LevelDB的实现类似于Bigtable，但文件的组织稍有不同。

DB的文件有几种类型：

1）日志文件log files

以.log结尾，每一个日志文件存储一系列最近更新，通过附加方式。在内存中有一个它的副本memtable，将更新附加到日志文件后（为了防止数据丢失），马上写入到这个内存副本，它是一个内存排序结构，即SkipList。当日志文件到达预设的大小，默认4M，则将创建一个新日志文件和新的memtable记录后面的更新，原来的内存副本不再操作，成为immutable memtable，后台将自动将它有序保存到磁盘，成为Level-0的有序文件，此时才将日志文件和immutable memtable删除。

2）有序文件sorted tables

以.ldb结尾，存储的是按Key有序的Key-Value，Value可以是Key的最新值，或者是Key的删除标志。

有序文件分多个Level，从日志文件直接写入的是Level-0。

当Level-0的有序文件超过设定的阈值，默认4个，则将所有Level-0有序文件（或者它们的一个子集，这个子集的最大最小Key与其他都不重叠）与Level-1中与它们有重叠的有序文件一起归并排序，重新生成一系列Level-1有序文件，每2MB一个文件。

Level-1内的文件没有Key重叠，更高层级内的文件也如此，只有Level-0内的文件可以有重叠。

当Level-L（>=1）的文件总大小超过了10^L MB，就取其中一个文件和Level-(L+1)内与它有重叠的文件合并，生成一系列新的Level-(L+1)文件。

合并完成后，参与Compact的老文件就被删掉，对应的文件增删信息以日志形式记录到清单文件。此过程中完成了老数据的删除，起到删除冗余的作用。

Level-L（>=1）的文件生成不超过2MB，但有时不足2MB也开始另一个此层级文件的生成，主要是为了防止它的范围覆盖了Level-(L+1)的超过10个文件，这主要是为了防止Compact Level-L层级的一个文件时，需要加入的Level-(L+1)层级文件过多。

Compact挑选用于合并的文件或文件子集（Level-0层级的）的规则是，根据Key空间轮流来。它的起始Key是上一次的最后一个Key的大于一点的，对于Level-0有重叠，所以会依次将有重叠的新文件或对新文件有重叠的新文件加进来，成为子集。

在一层级的一次Compact中，被覆盖的Key-Value会被删除（谁被覆盖，依据文件的最后修改时间？），如果最新的Value是删除标记，而且此层级就是最高层了，那么也会在最后时刻将此Key删除。

3）清单manifest

清单文件列出了所有的有序文件，包括文件名、层级、最小最大Key，和其他元数据。

每一次打开DB都会生成一个新的清单文件，并在文件名上有新的数字，如MANIFEST-000017，老的被删除。

它的内容以日志的形式附加，如一个有序文件的添加或移除。

4）current

它是一个文本文件，内容只有最新的清单文件的文件名。

**Timing**

Level-0 compactions will read up to four 1MB files from level-0, and at worst all the level-1 files (10MB). I.e., we will read 14MB and write 14MB.

Other than the special level-0 compactions, we will pick one 2MB file from level L. In the worst case, this will overlap ~ 12 files from level L+1 (10 because level-(L+1) is ten times the size of level-L, and another two at the boundaries since the file ranges at level-L will usually not be aligned with the file ranges at level-L+1). The compaction will therefore read 26MB and write 26MB. Assuming a disk IO rate of 100MB/s (ballpark range for modern drives), the worst compaction cost will be approximately 0.5 second.

If we throttle the background writing to something small, say 10% of the full 100MB/s speed, a compaction may take up to 5 seconds. If the user is writing at 10MB/s, we might build up lots of level-0 files (~50 to hold the 5\*10MB). This may significantly increase the cost of reads due to the overhead of merging more files together on every read.

Solution 1: To reduce this problem, we might want to increase the log switching threshold when the number of level-0 files is large. Though the downside is that the larger this threshold, the more memory we will need to hold the corresponding memtable.

Solution 2: We might want to decrease write rate artificially when the number of level-0 files goes up.

Solution 3: We work on reducing the cost of very wide merges. Perhaps most of the level-0 files will have their blocks sitting uncompressed in the cache and we will only need to worry about the O(N) complexity in the merging iterator.

**Number of files**

Instead of always making 2MB files, we could make larger files for larger levels to reduce the total file count, though at the expense of more bursty compactions. Alternatively, we could shard the set of files into multiple directories.

An experiment on an ext3 filesystem on Feb 04, 2011 shows the following timings to do 100K file opens in directories with varying number of files:

| **Files in directory** | **Microseconds to open a file** |
| --- | --- |
| 1000 | 9 |
| 10000 | 10 |
| 100000 | 16 |

So maybe even the sharding is not necessary on modern filesystems?

**Recovery**

* Read CURRENT to find name of the latest committed MANIFEST
* Read the named MANIFEST file
* Clean up stale files
* We could open all sstables here, but it is probably better to be lazy...
* Convert log chunk to a new level-0 sstable
* Start directing new writes to a new log file with recovered sequence#

**Garbage collection of files**

DeleteObsoleteFiles() is called at the end of every compaction and at the end of recovery. It finds the names of all files in the database. It deletes all log files that are not the current log file. It deletes all table files that are not referenced from some level and are not the output of an active compaction.

<http://www.cnblogs.com/haippy/archive/2011/12/04/2276064.html>