### 关于gluster分布式哈希--副本仲裁复用研究

目录

**[1. 背景 2](#_Toc22442)**

**[2. hash分布说明 2](#_Toc17907)**

[2.1hash的调用关系： 2](#_Toc23130)

[2.2 hash的核心算法为 3](#_Toc6545)

[2.3 hash数据落点分布图 4](#_Toc22548)

**[3. shard切片 5](#_Toc5061)**

**[4. 副本模式 5](#_Toc30707)**

[4.1 双副本模式 6](#_Toc20184)

[4.2 仲裁说明 6](#_Toc9295)

**[5. 实战测试 8](#_Toc10282)**

[5.1 无序复用data/ar 8](#_Toc6084)

[5.2有序复用data/ar 8](#_Toc32673)

[5.3 独立data/ar 9](#_Toc27513)

[5.4 结论 9](#_Toc10069)

**[6. 总结与展望 9](#_Toc30855)**

**[7. 参考资料 10](#_Toc11733)**

#### 背景

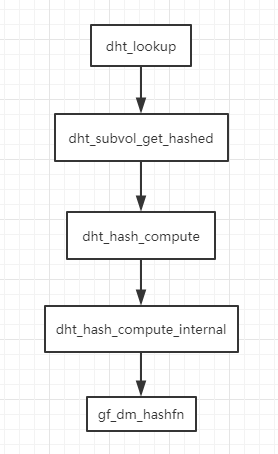
GlusterFS使用DHT模块来聚合多台机器的物理存储空间，形成一个单一的全局命名空间，并使用卷（Volume）这一逻辑概念来表示这样的空间。每个卷可以包含一个或多个子卷（Subvolume），子卷也可称为DHT子卷，同样是一个逻辑概念，一个子卷可以是单个brick、一个副本卷（Replica）或一个EC（Erasure Coding）卷，而副本卷和EC卷自身又都是由一组brick构成。而brick则是GlusterFS中的最基本存储单元，表示为一个机器上的本地文件系统导出目录。

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。

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。

#### hash分布说明

##### 2.1hash的调用关系：



##### 2.2 hash的核心算法为

gf\_dm\_hashfn (const char \*msg, int len)

{

uint32\_t h0 = 0x9464a485;

uint32\_t h1 = 0x542e1a94;

...

for (i = 0; i < full\_quads; i++) {

for (j = 0; j < 4; j++) {

word = \*intmsg;

array[j] = word;

intmsg++;

full\_words--;

full\_bytes -= 4;

}

dm\_round (DM\_PARTROUNDS, &array[0], &h0, &h1);

}

for (j = 0; j < 4; j++) {

if (full\_words) {

word = \*intmsg;

array[j] = word;

intmsg++;

full\_words--;

full\_bytes -= 4;

} else {

array[j] = pad;

while (full\_bytes) {

array[j] <<= 8;

array[j] |= msg[len - full\_bytes];

full\_bytes--;

}

}

}

dm\_round (DM\_FULLROUNDS, &array[0], &h0, &h1);

return h0 ^ h1;

}

再根据得到的hash值计算落点位置：

for (i = 0; i < layout->cnt; i++) {

if (layout->list[i].start <= hash

&& layout->list[i].stop >= hash) {

subvol = layout->list[i].xlator;

break;

}

}

##### 2.3 hash数据落点分布图

分别测试100，1000，3000，10000的输入观看落点分布情况，如下图。可以看到随着数据的增加落点会更加的均衡，最终保证各盘使用平衡。

#### shard切片

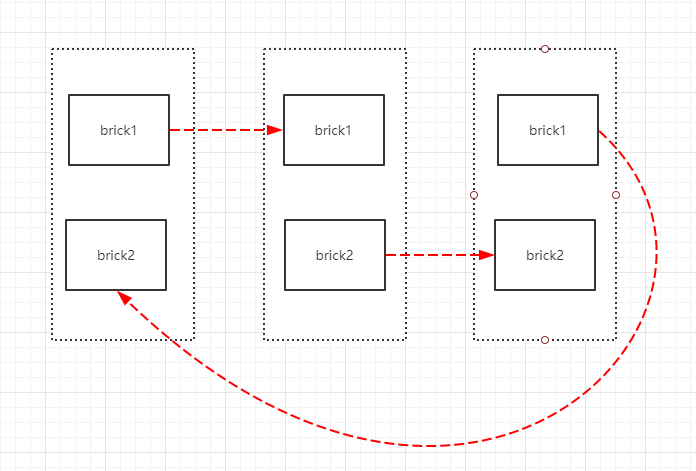
通过上面的分布测试图我们可以看到随着数据量的增加数据的分布更加的均衡，这样每个节点都可以承载业务端的读写压力，更好的利用集群的性能，不至于让某一个节点成为集群读写瓶颈。实际场景中经常会有大文件写入，因此我们使用gluster集群的shard功能，它可以将大文件切分为我们指定的大小，并且每个切片都可以通过hash分布到集群不同的子卷，性能大大提升。

#### 副本模式

副本模式作为一种高可用数据冗余方式被应用在各个场景，但是2副本天生的问题数据不一致性也是我们需要考虑的因素，而仲裁就可以很好的解决这个问题，并且仲裁功能只保留元数据占用的容量基本可以忽略不记，由此衍生出一种副本仲裁数据复用的方案场景。

##### 双副本模式

下面均以3节点，每节点2块50G盘为例。首先2副本我们需要考虑要跨节点，模型如下：

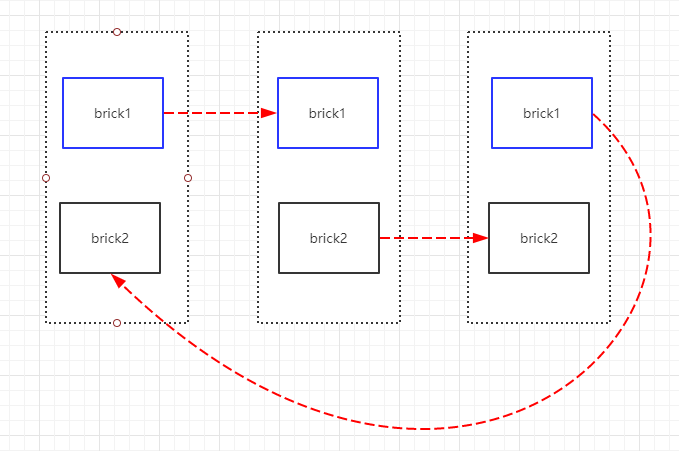


##### 仲裁说明

此时加入仲裁的场景有多个模式，优先要保证仲裁分布在3个节点。同时关于容量的说明为：每个子卷有3个数据盘，如果有2个做了复用则glus容量识别为25GB，如果1个盘做了复用则glus容量识别为50GB

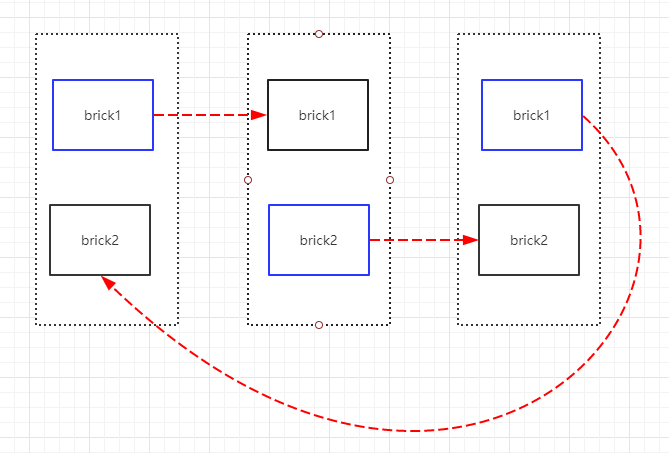
###### 4.2.1 无序复用

第一种我们直接以三个节点的brick1做为ar复用盘，如下如。（蓝色框表示复用磁盘）这时有个问题可以看到第一个子卷（1brick1+2brick1）同时复用了ar，而（2brick2+3brick2）没有复用ar，按照gluster识别容量方式vol3子卷（2brick2+3brick2）将呈现50GB的容量，而其他两个子卷只呈现25GB的容量，这就导致其哈希值的范围vol3是其余两个的2倍。这时候的数据写入vol3就会成为瓶颈。



###### 4.2.2 有序复用

第二种仲裁复用模式，如下图。此种方式可以看到每个子卷均有磁盘参与仲裁复用，最终3子卷容量均呈现为25GB，前端数据写入的时候数据能较均匀的同时分布到3个子卷，每个盘都能承载写压力。最终性能比较好。



#### 实战测试

##### 5.1 无序复用data/ar

|  |  |
| --- | --- |
| 服务器 | 3节点，每节点2盘，ar与data复用 |
| 磁盘复用 | 当2+1中指定的有2个是复用的，则容量显示其一半；当只有1个事复用，则容量为整容量 |
| 卷 | 3\*（2+1）副本仲裁 |
| Shard | 开启 |
| Shard-size | 8MB |

哈希图

|  |  |  |
| --- | --- | --- |
|  | Dht范围 | 系统识别容量 |
| 子卷1 | 7fff ae08 | 25G |
| 子卷2 | 4000 28f9 | 25G |
| 子卷3 | 4000 28fc | 50G |

测试

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 切片数 | 子卷1(25G) | 子卷2(25G) | 子卷3(50G) | 带宽 |
| 1000 | 247 | 262 | 490 | 45.6MB/s |
| 3000 | 751 | 734 | 1514 | 43.3MB/s |
| 5000 | 1214 | 1284 | 2501 | 54.5MB/s |

##### 5.2有序复用data/ar

|  |  |
| --- | --- |
| 服务器 | 3节点，每节点2盘，单盘50G，ar与data复用 |
| 磁盘复用 | 当2+1中指定的有2个是复用的，则容量显示其一半；当只有1个事复用，则容量为整容量 |
| 卷 | 3\*（2+1）副本仲裁 |
| Shard | 开启 |
| Shard-size | 8MB |

哈希图

|  |  |  |
| --- | --- | --- |
|  | Dht范围 | 系统识别容量 |
| 子卷1 | 5555 9e30 | 25G |
| 子卷2 | 5555 9e2e | 25G |
| 子卷3 | 5554 c39f | 25G |

测试

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 切片数 | 子卷1(25G) | 子卷2(25G) | 子卷3(25G) | 带宽 |
| 1000 | 340 | 311 | 348 | 76.6MB/s |
| 3000 | 954 | 1050 | 995 | 72.0MB/s |
| 5000 | 1625 | 1611 | 1763 | 65.8MB/s |

##### 5.3 独立data/ar

|  |  |
| --- | --- |
| 服务器 | 3节点，每节点3盘，ar与data独立 |
| 卷 | 3\*（2+1）副本仲裁 |
| Shard | 开启 |
| Shard-size | 8MB |

|  |  |  |
| --- | --- | --- |
|  | Dht范围 | 系统识别容量 |
| 子卷1 | 5555 5555 | 50G |
| 子卷2 | 5555 5554 | 50G |
| 子卷3 | 5555 5554 | 50G |

测试

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 切片数 | 子卷1 | 子卷2 | 子卷3 | 带宽 |
| 1000 | 332 | 339 | 329 | 70.6MB/s |
| 3000 | 969 | 1016 | 1014 | 70MB/s |
| 5000 | 1644 | 1666 | 1689 | 62.8MB/s |

##### 5.4 结论

从测试结果可以看到配置方案时选择ar/data独立配置或者有序复用效果最好，hash分布范围和压力都能很好的均衡。

#### 总结与展望

本文介绍了gluster哈希分布的原理，分析了数据均衡的背景，场景，同时结合实例说明了副本仲裁复用的优选方案与测试结果。

Guster的哈希分布充分考虑了磁盘容量的因素，但未考虑到节点cpu，内存等因素，总体来说数据分布比较均匀。由于能力有限，文中也忽略了很多细节，欢迎补充。

#### 参考资料

* <https://docs.gluster.org/>
* <http://www.taocloudx.com/index.php?a=shows&catid=4&id=106>
* glusterfs v3.13.2 source code