主要是从HBase应用程序设计与开发的角度，总结几种常用的性能优化方法。  
  
**Auto Flash**  
  
通过调用HTable.setAutoFlushTo(false)方法可以将HTable写客户端自动flush关闭，这样可以批量写入数据到HBase，而不是有一条put就执行一次更新，只有当put填满客户端写缓存的时候，才会向HBase服务端发起写请求。默认情况下auto flush是开启的。  
  
**Write Buffer**  
  
通过调用HTable.setWriteBufferSize(writeBufferSize)方法可以设置HTable客户端的写buffer大小，如果新设置的buffer小于当前写buffer中的数据时，buffer将会被flush到服务端。其中，writeBufferSize的单位是byte字节数，可以根基实际写入数据量的多少来设置该值。  
  
**WAL Flag**  
  
在HBase中，客户端向集群中的RegionServer提交数据时（Put/Delete操作），首先会写到WAL（Write Ahead Log）日志，即HLog，一个RegionServer上的所有Region共享一个HLog，只有当WAL日志写成功后，再接着写MemStore，然后客户端被通知提交数据成功，如果写WAL日志失败，客户端被告知提交失败，这样做的好处是可以做到RegionServer宕机后的数据恢复。  
  
对于不太重要的数据，可以在Put/Delete操作时，通过调用Put.setWriteToWAL(false)或Delete.setWriteToWAL(false)函数，放弃写WAL日志，以提高数据写入的性能。  
  
注：如果关闭WAL日志，一旦RegionServer宕机，Put/Delete的数据将会无法根据WAL日志进行恢复。  
  
**Compression 压缩**  
  
数据量大，边压边写也会提升性能的，毕竟IO是大数据的最严重的瓶颈，哪怕使用了SSD也是一样。众多的压缩方式中，推荐使用SNAPPY。从压缩率和压缩速度来看，性价比最高。

[Java] 纯文本查看 复制代码

[?](http://www.aboutyun.com/)

|  |  |
| --- | --- |
| 1  2 | HColumnDescriptor hcd = new HColumnDescriptor(familyName);  hcd.setCompressionType(Algorithm.SNAPPY); |

**批量写**  
  
通过调用HTable.put(Put)方法可以将一个指定的row key记录写入HBase，同样HBase提供了另一个方法：通过调用HTable.put(List<Put>)方法可以将指定的row key列表，批量写入多行记录，这样做的好处是批量执行，只需要一次网络I/O开销，这对于对数据实时性要求高，网络传输RTT高的情景下可能带来明显的性能提升。  
  
**多线程并发**  
  
**写**在客户端开启多个 HTable 写线程，每个写线程负责一个 HTable 对象的 flush 操作，这样结合定时 flush 和写 buffer（writeBufferSize），可以既保证在数据量小的时候，数据可以在较短时间内被 flush（如1秒内），同时又保证在数据量大的时候，写 buffer 一满就及时进行 flush。  
  
**批量读**  
  
通过调用HTable.get(Get) 方法可以根据一个指定的 row key 获取一行记录，同样 HBase 提供了另一个方法：通过调用HTable.get(List) 方法可以根据一个指定的 row key 列表，批量获取多行记录，这样做的好处是批量执行，只需要一次网络 I/O 开销，这对于对数据实时性要求高而且网络传输 RTT 高的情景下可能带来明显的性能提升。  
  
**缓存查询结果**  
  
对于频繁查询 HBase 的应用场景，可以考虑在应用程序中做缓存，当有新的查询请求时，首先在缓存中查找，如果存在则直接返回，不再查询 HBase；否则对 HBase 发起读请求查询，然后在应用程序中将查询结果缓存起来。至于缓存的替换策略，可以考虑 LRU 等常用的策略。  
  
**HBase数据表优化预分区**  
  
默认情况下，在创建HBase表的时候会自动创建一个Region分区，当导入数据的时候，所有的HBase客户端都向Region写数据，知道这个Region足够大才进行切分，一种可以加快批量写入速度的方法是通过预先创建一些空的Regions，这样当数据写入HBase的时候，会按照Region分区情况，在进群内做数据的负载均衡。  
  
**Rowkey优化**  
  
rowkey是按照字典存储，因此设置rowkey时，要充分利用排序特点，将经常一起读取的数据存储到一块，将最近可能会被访问的数据放到一块。  
  
rowkey若是递增生成的，建议不要使用正序直接写入，可以使用字符串反转方式写入，使得rowkey大致均衡分布，这样设计的好处是能将RegionServer的负载均衡，否则容易产生所有新数据都在集中在一个RegionServer上堆积的现象，这一点还可以结合table的与分区设计。  
  
**减少Column Family数量**  
  
不要在一张表中定义太多的column family。目前HBase并不能很好的处理超过2-3个column family的表，因为某个column family在flush的时候，它临近的column family也会因关联效应被触发flush，最终导致系统产生更过的I/O;  
  
**设置最大版本数**  
  
创建表的时候，可以通过HColumnDescriptor.setMaxVersions(int maxVersions) 设置表中数据的最大版本，如果只需要保存最新版本的数据，那么可以设置 setMaxVersions(1)。  
  
**缓存策略（setCaching）**  
  
创建表的时候，可以通过HColumnDEscriptor.setInMemory(true)将表放到RegionServer的缓存中，保证在读取的时候被cache命中。  
  
**设置存储生命期**  
  
创建表的时候，可以通过HColumnDescriptor.setTimeToLive(int timeToLive)设置表中数据的存储生命周期，过期数据将自动被删除。  
  
**磁盘配置**  
  
每台RegionServer管理10-1000个Regions。每个Region在1-2G，则每台server最少要10G，最大要1000\*2G=2TB，考虑3备份，需要6TB。方案1是3块2TB磁盘，2是12块500G磁盘，带宽足够时，后者能提供更大的吞吐率，更细力度的冗余备份，更快速的单盘故障恢复。  
  
**分配何时的内存给RegionServer**  
  
在不影响其他服务的情况下，越大越好。在HBase的conf目录下的hbase-env.sh的最后添加export HBASE\_REGIONSERVER\_OPTS="- Xmx16000m $HBASE\_REGIONSERVER\_OPTS"  
其中16000m为分配给REgionServer的内存大小。  
  
**写数据的备份数**  
  
备份数与读性能是成正比，与写性能成反比，且备份数影响高可用性。有两种配置方式，一种是将hdfs-site.xml拷贝到hbase的conf目录下，然后在其中添加或修改配置项dfs.replication的值为要设置的备份数，这种修改所有的HBase用户都生效。另一种方式是改写HBase代码，让HBase支持针对列族设置备份数，在创建表时，设置列族备份数，默认为3，此种备份数支队设置的列族生效。  
  
**客户端一次从服务器拉取的数量**  
  
通过配置一次拉取较大的数据量可以减少客户端获取数据的时间，但是他会占用客户端的内存，有三个地方可以进行配置

1. 在HBase的conf配置文件中进行配置hbase.client.scanner.caching;
2. 通过调用HTble.setScannerCaching(int scannerCaching)进行配置；
3. 通过调用Sacn.setCaching(int caching)进行配置，三者的优先级越来越高。

**客户端拉取的时候指定列族**scan是指定需要column family，可以减少网络传输数据量，否则默认scan操作会返回整行所有column family的数据  
**拉取完数据之后关闭ResultScanner**通过 scan 取完数据后，记得要关闭 ResultScanner，否则 RegionServer 可能会出现问题（对应的 Server 资源无法释放）。  
  
**RegionServer的请求处理IO线程数**  
  
较少的IO线程适用于处理单次请求内存消耗较高的Big Put场景（大容量单词Put或设置了较大cache的scan，均数据Big Put）或RegionServer的内存比较紧张的场景。  
  
较多的IO线程，适用于单次请求内存消耗低，TPS要求（每次事务处理量）非常高的场景。这只该值的时候，以监控内存为主要参考  
在hbase-site.xml配置文件中配置项为hbase.regionserver.handle.count  
  
**Region大小设置**  
  
配置项hbase.hregion.max.filesize，所属配置文件为hbase-site.xml，默认大小是256m。  
  
在当前RegionServer上单个Region的最大存储空间，单个Region超过该值时，这个Region会被自动split成更小的Region。小Region对split和compaction友好，因为拆分Region或compact小Region里的StoreFile速度非常快，内存占用低。缺点是split和compaction会很频繁，特别是数量较多的小Region不同的split，compaction，会导致集群响应时间波动很大，Region数量太多不仅给管理上带来麻烦，设置会引起一些HBase个bug。一般 512M 以下的都算小 Region。大 Region 则不太适合经常 split 和 compaction，因为做一次 compact 和 split 会产生较长时间的停顿，对应用的读写性能冲击非常大。  
  
此外，大 Region 意味着较大的 StoreFile，compaction 时对内存也是一个挑战。如果你的应用场景中，某个时间点的访问量较低，那么在此时做 compact 和 split，既能顺利完成 split 和 compaction，又能保证绝大多数时间平稳的读写性能。compaction 是无法避免的，split 可以从自动调整为手动。只要通过将这个参数值调大到某个很难达到的值，比如 100G，就可以间接禁用自动 split(RegionServer 不会对未到达 100G 的 Region 做 split)。  
  
再配合 RegionSplitter 这个工具，在需要 split 时，手动 split。手动 split 在灵活性和稳定性上比起自动 split 要高很多，而且管理成本增加不多，比较推荐 online 实时系统使用。内存方面，小 Region 在设置 memstore 的大小值上比较灵活，大 Region 则过大过小都不行，过大会导致 flush 时 app 的 IO wait 增高，过小则因 StoreFile 过多影响读性能。