HBase最佳实践－读性能优化策略

[日期：2016-11-16] 来源：有态度的HBase 作者：范欣欣 [字体：大 中 小]

任何系统都会有各种各样的问题，有些是系统本身设计问题，有些却是使用姿势问题。HBase也一样，在真实生产线上大家或多或少都会遇到很多问题，有些是HBase还需要完善的，有些是我们确实对它了解太少。总结起来，大家遇到的主要问题无非是Full GC异常导致宕机问题、RIT问题、写吞吐量太低以及读延迟较大。

Full GC问题之前在一些文章里面已经讲过它的来龙去脉，主要的解决方案目前主要有两方面需要注意，一方面需要查看GC日志确认是哪种Full GC，根据Full GC类型对JVM参数进行调优，另一方面需要确认是否开启了BucketCache的offheap模式，建议使用LRUBlockCache的童鞋尽快转移到BucketCache来。当然我们还是很期待官方2.0.0版本发布的更多offheap模块。

RIT问题，我相信更多是因为我们对其不了解，具体原理可以戳 这里 ，解决方案目前有两个，优先是使用官方提供的HBCK进行修复(HBCK本人一直想拿出来分享，但是目前案例还不多，等后面有更多案例的话再拿出来说)，使用之后还是解决不了的话就需要手动修复文件或者元数据表。

而对于写吞吐量太低以及读延迟太大的优化问题，笔者也和很多朋友进行过探讨，这篇文章就以读延迟优化为核心内容展开，具体分析HBase进行读延迟优化的那些套路，以及这些套路之后的具体原理。希望大家在看完之后能够结合这些套路剖析自己的系统。

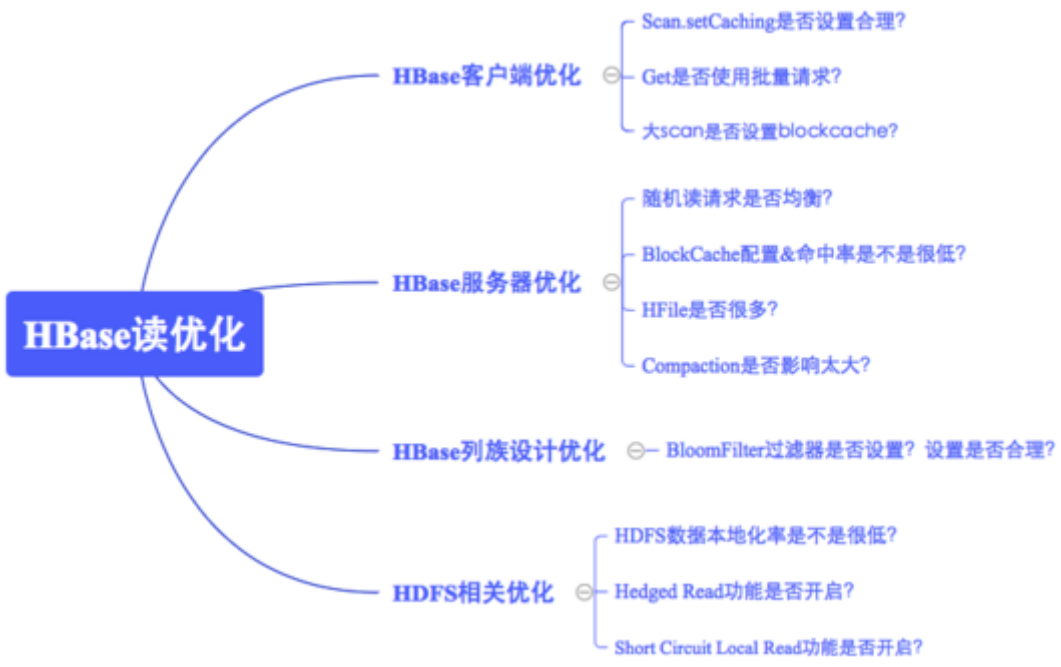
一般情况下，读请求延迟较大通常存在三种场景，分别为：

1. 仅有某业务延迟较大，集群其他业务都正常

2. 整个集群所有业务都反映延迟较大

3. 某个业务起来之后集群其他部分业务延迟较大

这三种场景是表象，通常某业务反应延迟异常，首先需要明确具体是哪种场景，然后针对性解决问题。下图是对读优化思路的一点总结，主要分为四个方面：客户端优化、服务器端优化、列族设计优化以及HDFS相关优化，下面每一个小点都会按照场景分类，文章最后进行归纳总结。下面分别进行详细讲解：



HBase客户端优化

和大多数系统一样，客户端作为业务读写的入口，姿势使用不正确通常会导致 本业务读延迟较高 实际上存在一些使用姿势的推荐用法，这里一般需要关注四个问题：

1. scan缓存是否设置合理?

优化原理：在解释这个问题之前，首先需要解释什么是scan缓存，通常来讲一次scan会返回大量数据，因此客户端发起一次scan请求，实际并不会一次就将所有数据加载到本地，而是分成多次RPC请求进行加载，这样设计一方面是因为大量数据请求可能会导致网络带宽严重消耗进而影响其他业务，另一方面也有可能因为数据量太大导致本地客户端发生OOM。在这样的设计体系下用户会首先加载一部分数据到本地，然后遍历处理，再加载下一部分数据到本地处理，如此往复，直至所有数据都加载完成。数据加载到本地就存放在scan缓存中，默认100条数据大小。

通常情况下，默认的scan缓存设置就可以正常工作的。但是在一些大scan(一次scan可能需要查询几万甚至几十万行数据)来说，每次请求100条数据意味着一次scan需要几百甚至几千次RPC请求，这种交互的代价无疑是很大的。因此可以考虑将scan缓存设置增大，比如设为500或者1000就可能更加合适。笔者之前做过一次试验，在一次scan扫描10w+条数据量的条件下，将scan缓存从100增加到1000，可以有效降低scan请求的总体延迟，延迟基本降低了25%左右。

优化建议：大scan场景下将scan缓存从100增大到500或者1000，用以减少RPC次数

2. get请求是否可以使用批量请求?

优化原理：HBase分别提供了单条get以及批量get的API接口，使用批量get接口可以减少客户端到RegionServer之间的RPC连接数，提高读取性能。另外需要注意的是，批量get请求要么成功返回所有请求数据，要么抛出异常。

优化建议：使用批量get进行读取请求

3. 请求是否可以显示指定列族或者列?

优化原理：HBase是典型的列族数据库，意味着同一列族的数据存储在一起，不同列族的数据分开存储在不同的目录下。如果一个表有多个列族，只是根据Rowkey而不指定列族进行检索的话不同列族的数据需要独立进行检索，性能必然会比指定列族的查询差很多，很多情况下甚至会有2倍～3倍的性能损失。

优化建议：可以指定列族或者列进行精确查找的尽量指定查找

4. 离线批量读取请求是否设置禁止缓存?

优化原理：通常离线批量读取数据会进行一次性全表扫描，一方面数据量很大，另一方面请求只会执行一次。这种场景下如果使用scan默认设置，就会将数据从HDFS加载出来之后放到缓存。可想而知，大量数据进入缓存必将其他实时业务热点数据挤出，其他业务不得不从HDFS加载，进而会造成明显的读延迟毛刺

优化建议：离线批量读取请求设置禁用缓存，scan.setBlockCache(false)

HBase服务器端优化

一般服务端端问题一旦导致业务读请求延迟较大的话，通常是集群级别的，即整个集群的业务都会反映读延迟较大。可以从4个方面入手：

1. 读请求是否均衡?

优化原理：极端情况下假如所有的读请求都落在一台RegionServer的某几个Region上，这一方面不能发挥整个集群的并发处理能力，另一方面势必造成此台RegionServer资源严重消耗(比如IO耗尽、handler耗尽等)，落在该台RegionServer上的其他业务会因此受到很大的波及。可见，读请求不均衡不仅会造成本身业务性能很差，还会严重影响其他业务。当然，写请求不均衡也会造成类似的问题，可见负载不均衡是HBase的大忌。

观察确认：观察所有RegionServer的读请求QPS曲线，确认是否存在读请求不均衡现象

优化建议：RowKey必须进行散列化处理(比如MD5散列)，同时建表必须进行预分区处理

2. BlockCache是否设置合理?

优化原理：BlockCache作为读缓存，对于读性能来说至关重要。默认情况下BlockCache和Memstore的配置相对比较均衡(各占40%)，可以根据集群业务进行修正，比如读多写少业务可以将BlockCache占比调大。另一方面，BlockCache的策略选择也很重要，不同策略对读性能来说影响并不是很大，但是对GC的影响却相当显著，尤其BucketCache的offheap模式下GC表现很优越。另外，HBase 2.0对offheap的改造(HBASE-11425)将会使HBase的读性能得到2～4倍的提升，同时GC表现会更好!

观察确认：观察所有RegionServer的缓存未命中率、配置文件相关配置项一级GC日志，确认BlockCache是否可以优化

优化建议：JVM内存配置量 < 20G，BlockCache策略选择LRUBlockCache;否则选择BucketCache策略的offheap模式;期待HBase 2.0的到来!

3. HFile文件是否太多?

优化原理：HBase读取数据通常首先会到Memstore和BlockCache中检索(读取最近写入数据&热点数据)，如果查找不到就会到文件中检索。HBase的类LSM结构会导致每个store包含多数HFile文件，文件越多，检索所需的IO次数必然越多，读取延迟也就越高。文件数量通常取决于Compaction的执行策略，一般和两个配置参数有关：hbase.hstore.compactionThreshold和hbase.hstore.compaction.max.size，前者表示一个store中的文件数超过多少就应该进行合并，后者表示参数合并的文件大小最大是多少，超过此大小的文件不能参与合并。这两个参数不能设置太'松'(前者不能设置太大，后者不能设置太小)，导致Compaction合并文件的实际效果不明显，进而很多文件得不到合并。这样就会导致HFile文件数变多。

观察确认：观察RegionServer级别以及Region级别的storefile数，确认HFile文件是否过多

优化建议：hbase.hstore.compactionThreshold设置不能太大，默认是3个;设置需要根据Region大小确定，通常可以简单的认为hbase.hstore.compaction.max.size = RegionSize / hbase.hstore.compactionThreshold

4. Compaction是否消耗系统资源过多?

优化原理：Compaction是将小文件合并为大文件，提高后续业务随机读性能，但是也会带来IO放大以及带宽消耗问题(数据远程读取以及三副本写入都会消耗系统带宽)。正常配置情况下Minor Compaction并不会带来很大的系统资源消耗，除非因为配置不合理导致Minor Compaction太过频繁，或者Region设置太大情况下发生Major Compaction。

观察确认：观察系统IO资源以及带宽资源使用情况，再观察Compaction队列长度，确认是否由于Compaction导致系统资源消耗过多

优化建议：

(1)Minor Compaction设置：hbase.hstore.compactionThreshold设置不能太小，又不能设置太大，因此建议设置为5～6;hbase.hstore.compaction.max.size = RegionSize / hbase.hstore.compactionThreshold

(2)Major Compaction设置：大Region读延迟敏感业务(100G以上)通常不建议开启自动Major Compaction，手动低峰期触发。小Region或者延迟不敏感业务可以开启Major Compaction，但建议限制流量;

(3)期待更多的优秀Compaction策略，类似于stripe-compaction尽早提供稳定服务

HBase列族设计优化

HBase列族设计对读性能影响也至关重要，其特点是只影响单个业务，并不会对整个集群产生太大影响。列族设计主要从两个方面检查：

1. Bloomfilter是否设置?是否设置合理?

优化原理：Bloomfilter主要用来过滤不存在待检索RowKey或者Row-Col的HFile文件，避免无用的IO操作。它会告诉你在这个HFile文件中是否可能存在待检索的KV，如果不存在，就可以不用消耗IO打开文件进行seek。很显然，通过设置Bloomfilter可以提升随机读写的性能。

Bloomfilter取值有两个，row以及rowcol，需要根据业务来确定具体使用哪种。如果业务大多数随机查询仅仅使用row作为查询条件，Bloomfilter一定要设置为row，否则如果大多数随机查询使用row+cf作为查询条件，Bloomfilter需要设置为rowcol。如果不确定业务查询类型，设置为row。

优化建议：任何业务都应该设置Bloomfilter，通常设置为row就可以，除非确认业务随机查询类型为row+cf，可以设置为rowcol

HDFS相关优化

HDFS作为HBase最终数据存储系统，通常会使用三副本策略存储HBase数据文件以及日志文件。从HDFS的角度望上层看，HBase即是它的客户端，HBase通过调用它的客户端进行数据读写操作，因此HDFS的相关优化也会影响HBase的读写性能。这里主要关注如下三个方面：

1. Short-Circuit Local Read功能是否开启?

优化原理：当前HDFS读取数据都需要经过DataNode，客户端会向DataNode发送读取数据的请求，DataNode接受到请求之后从硬盘中将文件读出来，再通过TPC发送给客户端。Short Circuit策略允许客户端绕过DataNode直接读取本地数据。(具体原理参考 此处)

优化建议：开启Short Circuit Local Read功能，具体配置戳 这里

2. Hedged Read功能是否开启?

优化原理：HBase数据在HDFS中一般都会存储三份，而且优先会通过Short-Circuit Local Read功能尝试本地读。但是在某些特殊情况下，有可能会出现因为磁盘问题或者网络问题引起的短时间本地读取失败，为了应对这类问题，社区开发者提出了补偿重试机制 – Hedged Read。该机制基本工作原理为：客户端发起一个本地读，一旦一段时间之后还没有返回，客户端将会向其他DataNode发送相同数据的请求。哪一个请求先返回，另一个就会被丢弃。

优化建议：开启Hedged Read功能，具体配置参考 这里

3. 数据本地率是否太低**?**

数据本地率：HDFS数据通常存储三份，假如当前RegionA处于Node1上，数据a写入的时候三副本为(Node1,Node2,Node3)，数据b写入三副本是(Node1,Node4,Node5)，数据c写入三副本(Node1,Node3,Node5)，可以看出来所有数据写入本地Node1肯定会写一份，数据都在本地可以读到，因此数据本地率是100%。现在假设RegionA被迁移到了Node2上，只有数据a在该节点上，其他数据(b和c)读取只能远程跨节点读，本地率就为33%(假设a，b和c的数据大小相同)。

优化原理：数据本地率太低很显然会产生大量的跨网络IO请求，必然会导致读请求延迟较高，因此提高数据本地率可以有效优化随机读性能。数据本地率低的原因一般是因为Region迁移(自动balance开启、RegionServer宕机迁移、手动迁移等),因此一方面可以通过避免Region无故迁移来保持数据本地率，另一方面如果数据本地率很低，也可以通过执行major_compact提升数据本地率到100%。

优化建议：避免Region无故迁移，比如关闭自动balance、RS宕机及时拉起并迁回飘走的Region等;在业务低峰期执行major_compact提升数据本地率

HBase读性能优化归纳

在本文开始的时候提到读延迟较大无非三种常见的表象，单个业务慢、集群随机读慢以及某个业务随机读之后其他业务受到影响导致随机读延迟很大。了解完常见的可能导致读延迟较大的一些问题之后，我们将这些问题进行如下归类，读者可以在看到现象之后在对应的问题列表中进行具体定位：

现象：某个业务慢，集群并不慢	
客户端问题	Scan缓存是否设置合理?
	Get请求是否设置为批量?
列族设计问题	Bloomfilter是否设置？是否合理?
	TTL是否设置合理?
HDFS相关问题	本地率是否太低?
现象：集群随机读慢	
HBase服务器端优化	BlockCache配置是否合理？GC参数是否配置合理?
	HFile数是否太多?
	Compaction是否影响很大？是否太过频繁?
HDFS相关优化	Short-Circuit Local Read功能是否开启?
	Hedged Read功能是否开启？（Hadoop 2.4+）
现象：某个业务起来之后其他业务很慢	
客户端问题	大scan是否设置setBlockCache=false?
服务器端问题	读请求是否均衡?

HBase读性能优化总结

性能优化是任何一个系统都会遇到的话题，每个系统也都有自己的优化方式。 HBase作为分布式KV数据库，优化点又格外不同，更多得融入了分布式特性以及存储系统优化特性。文中总结了读优化的基本突破点，有什么不对的地方还望指正，有补充的也可以一起探讨交流!

从零自学Hadoop(21)：HBase数据模型相关操作下

扫雷实用帖：HBase读延迟的12种优化套