LSM Tree 键值存储系统设计文档

1. 项目概况

（1）项目名称：LSM Tree 键值存储系统

（2）项目功能：实现一个键值存储系统，该键值存储系统将支持以下基本操作。

PUT(K, V)设置键 K 的值为 V。

GET(K)读取键 K 的值。

DELETE(K)删除键 K 及其值。 其中 K 是 64 位有符号整数，V 为字符串。

2. 代码模块

（1） IndexNode.h :存储索引节点，其中记录了键值对在 sstable 中的偏移量，所处文件目录，sstable 在文件目录中的标号，时间戳和存储的 string 的长度。

（2） QuadlistNode.h：存储跳表节点，是一个四联表节点类型。

（3） skiplist.h、skiplist.cpp 实现了跳表, bottomFirst()到 bottomLast()之间是链表存储，在两者之间（包括头尾）按 key 从小到大存储了在跳表中的键值对。调用函数 int size()可以得到跳表中存储的键值对的数量，int level()可以得到当前跳表的高度，int space()可以得到当前跳表中存储的键值对占据的空间，一但 2M，就出发磁盘操作。

（4） TableNode.h：保存了要写入 sstable 的内容，即时间戳，key 和 value。

（5） ScanNode.h：保存了保存的 sstable 中 key 的范围

（6） kvstore.h、kvstore.cpp：功能的主要实现模块

① list 为 SkipList 的实例化对象，实现了链表。

② vector<int>level 保存了每一个文件目录中有几个 sstable

③ map<uint_64,IndexNode> index 存储数据索引在内存中

④ ScanNode scan[15][10000] 存放每个 sstable 的 key 的范围

⑤ int maxfile(int level) //输入文件目录 level，获得该层最多可以容纳的文件数量

⑥ uint64_t gettime(); //获得时间戳

⑦ void remdup(vector<TableNode> &vec); //vec 中有相同 key 的 node，删除早的.前提是 vec 有序.

⑧ void handleDel(vector<TableNode> &vec); //处理 lazy 的删除.

⑨ void compaction(vector<int> &level, map<uint64_t, IndexNode> &index);完成 sstable 的归并和重新写会

3. 功能实现

(1) put：首先在跳表中进行插入，一但跳表的 space 大于 1048576，即 2M 时，就触发磁盘操作。将跳表中的键值对按照时间戳、key、value 的顺序写入 sstable，并且改名。之后使用 compaction 来进行调整。

(2) get：首先在跳表里查找，如果找不到的话到索引中寻找，在索引里找到了的话就读出来对应的 IndexNode，之后再读出对应的目录，对应的文件编号，根据 offset 去文件中寻找。

(3) del：进行懒删除操作。通过插入一个特殊的字符串“~”来表示删除。同时在索引中直接删除该 key 对应的 IndexNode，防止出现已经删除了该 key 对应的 IndexNode 但是还没来得及调整。

(4) 索引：利用 map<uint64_t,IndexNode> index 来实现。Map 本身的实现原理就是红黑树，所以查找索引时直接用 map 提供的查找，就不需要二分查找了。

(5) Scan（范围）：利用一个比较大的二维数组来记录每一个 sstable 中 key 的范围。其中每一行代表每一层目录。每一个 node 占的空间不超过 20 字节，就算开了 15*10000 个空间，占用内存空间也不超过 30M，可以接受.

(6) Compaction：这是这个 project 的核心，也是我最后实现中最可能出现问题的地方。实现思路是：

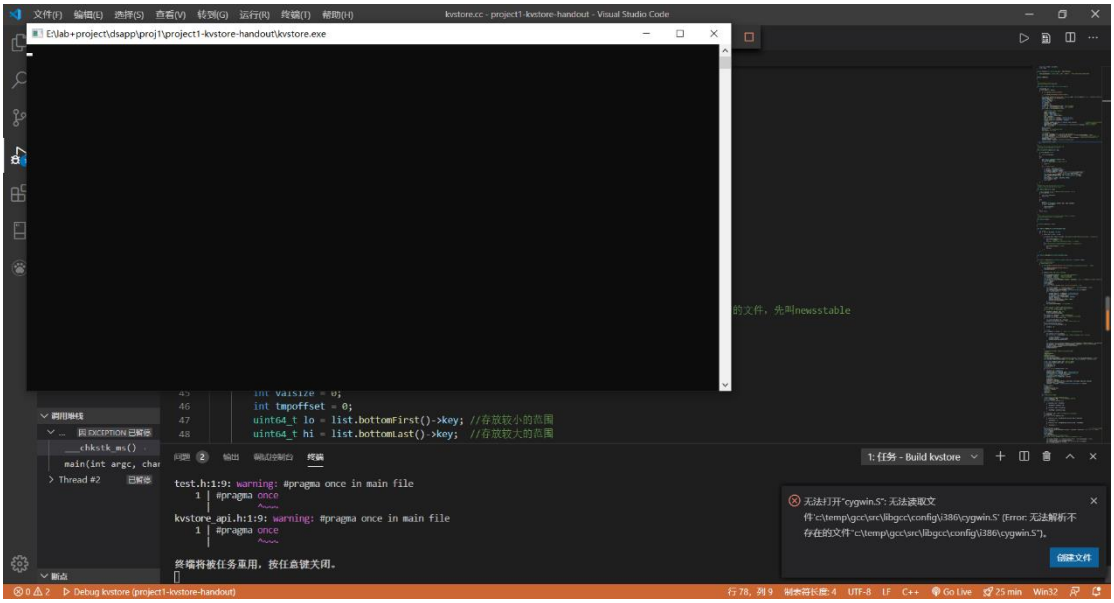
先检查是不是 level0 层出现了溢出。如果是的话，就把 level0 中所有的 sstable 取出，读到内存中，同时找到 level1 中与之范围有重合的 sstable，也取出读到内存里。取出 sstable 时，同时在磁盘中删除。之后对取出的键值对进行归并排序。得到一个较大的存储键值对的 vector，对 vector 处理重复和删除问题后，进行划分。划分策略是假设在 level1 层取出了 n 个 stable，就划分成 n+1 个 sstable。然后把 level1 层范围不相交的后边的 sstable 更名，腾出空间。然后将新划分好的 sstable 写进磁盘。

如果不是 level0 层发生溢出，就依次检查已经存在的所有文件目录。设文件目录的标号为 i，一但第 i 层文件目录中的文件数量大于 2^(i+1),就取出当前文件目录中最后三个文件，找到下一层文件中范围相交的文件，多个文件归并。具体操作与 level0 层发生溢出类似。之后的结果就是每次 compaction 都会使得文件总数-2。

4. 功能测试

我实现了两个版本。第一个版本是纯内存操作，只有一个跳表实现了键值对的存储删除查询等操作，也完成了正确性测试。持久性测试将数据量改小后也能通过。但是一但给出过大的数据，内存就不够用了，会崩掉。

第二个版本是带有 sstable 磁盘操作的实现。但是这个版本经过一个多星期的调试仍然无法使用，一开始是一直报 segment fault，之后经过修改，segment fault 的错误消除了，但是 IDEA 一直报以下的问题，我经过很长时间的调试仍然找不到原因



因为内存版本能够通过测试，所以我的 mingw 应该是没有问题的，出现这样的问题我也觉得很奇怪。我个人认为还是 compaction 的时候出现了内存上的问题。但是 IDEA 没有报 segment fault，让我无从下手。现在这个问题我无论在哪个地方设置断点，都没有用，

一但开始调试就会报出“无法打开 cygwin.S”的错误。一直到今天的 deadline，也没有找到最终的原因。所以希望助教学长，如果发现了我的错误所在，可以不吝赐教。

5. 优化方向

这个 project 对我来说，最大的优化方向应该是要先能正确的跑出来。这一点，就算 deadline 已经过去了，我也会继续探索的，希望可以把 Bug 弄好。

另外的优化方向应该还有应该控制内存的使用。我的索引和范围都保存在了内存中。其中范围用了一个静态数组来存储，一次分配了大约 30M 的空间，虽然对于几 G 的内存来说这个空间不算大，但是静态分配绝对不是最优解。一但测试数据量过大还是会崩掉。这一点需要优化。

Compaction 尤其是归并排序的地方也需要进一步优化。我的归并排序实际很低效，要不断寻找最大值最小值。如果要归并 m 个 sstable，平均每个 sstable 中含有 n 个键值对，时间复杂度应该是 O（m*n），这显然是低效率的。

再一个优化方向是就是应该把索引写入 sstable，但是我图方便就只把索引写到了内存中，而在磁盘中没有存储。这就造成了一但出现了断电等情况，索引就消失了。就无法再进行查询了。

6. 致谢

谢谢助教学长的支持
感谢 cppreference
感谢 CSDN、博客园