Atitit 数据库核心技术index索引技术 btree hash lsm fulltxt

目录

[1.1. HASH索引、 1](#_Toc20923)

[1.2. LSM索引 LSM算法(Log Structured-Merge Tree) 2](#_Toc27321)

[1.3. BTREE 2](#_Toc6264)

[1.4. Fulltxt比较 lsm适合高并发的数据写入，btree更适合大量查询 3](#_Toc10687)

## HASH索引、

先用最简单索引结构来分析，hash索引，hash是一个非常基本的算法，具体可以看链接。这里列举一个主要利用hash存储引擎的数据库Bitcask，官方文档详细的介绍了其实现原理，简单来说就是这个该数据库在内存中是一种hash表结构，hash结构维护key到文件存储的position，数据仅支持追加操作，所有的操作都是append增加而不更改老数据，数据文件分为新数据文件和老数据文件，老的数据文件只读不写，只有一个数据更新，及活跃数据文件。  
  
这里列举一个主要利用hash存储引擎的数据库Bitcask，官方文档详细的介绍了其实现原理，简单来说就是这个该数据库在内存中是一种hash表结构，hash结构维护key到文件存储的position，数据仅支持追加操作，所有的操作都是append增加而不更改老数据，数据文件分为新数据文件和老数据文件，老的数据文件只读不写，只有一个数据更新，及活跃数据文件。  
  
数据结构如下:  
  
Bitcask在内存中保存了key,value的位置关系，索引关系，key对应数据key, value保存不是实际的数据值，而是保存对应文件的相关信息，从图中可以看到是 file\_id, value\_sz, value\_ops, tstamp，由于内存实际不保存真实数据，所有存储容量就会大大提升。  
  
file\_id 代表保存的具体文件id  
  
value\_sz 代表value大小  
  
value\_ops 代表在文件中具体位置  
  
tstamp 保存时间  
  
然后还有一个点是，该数据实现的高性能是依赖于文件追加的方式，相当于LSM的数据追加，数据增加，删除，更新都是追加操作，形成新的数据版本，老的数据还是会在磁盘上，所以操作对应的io都是顺序io，不会随机访问磁盘，所以写入性能得到了保证，然后对于过期的数据，脏数据，bitcask会定期在后台进行一个merge操作，将inactive的数据清除。对于读操作就不需要过多介绍了，通过key可以直接读取到values\_文件位置，性能也能得到保证。

## LSM索引 LSM算法(Log Structured-Merge Tree)

引入LSM算法可以保证写入性能的情况下支持还范围排序的。  
  
首先说一下什么是LSM算法(Log Structured-Merge Tree), 最初LSM来自于Google分布式表格系统BigTable的论文，它是bigtables的文件组织方式。目前LSM已经应用到了多种数据库领域，最简单的是LevelDB，然后是facebook针对其更改的Roskdb，后来又产生了带上Hbase, cassandra数据库，在mongdb Wired Tiger也能看到，最近也出现在了MySQL领域，已经被广泛验证的存储引擎方式。  
  
SM会维护一个内存有序的Memtable，在内存中维护这样一个有序的数据结构会很容易，像红黑树，平衡树，当有数据插入是先写入到Memtable保持平衡，有序，当Memtable到达一定大小会触发Merge操作，首先将源Memtable指针赋值给一个不可变的MemTable, 这里叫Immutalbe Memtable，然后会重新生成一个Memtable，  
  
LSM的写操作和Bitcask一样，是通过append log的方式，只是会附加一点需要保持数据在内存中有序，不过插入，更新，删除都还是顺序io，性能会有一定保证。然后后台会定期合并各个sstable，清除不必要数据。这里可能引入的一个性能瓶颈是读取性能，如果是查询一个不存在的key，会先查找Memtable，然后会查询最新的sstable文件，这里会引入一个布隆过滤器来解决该问题，简单来说他能很快的判定某数据集中是否存在某个key。  
  
最后简单来说，在保证一定写入和读取性能的条件下，比hash index的优势在于，提供了有序的数据集，支持了范围查询。

## BTREE

上边介绍的hash和lsm都不是最应用广泛的索引，btree才是应用最广泛的数据结构。  
  
B+索引是数据库中最常见，也是使用最频繁的索引结构，下面这里可以通过一些基础数据结构来解释会好一点。  
  
相信大家都了解过一种基础的算法，二分查找法，以有序中点为间隔，使用跳跃式的方式，能快速将一组有序的数据结构中找到需要查找的值。  
  
有序性能提高查询效率，有序性也能给数据库带来支持像范围查询提供条件。  
  
其次为了更好的提升查询性能，我们应该减少树的层级结构，  
  
从这里看出，为了保持内存中树的平衡，维护btree+这样一个数据结构，需要做很多工作，性能会有一定降低，主要是插入，更新，删除都不是上边两种方式，通过append 文件，这里的插入，更新，删除，会查找到具体的内存位置，然后是磁盘位置，进行磁盘数据的更新，这回产生大量的随机IO, 写入，更新，删除，这类型的随机IO会更多。  
  
LSM和Btree由于实现原理的不同，所以不过也都有各有优缺点，LSM由于顺序写，提供更快的写入性能，Btree由于更加平衡，读取性能会更强。

Fulltxt ****FST Tree**** 倒排索引存储

五．

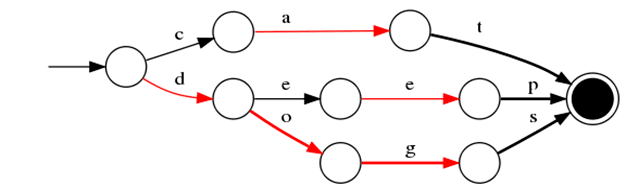
对 Mysql 来说，采用 B+树索引，对 Elasticsearch/Lucene 来说，是倒排索引。倒排索引(Inverted Index)也叫反向索引，有反向索引必有正向索引。通俗地讲，正向索引是通过 key 找 value，反向索引则是通过 value 找 key。

Elasticsearch 是建立在全文搜索引擎库 Lucene 基础上的搜索引擎，它隐藏了 Lucene 的复杂性，提供一套简单一致的 RESTful API。Elasticsearch 的倒排索引其实就是 Lucene 的倒排索引。

****FST Tree****

一种有限状态转移机，优点：1）空间占用小。通过对词典中单词前缀和后缀的重复利用，压缩了存储空间；2）查询速度快。O(len(str))的查询时间复杂度。

示例：对“cat”、 “deep”、 “do”、 “dog” 、“dogs”这 5 个单词进行插入构建 FST（必须已排序），得到如下一个有向无环图。



FST 压缩率一般在 3 倍~20 倍之间，相对于 TreeMap/HashMap 的膨胀 3 倍，内存节省就有 9 倍到 60 倍！

## 比较 lsm适合高并发的数据写入，btree更适合大量查询

虽然这些年层出不穷的出现了各种数据库，但是其实很多数据库底层的存储引擎还是有很多相似性，可以看到各类存储引擎实现方式应用的场景都有一定的不同，比如hash索引更适合简单的kv存储，适合做缓存，lsm适合高并发的数据写入，btree更适合大量查询的应用场景，所以在实际应用的时候，我们需要考虑我们的应用场景，