

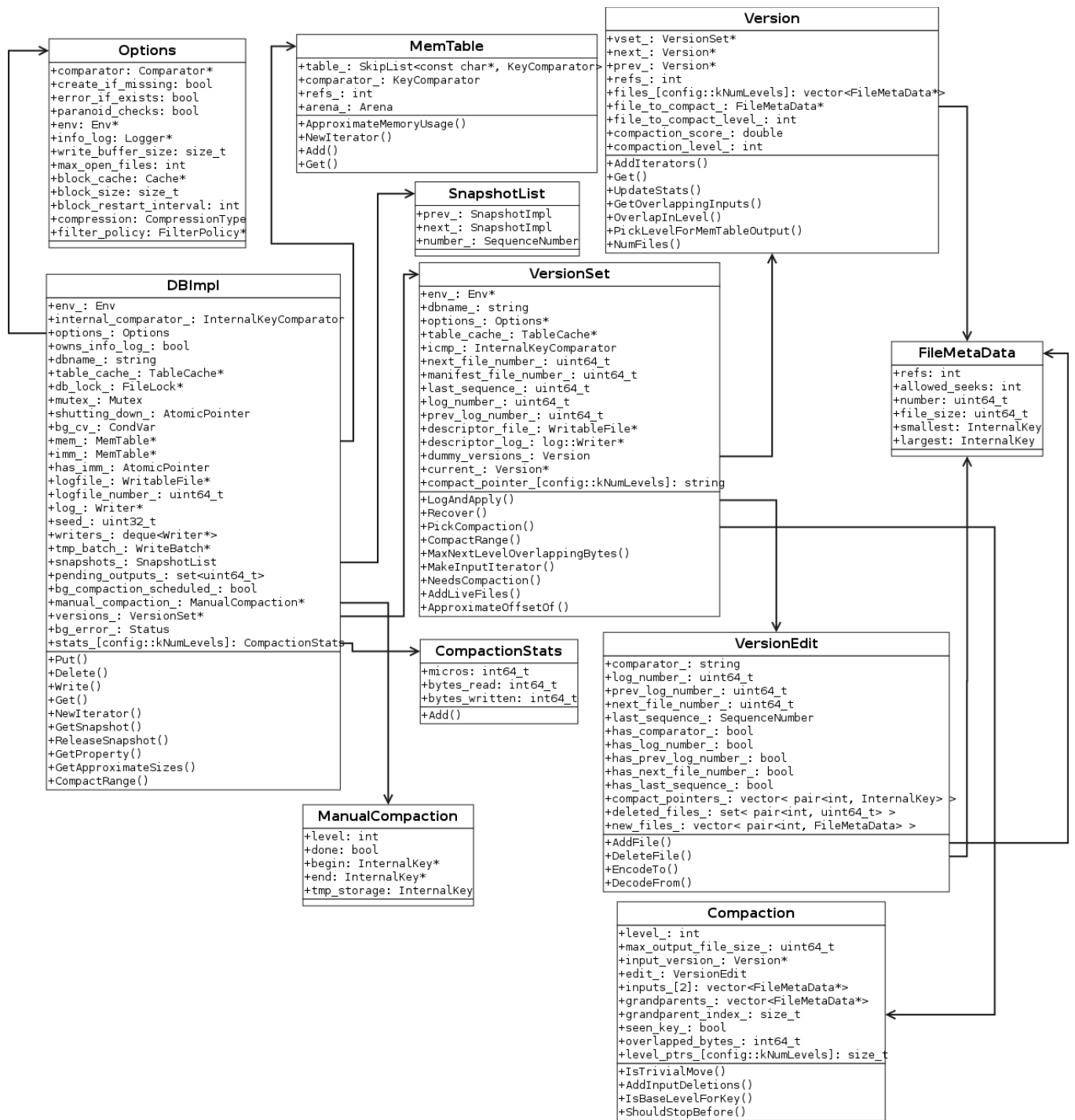
LevelDB-Notes

Aug 11, 2014

LevelDB特点

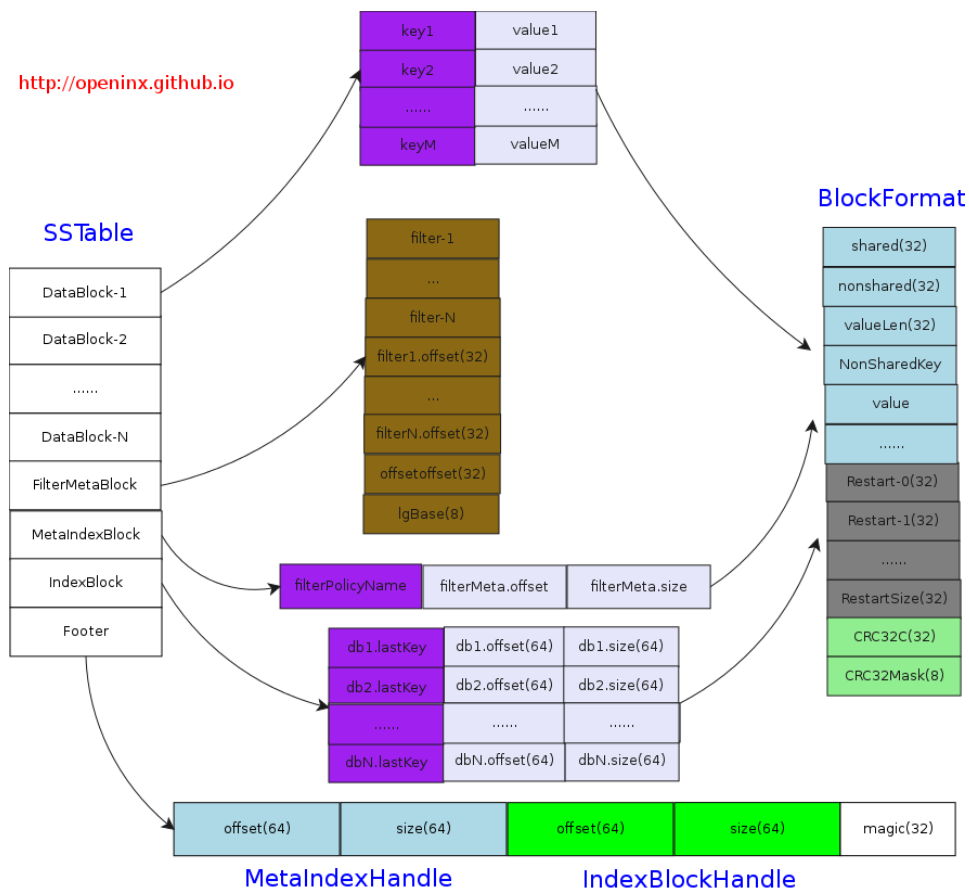
1. 持久化
2. 自定义xcompare比较函数
3. 接口简洁, Write, Read, Delete, Batch.
4. read一致性snapshot.
5. 数据压缩 - snappy
6. 写性能40W/s, 读性能6w/s.
7. 日志-系统崩溃恢复
8. Restart Point - 前缀压缩
9. MemTable是用SkipList实现的。[SkipList的期望复杂度的证明](#)
10. Cache: TableCache & BlockCache.
11. LevelDB在插入删除过程中, 版本如何维护?
12. Minor Compaction & Major Compaction
13. bloom filter
14. 元数据全部存放在内存上, 决定了磁盘数据存储上限。

LevelDB主要类图



LevelDB/SSTable编码图详细

<http://openinx.github.io>



db/build.cc

将一个串有序的key-Value编码写入到TableFile里，并且执行fsync。

db/dbformat.cc & db/dbformat.h

- namespace config定义了一些常量
 - leveldb的kNumLevels=7.
 - 当Level-0的sstable的个数大于等于kL0_CompactionTrigger(4)时，则Level-0需要做compact了。
 - 当Level-0的sstable的个数到达kL0_SlowdownWritesTrigger(8)时，则LevelDB会限制写速率。
 - 当Level-0的sstable的个数到达kL0_StopWritesTrigger(12)时，则LevelDB会停写。
- ParsedInternalKey & InternalKey & LookupKey
 - ParsedInternalKey由UserKey,SequenceNumber,ValueType三个成员组成，编码成字符串之后就是InternalKey的rep_成员。
 - InternalKey := [UserKey][SequenceNumber(56Bit)][ValueType(8Bit)]
 - ValueType := kTypeDeletion | kTypeValue
 - LookupKey由start,kstart,end_三个组成。

db/db_impl.cc & db/db_impl.h

- tablecache = options.max_open_files(1000) - kNumNonTableCacheFiles(10)= 990 * 2M.

- 查看当前LevelDB中LSM树每个Level的sstable的数量为多少：`tail -f LOG | grep 'compacted to'`
- DBImpl::NewIterator 在做Next的时候，如何过滤掉ValueType=KDeletion的key,因为这些key在高层是kDeletion的，但是在底层还是存在的，那么有可能在底层将key取出来？
- DBImpl在方法MakeRoomForWrite中将imm的log删掉了？没删，只是把imm的log文件fclose了。
- `DBImpl::Recover(VersionEdit* edit)`
 - a. 加文件锁
 - b. 读出CURRENT文件的manifest文件名，若没有CURRENT就按照 `create_if_missing` 决定是否新创。
 - c. 依次从manifest读出一条edit, 并应用到当前的versionSet. 即 `builder.Apply(edit)`。将当前versionSet的current_更新为所有edit应用后得到的v, 即 `versionSet.Recover()`。
 - d. 将当前活跃的log文件依次应用到memtable, 即 `DBImpl::RecoverLogFile`。
- `DB::Open()`
 - a. Recover 得到一个edit
 - b. 应用edit并更新当前版本LogAndApply.
 - c. DeleteObsoleteFiles
 - d. MaybeScheduleCompaction检查是否需要Compact.
- `DBImpl::WriteLevel0Table(MemTable* mem, VersionEdit* edit, Version* base)`
 - a. 用mem的迭代器将mem dump成一个sstable文件。
 - b. 按照PickLevelForMemTableOutput()策略返回sstable添加的level.
 - c. 将edit对应的level加上该sstable, 返回edit.
- `DBImpl::CompactMemTable()`
 - a. WriteLevel0Table(imm, &edit, base);
 - b. LogAndApply(&edit);
 - c. DeleteObsoleteFiles()
- 有以下几种情况会触发MaybeScheduleCompaction():
 - a. Get() 当mem和imm都没有找到key, `current_` 里第一次seek的sstable(我认为也可用第2次或第3次吧)的allowed_seek用光的时候，需要触发MaybeScheduleCompaction.
 - b. Write() 调用 `MakeRoomForWrite`。当imm表dump到磁盘完成(imm=NULL, 当imm为NULL时，说明versionSet已经维护好了prevLogNumber)，且mem表占用字节数超过了 `write_buffer_size` 时，需要将mem转成imm, 然后新开一个mem, 最后执行MaybeScheduleCompaction。
 - c. 当一次compaction完成之后，在某一层产生了很多sstable, 这样会继续MaybeScheduleCompaction.
- `DBImpl::BackgroundCompaction()`
 - a. `versionSet.PickCompaction()` 或者 `manual_compaction_`。
 - b. 当compaction与下层leve+1没有overlap，且与level+2的文件的字节总数不超过20M时，直接把sstable放level+1层。这叫做 `TrivialMove`, 无关紧要的移动。同时维护edit.
 - c. 当不是 `TrivialMove` 时，就做 `DoCompactionWork`。
- `DoCompactionWork` 将要compaction的level层sstable和level+1层sstable组织成一个有序的合并迭代器iter. 每次执行一次iter.Next(), 问题是哪些数据需要dropped掉呢？

```

if (last_sequence_for_key <= compact->smallest_snapshot) {
    // Hidden by an newer entry for same user key
    drop = true;    // (A)
} else if (ikey.type == kTypeDeletion &&
           ikey.sequence <= compact->smallest_snapshot &&
           compact->compaction->IsBaseLevelForKey(ikey.user_key)) {
    drop = true; // (B)
}

```

第一种情况: 是已经得到了一个更新(seq更大)的key, 所以丢弃掉现在得到的相同的key值。注意iter内对同一个key值seq是按照降序排列的, 降序也就是新鲜度降低。只能丢掉 `smallest_snapshot` 之前的, 之后的还有snapshot在用呢, 所以不能丢。

第二种情况: 要求是删除操作, 而且该key必须在level+2层以下没有出现。假设出现了却被删了, 那么下次在level+2层以下发现一个key, 就会被iter取出, 而实际在上层已经被删掉了, 造成错误。

Compaction & AllowedSeeks

- Compaction流程

```
if(imm != NULL){
    sst = BuildTable(imm);
    level = PickLevelForMemTableOutput(sst);
    edit = updateEdit();
    updateVersionSet(edit);
}else{
    c = PickCompaction();
    if(c.level.sstable == 1 && c.(level+1).sstable == 0 ){
        array = overlap(c.level, c.level+2);
        if(totalSize(array) < kMaxGrandParentOverlapBytes(20M)){
            place c.level.sstable to Level+ 1;
            return ;
        }
    }
    DoCompactionWork; // 每次合并的数据量在26M左右。
    edit = updateEdit();
    updateVersionSet(edit);
}
```

- AllowedSeeks的确定

```
// We arrange to automatically compact this file after
// a certain number of seeks.  Let's assume:
//   (1) One seek costs 10ms
//   (2) Writing or reading 1MB costs 10ms (100MB/s)
//   (3) A compaction of 1MB does 25MB of IO:
//       1MB read from this level
//       10-12MB read from next level (boundaries may be misaligned)
//       10-12MB written to next level
// This implies that 25 seeks cost the same as the compaction
// of 1MB of data.  I.e., one seek costs approximately the
// same as the compaction of 40KB of data.  We are a little
// conservative and allow approximately one seek for every 16KB
// of data before triggering a compaction.
```

db/version_set.h & db/version_set.cc

- level-0的sstable大小不能超过 `options_.write_buffer_size`。level-N(N>0)的sstable的最大空间不能超过kTargetFileSize(2M)。且第i(i>0)层的sstable的个数不能超过 10^i ，所以第1层到第kNumLevel-1(6)层，总共能容纳的数据量为 $(10+10^2+\dots+10^6) * 2 / 1024 = 4238G$
- `Version::PickLevelForMemTableOutput`
确定memtable dump到哪一层。假设与当前level有overlap,那么直接放到当前level；否则看与level+1是否有overlap，有就放level+1，没有就看level+2的overlap的files的总bytes数是否超过kMaxGrandParentOverlapBytes(2M),假设超过kMaxGrandParentOverlapBytes(20M)就放level+1算了，因为放level+2的话，要合并一大片数据IO划不来。
- VersionSet::Builder有三个成员，其中base_是一个全量，levels是一个增量（全量基础上要删除的文件和要新增的文件）。

```
VersionSet* vset_;
Version* base_;
LevelState levels_[config::kNumLevels];
```

`Builder.Apply(VersionEdit* edit)` 是将edit的增量合并到levels这个增量上来。

`Builder.SaveTo(Version* v)` 是将多次累计起来的增量levels,与base_全量做合并，得到一个版本v。

- `VersionSet::LogAndApply(VersionEdit* edit, port::Mutex* mu)`
 - 对VersionSet的当前版本执行edit增量，得到更新后的版本v
 - 更新v的 `compaction_score`，即 `Finalize(v)`
 - 将edit记日志到manifest文件，初次记manifest之前，会先写全量到manifest。
 - 更新当前版本 `current_` 为v，并将v加入版本维护队列表尾,即AppendVersion(v)。
- `Compaction* VersionSet::PickCompaction()`
 - 当level层的 `compaction_score` 超过1时，选择该层第一个 `largest>compact_pointer_[level]` 的sstable去做compaction；
 - 当level层的某个sstable的allowed_seeks用光时，选择该sstable去做compaction。
- `VersionSet::SetupOtherInputs(Compaction* c)`
 - 将c即将合并的level层sstable进行一次扩展，但是扩展后，必须满足：level层的sstable数据量之和 + (level+1)层的sstable数据量之和 <= kExpandedCompactionByteSizeLimit(25*kTargetFileSize=50M)。这样做的好处是让一次compaction合并不多不少的数据。
 - `compact_pointer_[level]` 意义：上次在level层参与compaction最大的key。
- `Compaction* VersionSet::CompactRange(int level,const InternalKey* begin,const InternalKey* end)`
 - 拿到在level层，与[begin,end]区间overlap的sstable列表。
 - 当选取sstable列表的一小段，保证选的这端sstable字节数之和不超过 `MaxFileSizeForLevel(level)`。
 - 按照 `SetupOtherInputs` 方式拓展。
- `Compaction::IsBaseLevelForKey(const Slice& user_key)`
如果 `user_key` 在 `i(i>=level+2)` 层落在了 `level_ptrs_[i]` 之后的某个sstable的range内，返回false，否则true。
- `Compaction::ShouldStopBefore(const Slice& internal_key)`
当internal_key与level+2层的overlap的sstable的字节数之和超过kMaxGrandParentOverlapBytes(10*kTargetFileSize=20M)返回true,否则返回false。

table/block.cc

Block块内搜索key的方式是(`iter->Seek()`): 在 `[RestartPoint-0, RestartPoint-1, ..., RestartPoint-N]` 之间用二分查找, 在 `[RestartPoint-(i), RestartPoint-(i+1)]` 之间用线性查找.

table/iterator.cc

注册多个清理function组织成链表, 在Iterator的回收时, 依次执行每个清理函数.

table/iterator_wrapper.h

IteratorWrapper缓存了iterator的`it->Valid()`, `it->Key()`两个状态. 当要多次用到valid和key的值时, 可以减少因调用`it->Valid()`和`it->Key()`而产生的开销. 对外暴露的interface和Iterator一样.

table/merger.h

将N个有序迭代器合并成一个有序的迭代器, 找N个中最小的值时作者用的线性查找, 可用MinHeap优化, 降低时间复杂度. 类似于这个问题[merge-k-sorted-lists](#).

table/table.cc

- 读取Block的时候, 会将该sstable的`cached_id(8Bytes)` + 该block的`offset(8Bytes)`拼成一个16 Bytes的key, 将该 (key, *Block)放入到LRU-Cache里面.
- LRU-Cache由`table.Open(&options, ...)`传入的 `options.block_cache` 确定, 默认的情况下 `options.block_cache=NewLRUCache(8 << 20)`。

table/builder.cc

- 从 `void TableBuilder::Add(const Slice& key, const Slice& value)` 实现看, 实际block的size有可能比4K大一点点, 而不是严格的4k. 因为是在插入(key,value)完成之后, 判断当前的blockSize是否大于4K, 假设大于4K就刷盘, 另起一个block.
- `lgBase=11`, 当`block.offset`在 `[i*lgBase, i*lgBase+1, ..., (i+1)*lgBase-1]` 这个范围的是否, 该block内的所有key生成的filter为 `filter-i`。

table/two_level_iterator.cc

用一个迭代器可以依次扫描一层Level的所有SSTable里面的所有block的所有(key,value)对. 这样的迭代器称之为 `two_level_iterator`

util/arena.cc

- 简单内存管理器，内存按照4K分配。一个个的4K块组成了整个的内存列表。提供内存对齐分配，对中间的空隙内存直接丢掉不管。

util/bloom.cc

- 用于判断一个给定的key是否在一个键值集合set中。假设set = ['hello', 'world', 'good', 'boy'], 将set的每一个单词hash成一个[0,63]之间的数, 假设h(hello)=3, h(world)=24, h(good)=54, h(boy)=59。那么得到一个bit串, 将这个串保存下来, 取名叫做bloomFilter。对给定一个key='caonima', 按照同样的hash函数, 得到一个[0,63]之间的数36, 发现在bloomFilter的36位不为1, 那么'caonima'肯定不在set这个集合中。为了更快的排除key不再set中, 可以将set中的key进行t(1<=t<=30)次哈希取或, 得到bloomFilter值。然后对key进行t次哈希取或, 假设某一次哈希值对应的bit位不为1, 就可以确定key不再set中。

```
00010..010..010..010000
  ^      ^      ^      ^
  3      24     54     59
```

- h是个32为整数, `(h >> 17) | (h << 15)` 将h的比特为向右Rotate17位。

util/cache.cc

- LRU-Cache. 作者自己实现了一个可以resize的hash表, 号称要比C++自带的哈希表快5%。个人觉的快就在hash表桶中存放的节点LRUHandle, 保存了key对应的hashValue。这样不论是在find操作还是resize操作都不要计算hash值, 快了好多。
- LRUCache用双向链表+哈希表实现的话, get和set的复杂度都为O(1)。
- ShardedLRUCache有4个普通的LRUCache组成, 对key的hashValue取高4位来决定到底落在哪个LRUCache表里面。

util/coding.cc

- 解决大端小端编码问题。

util/env_posix.cc

- 实现了随机读写文件, 顺序读写文件, mmap映射读写文件三种IO方式。
- 实现了生产者消费者模型的任务调度器。

util/histogram.cc

写了个Demo调用了下，发现生成了一个如下柱型图，用来做Benchmark的。

```
Count: 1000000  Average: 1.9963  StdDev: 1.42
Min: 0.0000  Median: 2.4955  Max: 4.0000
-----
[      0,      1 )  201310  20.131%  20.131% #####
[      1,      2 )  199654  19.965%  40.096% #####
[      2,      3 )  199869  19.987%  60.083% #####
[      3,      4 )  199772  19.977%  80.061% #####
[      4,      5 )  199395  19.940% 100.000% #####
```

util/random.h

- 简单的随机数生成器

0 Comments

openinx blog


1

Login ▾

♥ Recommend

🔗 Share

Sort by Best ▾



Start the discussion...

Be the first to comment.

ALSO ON OPENINX BLOG

MySQL Cluster工具关注的一些问题

6 comments • 3 years ago•

openinx — 这位朋友真是太可爱了！跨分片就是cross-shard...

一个滴滴老司机的故事

2 comments • 6 months ago•

openinx — 滴滴司机工作时间长这个是肯定的，像文中的司机这种情况，一方面确实很大经济压力要去还债， ...

Golang Src Notes

5 comments • 2 years ago•

openinx — 正在努力填哈。

the implement of redis dict

4 comments • 3 years ago•

ddd — 星星评价甚是到位，争哥听了心中小鹿乱撞。

