용한 옵션

...

분석 결과

...

Call Graph

실험 방식

Callgrind 결과

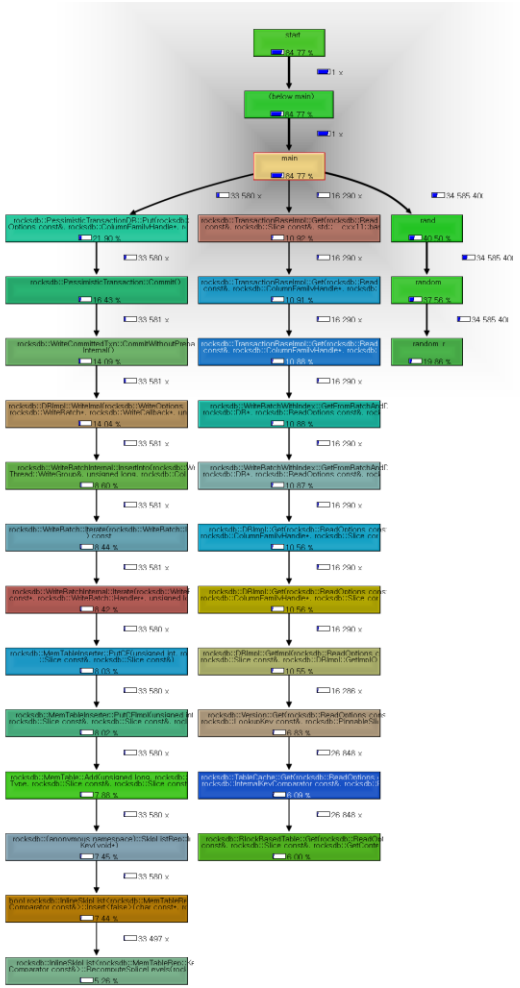
실험 방식

Call Graph를 구하기 위해 valrind에 내장된 툴인 callgrind를 사용했다.

callgrind의 결과는 단순한 텍스트이기 때문에 이를 시각화 하기 위해서 Kcachegrind를 사용했다.

사용한 결과는 오른쪽과 같다. 왼쪽이 PUT, 가운데가 GET, 오른쪽이 rand (무작위의 key-value 를 삽입/탐색하기 위해 사용)이다.

일정 이상 branch 가 나눠질 경우 축약된 상태로 그려지며, 각 블록을 클릭하면 해당 블록에서부터 시작되는 call graph 를 다시 그려준다. 이를 이용해서 좀더 자세한 call graph를 확인할 수 있다.

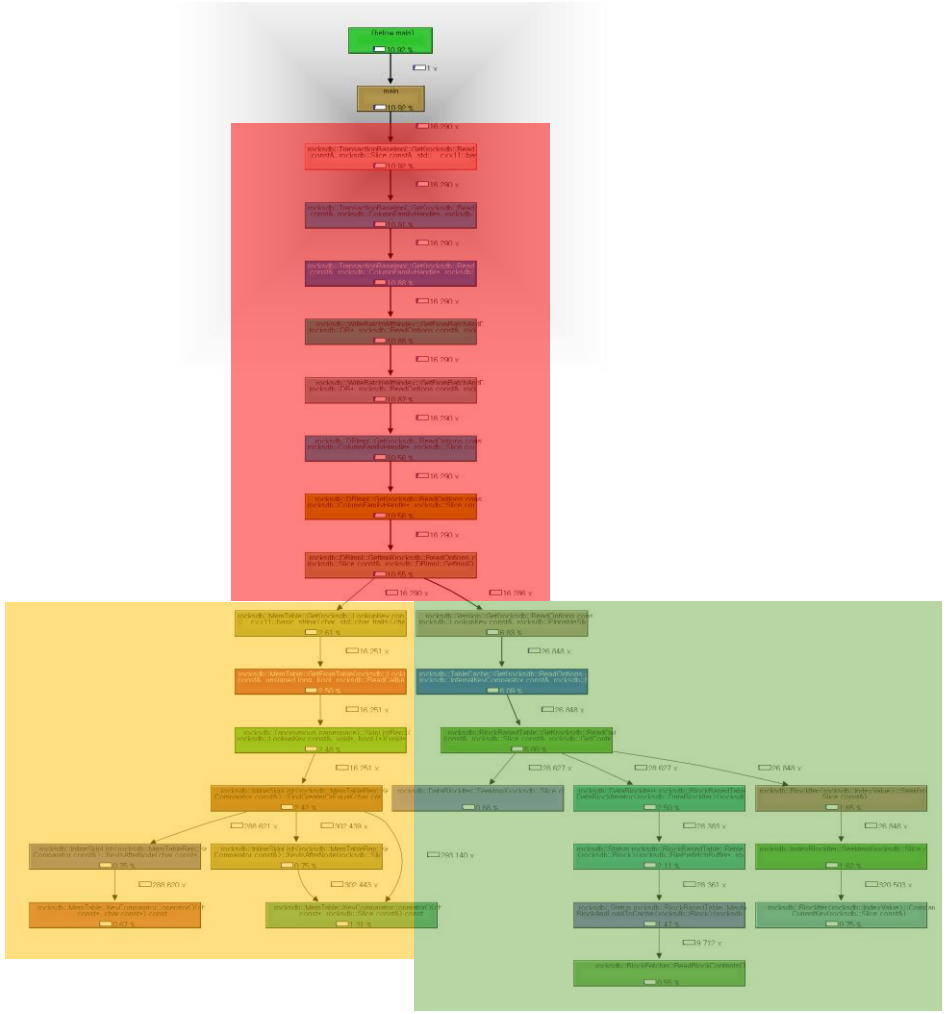


Callgrind 결과 (1)

Get의 전체적인 Call Graph 는 다음과 같다. Kcachegrind 가 아래쪽 일부 branch를 생략하고 보여준 상태이다.

크게 나눠서 보자면, 붉은 영역은 wrapper 함수들이 포함된 영역이다. 이 과정에서 WriteBatch 에서의 탐색이 수행된다.

그 아래의 노란 영역은 memtable 에서의 탐색이 수행되는 영역이고, 초록색 영역은 version 에서의 탐색, 즉 SST File 에서의 탐색이 수행되는 영역이다. 이 두 영역에는 생략된 path가 있으므로, 다음 페이지에서 더 자세한 call path graph를 보도록 하겠다.

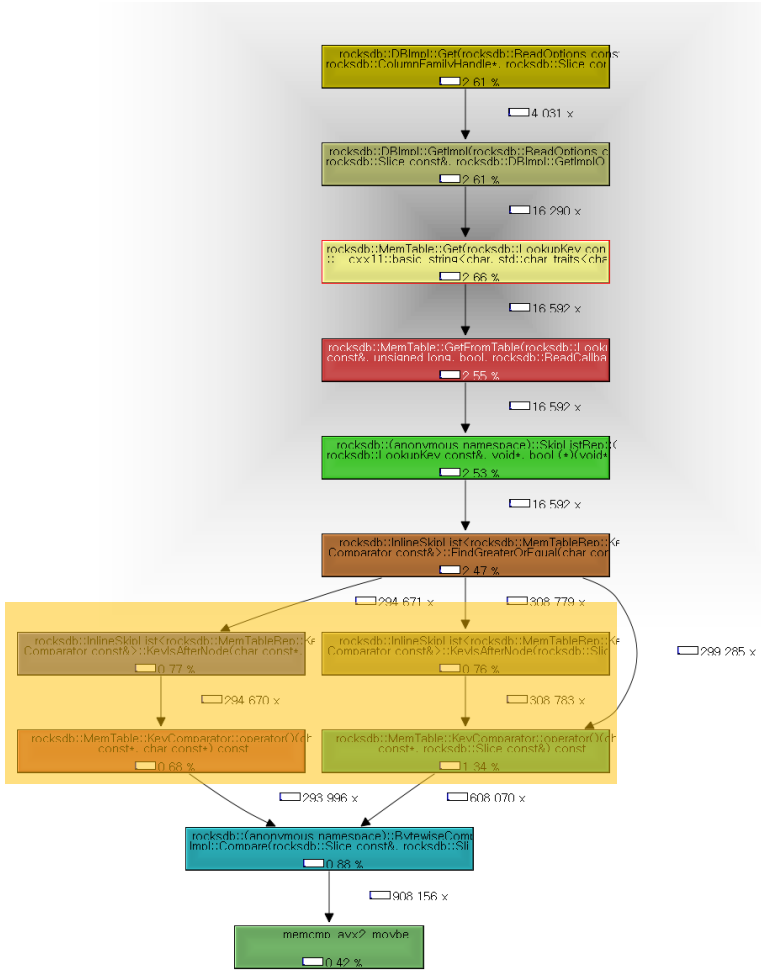


Callgrind 결과 (2)

memtable 에서의 탐색 path 의 call graph이다.

Skip list를 순회하면서 찾고자 하는 key 이상의 key를 갖는 노드를 찾고, 찾아낸 key와 target을 비교하는 과정을 갖고 있다.

노란색 영역에 해당하는 부분은 시스템의 무결성을 확인하는 assert 문에 사용된 것으로, 리스트가 정렬된 상태이고, 이미 target을 지나치지 않았는지 확인하는 과정이었다.

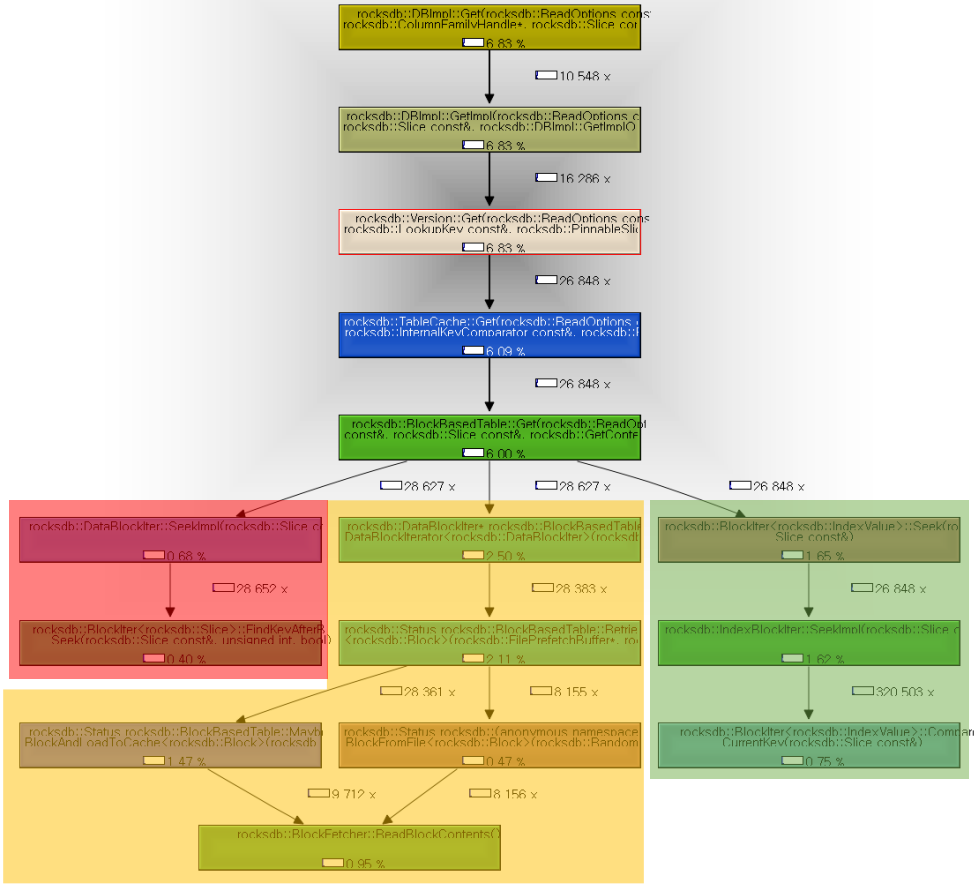


Callgrind 결과 (3)

SST File 에서의 탐색 path 의 call graph이다.

노란색 영역은 SST File 혹은 cache 에서 block 을 읽어오는 과정, 초록색은 인덱스 블록에서 데이터 블록을 찾아오는 과정, 붉은색 영역은 이렇게 찾아온 데이터 블록에서 원하는 key-value 를 탐색하는 과정이었다. (callgrind의 그래프가 순차적인 형태를 띄지는 않는 것 같다.)

기본적으로 bloom filter 를 사용해서 의미없는 탐색을 줄이고, 인덱스 블록을 사용해서 데이터 블록을 읽어와서 탐색을 진행하는 방식을 사용하고 있었다.



To Do

해결해야 할 문제

질문

해결해야 할 문제 및 질문

1. perf 를 사용해서 병목이 되는 부분을 찾으려 하는데, 현재 사용하려는 방식이 맞는 것인지 궁금하다.

THANK YOU!

감사합니다!