6. 第六章 运维之路

6.1. 难受的运维经历

对于 glusterfs 的使用，在生产环境中使用的版本是 7.5,而到了写下这段话的时候，目前最新的版本已经是 9.2 了,短短的一两年时间内,glusterfs 的版本迭代非常的迅速，同时也有了很多的优化改进，而因为生产环境的追求稳定，加上当时对 glusterfs 的熟悉程度不够，还有使用场景的特殊性，遇到了很多惨痛的生产环境问题和棘手的需求，下面分享其中一些故事和事故，希望能够给大家在日常运维使用过程中规避一些问题。同时如果你是准备使用 glusterfs，那么能够很好地规划与设计集群的使用。

1) 挂载突然掉线

这个问题是当时生产环境在使用 k8s+glusterfs+heketi 的时候遇到的，时不时会出现挂载掉线的问题，挂载点提示 transport endpoint is not connected,而容器 pod 也并不会因为这个而重启，并且已经无法正常读写数据了，但是这时候 volume 状态是正常的。

后来通过排查日志和官方的 issue 发现，这里是因为存在了一个版本 bug 导致的，这里出现问题的原因是以前实现的时候，文件句柄在使用时并没有添加引用到 fd_t 这个结构中，导致可能还在被使用的连接被销毁。这个问题如果感兴趣的话，可以查看 issude 为 1225 的文档[1]，这里给出了相关的内容。

这个问题在 7.7 以前的版本中会存在 ,对于旧版本的集群 ,可以通过关闭 open-behind 这个参数来进行规避。

2) brick 进程突然收到关闭信号

这个问题是一个很有趣的问题，当时是计划打算把 minio 放到 k8s 中运行，然后底层

存储使用 glusterfs 的，而在测试过程中，突然发现有时候 volume 的 brick 进程会异常，然后通过排查日志发现，在服务端会收到客户端发送的日志，其中关键是有一段信息中包含 Shutting down connection,显示这个是从客户端发送过来的。 后来通过排查客户端的日志，包括查看了 pvc 的日志(日志的位置则是在 k8s 节点中的，前面提到过)，但是并没有发现发送相关信息。接着通过去咨询官方才发现，这是一个遗留问题[2]，同时这个问题在比较早之前 minio 官方的 issue[3]中也曾经有其他使用者提到过,然而目前并没有找到具体的场景来准确复现，因此属于一个长期遗留的问题。而 glusterfs 官方为了进一步排查这些问题，在最新的 9.x 版本中，添加了 core dump 相关的内容[4]，方便进一步排查。

因此后续中就不再考虑把 minio 放在 k8s 中使用了，而其他的应用服务，也暂时没有遇到过该问题。

3) 应用服务对 glusterfs 的不支持

在生产环境中，使用的架构是 k8s+glusterfs+zfs 的模式，而很多服务也是部署在 k8s 上面，在使用一些服务的过程中，调研发现部分服务官方是明确提出不太建议使用 glusterfs 作为存储后端的，这里 nexus 官方的文档中也有提到这个问题[5] 。因此在日常运维使用过程中，对于服务组件的要求，也是需要调研清楚的。

4) 集群节点 peer 状态异常

在一次日常运维集群中，因为集群的部分节点被异常重启，接着检查发现，其中有一个节点 A 出现异常了，使用 peer status 的时候发现，对另外一个节点 B 的连接状态显示为 peer is connected and accepted， 后来检查日志，发现这里是 peer 状态元数据信息不一致，异常的节点 A 的文件/var/lib/glusterd/peers，记录的 B 节点的文件中的 state 和其他节点是不一样的。接着通过咨询官方发现[6]，这里是当前开启了 quota 功能之后，

7.8 以前的版本存在 checksum 异常的 bug，后续通过修改这个异常问题，同时检查 volume 元数据是否一致，重启该异常节点的 glusterd 进程解决的。

5） glusterfshd 进程连接所有的 brick 进程

这里其实属于一个优化内容，因为以前的设计和实现，目前对于每个节点上的 glustershd 进程，都会去连接所有 brick，哪怕当前 brick 不是该节点的，但是对于该 glustershd 进程来说，因为 brick 不在该节点上，因此却又不做自愈等操作，属于一个多余的连接情况，这一点目前也提了一个 issue 给官方[7]，后续待官方修复该问题。

6） zfs+glusterfs 的使用在大量文件下 heal 速度很慢

在生产环境中，这里使用的是 zfs+glusterfs+k8s 的架构，而当 volume 数量逐渐增加，并且有些 volume 存放了一些小文件之后，可以明显感受到 heal 的速度非常慢，而官方有也有使用者提出如果是 zfs+glusterfs，使用 xfs 的文件格式的话，这个问题会比较明显,而目前来说，关于 zfs 文件系统对于小文件的支持,也不是特别理想。当然对于小文件的支持问题，这也是目前工业界的一个难题。

7） 节点负载过高影响集群稳定性

这个问题的出现很有意思,生产环境的物理节点其中有一台出现了很高的负载,64 核的物理机,top 命令显示的 1 分钟和 5 分钟平均负载高达三百多，直接导致了这台机器无法远程登录，也无法进行任何的命令操作，但是这时候还没有导致机器宕机，也就是说其他 glusterfs 集群节点使用 peer status 显示还是正常的，但是这时候，如果 volume 需要同步信息的话，是需要所有节点都有响应的，会导致集群中其他的 volume 服务会受到严重的影响。

对于这个的问题，根本原因就是因为 glusterfs 的无中心架构设计，所有的 volume 信息是每个节点都需要同步的，所以如果一旦出现某个节点负载很高响应很慢，这样对整个集群的影响是非常大的，这一点需要特别留意的。

6.2. 周边生态项目

在使用一个工具的时候，除了当前工具的功能与特性以外，还要考虑工具的周边生态项目，如果周边生态项目的活跃度，也会直接影响到当前项目的使用量与便捷性的，而对于 glusterfs 来说，这里也有一些相关的生态项目，下面简单分享一些使用和遇到的项目。

1) heketi 和 kadalu

在使用 k8s+glusterfs 的架构中,k8s 管理 glusterfs 的 volume 工具，可以使用 heketi 来进行管理,而 heketi 的原理，其实是对块设备进行格式化，做成 lvm2，弄成 vg，而有 pod 要申请 volume 的时候，则从 vg 中创建 lv，然后格式化成 xfs 文件系统格式，接着挂载到系统的目录上面使用，对于这种方式，优缺点是非常明显的。优点是使用这方式创建的 volume,非常方便进行容量监控，因为底层的 vg 信息创建出来的 lv 已经限制了容量，在挂载之后，可以直接使用 exporter 进行检测。同时使用这种方式进行创建的 volume,可以很方便地使用快照功能。

当然这样做的缺点也是显而易见的，首先如果一旦 volume 容量不足，因为底层是 lvm2 的，因此 heketi 的扩容是再次从 vg 中创建一个新的 lv 出来，并且再次格式化挂载，对原来的 volume 进行 add-brick 操作，但是对于 volume 来说则需进行数据的 rebalance 操作，而 rebalance 操作则是一个不确定性非常大的操作，因为对于

容量很大的volume,或者小文件非常多的volume,数据的重平衡rebalance操作时间是没有办法准确预估的。曾经在测试环境中对 nexus 进行扩容操作，两百多 G 的数据重平衡花费了两个多小时，当然这个时间可以通过设置一些参数来减少，但是仍然不是一个短时间内可以完成的操作。另外对于重平衡后的 volume，没有办法完全做到数据的平均分布的，尤其是对于一些数据大小并不是一致的时候，有出现数据倾斜的风险。

其次对于使用块设备进行格式化为 lvm2 的话，这里如果多个服务都是部署在当前的块设备上面，那么对于节点的读写负载是比较高的，同样会影响到其他服务的使用，而 heketi 为了解决这个问题，有一个 tag 标签的功能，可以对不同的块设备在格式化之后打上 tag 标签，然后 storageclass 可以指定对应的标签的块设备来进行创建 volume，实际上就是一个分组的功能，因此这里也需要额外去规划好块设备的分组问题。

另外对于 heketi 来说，因为目前官方已经暂时停止更新了，从最新的 10.3 版本以后，基本上没有了更新，目前官方也不再推荐使用该工具了，而是推荐使用 kadalu 这个相对轻量级一点的工具。相比起 heketi,kadalu 属于一个非常新的工具，在写下目前这段话的时候，目前也并没有 1.x 版本的出现，kadalu 目前最新的是 0.8.3 版本，因此建议可以继续跟进了解。而 kadalu 的使用其实是放弃了块设备进行格式化为 lvm2 的方式，采用了类似直接使用命令 create volume 的方式，还有手动创建之后，挂载到 pod 等方式，会更加直接简单方便管理。

2) ganesha

Nfs-ganesha 是 NFS v3、4.0、4.1 和 4.2 的用户模式文件服务器，考虑使用这个项目的原因，是出于对 volume 的安全管理，因为对于 glusterfs 集群来说，只要知道了 volume 和任意一个节点的 ip，那么就可以随意挂载并且使用了，只要防火

墙允许，因此对于生产环境来说，如果一个 volume 是一个团队使用的话，那么权限管理就比较混乱和麻烦了，而使用 ganesha 项目则可以隐藏 volume 信息。当然因为 ganesha 项目是基于 NFS 的，而 NFS 对于读写性能的损耗也是比较大的，因此如果生产环境使用的话，建议多进行压测。

6.3. 未来还能做什么

Glusterfs 作为无中心架构的分布式文件系统代表，目前来说经历过很多版本的迭代发展，在性能和架构方面有了很大的改变，然而需要优化和改进的地方其实也是很多的，其中一点就是对于很多小文件的读取问题，在测试环境中经过测试，volume 创建了上万个文件以后，执行 ls 等命令都会比较慢，这是因为目前对于文件的元数据信息缓存，并没有做的很好，而国内的 TaoCloud 公司团队，通过给 server 端加入了一个 dcache 层[9]，仅将目录结构信息保存于内存当中，而将元数据保存到嵌入式 KV 数据库 levelDB 当中。这种架构方案的实现，可以大幅度提高文件元数据的读取速度,关于这个架构方案的实现，建议可以具体阅读一下 TaoCloud 团队的技术博客文档。

那么除了这个以外，glusterfs 官方提到了一个 RIO 的技术实现方案[10]，这个方案的核心就是考虑把 volume 的元数据管理和真实数据信息两部分拆开，因为有些 fop 操作是只需要访问元数据的，有些则是两者都访问等，而拆开之后，对于元数据管理部分，可以考虑引入其他的工具进行管理和缓存了，这样未来对于扩展 glusterfs 的架构层次也有了更大的空间，另外关于这个方案的实现思路，可以查看文档[11]。

章节语:

这一章主要关于了 glusterfs 运维过程中遇到的一些问题，和 glusterfs 配合使用的一些相关生态项目，还有未来 glusterfs 的一些架构发展等，对于 glusterfs，目前来说不太建议将数据库、conflunce 和 gitlab 这些作为后端存储使用，而 glusterfs 更加适合那种冷存储数据系统，或者是很大的文件存储系统使用。同时作为运维和开发团队来说，如果使用 glusterfs，建议需要考虑团队技术熟悉程度，否则无中心架构的分布式存储系统，特点与常见的 hdfs 那些有着很多不同的地方，系统规划和设计方面要考虑的内容也会不一样。

附录引用:

[1] https://github.com/gluster/glusterfs/issues/1225
[2] https://github.com/gluster/glusterfs/issues/2473
[3] https://github.com/minio/minio/issues/4993
[4] https://github.com/gluster/glusterfs/issues/1810
[5] https://help.sonatype.com/repomanager3/installation/system-requirements#SystemRequirements-FileSystemstoavoid
[6] https://github.com/gluster/glusterfs/issues/2498
[7] https://github.com/gluster/glusterfs/issues/2594
[8] https://github.com/gluster/glusterfs/issues/11764
[9] https://mp.weixin.qq.com/s/qaRGjUtjG1JUsz43ktIwTw
[10] https://github.com/gluster/glusterfs/issues/243
[11] https://docs.google.com/document/d/1KEwVtSNvDhs4qb63gWx2ulCp5GJjge77NGJk4p_Ms4Q/edit#
[12]