设计文档

**前言**

这次的proj是我有史以来做的最提心吊胆的一次，不仅因为刚开学时的icslab和web迭代，也因为这学期在家上课使得我有所懈怠，直到原先ddl 4.11前一周的周日才开始写，最终用了四天时间写完整个部分，又用了四天时间debug（期间由于其他课程的作业耽误了两到三天，导致proj写与debug不连贯，带来不少麻烦），才在ddl前一天完成。

**设计部分**

LSM-tree的设计思路跟文档中给出的一致，数据刚开始写入时保存在用跳表实现的memtable中，当memtable转换成sstable的大小达到2M后写入磁盘的sstable中。磁盘分为多层，每层能够存储的sstable的最大容量为等比数列递增，当某层中sstable超过该层最大容量时，进行合并（compaction）。LSM-tree主要有四个功能函数：put、get、del、reset，分别是存入数据，查询数据，删除数据和重置LSM-tree。下面重点介绍我设计的数据的结构和功能的实现。

memtable（skiplist）：声明在SkipList.h中的一个类，除了一般跳表中包含的成员变量之外，他还包含了numOfNode，sizeOfKVInSs，mem\_searchtable三个变量，分别用来储存跳表中已有的数据的数量、跳表转换为sstable后的大小、以及将跳表中的数据写成含有key、value、offset的列表暂时储存，便于转化为索引区和向文件中写入

sstable：声明在sstable.h中的一个类，有filename、id、searchtable三个成员变量，分别记录sstable文件名、文件名的数字部分（文件以0.txt、1.txt等依次命名）以及文件的索引区（只含key和offset）

tablenode.h：一个声明了searchtablenode和mem\_searchtablenode的头文件，顾名思义，前者是memtable中mem\_searchtable的节点，一个节点包含三个变量，后者是sstable中searchtable的节点，一个节点只包含两个变量（无value）

kvstore.h：五个成员变量dir、memtable、disk、sort\_table、sort\_disk。dir记录sstable储存的路径名，此porj中为./data。memtable为skiplist类的指针。disk为sstable类的二维vector，我认为这样可以很好的表明文件储存时的层次结构。sort\_table在合并时存入所有需要合并的mem\_searchtablenode，便于排序、输出到sstable、控制key的单一性等。sort\_disk在合并时存入所有需要合并的sstable类对象，然后将其转化为一个个mem\_searchtablenode读入sort\_table中。

值得一提的是，我在思考后认为每个kv-pair的时间戳没有太大必要，甚至连每个sstable都不需要加上时间戳，只需保证旧的同key kv-pair永远不会先于新pair被读到即可。因此我并没有加入时间戳，具体控制数据新旧与单一性的操作在接下来会详细讲述。

put操作：首先判断memtable中是否已经存在含有这个key的kv-pair，若有直接删除。随后插入kv-pair，同时判断是否达到写入sstable的临界。若达到临界，写入sstable并存入L0层，同时遍历0至disk.size()-1层，是否有文件数目溢出。若有，确认溢出层i中需要合并的文件数量，从disk[0]开始确认文件（因为后写入的文件新，所以先合并排在前面的旧文件），根据i层文件中key范围确定i+1层中需要合并的文件，然后先将i+1层中文件push\_back进sort\_disk，再将i中文件push\_back进sort\_disk，这样确保下面操作中删除的同key的kv-pair是旧的。随后，从后往前依次读出sort\_disk中的sstable，读出后根据索引区searchtable将文件中的kv-pair读取出来，在将其压进sort\_table中前，先判断sort\_table中是否含有同key的pair，若有则不压入。当sort\_disk中所有sstable的kv-pair均已入列后，使用list自带的sort函数进行排序（我万万没想到要使用的是多个list的归并排序，我只用了一个list自身的归并），然后依次出列，每达到2M便生成一个新的sstable，从disk[i]层push\_back，调用exporting函数导出到disk[i+1]层（由于我每次取文件都是使用disk[i][0]，disk[i][1]，disk[i][disk[i].size()-1]等的方式，所以其实文件名我并不关心，但是我又需要生成不一样的文件名，因此我选择记录每个sstable的id，然后用id命名，这样会导致运行多次后的id达到一个很大的数，但还好都没有达到LONG\_MAX。我认为我肯定可以找出与贪吃蛇proj中蛇身周期性变色类似的命名方法，可惜时间不足，以后的proj我一定会尽早开始......）

get操作：先检查memtable中是否含有该key，若有则直接取出，否则从L0层开始，从后(disk[i][disk[i].size()-1])往前(disk[i][0])遍历sstable的索引区，找到相同key后不再继续遍历(后面即使有也是旧的)

del操作：先检查memtable中是否含有该key，若有先取出value——value为“”则代表已被删除，直接返回false，否则删除kv-pair并返回true。若memtable中不存在，则与get操作类似遍历sstable，若无此key或value为“”则代表已删除，返回false，若value不为空则put(key,“”)。

reset操作：调用每次写入sstable后都会调用的memtable->reset()重置跳表与memtable自带的参数，同时清空sort\_table、sort\_disk，并remove\_all(dir)以及create(dir/L0)

KVStore构造函数：先检查dir/L0是否存在，若存在则代表磁盘中已有文件，于是从L0开始依次遍历每层下的所有文件（使用struct \_finddata\_t）并依靠filename构造每个sstable对象。若dir/L0不存在，则新建dir/L0目录并结束构造

**测试部分**

开发环境：

Microsoft VS Code 1.44.2.0

测试环境：

机型：Surface Pro 5

处理器：Intel Core i7-7660U

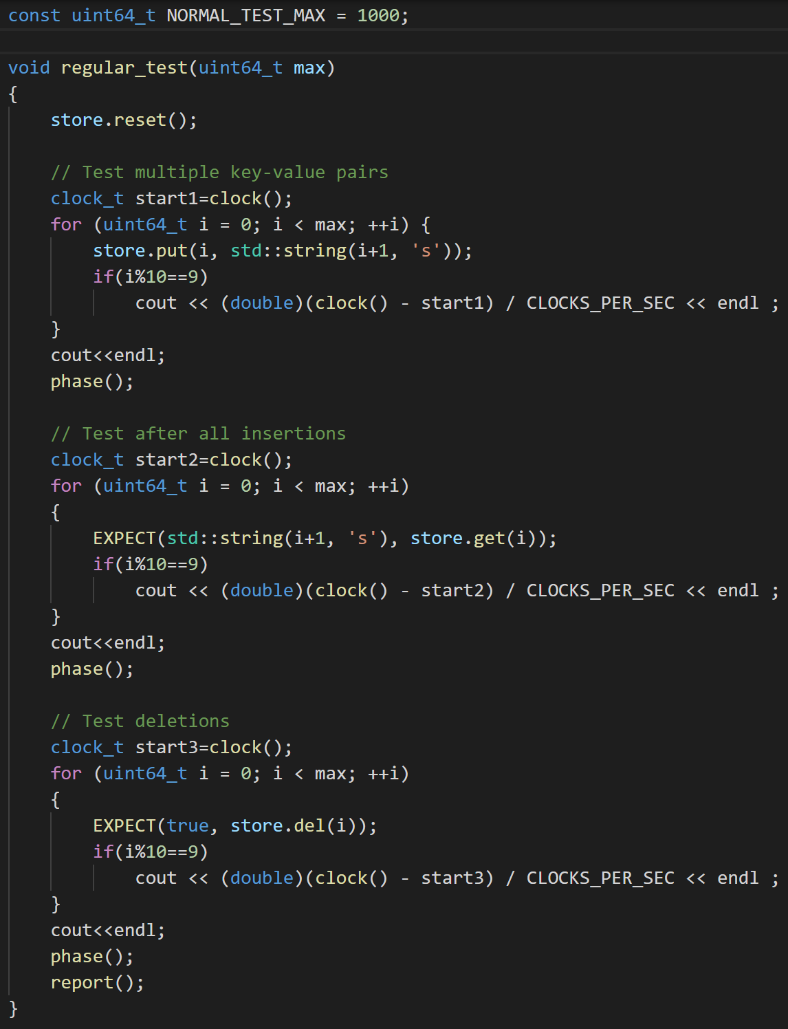
内存：16.0 GB

硬盘：512GB SSD

系统：Windows 10

时延：在正常情况下（无compaction操作）PUT、GET、DELETE 操作所需要的平均时间的柱状图；

测试程序：

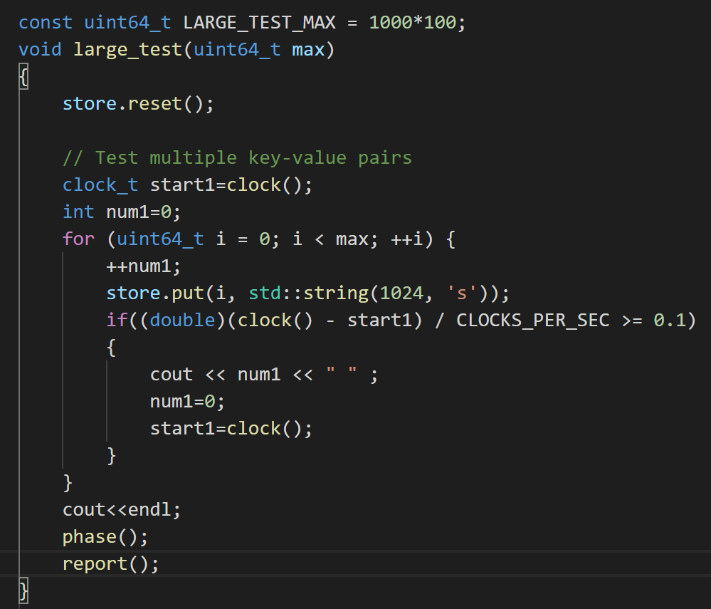


条形图：

由于每次操作时间过短，因此我选择10次操作计算一次时间，可以看出三种操作的平均时间都在0.001秒左右。put时时间过长可能是因为某些塔的层数较高，而get时间过长则可能是某些塔时间过低。可以看到这两者的较长时间不重合。由于del时是遍历整个跳表，所以del出现过长的时间我也有点不能理解，可能是不同时间cpu状态不同吧。但越到后面时间越总体下降是符合规律的，因为节点数变少了。

2）吞吐量：不断插入数据的情况下，每秒钟处理的PUT请求个数（即吞吐量）随时间变化的折线图。

测试程序：



折线图：

由于我刻意安排每次插入的value的长度都是1024，这就导致发生compaction是有一定周期性的，可以看出图像较符合此规律。当吞吐量最大（3000左右）时是不发生写入的时候，当吞吐量较大（2500左右）时是发生写入sstable但不发生compaction时，当吞吐量最小（低于1000）时是发生compaction时。

**总结**

总结几个浪费了不少时间的bug：

1. 合并时检查下层sstable中key的范围与上层是否有交集，忽略了上层整个的范围是下层文件范围的真子集的情况
2. string写入文件、从文件读出到string和uint\_64t的方法有误，多亏了谭世炜同学的讲解和帮助，我才能正确读入和写入文件
3. del操作的时候先检查memtable中是否有key，检查过之后我不论value是否为空都直接删去并返回true了，实际上若value为空则不应删掉并返回false
4. 合并时将kv-pair写入sort\_table时要先判断sort\_table中是否含有该key，应该先判断if(contain(it->key))再让迭代器it++，我不知道怎么的先让it++再判断，由于我debug的时候与写的时候相隔了近一个星期，导致我一个一个翻sstable文件发现一些key压根没有，才得以找出这个bug...
5. sstable类我手贱加了个析构函数，会remove掉相应文件，导致我写完一个文件就被删，写完一个就被删，我一度以为闹鬼了，后来单步调试时才意识到是析构函数（这又是因为我debug的时候早就忘了写的时候心里是怎么想的）
6. sort\_disk与sort\_table俩名字太像，导致我合并时稍微不注意clear错了，我一度以为list容器限定大小，溢出后会全部清除，搞得我怀疑人生...
7. 其实本来这些都是小bug，应该很容易看出，但是由于刚开始我debug时打断点无效（程序忽略断点继续向下执行），所以只能采取原始的cout法以及很麻烦的gdb，心态崩了......后来不知道什么原因又可以打断点了（好像是因为我点了一下启用所有断点？），于是我才可以欢乐地debug
8. correctness通过之后我自认为KVStore的构造函数写的天衣无缝，persistence肯定一遍过，结果由于①和④，prepare时就fail了，我一度逃避艰难的人生，开始追番......后来因为ddl的催赶，我不得不一个一个翻sstable文件并且重新理逻辑，才终于找出bug

特别鸣谢：斯金泽。斯金泽同学在比我早做完两到三天的情况下向我施以援手，我得以和他深入地讨论具体结构与细节，这对于我对整个项目的理解的加深有很大帮助。另外他还指正过许多我忽略的bug：比如上述的①和③。可以说如果没有他的帮助，我可能每天要晚睡3个小时。