Leveldb原理与源码剖析

youngsterxyf

2014.08.31

Leveldb简介

特点

- 持久化存储的KV系统
- 记录按Key有序存储
- 应用场景：写远远多于读
- LIB, NO SERVER — 好处是？
- 单进程

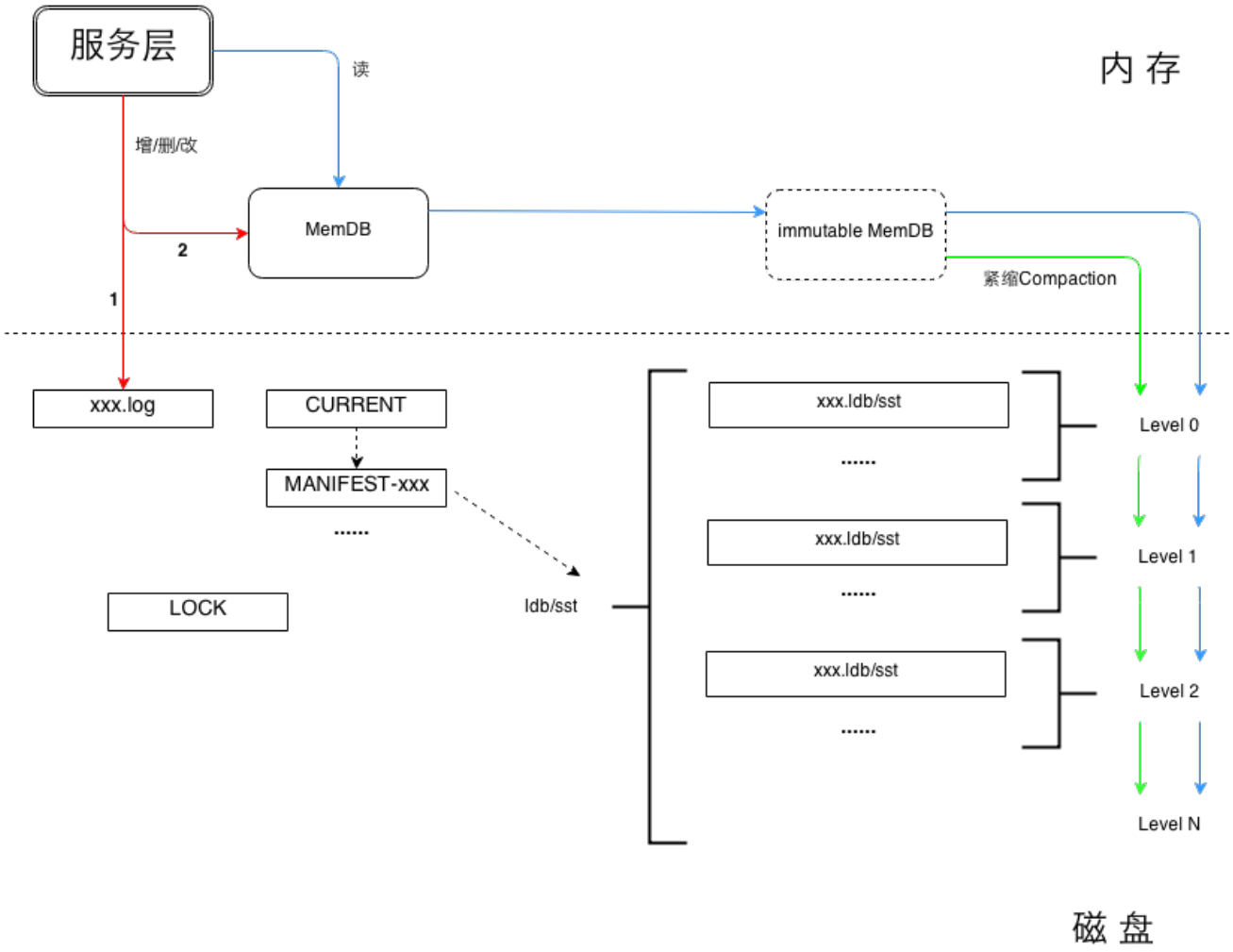
测评

- 随机写：40万条记录每秒
- 随机读：6万条记录每秒
- 顺序读？

应用案例

- InfluxDB
- BigTable
- Google Chrome IndexDB
- 淘宝Tair的持久化存储引擎
- 可作为很多存储方案的存储引擎(如cayley、Riak)
- 异步消息队列的Broker

Leveldb基本原理



LevelDB的使用

如何使用LevelDB？程序演示... 源码见Github
逐个接口探索实现...

基本逻辑

初始化的过程就是打开数据库的过程-

1. 首先初始化一些数据结构，如d.tableCache、d.mem
2. 然后尝试创建数据存储目录，并创建LOCK文件，加文件锁。
3. 如果不存在CURRENT文件，则说明应该是首次打开数据库，需要创建manifest文件，并向文件中写入用户key比较方法的名称以及下一个可用的文件序号，然后创建CURRENT文件，在其中写入新manifest文件的文件名。
4. 接着，不管是否是首次打开数据库，都要先读取CURRENT文件内容，继而读取CURRENTS所指向的manifest文件的内容，对于其中的每条记录解析得到comparatorName、logNumber、nextFileNumber、lastSequence、compactPointer、deletedFiles、newFiles、preLogNumber（不一定都有），根据deleteFiles、newFiles计算出每个level持有的文件列表
5. 根据前一步骤得到的数据生成一个version（并且会计算该version的compactionScore和compactionLevel），放入d.versions链表中
6. 然后读取原来记录写入mem的操作的日志文件，根据日志内容重做其中的操作，存入一个临时的memtable中，然后转存入磁盘的level0 db文件中
7. 然后创建新的日志文件，设置d.log、d.logFile，并根据前一步骤重做日志产生的versionEdit和d.versions.currentVersion()生成一个新的version添加到d.versions上，并将versionEdit的内容写入manifest文件
8. 删除多余的文件（老log文件、当前version不需要的db文件等）
9. 尝试发起compaction（compaction的具体细节见“紧缩”一节）

基本逻辑

对于Leveldb来说，插入/更新是同一个操作SET，过程如下：

1. 先将SET操作的key、value封装进一个batch，然后将batch的数据存储到d.mem所指的内存空间中，在存入d.mem之前需要检测d.mem是否还有剩余可用空间
2. 若没有，则将d.imm指向原来d.mem指向的内容空间，为d.mem申请一块新的内容空间，并创建新log文件设置d.logNumber、d.log、d.logFile，然后尝试发起compaction
3. 在存入d.mem之前还需要先把数据(batch.data)写入log文件并持久化。
4. 最后从batch中逐个读取kind、ukey、value，根据ukey生成内部key，然后以内部key为key想d.mem中写入value

基本逻辑

删除操作(DELETE)的过程与插入/更新的过程基本一致。因为Leveldb并不会真的去删除key、value对。DELETE操作，用户只提供了key，在封装成batch时，仅将key封装进去，同时将操作类型（internalKeyKindDelete=0）也封装进去。

基本逻辑

由前述内容可知，在初始化或SET操作d.mem已写满时，可能会有compaction过程。

- compaction是否发起，和d.imm是否为nil以及d.versions.currentVersion().compactionScore是否大于1有关
- 实际的compaction过程是在一个新的goroutine中执行的

基本逻辑(续)

当d.imm不为nil时，

1. 先将d.imm中的数据转存入level0的一个新的db文件中，转存的过程就是从d.imm逐项读出数据写入该新db文件中，然后返回该新db文件的元信息（fileNum、smallest、largest、size）
2. 接着根据上一步返回的元信息封装一个versionEdit，与d.versions.currentVersion()生成一个新的version，并计算该version的compactionScore和compactionLevel，计算方法为：
   1. 对于level0，计算$float64(len(v.files[0]))/l0CompactionTrigger$
   2. 对于非0 level，计算$float64(totalSize(v.files[level]))/maxBytes$，maxBytes初始为$float64(10 * 1024 * 1024)$，然后level每增大1，maxBytes则增大10倍
   3. 从1和2中找出最大的一个值作为version的compactionScore，这个最大的值对应的level作为version的compactionLevel
3. 将versionEdit信息写入manifest文件中

基本逻辑(续)

除了d.imm不为nil时，需要将d.imm compaction到level0外，紧缩过程都是根据当前version的compactionLevel（假设为level n）（若*compactionScore* >= 1）

1. 先从level n、level n+1、level n+2找出内部key范围有重合的所有文件
2. 然后将这些文件的内容写到一个新的db文件中，并将该新db文件加到level n+1持有的文件列表中，从level n和level n+1持有的文件列表中删除找出来的文件（咦，level n+2呢？），从而得到一个新的versionEdit
3. 将新得到的versionEdit与当前的version合并生成一个新的version，放到d.versions中
4. 真正删除无用的db文件、log文件等
5. 在完成一次compaction后，由于某个level的文件数和文件内容大小有所变化，也生成了新的version、新的compactionScore和compactionLevel，所以会再次尝试发起compaction

基本逻辑

查找流程

1. 根据用户提供的key，封装成内部key
2. 根据内部key，依次从d.mem、d.imm中查找
3. 第2步若没有找到，则接着依次从level 0、level 1、...中查找

- 由于level 0和其他非0 level的db文件组织方式不相同，所以查找的方式也不一样
- 为什么查找的顺序依次为d.mem、d.imm、level 0、level 1、...呢?

参考资料

- 数据分析与处理之二（Leveldb 实现原理）
- Leveldb官方文档
- Leveldb - Google Code
- Leveldb - dirlt.com

Q & A

谢谢！