第 7 章 HBase 优化

7.1 高可用

在 HBase 中 Hmaster 负责监控 RegionServer 的生命周期，均衡 RegionServer 的负载，如果 Hmaster 挂掉了，那么整个 HBase 集群将陷入不健康的状态，并且此时的工作状态并不会维持太久。所以 HBase 支持对 Hmaster 的高可用配置。

1） 关闭 HBase 集群（如果没有开启则跳过此步）

```
[atguigu@hadoop102 hbase]$ bin/stop-hbase.sh
```

2） 在 conf 目录下创建 backup-masters 文件

```
[atguigu@hadoop102 hbase]$ touch conf/backup-masters
```

3） 在 backup-masters 文件中配置高可用 HMaster 节点

```
[atguigu@hadoop102 hbase]$ echo hadoop103 > conf/backup-masters
```

4） 将整个 conf 目录 scp 到其他节点

```
[atguigu@hadoop102 hbase]$ scp -r conf/ hadoop103:/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/
[atguigu@hadoop102 hbase]$ scp -r conf/ hadoop104:/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/
```

5） 打开页面测试查看

0.98 版本之前：http://hadooo102:60010

0.98 版本及之后：http://hadooo102:16010

7.2 预分区

每一个 region 维护着 startRow 与 endRowKey，如果加入的数据符合某个 region 维护的 rowKey 范围，则该数据交给这个 region 维护。那么依照这个原则，我们可以将数据索要投放的分区提前大致的规划好，以提高 HBase 性能。

1）手动设定预分区

```
hbase> create 'staff','info','partition1',SPLITS => ['1000','2000','3000','4000']
```

2）生成 16 进制序列预分区

```
create 'staff2','info','partition2',{NUMREGIONS => 15, SPLITALGO => 'HexStringSplit'}
```

3）按照文件中设置的规则预分区

创建 splits.txt 文件内容如下：

```
aaaa
bbbb
cccc
dddd
```

然后执行：

```
create 'staff3','partition3',SPLITS_FILE => 'splits.txt'
```

4）使用 JavaAPI 创建预分区

```
//自定义算法，产生一系列 Hash 散列值存储在二维数组中
byte[][] splitKeys = 某个散列值函数
//创建 HBaseAdmin 实例
HBaseAdmin hAdmin = new HBaseAdmin(HBaseConfiguration.create());
//创建 HTableDescriptor 实例
HTableDescriptor tableDesc = new HTableDescriptor(tableName);
//通过 HTableDescriptor 实例和散列值二维数组创建带有预分区的 HBase 表
hAdmin.createTable(tableDesc, splitKeys);
```

7.3 RowKey 设计

一条数据的唯一标识就是 rowkey，那么这条数据存储于哪个分区，取决于 rowkey 处于哪个一个预分区的区间内，设计 rowkey 的主要目的 ，就是让数据均匀的分布于所有的 region 中，在一定程度上防止数据倾斜。接下来我们就谈一谈 rowkey 常用的设计方案。

1）生成随机数、hash、散列值

```
比如：
原本 rowKey 为 1001 的，SHA1 后变成：dd01903921ea24941c26a48f2cec24e0bb0e8cc7
原本 rowKey 为 3001 的，SHA1 后变成：49042c54de64a1e9bf0b33e00245660ef92dc7bd
原本 rowKey 为 5001 的，SHA1 后变成：7b61dec07e02c188790670af43e717f0f46e8913
在做此操作之前，一般我们会选择从数据集中抽取样本，来决定什么样的 rowKey 来 Hash 后作为每个分区的临界值。
```

2） 字符串反转

```
20170524000001 转成 10000042507102
20170524000002 转成 20000042507102
```

这样也可以在一定程度上散列逐步 put 进来的数据。

3）字符串拼接

```
20170524000001_a12e
20170524000001_93i7
```

7.4 内存优化

HBase 操作过程中需要大量的内存开销，毕竟 Table 是可以缓存在内存中的，一般会分配整个可用内存的 70%给 HBase 的 Java 堆。但是不建议分配非常大的堆内存，因为 GC 过程持续太久会导致 RegionServer 处于长期不可用状态，一般 16~48G 内存就可以了，如果因为框架占用内存过高导致系统内存不足，框架一样会被系统服务拖死。

7.5 基础优化

1）允许在 HDFS 的文件中追加内容

hdfs-site.xml、hbase-site.xml

属性：dfs.support.append 解释：开启 HDFS 追加同步，可以优秀的配合 HBase 的数据同步和持久化。默认值为 true。

2）优化 DataNode 允许的最大文件打开数

hdfs-site.xml

属性：dfs.datanode.max.transfer.threads 解释：HBase 一般都会同一时间操作大量的文件，根据集群的数量和规模以及数据动作，设置为 4096 或者更高。默认值：4096

3）优化延迟高的数据操作的等待时间

hdfs-site.xml

属性：dfs.image.transfer.timeout 解释：如果对于某一次数据操作来讲，延迟非常高，socket 需要等待更长的时间，建议把该值设置为更大的值（默认 60000 毫秒），以确保 socket 不会被 timeout 掉。

4）优化数据的写入效率

mapred-site.xml

属性： mapreduce.map.output.compress mapreduce.map.output.compress.codec 解释：开启这两个数据可以大大提高文件的写入效率，减少写入时间。第一个属性值修改为 true，第二个属性值修改为：org.apache.hadoop.io.compress.GzipCodec 或者其他压缩方式。

5）优化 DataNode 存储

属性：dfs.datanode.failed.volumes.tolerated 解释：默认为 0，意思是当 DataNode 中有一个磁盘出现故障，则会认为该 DataNode shutdown 了。如果修改为 1，则一个磁盘出现故障时，数据会被复制到其他正常的 DataNode 上，当前的 DataNode 继续工作。

6）设置 RPC 监听数量

hbase-site.xml

属性：hbase.regionserver.handler.count 解释：默认值为 30，用于指定 RPC 监听的数量，可以根据客户端的请求数进行调整，读写请求较多时，增加此值。

7）优化 HStore 文件大小

hbase-site.xml

属性：hbase.hregion.max.filesize 解释：默认值 10737418240（10GB），如果需要运行 HBase 的 MR 任务，可以减小此值，因为一个 region 对应一个 map 任务，如果单个 region 过大，会导致 map 任务执行时间过长。该值的意思就是，如果 HFile 的大小达到这个数值，则这个 region 会被切分为两个 Hfile。

8）优化 hbase 客户端缓存

hbase-site.xml

属性：hbase.client.write.buffer 解释：用于指定 HBase 客户端缓存，增大该值可以减少 RPC 调用次数，但是会消耗更多内存，反之则反之。一般我们需要设定一定的缓存大小，以达到减少 RPC 次数的目的。

9）指定 scan.next 扫描 HBase 所获取的行数

hbase-site.xml

属性：hbase.client.scanner.caching 解释：用于指定 scan.next 方法获取的默认行数，值越大，消耗内存越大。

10）flush、compact、split 机制

当 MemStore 达到阈值，将 Memstore 中的数据 Flush 进 Storefile；compact 机制则是把 flush 出来的小文件合并成大的 Storefile 文件。split 则是当 Region 达到阈值，会把过大的 Region 一分为二。

涉及属性：

即：128M 就是 Memstore 的默认阈值

hbase.hregion.memstore.flush.size：134217728

即：这个参数的作用是当单个 HRegion 内所有的 Memstore 大小总和超过指定值时，flush 该 HRegion 的所有 memstore。RegionServer 的 flush 是通过将请求添加一个队列，模拟生产消费模型来异步处理的。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。

hbase.regionserver.global.memstore.upperLimit：0.4 hbase.regionserver.global.memstore.lowerLimit：0.38

即：当 MemStore 使用内存总量达到 hbase.regionserver.global.memstore.upperLimit 指定值时，将会有多个 MemStores flush 到文件中，MemStore flush 顺序是按照大小降序执行的，直到刷新到 MemStore 使用内存略小于 lowerLimit