2019-6-11 simplekvsvr 文档

小書匠

目录

一, 关键设计

1.1 数据分片

假定数据分布比较均匀，如果不均匀，也要通过某种hash函数使其在整体上看起来是分布均匀的，根据key的将6400万条数据划分到64个分区，对数据进项分区对数据进行分区会带来以下几个好处：

降低冲突：将每个分区的数据单独存储，在写入数据时，每个分区具有一个锁控制并发写入的一致性。相比于不做分区的情况，理论上锁的冲突降低了 64 倍。
并行计算：可以使用多线程并发地构建各个分区的索引，理论上索引构建的时间降低了64倍。
快速定位：在根据 key 查找数据时，可以根据 key 直接定位到所在分区，然后在对应分区中进行查找。
加载分区：可以将整个分区的数据完全加载进内存，读大块数据的操作有利于发挥 SSD 性能。

当然了，由于没有专门设计存储的数据，也没有专门的业务场景，所以分为64个分片并没有其他实际上的考量，具体的参数设置还是需要根据实际场景去设定,理论上似乎是越多越好，但是文件描述符的数量与之还是有一些关系的，需要试验去获得,本机测得文件数最合理为240。

1.2 Key/Value 分离存储

一般情况下，key 的长度都是要小于 value 的长度的,分离存储的话，索引文件要远小于value文件，还可以解除构建索引和数据存储之间的耦合，具体有以下优点。

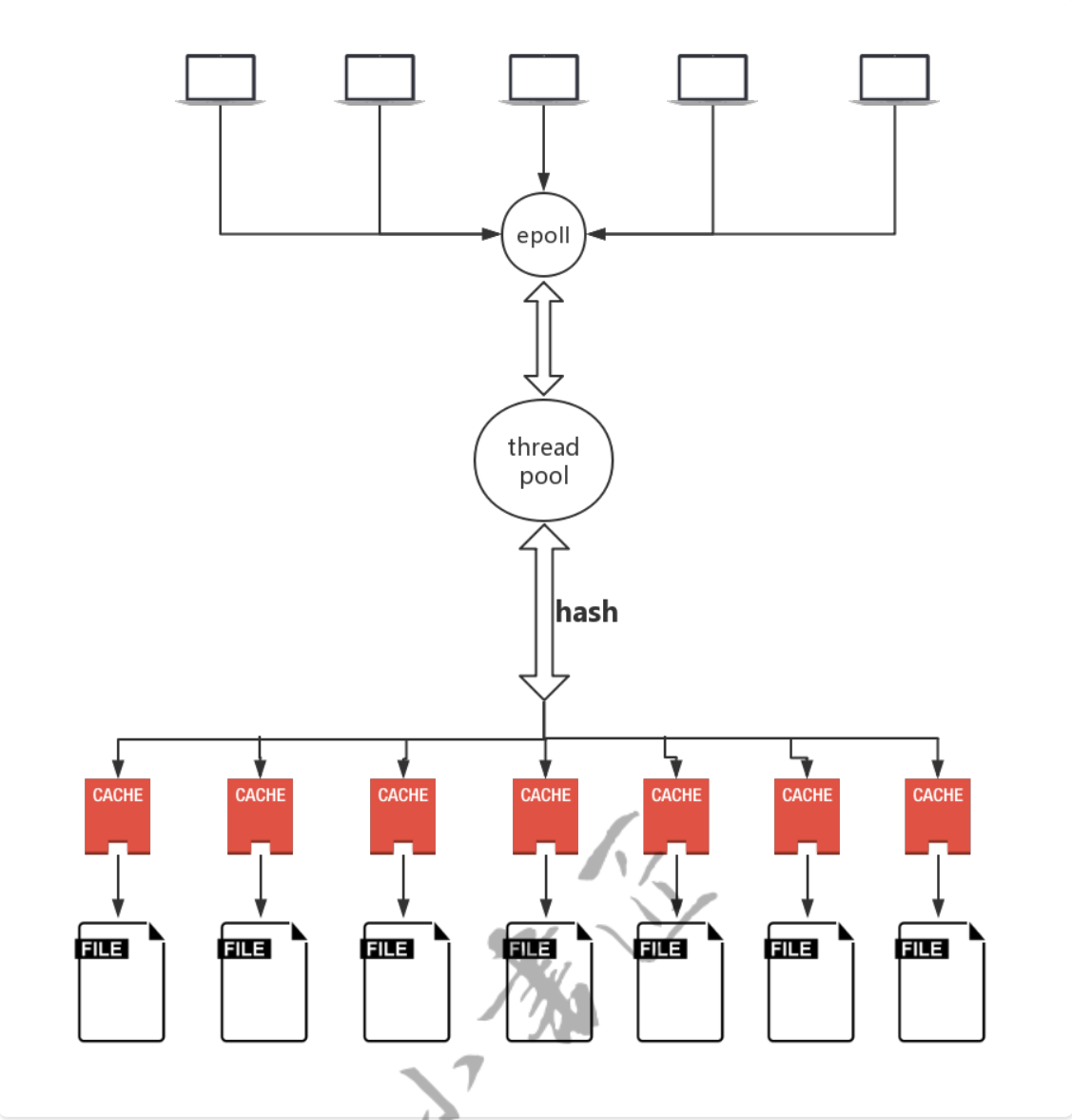
- 索引可以全部加载到内存，格式为 <key， mapEntry>，在内存中可以读取索引，其中，mapEntry 是 value 在 valuelog 中的分布信息。
- 恢复阶段只需读取体积很小的索引文件，不用访问较大的 value 文件。

由于 key 和 value 不同时写入文件，可能会导致进程退出时的数据不一致。解决的策略是先写 value 文件，最后写 key 文件.

1.3 网络模块

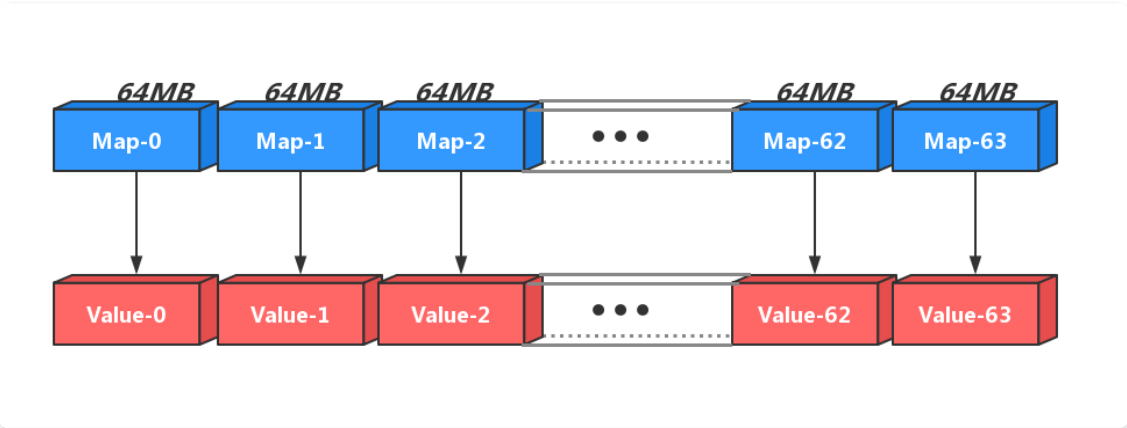
使用 epoll 多路复用技术，可以提升服务器的并发能力，epoll_wait 返回需要处理的事件集，省去遍历大量无事件套接字。 通过其返回的事件集，找到对应的fd 再添加至线程池任务队列中去，充分利用多线程并发。

二, 整体流程



整体流程图 (2)

2.1 文件架构



文件架构图

2.2 索引架构

由于此数据库功能仅需支持增删改查，不需要排序功能，那索引的最佳人选当然是一个性能优异的哈希表——google_hash_map, 由于我把数据分成了64个部分，本着内存不要钱的想法，那当然是对每一个部分都建立一张表，好处有以下几点:

- 提升查询效率
- 每个分片对应一个缓存，缓存的效率也随之提升
- 可以与之后的扩容解耦，即只需扩容那个需要扩容的那个分片所对应的文件即可。

2.2.1 索引的构建

由于设计了 K/V 分离存储的模式，加载索引时可以直接将 map 文件 **mmap** 加载到内存中去，直接对字符串解析，mmap 这一读取方式因为少了一次拷贝，以及省去与 map 无关的大量的磁盘寻址操作，理论上来讲，索引构建可以减少一半以上的时间。

2.3 缓存

这里选取了比较常用的 LRU 缓存策略，依然是针对每一个分片设计一个缓存，使得命中更加有效，在数据结构上选取了 std::list + std::unordered_map 的组合，有些类似于java 中 linkedlist ，目的就是为了避免读取缓存时性能降到 O(N) 的尴尬局面，同索引一样，每一个分片对应一个缓存。

三, 关建功能代码部分

3.1 数据读取 GET

根据key 哈希找到对应的桶，先查找 Cache，有则返回，无则查询索引。

3.2 数据写入 SET

根据key 哈希找到对应的桶，追加写入，超过桶容量的70%，即进入扩容。

3.3 存储文件重整

当某个分片的存储占用达到 70% 以上，就将此分片扩容为原来的两倍，这里有一个小细节上的设计，扩容时，首先创建新的文件，遍历此分片所对应的索引，根据索引项遍历去查找 value 分别写到新文件中，移动过程中，写请求也随机转移到新的文件中去，**待转移完毕之后直接 rename 扩容文件名为原来的文件名**，这样在重新恢复的时候省去了加载文件表，提高了效率。

3.4 数据删除 DELETE

删除操作与写文件基本相同，唯一区别就是 valueLength 置为 0 来标记这是一条删除的记录，加载这条数据时根据 value=0, 即可读取 key ,根据 key 去索引中删除已存在项。

3.5 获取服务器状态 STATAS

遍历每个分片中的数据叠加起来即可。
