

深入理解GlusterFS之数据均衡

开源云工具

作者: danny_2018

时间: 2018-07-31 17:59:57

1420

0

GlusterFS是一个免费的开源分布式文件系统，具有无中心节点、堆栈式设计、全局统一命名空间、高可用、高性能和横向扩展等特点，在业界被广泛使用。本文主要介绍GlusterFS的数据均衡功能（即rebalance），内容涉及数据均衡的产生背景、使用场景、基本原理、程序实现剖析、操作命令实践、存在的问题以及需要优化的地方等，希望能够抛砖引玉，为读者深入学习和理解GlusterFS起到一定的参考作用。

本文假定读者已经熟悉了GlusterFS的一些基本概念术语以及架构组成。

1. 问题引入

1.1. 问题背景

在当今大数据时代，人们每天都在产生大量的数据，其中需要持久化的数据也越来越多，而一台计算机的存储容量是比较有限的，尽管可以为其增加更多的内存和磁盘，但再怎么增加也是有上限的，因此分布式存储系统便应运而生。分布式存储系统就是利用多个独立的计算机来解决单个计算机无法解决的存储数据量大的问题，对于这种系统而言，扩容（增加节点）和缩容（减少节点）是不可避免会遇到的情况，GlusterFS、Ceph、HDFS和OpenStack Swift等分布式存储系统均如此。

以扩容为例，这是一个十分常见的场景，随着数据量的增多，必然要对系统进行扩容来满足实际需求，而扩容系统后会出现集群内数据分布不均衡的情况，已有数据只存在原有节点上，新增节点只会存储后续新增的数据，这样发展下去会造成原有节点负载过高，而新增节点可能还有很多可用空间，这就产生了不均衡（如图1），为了解决该问题，很多分布式存储系统都支持数据均衡（即rebalance）功能，GlusterFS、Ceph、HDFS和Swift等也是如此，执行数据均衡后，可以使集群中的数据进行重新分布，并且分布的更加均匀（如图2）。

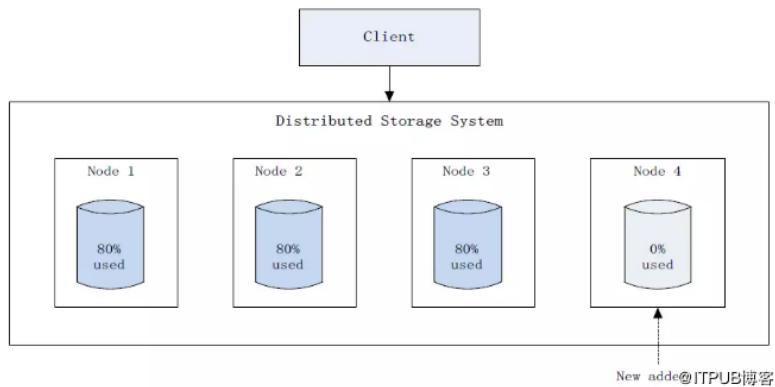


图1 扩容后数据均衡前示意图

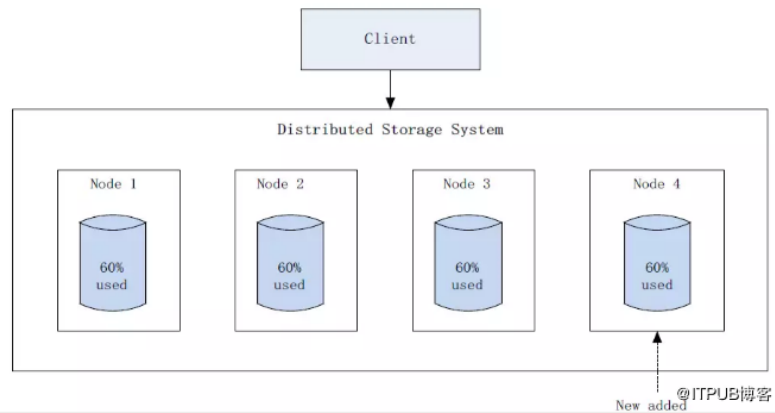


图2 扩容后数据均衡后示意图

本文主要讨论GlusterFS数据均衡，接下来先介绍下GlusterFS数据均衡涉及到的相关内容，尤其是DHT（Distributed Hash Table）部分，因为数据均衡和DHT结合的非常紧密。



danny_2018

注册时间: 2018-07-30

博文量

73

访问量

77569

最新文章

IBM邵萍：数字化重构传统企业IT架构

漂洋过海，与Splunk第一次近距离接...

架构成长这十年，人生沉浮无限多！

中国系统架构师大会精彩继续，“四大...

Kubernetes风头正盛，Docker何去何...

如何让软件开发从功能测试转入应用...

HyperLedger Fabric和区块链是什么关...

为什么说容器和DevOps不分彼此？

10大重点推荐的区块链技术

Kubernetes 再升级，安全和稳定性是...

GlusterFS使用DHT模块来聚合多台机器的物理存储空间，形成一个单一的全局命名空间，并使用卷（Volume）这一逻辑概念来表示这样的空间。每个卷可以包含一个或多个子卷（Subvolume），子卷也可称为DHT子卷，同样是一个逻辑概念，一个子卷可以是单个brick、一个副本卷（Replica）或一个EC（Erasure Coding）卷，而副本卷和EC卷自身又都是由一组brick构成。而brick则是GlusterFS中的最基本存储单元，表示为一个机器上的本地文件系统导出目录。例如，GlusterFS分布式副本卷（双副本）的简化组成如图3所示。

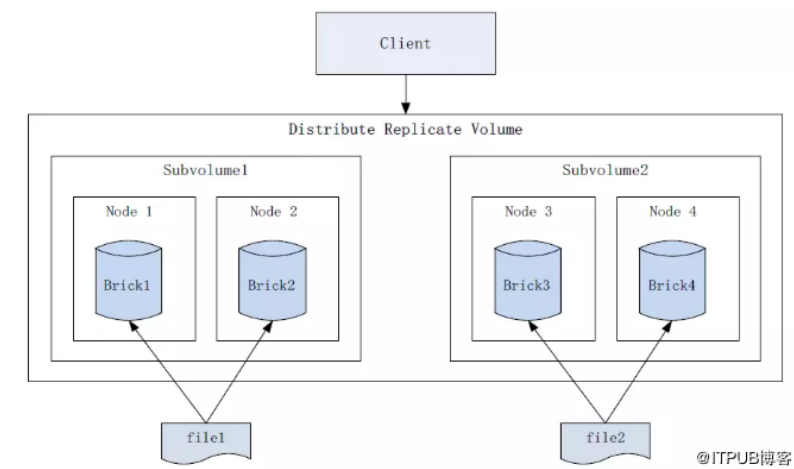


图3 GlusterFS分布式副本卷示意图

DHT模块使用基于32位哈希空间的一致性哈希算法（Davies-Meyer算法）计算文件的哈希值，并将文件存储到其中一个DHT子卷，而目录是GlusterFS中哈希分布（layout）的基本单位，会在所有DHT子卷中都创建，哈希范围保存在目录的扩展属性中。根据DHT算法原理，每一个DHT子卷的目录都会被分配一个哈希子空间，即32位哈希空间（十六进制表示为0x00000000~0xffffffff）中的一段哈希范围，例如，0x00000000~0x55555554。

为了在集群内均衡地分布文件，GlusterFS在每个目录层次上重新划分一次哈希空间，并且子目录层和父目录层的哈希分布并无关联。而文件的实际存储位置，只由其父目录上的哈希范围决定，与其他目录层次无关。子卷的目录哈希空间分布示意图如图4所示。

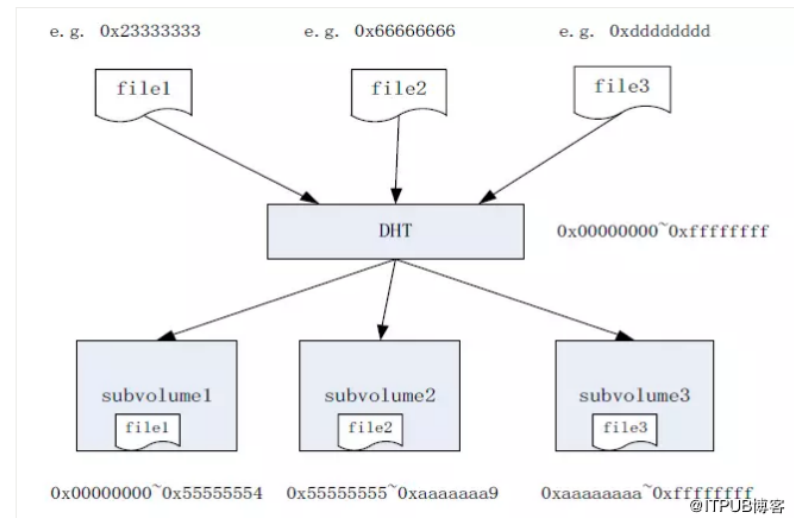


图4 GlusterFS卷目录哈希空间分布示意图

根据以上介绍，可以概括GlusterFS中文件访问的一般流程如下：

- 1、首先，DHT根据文件名称计算出一个哈希值（假设图4中file1哈希值为0x23333333）；
- 2、其次，根据该哈希值找到其所在的目录哈希子空间（即0x00000000~0x55555554），也就是找到了对应的DHT子卷（即subvolume1）；
- 3、最后，访问对应的DHT子卷（即subvolume1）。

1.2. 使用场景

理想情况下，在一个GlusterFS卷中，会尽可能地在DHT子卷之间均衡地存储文件，这样可以充分发挥GlusterFS的高可靠性、高可用性和高性能。但现实中总有一些特殊情况存在，比如，集群需要扩容等，从而导致文件分布不均匀，因此也就需要进行数据均衡。

那么GlusterFS中数据均衡具体指的是什么呢？简而言之，GlusterFS数据均衡就是必要的时候，在DHT各子卷之间迁移数据的过程，其目的是使数据在集群中的不同节点之间尽量均匀分布，从而使得集群处于最佳状态。

那么问题来了，何时才是数据均衡的“必要的时候”呢？主要有如下两类场景：

- (一) 扩容或缩容文件系统
- (二) 重命名文件

1.2.1 扩容或缩容

扩容或缩容GlusterFS按照子卷为单位来做增减，这会使得DHT子卷的数量发生变化，从而导致每个子卷的目录哈希范围会被重新计算和分配，即每个子卷的目录哈希范围会改变。而文件的哈希值并没有变化，如果此时有一些文件的哈希值落到了其他子卷（即不同于文件当前所在子卷），那么这些文件应该被迁移到正确的子卷。

值得注意的是，在扩容GlusterFS后，需要手动执行gluster rebalance命令来触发数据均衡功能。前文已经介绍了，扩容后会带来新旧节点数据不均衡的问题，进一步发展可能会导致旧节点负载过高而出现性能问题，甚至最终影响到数据可靠性和可用性，因此，扩容后进行数据均衡是非常必要的。

而在缩容GlusterFS后，并不需要手动执行命令，缩容时会自动触发执行数据均衡过程，这是因为如果缩容时没有自动进行数据均衡，那么被剔除掉的节点或子卷上的数据将不再可用，从而会导致数据的丢失，这对于用户来说是不可接受的，因此数据均衡在缩容时是不可或缺的，程序实现采用自动触发方式也就理所当然了。

下面分别给出扩容和缩容后，并且执行数据均衡后的子卷目录哈希空间分布变化示意图，如图5和图6所示。

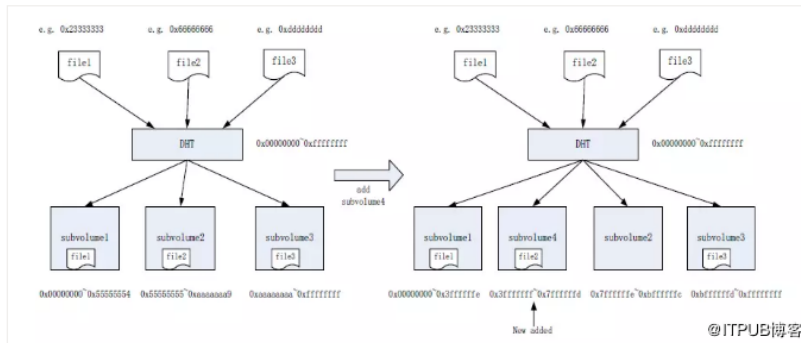


图5 扩容后子卷的目录哈希空间分布变化

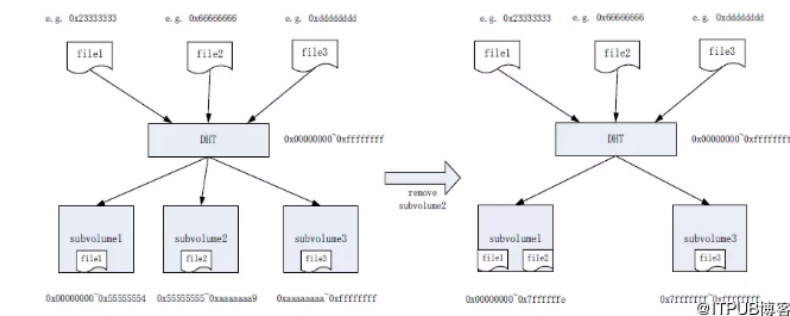


图6 缩容后子卷的目录哈希空间分布变化

1.2.2 重命名文件

在GlusterFS中，重命名文件会导致该文件的哈希值发生变化，假设此时DHT子卷的数量并没有变化，即没有扩容和缩容，那么每个子卷的目录哈希范围也就没有变化，根据DHT算法判断该文件的新存储位置，如果文件的新存储位置与当前所在子卷不同（如图7），则该文件应该被迁移到正确的子卷。

重命名文件后，系统不会自动进行均衡，而是会在目标子卷上产生一个同名的内容为空的链接文件（Link file），该链接文件的扩展属性上会记录文件的实际存储位置（也就是原来所在的子卷）。假设此时客户端访问重命名后的文件，根据前面介绍的文件访问流程，则DHT会先将请求转到哈希计算得出的子卷去查找该文件，并获取到链接文件信息，DHT模块懂得链接文件的意义，从链接文件信息中得出文件的实际位置，然后再到实际的子卷获取文件。

可以发现，如果重命名文件后不进行数据均衡，则客户应用程序在访问文件时会增加额外的步骤，从而造成一定程度的访问延迟，当系统有大量链接文件时，则会导致访问性能的大幅下降，对应用程序造成影响。而执行数据均衡后则会将文件迁移到正确的位置，消除了链接文件带来的访问延迟问题，因此数据均衡对于文件重命名来说也是很有必要的。

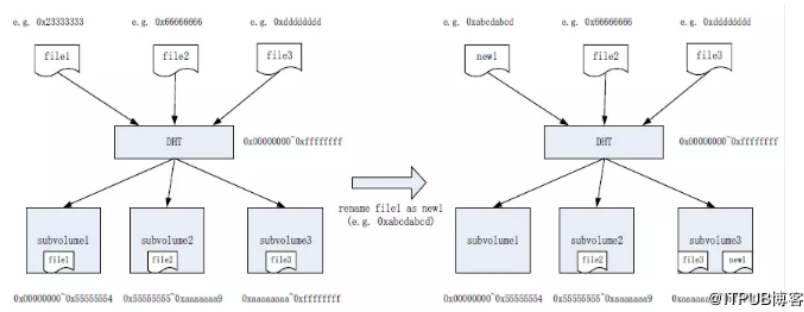


图7 重命名文件导致的文件位置变化

2. 基本原理

前面简要介绍了DHT模块的基本作用和文件的一般访问流程，目的是为了读者能够更好地理解GlusterFS数据均衡。本节将介绍当前数据均衡功能是如何工作的。

随着GlusterFS项目的发展，其数据均衡功能也在不断的完善，其最初的工作机制和现在的工作机制已经相去甚远，下面高度概括了数据均衡当前的流程，如图8所示：

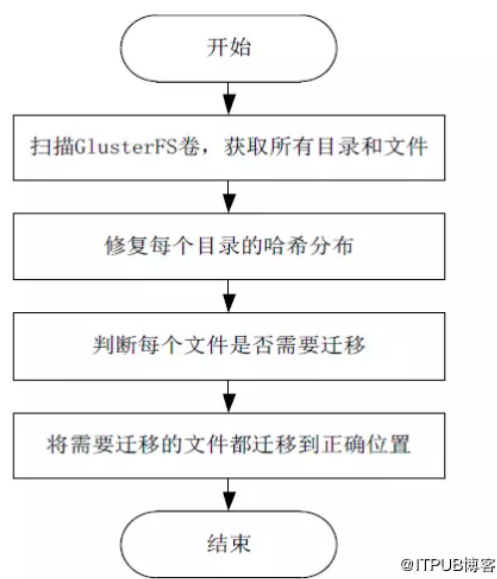


图8 数据均衡的概括流程

为了便于理解，以上流程是站在集群的视角，对数据均衡流程的高度总结，具体如下：

- 1、扫描整个GlusterFS卷，获取到卷中所有目录和文件；
- 2、对每一个目录，修复该目录的哈希分布；
- 3、对每一个文件，判断该文件是否需要迁移；
- 4、对于需要迁移的文件，将其迁移到正确的位置。

可以看到，概括总结后的流程非常简单，但是实际代码实现却比较复杂，一方面是考虑了各种异常情况，会有大量的细节需要处理，另一方面是数据均衡功能的代码与DHT的代码结合的非常紧密，从而增加了代码的复杂度，不便于理解。后面的实现剖析部分会具体介绍代码级别的处理流程。

本节接下来先介绍数据均衡功能稍微细化后的工作机制，然后再给出一个实例分析，用于帮助读者理解。

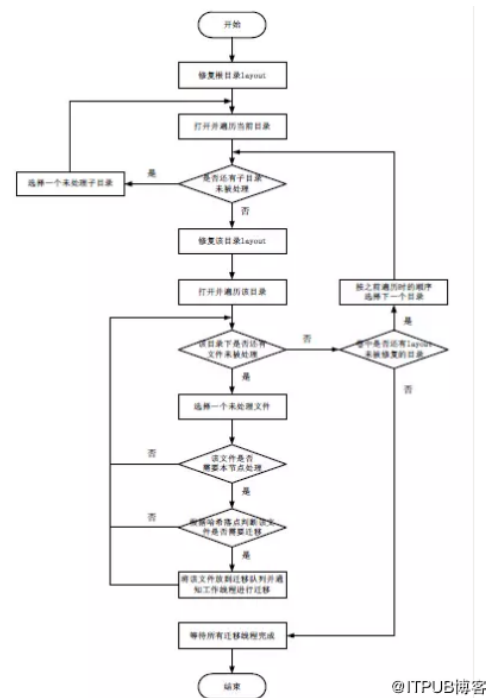
2.1. 工作机制

在GlusterFS的数据均衡功能实现中，每个节点采用单进程多线程的实现方式，其中，主线程使用类似深度优先算法，从根目录开始，遍历GlusterFS卷在本节点上的目录并修复其哈希分布，同时爬取目录下的所有文件，根据算法将相应文件放到迁移队列里，并通知等待的工作线程进行迁移处理。

而工作线程则负责检查迁移队列里是否有文件待迁移，若队列不空则迁移其中的文件，一次迁移一个文件；若队列为空则自我睡眠，等待主线程唤醒。

主线程的文件爬取工作和工作线程的数据迁移工作同时进行，并且支持多个线程并行迁移文件。当主线程完成修复目录和爬取文件工作后，将等待其他工作线程完成数据迁移工作，当迁移队列里的所有文件都被迁到正确位置后，所有工作线程结束退出，主线程收到所有工作线程的退出消息后，做一些清理工作，然后结束本节点的数据均衡进程。

下面给出稍微细化后的数据均衡主线程和迁移线程的工作流程如图9和图10所示。



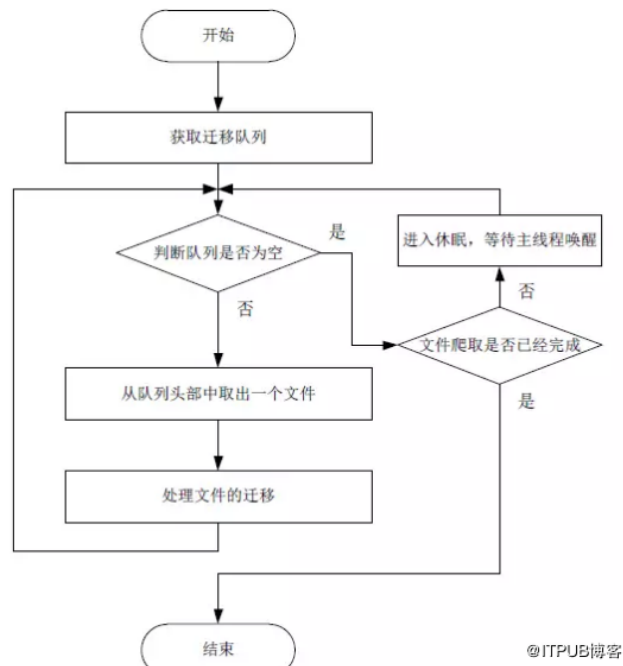
@ITPUB博客

图9 数据均衡主线程的工作机制

启动数据均衡功能后，每个节点的数据均衡进程的主线程都按照上图流程进行，并且每个节点只处理属于本地brick上的文件，上述流程简要说明如下：

- 1、首先，修复卷的根目录哈希分布；
- 2、打开并遍历当前目录，获取到所有子目录；
- 3、对于当前目录中的每个子目录，按照如下步骤递归处理：
 - 3.1、如果有未被处理的子目录，从中选择一个目录，并返回到步骤2处理该子目录；否则，下一步；
 - 3.2、所有子目录都已经被处理过了或者没有子目录，修复当前目录的哈希分布；
 - 3.2.1、打开并遍历当前目录，获取所有文件；
 - 3.2.2、对于当前目录中的每个文件，按照如下步骤处理：
 - 3.2.2.1、如果有未被处理的文件，从中选择一个文件，进入下一步，否则，转到步骤3.2.2.4；
 - 3.2.2.2、如果文件应该由本节点处理，则进入下一步，否则，转到步骤3.2.2；
 - 3.2.2.3、如果文件满足迁移条件，则将文件放到一个迁移队列，并通知迁移线程执行迁移，否则，返回步骤3.2.2；
 - 3.2.2.4、当前目录的所有文件已被处理过了，转入下一步；
- 4、如果本节点还有目录的哈希分布未被修复，选择其中的一个目录（按照之前遍历时的顺序），并转到步骤3，否则，进入下一步；
- 5、等待所有迁移线程完成数据迁移工作，然后做些清理工作，最后结束数据均衡进程。

工作线程的处理流程如下：



@ITPUB博客

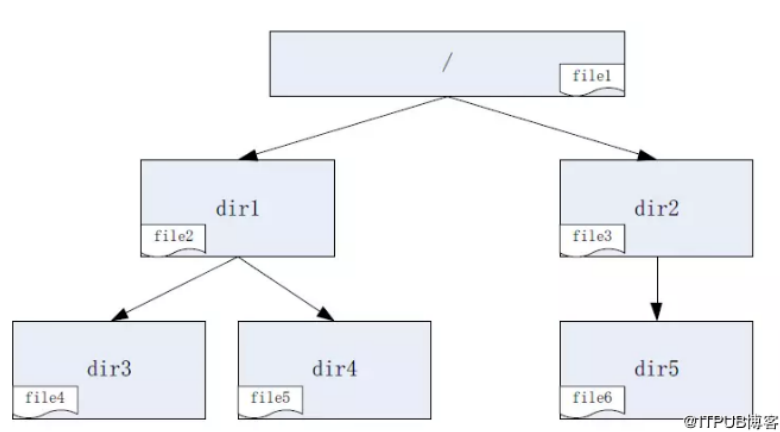
图10 数据均衡迁移线程的工作机制

数据均衡的迁移线程处理流程简介如下：

- 1、获取到数据均衡进程的迁移队列；
- 2、检查队列是否为空；
 - 2.1、如果队列不空，则从队列头部取出一个文件，并处理该文件的具体迁移过程，迁移完成后返回步骤2；
 - 2.2、如果队列为空，则继续判断主线程是否已经完成对整个卷的文件爬取；
 - 2.2.1、如果主线程未完成文件爬取，则迁移线程进入休眠状态，等待主线程的唤醒；
 - 2.2.2、如果主线程已完成文件爬取，则退出线程。

2.2. 实例分析

以上内容可能有些晦涩难懂，为了便于理解，下面举个例子，假设GlusterFS卷中的目录文件布局如图11所示：



@ITPUB博客

图11 GlusterFS卷的目录文件布局示意图

根据上述数据均衡的工作机制，其处理顺序如下：

- 1、数据均衡进程启动，主线程首先修复根目录的哈希分布；
- 2、打开并遍历根目录，获取到文件file1、目录dir1和dir2；
- 3、选择根目录的一个子目录，假设选择dir1；
- 4、打开并遍历dir1，获取到文件file2、目录dir3和dir4；
- 5、选择dir1目录的一个子目录，假设选择dir4；
- 6、打开并遍历dir4，获取到文件file5；
- 7、因为dir4没有子目录，所以不再进行递归遍历，而是修复dir4的哈希分布；
- 8、遍历dir4，获取其下所有文件，本例为file5；
- 9、判断file5是否应该由本节点处理，假设由本节点处理；
- 10、判断file5是否需要迁移，假设需要迁移；

- 11、将file5放到迁移队列，通知工作线程进行迁移（迁移线程负责处理具体的迁移过程）；
- 12、dir4目录下没有其他文件需要处理了，即对dir4的处理完成了，按照之前的遍历顺序，继续处理下一个哈希分布未修复的目录，也就是优先选择与dir4同一层次的其他目录；
- 13、选择dir3，由于dir3也没有子目录，修复dir3的哈希分布；
- 14、打开并遍历dir3，获取到文件file4，然后同file5的处理流程一样，不再赘述；
- 15、处理完dir3后，则当前同一层次的目录都处理完了（即dir3和dir4），开始处理当前目录层次的父目录，即dir1；
- 16、完成对dir1的修复目录、遍历文件和迁移文件，具体步骤同dir4，不再赘述；
- 17、接着上面的步骤3，对dir1的同一层次的其他目录（本例为dir2）进行处理，过程与dir1类似，即先修复dir5哈希分布，迁移其下文件file6，再修复dir2哈希分布，迁移文件file3，具体过程不再赘述；
- 18、最后处理根目录的文件file1，不再赘述；
- 19、数据均衡进程等待所有迁移线程完成数据迁移，然后正常结束。

以上处理顺序的简化示意图如下图12所示：

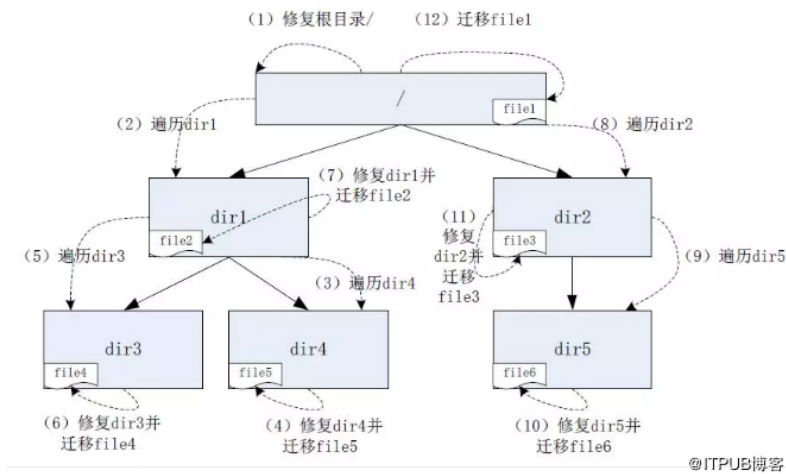


图12 目录处理与文件迁移顺序

总结一下，数据均衡的当前工作机制就是多个节点同时参与处理，每个节点以单进程多线程的方式，同时扫描文件和迁移文件，并且扫描文件和迁移文件分别由不同线程处理，每个节点可以并行迁移文件，相比以往的数据均衡流程，大大增加了并行度，并且更加可扩展，使得集群数据均衡时的系统负载分布的更加均匀，同时效率也更高了。

3. 最佳实践

GlusterFS为数据均衡功能提供了相应的命令行程序，通过该程序可以和glusterd服务进程之间进行通信，用于查询数据均衡状态信息和控制数据均衡相关操作，可以辅助相关人员更好地控制管理数据均衡过程。

3.1. 操作命令

命令格式为：

```
gluster volume rebalance <VOLNAME> {{fix-layout start} | {start [force]]stop|status}}
```

常用命令介绍如下：

(1) 只修复目录layout命令

```
gluster v rebalance VOL_NAME fix-layout start
```

例如：

(2) 开启数据均衡命令

```
gluster v rebalance VOL_NAME start
```

例如：

也可以使用强制选项，这样可以忽略一个容量限制，即不带强制选项时会比较文件所在的原子卷和目标子卷的剩余容量，如果原子卷大于目标子卷，则不迁移该文件。使用强制选项则会跳过这个限制，命令如下：

```
gluster v rebalance VOL_NAME start force
```


例如：

（3）设置数据均衡迁移速度

为了良好的迁移性能，数据均衡进程中使用多线程，并且支持并行迁移多个文件，但是这样占用一定的系统资源，对存储系统本身性能带来一定影响，因此，GlusterFS提供了相关控制命令，用于调控实际迁移速度的快慢，目前主要有三种模式，lazy、normal和aggressive。顾名思义，lazy是懒惰的慢速模式（较少线程迁移），normal是正常模式（线程数量适中），aggressive是激进模式（较多线程迁移），默认采用normal模式，可以根据实际需要去做具体设置，命令格式如下：

```
gluster volume set <VOLNAME> rebal-throttle lazy|normal|aggressive
```

例如：

（4）查看数据均衡状态等信息

```
gluster v rebalance VOL_NAME status
```

例如：

可以查看到数据均衡在每个节点的当前状态（status）、运行了多久（run time）、扫面的文件数量（scanned）、已经迁移的文件数量（rebalanced-files）、已经迁移的数据量（size）、迁移失败的文件数量（failures）和跳过的文件数量（skipped）。

（5）停止数据均衡进程

```
gluster v rebalance VOL_NAME stop
```

例如：

3.2. 均衡建议

当集群需要进行数据均衡时，建议参考如下内容：

- （1）尽量提前做规划，例如，别等到集群存储空间快用完了才扩容，一方面会导致时间紧迫，部署准备时间匆忙，容易忙中出错；另一方面也容易导致旧节点之间的文件迁移失败，最好预留出一定的剩余空间；
- （2）确保集群所有节点处于正常状态，卷处于启动状态，glusterd服务进程和brick进程状态正常，节点之间通信正常；
- （3）检查GlusterFS卷中是否有文件损坏，如果有，则先对其进行修复；
- （4）在执行数据均衡时，确保集群没有自修复操作正在进行，否则会影响到数据正确性和迁移效率；
- （5）如果允许的话，在执行数据均衡前，停止客户端应用，可以提高均衡效率；
- （6）先执行fix-layout操作，再执行数据迁移，可以在一定程度上提高迁移效率；
- （7）根据实际需要选择迁移模式，默认是normal模式，aggressive模式可能占用的系统资源较多，进而影响到存储性能；
- （8）数据均衡过程中，通过命令行程序定时关注均衡的当前状态，以便及时发现问题并做相应调整；
- （9）当集群规模较大时，可能偶尔会出现某个节点均衡失败的情况，一般重新开始执行均衡即可；
- （10）如果执行数据迁移对应用程序影响较大，可以只执行fix layout，这样可以只修复目录的哈希分布，并不会实际迁移文件，此时新文件可以存储到新增节点（或brick）上，之后再找适当时机（系统比较空闲的时候）执行数据迁移操作。

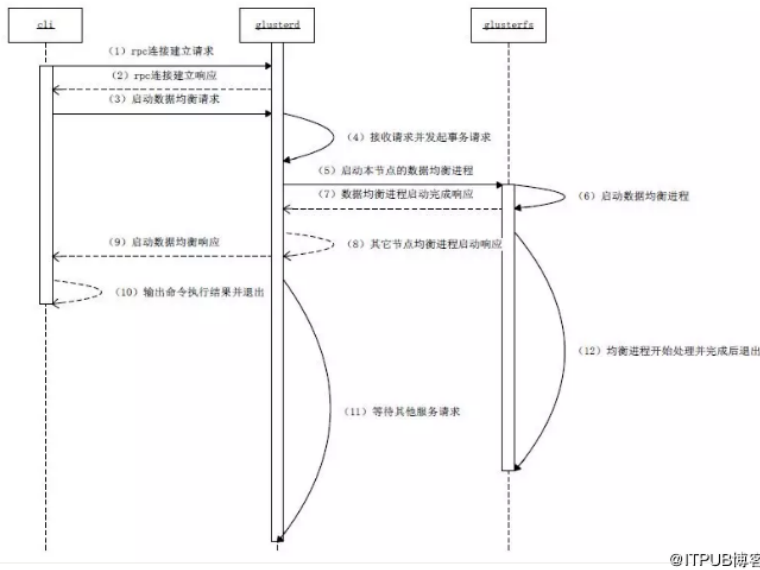
4. 实现剖析

GlusterFS数据均衡功能的代码与DHT层代码紧密结合在一起，主要的功能逻辑代码位于dht-rebalance.c文件中，剩余部分代码包含在DHT层的其他代码逻辑中。在数据均衡代码的实现中，采用了syncop框架，这是一套基于协程的合作式多任务同步框架，它的存在简化了数据均衡功能的实现。

前面的工作机制部分介绍了数据均衡的基本流程，本节则主要侧重代码级别的讨论，首先简要介绍数据均衡功能中涉及的进程交互，然后讨论syncop框架，最后给出数据均衡的几个关键流程。限于时间原因，忽略了很多代码处理细节，留给读者自行分析实践。

4.1. 进程交互过程

下面以命令行手动触发数据均衡为例，简要说明其中涉及的进程交互步骤，在命令执行的节点上，简化的进程交互过程如下图13所示。



@ITPUB博客

图13手动触发数据均衡的进程交互过程

交互过程主要涉及到cli、glusterd和glusterfs三个进程，简要说明如下：

- 1、在集群的一个节点上，执行gluster rebalance start命令（即启动cli进程），该命令向本节点glusterd服务进程发起rpc连接建立请求；
- 2、本节点glusterd进程接收到该请求，与之建立rpc连接并回复响应；
- 3、cli命令进程收到rpc连接建立响应，然后向本节点glusterd进程发起启动数据均衡请求；
- 4、本节点glusterd进程收到请求后，做一些初始化等准备工作，并以事务方式向集群其他相关节点（当前GlusterFS卷的所有其他节点）的glusterd进程，发起启动数据均衡的请求；
- 5、与此同时，本节点glusterd进程启动本节点的数据均衡进程；
- 6、数据均衡进程以gluster客户端进程（即glusterfs）的形式启动，该进程使用的客户端配置文件（即.vol文件）与正常客户端配置文件不同，去掉了其中的性能相关模块；
- 7、本节点的数据均衡进程启动完毕后，返回响应给glusterd进程；
- 8、本节点glusterd进程等待并接收所有其他相关节点的响应，即数据均衡进程已经启动完毕的消息；
- 9、本节点glusterd进程返回启动数据均衡的响应，通知命令行程序启动已经完成；
- 10、命令行程序从本地glusterd进程接收响应，打印输出命令执行结果，然后退出cli程序；
- 11、本节点glusterd进程完成了对命令行请求的处理，等待其他请求到来；
- 12、本节点数据均衡进程开始处理，具体处理流程参见前文中的工作原理部分，均衡进程在处理完成后退出。

4.2. syncop框架

在GlusterFS的几乎所有线程中，文件操作（fops）都是异步执行的，使用的基本原语是STACK_WIND和STACK_UNWIND这一类宏定义，这类宏定义像调用函数和回调函数一样工作，有效地实现了集群和并行环境下的文件操作。

但在数据均衡功能中，如果直接采用这类宏定义实现相关操作，则会使程序变得非常复杂而不好控制，因此，GlusterFS开发人员引入了syncop同步框架，并广泛使用于程序中，以方便数据均衡功能的实现，例如，用于文件爬取程序和流程控制等方面。

4.2.1. syncop术语

- syncenv

Syncenv是同步环境对象，该对象拥有一个可调度的工作线程池，同步任务（synctask）在该环境中调度执行。

- synctask

Synctask是同步任务对象，可以理解为一对C语言函数指针，即调用函数指针synctask_fn_t（如syncop_lookup）和回调函数指针synctask_cbk_t（如syncop_lookup_cbk）。

Synctask有两种操作模式：

- (1) 调用线程等待synctask完成；
- (2) 调用线程调度synctask，使之继续运行；

Synctask能够确保一个函数调用完成后，其对应的回调函数也会被执行到。

4.2.2. synctask生命周期

一个synctask的完整生命周期包括如下几个阶段：

- CREATED

当调用synctask_create或synctask_new函数创建一个synctask时；

- RUNNABLE

当把synctask加入到syncenv的runq队列时；

- RUNNING

当syncenv的工作线程调用synctask_switch_to函数时；

- WAITING

当调用synctask_yield函数将一个synctask加入到syncenv的waitq队列时；

- DONE

当一个synctask完成运行时。

以上几个阶段的状态转换如下图14所示：

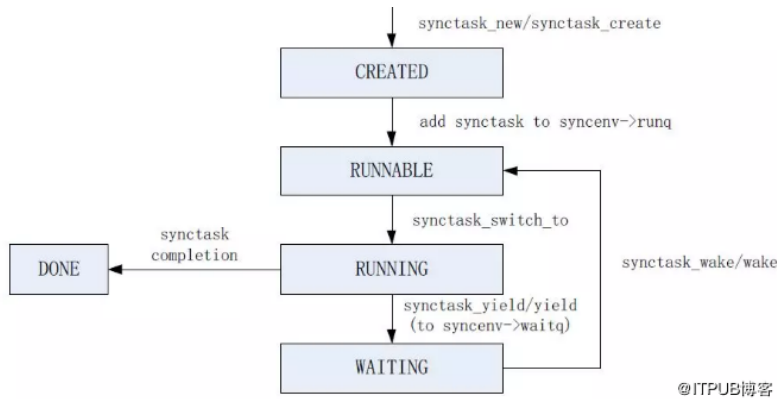


图14 synctask状态转换

需要注意的是，在synctask的生命周期内，并不能保证该任务是一直在同一个线程中运行，每次调用synctask_yield函数时，可能会使其运行于一个不同的线程中。

总之，可以这样简单地理解，使用syncop框架执行一个任务时（例如syncop_opendir函数用于打开目录），调用函数会等待该任务完成后（目录已被打开），再继续往下进行。相对而言，syncop框架大大简化了数据均衡功能的实现。

4.3. 关键处理流程

每个节点的数据均衡进程，在处理流程上都是一样的，下面结合实际代码，分别简要分析一下数据均衡功能的主线程与迁移线程的处理流程。

主线程的入口函数是gf_defrag_start，该函数主要就是使用syncop框架创建一个synctask，用于本次数据均衡的处理工作。创建synctask同步任务时，指定了实际处理函数gf_defrag_start_crawl及其完成后的回调函数gf_defrag_done，下面首先给出从gf_defrag_start_crawl函数开始的一些主要处理流程，如图15、图16和图17所示。然后再列出迁移线程的主要处理流程，如图18所示。

4.3.1. 主线程处理流程

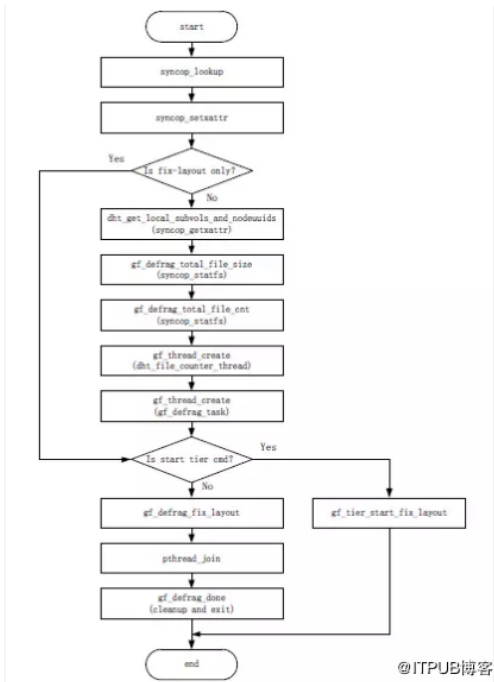
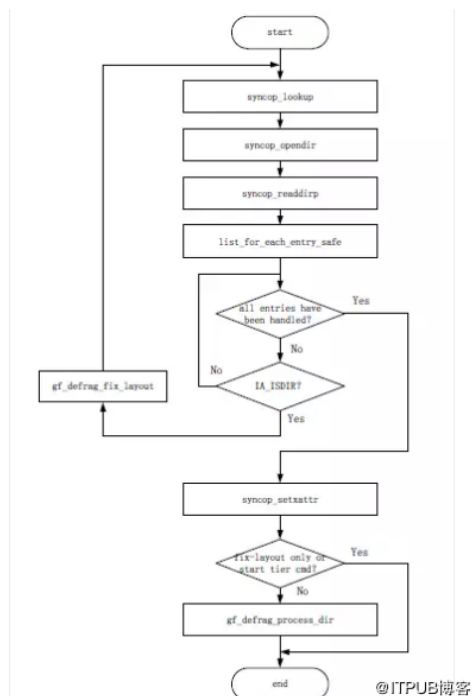


图15 gf_defrag_start_crawl函数处理流程

- 1、调用syncop_lookup，查询根目录的相关信息，位置、元数据和父目录等信息；
- 2、调用syncop_setxattr，修复根目录的哈希分布；
- 3、根据命令类型判断，如果命令类型只是修复目录哈希分布，则跳过迁移文件相关函数，转到步骤9；否则，下一步；
- 4、调用dht_get_local_subvols_and_nodeuuids，其内部使用了同步函数syncop_getxattr，获取本节点参与的子卷和每个相关子卷中涉及到的节点的uuid信息，并且在每个节点上的相同子卷返回的节点uuid的顺序一致；
- 5、调用gf_defrag_total_file_size，其内部使用了syncop_statfs，获取本节点参与子卷的总共文件大小；
- 6、调用gf_defrag_total_file_cnt，其内部使用了syncop_statfs，获取本节点参与子卷的总共文件数量；
- 7、调用gf_thread_create，创建一个文件计数线程，用于在数据均衡过程中统计文件迁移数量，方便用户命令行查看；
- 8、调用gf_thread_create，根据节点cpu活动processor的数量，创建多个迁移线程，负责具体文件迁移工作，其入口函数为gf_defrag_task，后面会给出该函数处理流程（图18）；
- 9、根据命令类型判断，如果命令类型是开启分层（tier），则调用tier相关处理函数，由于本文不考虑tier相关操作，在此略过；否则，下一步；
- 10、调用gf_defrag_fix_layout，开始进行目录的修复与文件的爬取，这也是主线程的主要工作，下面会单独给出该函数的处理流程（图16）；



- 11、在修复目录和爬取文件完成后，调用pthread_join，等待所有迁移线程完成；

根据以上流程的说明，并结合前面介绍的工作机制等内容，要理解下面给出的gf_defrag_fix_layout和gf_defrag_process_dir函数处理流程并不会很难，具体过程不再赘述。

图16 gf_defrag_fix_layout函数处理流程

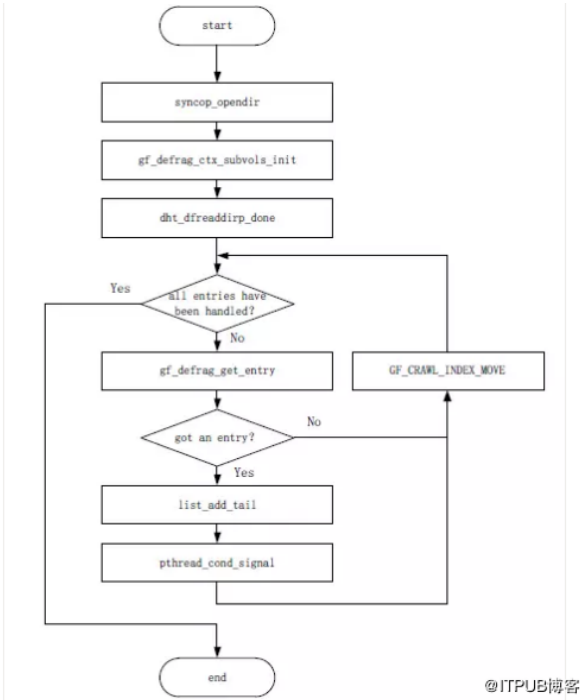


图17 gf_defrag_process_dir函数处理流程

4.3.2. 迁移线程处理流程

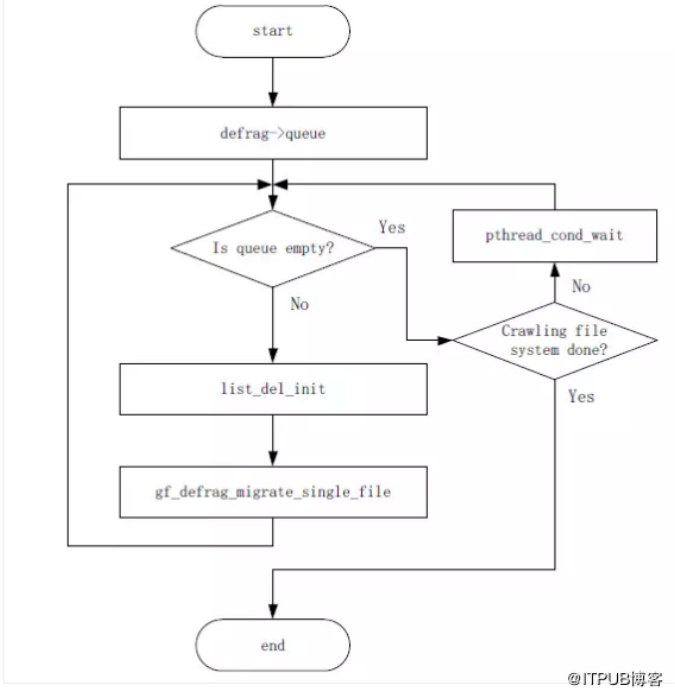


图18 gf_defrag_task函数处理流程

多个迁移线程同时运行，每个线程的处理流程都由gf_defrag_task函数负责，简要说明如下：

- 1、获取待迁移文件的队列defrag->queue；
- 2、进入如下while循环模式；
- 3、判断队列是否为空，如果不空，下一步；否则，转到步骤7；
- 4、调用list_del_init，从队列头取出一个文件，并更新相关计数；

- 5、调用gf_defrag_migrate_single_file，该函数负责单个文件迁移的整个过程，包括判断是否需要迁移、数据内容迁移、各种属性迁移、特殊文件处理和迁移后的清理工作等内容，此部分内容需要读者自行分析，不再赘述；
- 6、gf_defrag_migrate_single_file函数完成后，返回到步骤3；
- 7、判断文件爬取任务是否完成，如果没完成，则调用pthread_cond_wait，等待新文件迁移任务到来的通知；否则，下一步；
- 8、迁移任务都已完成，线程返回。

5. 问题与优化

尽管数据均衡功能经过了不断的完善，已经具有了良好的性能和扩展性，同时大大加快了扩容或缩容后集群数据均衡的进度，但仍然存在一些限制和问题，具体说明如下：

（1）数据迁移速度可进一步提升

原因是在迁移数据的时候，工作线程在读取原始文件和写入目标文件是串行的，如果能够将两者有效的分开独立进行，则可以进一步提升迁移速度；

（2）扩容或缩容时的数据迁移量大

在增加brick或删除brick的时候，需要对整个卷做数据均衡，这样一来，增加或删除很少brick时，也会触动整个集群均衡一次，给人一种牵一发而动全身的感觉，并且需要迁移的数据量较大，而在OpenStack Swift的数据均衡中则采用了带有虚拟节点的一致性哈希，可以一定程度减少数据的迁移量，因此可以考虑借鉴这种做法。

（3）数据迁移速度控制能力较弱

虽然目前命令行有三种模式（lazy、normal和aggressive），用于控制迁移速度，但还是属于比较粗略的控制，当有大量文件迁移时，还是会占用较多的系统资源，影响到存储系统性能。

（4）代码存在bug

例如，在GlusterFS的3.12系列之前的版本中，对分布式EC卷做扩容后做数据迁移的时候，每组EC卷只有一个节点在迁移文件，组内其他节点并没有参与该过程，这是和其预期工作机制是不相符的，并且影响了一部分文件的迁移。

上面只是列出了数据均衡目前存在一些限制，可能还有其他问题，等待读者去发现和解决。总之，未来还是有很多工作要做的，结合GlusterFS社区的讨论，列出主要内容如下：

- （1）优化迁移速度，迁移数据的时候，读文件和写文件并行化；
- （2）更精细地控制数据迁移，根据实际需要灵活地调控迁移速度；
- （3）优化目录哈希分布，减少每次扩容或缩容后需要迁移的数据量；
- （4）支持迁移暂停功能，再次开启后可以继续从当前暂停位置开始迁移；
- （5）在数据均衡进程中增加性能模块（xlator），提高迁移效率；
- （6）修复数据均衡代码中的bug。

6. 总结与展望

本文介绍了GlusterFS当前的数据均衡功能，内容包括数据均衡的产生背景、使用场景、基本原理、程序实现剖析、操作命令实践、存在的问题以及需要优化的地方等，同时结合实例说明了数据均衡的工作机制。

GlusterFS数据均衡功能还在不断的完善中，相信未来的数据均衡功能一定会更加高效和可控，并且占用较少的系统资源。

由于时间和精力有限，文中省略了很多细节，也包括一些重要的算法和流程，期望读者能够给予补充，同时理解不正确的地方也在所难免，欢迎指正。

（文章来自：大魏分享 郭忠秋）

参考资料

[1] <https://docs.gluster.org/>

[2] <http://pl.atyp.us/hekafs.org/index.php/2012/03/glusterfs-algorithms-distribution/>

[3] <http://staged-gluster-docs.readthedocs.io/en/release3.7.0beta1/Features/rebalance/>

[4] <https://github.com/gluster/glusterfs-specs/blob/master/done/GlusterFS%203.7/Improve%20Rebalance%20Performance.md>

[5] glusterfs v3.12.9 source code

来自 " ITPUB博客 "，链接：<http://blog.itpub.net/31547898/viewspace-2168800/>，如需转载，请注明出处，否则将追究法律责任。


👍 点赞 | 1

☆ 收藏 | 0

分享到：

上一篇：数据库审计为何沦为僵尸级应用？

下一篇：Harbor开源项目加入CNCF基金会!



请登录 after 发表评论

登录

全部评论

