HBase 性能优化

1、修改Linux配置
Linux系统最大可打开文件数一般默认的参数值是1024，如果你不进行修改并发量上来的时候会出现“Too Many Open Files”的错误，导致整个HBase不可运行，你可以用ulimit -n 命令进行修改，或者修改/etc/security/limits.conf 和/proc/sys/fs/file-max的参数，具体如何修改可以去Google 关键字“linux limits.conf ”

2、修改 JVM 配置
修改hbase-env.sh文件中的配置参数
HBASE_HEAPSIZE 4000 #HBase使用的 JVM 堆的大小
HBASE_OPTS "-server -XX:+UseConcMarkSweepGC"JVM #GC 选项
HBASE_MANAGES_ZK false #是否使用Zookeeper进行分布式管理

3、修改HBase配置
zookeeper.session.timeout
默认值：3分钟（180000ms）
说明：RegionServer与Zookeeper间的连接超时时间。当超时时间到后，ReigonServer会被Zookeeper从RS集群清单中移除，HMaster收到移除通知后，会对这台server负责的regions重新balance，让其他存活的RegionServer接管.
调优：
这个timeout决定了RegionServer是否能够及时的failover。设置成1分钟或更低，可以减少因等待超时而被延长的failover时间。
不过需要注意的是，对于一些Online应用，RegionServer从宕机到恢复时间本身就很短的（网络闪断，crash等故障，运维可快速介入），如果调低timeout时间，反而会得不偿失。因为当ReigonServer被正式从RS集群中移除时，HMaster就开始做balance了（让其他RS根据故障机器记录的WAL日志进行恢复）。当故障的RS在人工介入恢复后，这个balance动作是毫无意义的，反而会使负载不均匀，给RS带来更多负担。特别是那些固定分配regions的场景。

4、修改HBase配置:hbase-site.xml
hbase.regionserver.handler.count
默认值：10
说明：RegionServer的请求处理IO线程数。
调优：
这个参数的调优与内存息息相关。
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（大容量单次PUT或设置了较大cache的scan，均属于Big PUT）或ReigonServer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要求非常高的场景。设置该值的时候，以监控内存为主要参考。
这里需要注意的是如果server的region数量很少，大量的请求都落在一个region上，因快速充满memstore触发flush导致的读写锁会影响全局TPS，不是IO线程数越高越好。
压测时，开启Enabling RPC-level logging，可以同时监控每次请求的内存消耗和GC的状况，最后通过多次压测结果来合理调节IO线程数。

5、修改HBase配置

hbase.hregion.max.filesize

默认值：256M
说明：在当前ReigonServer上单个Reigon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。
调优：
小region对split和compaction友好，因为拆分region或compact小region里的storefile速度很快，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不停地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，甚至会

引发一些Hbase的bug。
一般512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。
当然，大region也有其用武之地。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
既然split和compaction如此影响性能，有没有办法去掉？
compaction是无法避免的，split倒是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，相反，管理成本增加不多，比较推荐online实时系统使用。平稳的读写性能。
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store file过多影响读性能。

6、修改HBase配置
hbase.regionserver.global.memstore.upperLimit/lowerLimit

默认值：0.4/0.35
upperlimit说明：hbase.hregion.memstore.flush.size 这个参数的作用是当单个Region内所有的memstore大小总和超过指定值时，flush该region的所有memstore。RegionServer的flush是通过将请求添加一个队列，模拟生产消费模式来异步处理的。那这里就有一个问题，当队列来不及消费，产生大量积压请求时，可能会导致内存陡增，最坏的情况是触发OOM。
这个参数的作用是防止内存占用过大，当ReigonServer内所有region的memstores所占用内存总和达到heap的40%时，HBase会强制block所有的更新并flush这些region以释放所有memstore占用的内存。
lowerLimit说明： 同upperLimit，只不过lowerLimit在所有region的memstores所占用内存达到Heap的35%时，不flush所有的memstore。它会找一个memstore内存占用最大的region，做个别flush，此时写更新还是会被block。lowerLimit算是一个在所有region强制flush导致性能降低前的补救措施。在日志中，表现为“** Flush thread woke up with memory above low water.”
调优：这是一个Heap内存保护参数，默认值已经能适用大多数场景。
参数调整会影响读写，如果写的压力大导致经常超过这个阀值，则调小读缓存hfile.block.cache.size增大该阀值，或者Heap余量较多时，不修改读缓存大小。
如果在高压情况下，也没超过这个阀值，那么建议你适当调小这个阀值再做压测，确保触发次数不要太多，然后还有较多Heap余量的时候，调大hfile.block.cache.size提高读性能。
还有一种可能性是hbase.hregion.memstore.flush.size保持不变，但RS维护了过多的region，要知道 region数量直接影响占用内存的大小。

7、修改HBase配置
hfile.block.cache.size

默认值：0.2
说明：storefile的读缓存占用Heap的大小百分比，0.2表示20%。该值直接影响数据读的性能。
调优：当然是越大越好，如果写比读少很多，开到0.4-0.5也没问题。如果读写较均衡，0.3左右。如果写比读多，果断默认吧。设置这个值的时候，你同时要参考“hbase.regionserver.global.memstore.upperLimit”，该值是memstore占heap的最大百分比，两个参数一个影响读，一个影响写。如果两值加起来超过80-90%，会有OOM的风险，谨慎设置。
HBase上Regionserver的内存分为两个部分，一部分作为Memstore，主要用来写；另外一部分作为BlockCache，主要用于读。
写请求会先写入Memstore，Regionserver会给每个region提供一个Memstore，当Memstore满64MB以后，会启动 flush刷新到磁盘。当Memstore的总大小超过限制时（heapsize * hbase.regionserver.global.memstore.upperLimit * 0.9），会强行启动flush进程，从最大的Memstore开始flush直到低于限制。
读请求先到Memstore中查数据，查不到就到BlockCache中查，再查不到就会到磁盘上读，并把读的结果放入BlockCache。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。
一个Regionserver上有一个BlockCache和N个Memstore，它们的大小之和不能大于等于heapsize * 0.8，否则HBase不能启动。默认BlockCache为0.2，而Memstore为0.4。对于注重读响应时间的系统，可以将 BlockCache设大些，比如设置BlockCache=0.4，Memstore=0.39，以加大缓存的命中率。

8、修改HBase配置
hbase.hstore.blockingStoreFiles

默认值：7
说明：在flush时，当一个region中的Store（Coulmn Family）内有超过7个storefile时，则block所有的写请求进行compaction，以减少storefile数量。
调优：block写请求会严重影响当前regionServer的响应时间，但过多的storefile也会影响读性能。从实际应用来看，为了获取较平滑的响应时间，可将值设为无限大。如果能容忍响应时间出现较大的波峰波谷，那么默认或根据自身场景调整即可。

9、修改HBase配置
hbase.hregion.memstore.block.multiplier

默认值：2
说明：当一个region里的memstore占用内存大小超过hbase.hregion.memstore.flush.size两倍的大小时，block该region的所有请求，进行flush，释放内存。
虽然我们设置了region所占用的memstores总内存大小，比如64M，但想象一下，在最后63.9M的时候，我Put了一个200M的数据，此时memstore的大小会瞬间暴涨到超过预期的hbase.hregion.memstore.flush.size的几倍。这个参数的作用是当memstore的大小增至超过hbase.hregion.memstore.flush.size 2倍时，block所有请求，遏制风险进一步扩大。
调优： 这个参数的默认值还是比较靠谱的。如果你预估你的正常应用场景（不包括异常）不会出现突发写或写的量可控，那么保持默认值即可。如果正常情况下，你的写请求量就会经常暴长到正常的几倍，那么你应该调大这个倍数并调整其他参数值，比如hfile.block.cache.size和hbase.regionserver.global.memstore.upperLimit/lowerLimit，以预留更多内存，防止HBase server OOM。

10、修改HBase配置

hbase.hregion.memstore.mslab.enabled

默认值：true
说明：减少因内存碎片导致的Full GC，提高整体性能。
调优： Arena Allocation，是一种GC优化技术，它可以有效地减少因内存碎片导致的Full GC，从而提高系统的整体性能。本文介绍Arena Allocation的原理及其在Hbase中的应用-MSLAB。
开启MSLAB ：
hbase.hregion.memstore.mslab.enabled=true // 开启MSALB
hbase.hregion.memstore.mslab.chunksize=2m // chunk的大小，越大内存连续性越好，但内存平均利用率会降低
hbase.hregion.memstore.mslab.max.allocation=256K // 通过MSLAB分配的对象不能超过256K，否则直接在Heap上分配，256K够大了

-

1.