Code Review

RockDB 코드 리뷰

컴퓨터소프트웨어학부

김현정 박정호

CONTENTS

분석

실험 세팅

결과 분석

실험 세팅

지난 실험에서 쓰인 코드를 그대로 사용하여 분석을 진행하였다.

CPU cycle을 측정하는 것이 목표이므로 cpu cycle을 저장하는 TSC(time stamp counter)의 값을 eax와 edx레지스터를 통해 받아오는 rdtsc라는 함수를 사용하였다.

```
#if defined(OS_WIN)
std::string kDBPath = "C:\\Windows\\TEMP\\rocksdb_read_bottleneck_breakdown_test";
/*#include <intrin.h>
uint64_t rdtsc(){
    return __rdtsc();
}*/
#else
std::string kDBPath = "/tmp/rocksdb_read_bottleneck_breakdown_test";

uint64_t rdtsc(){
    unsigned int lo, hi;
    __asm__ __volatile__ ("rdtsc" : "=a" (lo), "=d" (hi));
    return ((uint64_t)hi << 32) | lo;
}

#endif
```

실험 세팅

- 컴파일 과정에서 앞전에 추가한 함수를 인식하지 못하고 충돌하는 문제가 계속 발생하였다.
  Rocksdb코드에 같은 이름을 사용하는 함수가 구현되어 있는지를 보니,
  Hardware.h파일에서 hardware_timestamp라는 함수내부에서 rdtsc를 사용하고 있음을 확인했다.
- Test code에서는 hardware_timestamp() 파일을 인식하지 못해서 앞전에 직접 선언한 rdtsc()함수로 Get함수의 cycle을 측정하고 Get함수 내부에서는 hardware_timestamp()로 측정을 시도했다.

```
inline std::uint64_t hardware_timestamp() {
#if _MSC_VER && (defined(_M_IX86) || defined(_M_X64))
  return __rdtsc();
#elif __GNUC__ && (__i386__ || FOLLY_X64)
  return __builtin_ia32_rdtsc();
#else
  // use steady_clock::now() as an approximation for the timestamp counter on
  // non-x86 systems
  return std::chrono::steady_clock::now().time_since_epoch().count();
#endif
}
```

결과 및 분석

- 실험도중에 컴파일이 되지 않는 문제가 연이어 발생하면서 많은 부분을 측정하지는 못하였다.
  코드 수정을 시도하지 않은 파일에서 'unable to open', 'undefined function' 등의 문제가 발생하여 제대로 된 분석을 시도하지 못했다.

- Get함수의 cycle을 측정해본 결과 get cycle이 보통과 다르게 10배 이상으로 길어지는 경우를 발견하고 확인해보니 sv->imm->Get()을 호출한 case였다.
- 보통 5~6자리의 cycle이 소요되고 6자리의 경우에도 가장 큰 자릿수가 3을 넘지 않았다.
  Sv->imm->Get을 시도한 경우에 사이클이 7자리로 증폭되었음을 확인했다.
- 이경우 ssd파일로 flush를 시도하면서 Merge가 연관될 가능성을 추측해 볼 수 있다.

```
----
GET:32124
----
GET:31234
----
GET:32610
----
sv-imm->Get():12942
GET:1211262
----
GET:35079
----
GET:32716
----
GET:34327
----
GET:25435
----
```

Merge

Merge의 종류 및 동작

Merge의 종류

1. Full merge
   중복된 데이터, 혹은 기존에 존재하지 않았던 데이터가 추가되었을때 처리를 담당한다.
   (아마도 tombston이나, 특정 Key에 대한 write가 쌓였을 때 이를 하나로 축약하는 과정인 듯하다.
   일종의 Garbage collection 기능.)

2. Partial merge
   두개의 대상을 합치는 일반적으로 Merge라 일컫는 과정을 말한다.

Merge의 동작

1. DB::Merge, DB::Put은 merge를 강제하지 않는다. 단지 merge를 해야 한다는 조건을 만들어줄 뿐, 실제 merge 동작은 Get의 실행이나, system이 compaction을 수행할 때 발생한다.
   즉, 실제로 merge가 동작할 때 merge operand가 하나일 것이란 보장이 없으며, merge operator는 이 merge operand를 하나씩 순차적으로 처리한다.

2. 단 위의 언급은 full merge에 해당하며, partial merge의 경우 가능하다면 즉시 수행된다.

3. Full merge가 false를 return하는 것은 데이터에 잘못된 값이 들어갔다는 의미이다.

4. 전체적인 흐름은 다음과 같다.
   1. Merge operan가 발생할 경우, 이것에 partial merge적용을 시도하고, 불가능할 경우, 해당 operan를 쌓아 둔다.
   2. 단, 해당 operand에 Put/Delete관련 값이 있을 경우, Full merge를 시도한다.

To do

Questions. (앞으로 해결해야 할 문제들)

Questions.

1. 특정 경우에 대한 코드를 자세히 관찰할 필요가 있는가.
    1. Sv->imm->Get()이 호출되었을 때, 사이클이 크게 증폭한 경우를 자세히 볼 가치가 있는가.
    (100초동안의 실험시간동안 10번도 채 나오지 않음.)
    2. 디버깅 과정에서 merge하는 코드를 확인하지 못했는데 이 함수를 위키나 text검색을 통해 찾아내어 측정에 포함시킬 필요가 있을까.

2. 컴파일을 할 때마다 make를 처음부터 다시 해야해서 1시간 정도가 소요되는데 더 빠르고 효율적으로 할 수 있는 방법이 있을까.

THANK YOU!

감사합니다!