HBase的读写缓存

参考答案：

HBase上RegionServer的cache主要分为两个部分：**MemStore & BlockCache。**

MemStore是写缓存，BlockCache是读缓存。

当数据写入HBase时，会先写入memstore，RegionServer会给每个region提供一个memstore，memstore中的数据达到系统设置的阈值后，会触发flush将memstore中的数据刷写到磁盘。

客户的读请求会先到memstore中查数据，若查不到就到blockcache中查，再查不到就会从磁盘上读，并把读入的数据同时放入blockcahce。由于BlockCache采用的是LRU策略，因此BlockCache达到上限heapsize*hfile.block.cache .size * 0.85后，会启动淘汰机制，淘汰掉最老的一批数据。

BlockCache

为了高效获取数据，HBase设置了BlockCache机制，内存中缓存block，Block大体来分为两类，一类是JVM的heap内存，一类是heap off内存；**第一类的cache策略叫做LRUCache**，**第二类Cache策略有SlabCache以及BucketCache两类**。BlockCache是Region Server级别的，一个Region Server只有一个Block Cache，在Region Server启动的时候完成Block Cache的初始化工作。到目前为止，**HBase先后实现了3种Block Cache方案**，LRUBlockCache是最初的实现方案，也是默认的实现方案；HBase 0.92版本实现了第二种方案SlabCache；HBase 0.96之后官方提供了另一种可选方案BucketCache。

1、LRUBlockCache

LRUBlockCache是目前hbase默认的BlockCache机制，实现机制也比较简单，是使用一个ConcurrentHashMap管理BlockKey到Block的映射关系，缓存Block只需要将BlockKey和对应的Block放到该HashMap中，查询缓存就根据BlockKey从HashMap中获取即可。同时该方案采用严格的LRU淘汰算法，当BlockCache总量达到一定阈值之后就会启动淘汰机制，最近最少使用的Block会被置换出来。

LRUBlockCache将缓存分为三块：**single-access区、mutil-access区、in-memory区**，分别占到整个BlockCache大小的25%、50%、25%。Block Cache的实现机制核心思想是将Cache分级，这样的好处是避免Cache之间相互影响，尤其是对HBase来说像Meta表这样的Cache应该保证高优先级。

- single-access 优先级：当一个数据块第一次从HDFS读取时，它会具有这种优先级，并且在缓存空间需要被回收（置换）时，它属于优先被考虑范围内。它的优点在于：一般被扫描（scanned）读取的数据块，相较于之后会被用到的数据块，更应该被优先清除。
- mutil-access优先级：如果一个数据块，属于Single Access优先级，但是之后被再次访问，则它会升级为Multi Access优先级。在缓存里的内容需要被清除（置换）时，这部分内容属于次要被考虑的范围。
- in-memory-access优先级：表示数据可以常驻内存，一般用来存放访问频繁、数据量小的数据，比如元数据，用户也可以在建表的时候通过设置列族属性IN-MEMORY= true将此列族放入in-memory区。

加入Block Cache

- 这里假设不会对同一个已经被缓存的BlockCacheKey重复放入cache操作。
- 根据inMemory标志创建不同类别的CachedBlock对象：若inMemory为true则创建BlockPriority.MEMORY类型，否则创建BlockPriority.SINGLE；注意，这里只有这两种类型的Cache，因为BlockPriority.MULTI在Cache Block被重复访问时才进行创建。

公众号 旧时光大数据

- 将BlockCacheKey和创建的CachedBlock对象加入到全局的ConcurrentHashMap map中，同时做一些更新计数操作。
- 最后判断如果加入后的Block Size大于设定的临界值且当前没有淘汰线程运行，则调用runEviction()方法启动LRU淘汰过程。其中，EvictionThread线程即是LRU淘汰的具体实现线程。

淘汰Block Cache

EvictionThread线程主要用于与主线程的同步，从而完成Block Cache的LRU淘汰过程。EvictionThread线程启动后，调用wait被阻塞住，直到EvictionThread线程的evict方法被主线程调用时执行notify，开始执行LruBlockCache的evict方法进行真正的淘汰过程：

1. 首先获取锁，保证同一时刻只有一个淘汰线程运行；
2. 计算得到当前Block Cache总大小currentSize及需要被淘汰释放掉的大小bytesToFree，如果bytesToFree小于等于0则不进行后续操作；
3. 初始化创建三个BlockBucket队列，分别用于存放Single、Multi和InMemory类Block Cache，其中每个BlockBucket维护了一个CachedBlockQueue，按LRU淘汰算法维护该BlockBucket中的所有CachedBlock对象；
4. 遍历记录所有Block Cache的全局ConcurrentHashMap，加入到相应的BlockBucket队列中；
5. 将以上三个BlockBucket队列加入到一个优先级队列中，按照各个BlockBucket超出bucketSize的大小顺序排序（见BlockBucket的compareTo方法）；
6. 遍历优先级队列，对于每个BlockBucket，通过Math.min(overflow, (bytesToFree - bytesFreed) / remainingBuckets)计算出需要释放的空间大小，这样做可以保证尽可能平均地从三个BlockBucket中释放指定的空间；具体实现过程详见BlockBucket的free方法，从其CachedBlockQueue中取出即将被淘汰掉的CachedBlock对象；
7. 进一步调用了LruBlockCache的evictBlock方法，从全局ConcurrentHashMap中移除该CachedBlock对象，同时更新相关计数；
8. 释放锁，完成善后工作。

弊端：随着数据从single-access区晋升到multi-access区或者长时间停留在single-access区，对应的内存对象会从young区晋升到old区，晋升到old区的Block被淘汰后变为内存垃圾，最终由CMS回收。使用LRUBlockCache缓存机制会因为CMS GC策略导致内存碎片过多，从而可能引发Full GC，触发stop-the-world。

2、SlabCache

内部结构是划分为两块，80%和20%；缓存的数据如小于等于blocksize，则放在在前面的区域（80%区域）；如果block大于1x但是小于2x将会放置到后面区域（20%区域）；如果大于2x则不进行缓存。和LRUBlockCache相同，SlabCache也使用LRU算法对过期Block进行淘汰。和LRUBlockCache不同的是，SlabCache淘汰Block的时候只需要将对应的bufferbyte标记为空闲，后续cache对其上的内存直接进行覆盖即可。

线上集群环境中，不同表不同列族设置的BlockSize都可能不同，很显然，默认只能存储两种固定大小Block的SlabCache方案不能满足部分用户场景。因此HBase实际实现中将SlabCache和LRUBlockCache搭配使用，称为DoubleBlockCache。一次随机读中，一个Block块从HDFS中加载出来之后会在两个Cache中分别存储一份；缓存读时首先在LRUBlockCache中查找，如果Cache Miss再在SlabCache中查找，此时如果命中再将该Block放入LRUBlockCache中。

弊端：SlabCache设计中固定大小内存设置会导致实际内存使用率比较低，而且使用LRUBlockCache缓存Block依然会因为JVM GC产生大量内存碎片。因此在HBase 0.98版本之后，该方案已经被不建议使用。

公众号·知识星球·旧时光大数据

3、BucketCache

BucketCache通过配置可以工作在三种模式下：heap，offheap和file。无论工作在那种模式下，BucketCache都会申请许多带有固定大小标签的Bucket，和SlabCache一样，一种Bucket存储一种指定BlockSize的数据块，但和SlabCache不同的是，BucketCache会在初始化的时候申请14个不同大小的Bucket，而且即使在某一种Bucket空间不足的情况下，系统也会从其他Bucket空间借用内存使用，不会出现内存使用率低的情况。heap模式表示这些Bucket是从JVM Heap中申请，offheap模式使用DirectByteBuffer技术实现堆外内存存储管理，而file模式使用类似SSD的高速缓存文件存储数据块。

弊端：HBase将BucketCache和LRUBlockCache搭配使用，称为CombinedBlockCache。和DoubleBlockCache不同，系统在LRUBlockCache中主要存储Index Block和Bloom Block，而将Data Block存储在BucketCache中。因此一次随机读需要首先在LRUBlockCache中查到对应的Index Block，然后再到BucketCache查找对应数据块。BucketCache通过更加合理的设计修正了SlabCache的弊端，极大降低了JVM GC对业务请求的实际影响，但也存在一些问题，比如使用堆外内存会存在拷贝内存的问题，一定程度上会影响读写性能。

欢迎加入知识星球，获取《大数据面试题 V4.0》以及更多大数据开发内容

公众号《知识星球·旧时光大数据》