5. 第五章 参数与性能优化

5.1. 那些有趣的参数

这里 glusterfs 中提供了非常多的参数，其中不同的参数作用是不同的，而下面将分享一下，关于本人在使用过程中了解和遇到过的使用的一些参数。当然这里要查询不同的参数的作用，想了解参数的用法，可以使用命令 gluster volume set help 输出内容。

5.1.1. rebalance 相关

这里在 rebalance 中有一些比较重要的参数需要关注一下，这个参数就是 rebal-throttle，这个参数就是控制 rebalance 的时候在节点并行处理的文件数量，这里受限于节点的 cpu 数量的，计算的公式是 [($(processing units) - 4) / 2), 4]，在进行 rebalance 的时候，适当提高该参数，可以加快处理的速度。

5.1.2. 服务和性能相关

这里有很多关于读写性能的优化参数，而优化读写的速度，可以优化的地方也是比较多的，下面分享部分常见的。

1. open-behind

这个参数是用于提高文件打开速度的，和匿名 inode 那些有关，因为对于很多文件需要频繁读写打开的时候，每次的文件句柄地创建申请都会比较麻烦，尤其是一些临时文件，因此把该参数设置为 on，可以优化文件的打开参数。

然而这个参数，因为在历史版本中，曾经出现过一个比较大的 bug，该 bug 在 7.x 版

本中出现，表现的现象主要是在 k8s 中挂载 volume 的时候，会突然间挂载掉线，但是 volume 是正常的。原因是因为使用了匿名的 fd，在以前的实现方式中，并没有在 fd_t 的结构中添加引用，因此有可能会导致还在使用的文件句柄被杀死。该问题可以通过关闭该参数解决，如果想了解具体的情况，可以看官方 github 上 issues 为 1225 的内容，那里具体讨论了该情况。

2. config.brick-threads 和 config.client-threads

这个参数是需要设置 config.global-threading 为 on 开启之后才会生效的，该参数的作用就是控制每个 brick 和 clients 最大的线程数量，但是这里设置了，并不代表立马生效的，因为具体的情况，glusterfs 会根据目前的负载来进行处理的。如果当前默认在 120 秒内没有新的请求过来，当前的 xlator 的线程池就会维护最小的数量运行。

3. ctime

这个参数默认是开启的,在生产环境中，我们曾经尝试关闭过该参数，但是发现对于一些依赖于此类时间戳属性的应用程序会中断，如 ES 在检测到 ctime 的差异（如果统计信息来自不同的块）时，将会出现“file changed as we read it”和“Underlying file changed by an external force”的警告,因为不一定总是从副本集的同一块中返回时间属性，而目前该参数发现会对 jira 日志，confluence、elasticsearch 服务有影响。

4. read-hash-mode

因为 glusterfs 的无中心架构，对于每一个节点都是平等地对待，因此这里如何避免解决读写热节点则天然不具备优势，这里不像 k8s 集群那样，可以设置节点标签或者其他的方式，但是对于读取数据的时候，该参数可以做到一定程度避免读取热节点的数据，这个参数有 5 种情况可以选择，1 是默认值，就是根据 hash 来读取其中一个 brick，而 3 则可以找到具有较少没有完成的读请求的 brick 来进行操作，也就是这里会考虑读请求队列的正

在运行的任务情况。而 4 则是考虑网络延迟的，5 则是介于 3 和 4 之间的一个方法。

5. 数据库相关

对于数据库，其实是不推荐使用 glusterfs 作为存储的，因为 glusterfs 相比起 ceph 和 openEBS 这些存储方案，其独特的架构方式和元数据管理方式，会让 k8s+mysql+glusterfs 的组合，使用效率会显得非常低。当然，对于数据库使用 glusterfs 作为存储的话，因为数据库本身是有应用缓存的，因此这里需要进行一些参数优化，主要就是避免使用 io cache 那些，尽量使用直接 IO 那样的方式，加快读写速度，因此下面有一些参数是可以进行参数设置的。

参数名称	参数值
performance.stat-prefetch	off
performance.read-ahead	
performance.write-behind	
performance.readdir-ahead	
performance.io-cache	
performance.quick-read	
performance.open-behind	
performance.strict-o-direct	on

那么这里对于一些参数，如 write-behind，如果写入请求很小，它还会合并连续写入以形成更大的缓冲区，以便它可以一次性发送所有数据。所以 write-behind 的目的是减少 fuse 写路径中网络写的次数。

6. min-free-disk

这个参数是一个比例,brick 对应的节点的磁盘剩余空间比例，默认是 10%。如果是多个 subvolume ,其中某个 subvolume 如果空间达到这个阈值，然后在 gluster 的 client 创建文件夹 那么新的文件夹下这个 subvolume 对应的 dht 的 hash 值就是 0，所有以后在这个文件夹下创建的文件都不会落到这个 subvolume。这个参数是一个优化的参数，但是同时这样的优化也会带来一定的麻烦，那就是当空间快满了以后，调用链会变长了，会导致创建文件变慢，系统表现地比较卡了。

5.1.3. 日志相关

在日常运维过程中，有时候会遇到一些问题，那么正常的日志输出信息是不够的，因此需要 debug 一些日志信息的时候，就需要用到下面两个参数了。

参数	作用
diagnostics.brick-log-level	控制 brick 端的日志级别,影响 glusterd 和 brick 日志输出
diagnostics.client-log-level	控制 client 日志输出级别

5.1.4. heal 相关

关于 heal 自愈，属于一个低优先级的功能，同时默认中每个节点都是一个 heal 进程的，因此这里如果想加快 heal 自愈的速度，那么下面有一些参数可以进行调整。

1. cluster.shd-max-threads 和 cluster.shd-wait-qlength

这两个参数的作用就是设置最大的 glustershd 的进程数量，也就是每个节点的 heal

进程数据，还有每个进程处理的队列长度。另外这里还可以手动进行增加 shd 进程，这里的命令类似如下所示。

```
1. root@gfs01:~#/usr/sbin/glusterfs -s localhost --volfile-id gluster/glustershd -p /var/run/gluster/glustershd/<glustershd-pidfile> -l /var/log/glusterfs/<glustershd-logfile> -S /var/run/gluster/<glustershd-socketid>.socket --xlator-option *replicate*.node-uuid=<gluster-node-uuid>
```

下面的这几个内容是需要进行替换的

① <glustershd-pidfile>：新的存放 pid 信息的文件路径

② <glustershd-logfile>： 新的保存日志的地方

③ <glustershd-socketid>：socket 文件,这里 id 是可以使用 uuidgen 命令获取得

④ <gluster-node-uuid>：节点的 uuid 信息，这里可以通过 peer status 得到

那么实际执行类似下面这样的，然后这里再使用 ps -ef |grep glustershd 就可以看到多出来一个进程了。

```
1. root@gfs01:~#/usr/sbin/glusterfs -s localhost --volfile-id shd/test-replica -p /var/run/gluster/shd/test-replica/test-replica-shd-01.pid -l /var/log/glusterfs/glustershd-01.log -S /var/run/gluster/53fc946f9b1443bf.socket --xlator-option *replicate*.node-uuid=f1c72979-3502-41fb-83f2-45f9d2e47cc2 --process-name glustershd-01 --client-pid=-6
```

另外这里注意的是，进程数和链接数是无关的，也就是说当这里的进程增加之后，新的进程并不会再和所有的 bricks 相连接。

2. cluster.data-self-heal-algorithm

这是选择 heal 的算法，有两个选项，分别是 full 和 diff,区别就是是否全量校验，默认情况下是 full,而使用 diff 的话则可以加快自愈的速度，因为会减少一部分检查。

5.2. linux 系统优化

对于 linux 系统，Glusterfs 因为基于 fuse 开发的，而 IO 和文件系统的一些参数配置优化，有时候可以明显提高系统性能，因此下面来简单了解一下。

1) vm.swappiness

这个参数是控制如何使用 swap 分区的，对于很多分布式系统，大多数建议是直接禁用 swap 分区,避免因为使用该分区而导致性能下降，因此建议对于该参数可以考虑设置为 0。

2) vfs_cache_pressure

该文件表示内核回收用于 directory 和 inode cache 内存的倾向 ;缺省值 100 表示内核将根据 pagecache 和 swapcache，把 directory 和 inode cache 保持在一个合理的百分比；降低该值低于 100，将导致内核倾向于保留 directory 和 inode cache；增加该值超过 100，将导致内核倾向于回收 directory 和 inode cache。因此这里建议可以考虑将该值设置大于 100。

3) vm.dirty_background_ratio 和 vm.dirty_ratio

两者中的第一个（vm.dirty_background_ratio）定义了在后台将页面刷新到磁盘之前可能变脏的内存百分比。在达到此百分比之前，不会将页面刷新到磁盘。但是，当刷新开始时，它会在后台完成，而不会中断前台的任何正在运行的进程。

现在，两个参数中的第二个（vm.dirty_ratio）定义了在强制刷新开始之前脏页可以占用的内存百分比。如果脏页的百分比达到这个阈值，那么所有进程都变得同步，并且在它们请求的 io 操作实际执行并且数据在磁盘上之前不允许它们继续。在高性能 I/O 机器的情况

下，这会导致问题，因为数据缓存被切断并且所有执行 I/O 的进程都被阻塞以等待 I/O。这将导致大量挂起进程，从而导致高负载，从而导致系统不稳定和糟糕的性能。

4) scheduler

这个参数是/sys/block/{DEVICE-NAME}/queue/scheduler 的配置，这里设置的是 IO 调度算法，操作系统中大部分默认的是 none。

```
$ sudo cat /etc/issue
Ubuntu 18.04.5 LTS \n \l

$ sudo cat /sys/block/sda/queue/scheduler
[mq-deadline] none
```

这里默认的调度是 none,那么这里可以考虑 deadline 算法，该算法主要考虑调度任务荒饿问题，可以根据业务和 IO 情况来进行调整。

5) nr_requests 和 queue_depth

nr_requests 是请求的 IO 调度队列大小,queue_depth 是请求在磁盘设备上的队列深度,I/O 调度器中的最大 I/O 操作数是 nr_requests * 2。操作系统中 nr_requests 参数，可以提高系统的吞吐量，似乎越大越好，但是该请求队列的也不能过大，因为这样会消耗大量的内存空间。该值的调整需要综合多处因素。

章节语:

这一章主要关注了 glusterfs 的一些参数和 linux 中的系统优化配置，对于这两部分的内容，因为涉及内容比较多，需要经过大量测试和了解原理后建议才在生产环境中使用。