## Weed-fs架构理解

### 概述

Weedfs架构中存在如下几种角色：1). Data center；2). Rack ；3). Master server；4). Volume server。

Weedfs有个系统配置文件(/etc/weedfs/weedfs.con)，在该文件中可指定系统的拓扑信息，配置文件形式如下：

  <Configuration>  
    <Topology>  
      <DataCenter name="dc1">  
        <Rack name="rack1">  
          <Ip>192.168.1.1</Ip>  
        </Rack>  
      </DataCenter>  
      <DataCenter name="dc2">  
        <Rack name="rack1">  
          <Ip>192.168.1.2</Ip>  
        </Rack>  
        <Rack name="rack2">  
          <Ip>192.168.1.3</Ip>  
          <Ip>192.168.1.4</Ip>  
        </Rack>  
      </DataCenter>  
    </Topology>  
  </Configuration>

可以看到，weedfs对整个系统的层级规划还是比较好的，比如多数据中心管理，多机架管理等，而且根据作者的描述，整个weed-fs集群可以跨多个数据中心。

### 实现

有关weedfs拓扑结构的实现代码全部位于源代码子目录的go/topology中，包括了配置文件加载、datacenter、racker、datanode的定义、volume的分配等。相对来说，整个package比较简单，对各种数据结构的定义也比较粗糙。

但是volume的分配值得好好研究一番。

#### weedfs 的volume分配策略

1. 在分配策略中的collectionName到底指的是什么呢？我看到分配的时候会指定这个选项？

猜测：应该对应的是HayStack中的逻辑卷的概念吧

1. 分配策略异常简单：随机分配。随机挑选可分配的volume，然后分配即可。当然，如果客户端指定了DataCenter或者rack，那么挑选的volume所在的DataNode必须满足这些条件。

#### volume server存储结构

* 启动volume server时可指定该volume server中存储volume使用的文件夹。默认使用/tmp目录，同时还可以指定最大的volume数量。但是有一点不确定的就是这个最大指的是每个目录下最大的volume数量还是整个volume server可使用的最大volume数量。
* Volume server下的每个volume都有一个数据文件和一个索引文件。这个结构已经在Facebook的Haystack论文中作过描述。
* Collection： Volume server与master交互
* 注册：volume server启动时会将自己的信息注册到master上，包括了自身的IP、port，以及上面的volume信息；而master在收到这些信息的时候会更新本地存储的volume server的这些信息。
* Heartbeat：volume server会定期和master发起心跳信息，将本地的信息更新至master上。但是有一个疑问：好像没有看到volume server更新自己的本地volume信息。

### 关键问题

#### master单点故障

作者本意是想使用ZK来管理成员的，但是好像当时golang的ZK客户端有点问题，因此自己实现了raftserver来管理master servers。

关于raft的介绍可参考这里：<http://blog.gopheracademy.com/writing-a-distributed-system-library>

#### 负载均衡

增加server时好像没有作数据迁移以实现负载均衡的功能。而且在分配的时候好像也只是采用了随机分配的算法。不一定能做到比较好的负载均衡效果。

#### Volume同步

当一个volume的副本损坏的时候，好像也没有看到进行volume副本同步的工作，不过需要继续研究看看。

#### Weed-fs坏盘了该如何处理？

#### 副本一致性检测

好像没有什么副本一致性检测的工具

#### 运维工具

#### Volume server与master心跳

### MDS实现

#### 数据结构

* DataCenter
* Rack
* DataNode
* **Collection**

这个概念刚开始理解有点错误。现在想想，应该是和nefs租户差不多的概念：是一组volume的集合。在实现上，其中存储了一个数组：

replicaType2VolumeLayout []\*VolumeLayout

该数组以复制策略为索引，每项存储了volume布局信息，这个我们会在下面讲到。

复制策略比如：副本放置在不同数据中心，副本放置在相同数据中心的不同rack……，另外一篇文档中有对weedfs复制策略的定义。

* **VolumeLayOut**

是对volume布局信息的抽象。其中主要包含了一个map：

vid2location map[storage.VolumeId]\*VolumeLocationList

记录了本volume layout所包含的所有的volume以及其物理位置信息。

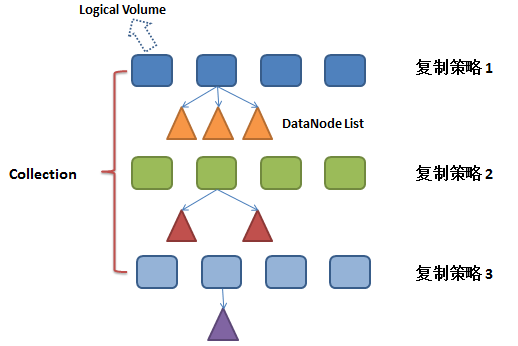
* **VolumeLocationList**

type VolumeLocationList struct {

list []\*DataNode

}

记录了volume所在的节点。



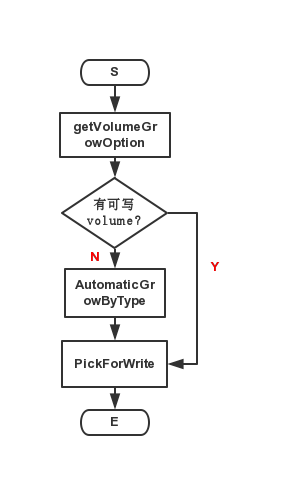
#### 关键流程

* **分配volume**

客户端通过http请求向MDS申请分配写fid，其url形式如下：

<http://localhost:9333/dir/assign>

MDS端处理assign请求的函数为dirAssignHandler()。



**getVolumeGrowOption(r)**

该函数根据http请求中的参数构造一个volume分配策略的结构体，主要包含了副本所在的DataCenter、Rack、DataNode、collection以及副本的存放策略，将这些分配选项打包返回以作后面的分配volume的指导原则。

另外，系统默认的volume副本存放策略是000。

**PickForWrite()**

PickForWrite()实现了主要的volume分配策略：它从满足条件的所有volume(逻辑volume)中挑选一个合适的，并生成一个FileID返回给客户端。主要分如下步骤：

GetVolumeLayout：首先根据volume分配的指导原则选择出特定的Volume Layout；

PickForWrite：再从Volume Layout中选择一个合适的volume。

该函数的实现比较有意思：

1). 如果我们对于副本的位置（如DataCenter、Rack、DataNode等）没有限制的话，那么直接从该volume layout的可写volume中随机挑选一个；

2). 如果我们对副本的位置有限制，那么需要对所有可写volume进行筛选，过滤出那些满足条件的volume，再从中随机挑选一个，随机算法比较有意思。

for \_, v := range vl.writables {

volumeLocationList := vl.vid2location[v]

for \_, dn := range volumeLocationList.list {

if … {

if …

}

counter ++; // 记录满足条件的volume个数

if rand.Intn(counter) < 1 {

vid, locationList = v, volumeLocationList

} // 随机挑选算法

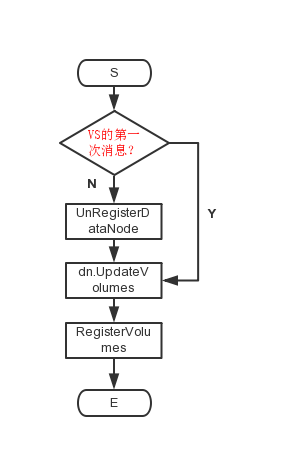
NextFileId()：挑选出合适的volume后，接下来就要根据volume信息生成fileID。

### VS与master交互

#### 注册

VS(volume server)启动时会向MS注册自己的信息，包括自己的IP、端口以及自身的volume信息等。

MS在收到VS的注册信息时，会调用ms.Topo.RegisterVolumes(joinMessage)将VS发送过来的volume注册到MS中。接下来我们主要看看这个函数内部的处理逻辑：



**UnRegisterDataNode**

将DN上的所有volume设置为Unavailable，并将该DN节点从其父节点上移除。主要调用了SetVolumeUnavailable和UnlinkChildNode。

SetVolumeUnavailable：将该volume所在的节点从其所在的collection的复制策略组的volume\_layout的逻辑volume中移除，并判断如果当前副本数少于要求的副本数，那么就将该逻辑volume从可写volume中移除；

UnlinkChildNode：将DN从其父节点列表中删除。

**Dn.UpdateVolumes**

更新DN的volume信息，将DN汇报的最新的volume信息与当前MS存储的对比，删除那些已经不复存在的volume。然后将最新的volume信息存储在DN里面。

**RegisterVolumeLayout**

这个过程与第一个过程正好相反，就不再赘述。

#### Heartbeat

DN与MS的heartbeat逻辑与注册逻辑完全一致，每隔一定的时间，DN都会向MS发送一次本地最新的volume信息，而这个时间其实不是一定的，会根据上一次heartbeat成功与否而不一致。

### Questions

* VS端的volume是什么时候被创建的呢？是预分配好的？
* VS端的volume信息在什么时候会发生改变呢？比如增加或者减少？
* volumeID是谁生成的？按照道理应该是MS.

VS启动的时候会从配置的存放volume的目录中读出当前存在的volume。每个volume其实就是一个大文件。文件命名规则如下：

“collection\_volumeID.dat”

另外，每个volume还有super\_block，记录了很多volume元数据信息，在VS启动的时候也需要将他们一起读出。

* MDS如何检测DN离线

MDS根据DN的心跳汇报时间来检测DN是否离线。MDS有个后台协程专门检测那些离线的DN，并改变这些DN的状态。

这个后台协程是在StartRefreshWritableVolumes()中被启动的。对于离线DN的判断方法是上次心跳时间距离当前时间超过了3 \* 心跳时间内间隔。

具体对DN离线判断的函数是CollectDeadNodeAndFullVolumes()。这个函数其实会做两件事：

1. 将那些已经离线的DN设置状态为离线；
2. 将那些当前容量已经超过最大容量限制的volume标记为满。
3. 但是volume的当前容量怎么更新呢？因为写只是客户端与DN的交互，难道DN在写入完成后会通知MDS吗？还是DN在心跳信息中会汇报给MDS? 查看了下代码，确实是DN在心跳信息中汇报给MDS的。

该函数会将上面的两个信息分别写入chanDeadDataNodes以及chanFullVolumes两个channel，后面会有专门的协程来处理这些事件。

* DN重新上线后如何恢复

这个过程与上面的过程相反：如果一个DN在离线一段时间后又重新发送心跳了，那么如果此前判定了该DN离线，此时势必要启动离线恢复流程。

在GetOrCreateDataNode()函数中如果dn状态是dead的话，那么需要设置回来，并且将该信息写入chanRecoveredDataNodes该管道中，后面有专门的协程来处理这个事件。

* Weed-fs对上述事件的处理

1. 对于volume full事件：调用函数SetVolumeCapacityFull()；
2. 对于DN离线，调用UnRegisterDataNode ()；
3. 对于DN重新上线，调用RegisterRecoveredDataNode()；

* 对于weedfs replacement的理解：

type ReplicaPlacement struct {

SameRackCount int // 代表了同一个racker下的DN数量

DiffRackCount int // 代表了不同rack的数量

DiffDataCenterCount int // 代表了不同DataCenter的数量

}

根据作者的文档，理解起来非常费力。

### 理解

1. **对weedfs副本分配算法的理解：**

首先，根据代码以及作者文档的理解，举例吧

假如存储策略是110：

1. 说明volume有三副本；
2. 三副本的存放策略是：第一个副本存储于DC1(随机挑选)的Racker1的某个server上，第二个副本存放于DC1上的Racker2的某一个server上，第三个副本存放于DC1外随机的一个DC2上；
3. 本例中存在一个主DataCenter，一个次DataCenter，而且主DataCenter上存在一个主Racker（Racker1），一个次Racker（Racker2）；
4. 在分配volume的时候，首先会挑选主DC，但是主DC的选择需要有限制条件：
5. 要有能满足条件的racker：本例中至少要有2个Racker；
6. 空闲volume数量要足够：本例中至少要有两个volume ；
7. DC上的有空闲volume的racker数量必须满足要求。本例中至少要由两个racker拥有至少一个空闲volume。作者在实现的时候也非常巧妙，我们在后面会详细说到。
8. 分配完成DC后，再从主DC上分配racker，跟上面一样，主racker必须满足以下限制条件：
9. 空闲volume数量必须大于一定值，本例中为1；
10. 子节点数量必须大于一定值，本例中也为1；
11. 至少拥有N个空闲volume的子节点数量至少为M，本例中N为1，M为1;
12. 分配完成Racker以后，接下来分配server，即volume所在的数据节点，主server同样满足条件：
13. 至少拥有一个空闲volume
14. 上面的这些限制条件是针对主节点而言，主节点指的是主server，主racker，主DC，主server上存储主副本，主racker是指主副本所在racker，主DC指的是主副本所在DC，而其他副本是相对于主副本而言的。如011值的就是第一个次副本和主副本位于相同的racker，第二个次副本和主副本位于相同DC的不同racker。
15. 根据上面7的描述，我们就可以知道，对于次节点的分配，只要其上面有空闲空间即可。因为一般会在主节点上分配多个副本，而不会在次节点上分配多个副本，比如011，主racker上有两个副本，而次racker上只需要保存一个副本。比如110，主DC上有两个副本，而次DC上只需一个副本。
16. 上面的逻辑理解起来真是不容易啊。
17. **Weed-fs9种分配策略的理解**

要理解weed-fs的9种分配策略，我们首先必须搞清楚一点：weedfs最多只支持三副本，当然只是目前，很容易扩展为更多副本。

000：单副本；

001：两副本，存放于相同racker；

002：三副本存放于相同racker；

010：两副本，存放于相同DataCenter的不同racker；

020：三副本，存放于相同DC的不同racker；

011： 三副本，两个副本存放于相同racker，另一个副本存放于相同DC下面的不同racker；

100：两副本，存放于不同DC；

200：三副本，存放于不同DC；

110：三副本，第一个副本随机挑选位置，第二个副本和第一个副本存放于相同DC的不同racker下，第三个副本存放在不同的DC。

000：单副本

001：两副本，存放在相同rack

002：三副本，存放在相同racker

010：两副本，存放于相同DC的不同racker

020：三副本，存放于相同DC的不同racker

011：三副本，两个副本存放于相同racker，另一个副本存放于相同DC下面不同racker

100：两副本，存放于不同DC

200：三副本，存放于不同DC

110：三副本，第一个副本随机挑选位置，第二个副本和第一个副本存放于相同DC的不同racker，第三个副本存放于不同DC