

基于 FastDFS 的数字媒体系统设计与实现技术研究

张祥俊¹, 伍卫国²

(1. 西安交通大学 软件学院, 陕西 西安 710000;
2. 西安交通大学 电信学院, 陕西 西安 710000)

摘 要:海量的数据存储一般采用分布式文件系统 HDFS, Hbase 等作为存储工具, 但对海量媒体资产数据这种中小文件的存储, 存在性能瓶颈, 并且随着后期小文件越来越多, 会造成系统元数据占用过大。对于海量媒体资产管理而言, 检索是关键, 设计良好的查询体系很重要。结合国家重点研发课题海量数字媒体与应用的实际应用需求, 通过分析分布式文件系统 FastDFS 的原理与缺陷, 提出了一种基于容器技术、适合于存储海量中小型文件的分布式存储系统, 同时设计了集群服务器之间的同步目录组方案, 确保在宕机的情况下, 用户的数据不会丢失, 且 Docker 支持多租户, 确保了不同租户数据的隔离性。最后对设计实现的原型系统进行了测试与分析, 测试结果表明, 基于 FastDFS 的数字媒体资产在线管理系统符合预期设计, 性能和可靠性较好, 可适用于超算环境下各种渲染任务的使用。

关键词: FastDFS; Docker; 媒体资产管理; 分布式存储; 数字媒体制作

中图分类号: TP302.1

文献标识码: A

文章编号: 1673-629X(2019)05-0006-06

doi: 10.3969/j.issn.1673-629X.2019.05.002

Research on Design and Implementation of Digital Media System Based on FastDFS

ZHANG Xiang-jun¹, WU Wei-guo²

(1. School of Software, Xi'an Jiaotong University, Xi'an 710000, China;

2. School of Electronics and Information Engineering, Xi'an Jiaotong University, Xi'an 710000, China)

Abstract: Distributed file systems such as HDFS and Hbase are commonly used as storage tools to store massive data. However, there exists bottlenecks in the storage of medium and small files for massive media asset data, and as small files increase, system metadata may occupy too much space. For mass media asset management, retrieval is the key, and well-designed query system is very important. Combined with actual application requirements of huge amounts of digital media and application of the national key project, based on the analysis of the principles and defects of distributed file system FastDFS, we propose a distributed storage system based on container technology which is suitable for storing large and medium-sized files. At the same time, we design a synchronized directory group scheme between cluster servers to ensure that users' data will not be lost in case of downtime. Docker supports multi-tenancy and can ensure the isolation of different tenant data. Finally, the prototype system designed and implemented is tested and analyzed. The test results show that the digital media asset online management system based on FastDFS is in line with the expected design with better performance and reliability, which is applicable for various rendering tasks in a supercomputing environment.

Key words: FastDFS; Docker; media asset management; distributed storage; digital media production

0 引言

高性能计算机已跨入了 E 级计算时代, 数字媒体产业作为信息技术和人文艺术结合的内容创意产业已成为最活跃的高性能计算应用之一, 用户众多。数字媒体作品的制作中, 计算机 3D 技术的作用愈加重要, 3D 建模和渲染成为全球电影、动漫工业界关注的核心

技术^[1]。动漫作品团队使用渲染农场后处理好的数据需要存放在一个共享的平台^[1], 而在此平台上对数据的安全有效管理成为目前需要考虑的问题。因此, 在渲染流程处理中的数据量很大, 众包模式下的数据安全性、可靠性、高效存储、便捷管理是一个问题, 为了节约成本, 使用统一的开放平台接口处理渲染数据的问

收稿日期: 2018-05-30

修回日期: 2018-09-06

网络出版时间: 2018-12-21

基金项目: 国家重点研发计划重点专项项目(2017YFB0203000); 海外及港澳学者合作研究基金(61628210)

作者简介: 张祥俊(1993-), 男, 硕士研究生, 研究方向为云计算与大数据; 伍卫国, 教授, 博导, 研究方向为高性能计算机体系结构、海量存储。

网络出版地址: <http://kns.cnki.net/kcms/detail/61.1450.TP.20181221.1624.062.html>

题也亟待解决。

FastDFS 是一种分布式存储系统,特点在于以中小文件为载体^[2],同时配合 NGINX,两者作为图片服务器提供对应的在线服务,客户端使用 API 读写文件,基本解决了大容量存储的问题,并且这套高性能的文件服务器集群可以提供文件上传、下载等操作。虽然 FastDFS 能满足高性能的存储,但是针对渲染流程中海量媒体的管理需求,还没有办法完全达到。首先, FastDFS 不支持断点续传,在大文件上传时,一般采用切片上传形式;第二, FastDFS 对文件的上传版本没有控制,客户端上传的文件都保存在服务器中,没有进行版本化的管理;第三,完成一个数字产品所需要的数据量很大,同时资产数据的个数也很多,为了能够方便管理,需要有每个客户所上传的目录分级,而 FastDFS 不能完成这一点;第四,近年来,文创众包走进了大众视野,在此新制作模式下,多个制作团队之间需要资源共享与权限控制,显然当前的 FastDFS 无法满足这种多租户的存储即服务的需求。因此希望可以通过容器技术 Docker 支持多租户,实现用户环境隔离;第五,在数据安全方面,自动化恢复机制不完善:如果 storage 的某块磁盘发生故障发生,且故障时间较长,只能手动恢复原有数据并且换磁盘;需要预先准备好热备磁盘,否则很难立即向外提供正常服务。不能自动化恢复会增加管

理员的运维成本。国内的 FastDFS 与 Google FS 相似,具有存储小文件优化和简洁的特性,比较 mogileFS,维护和使用体验更好的开源分布式文件系统^[3]。主要用于大中网站,为文件上传和下载提供在线服务。所以在负载均衡、动态扩容等方面都支持得比较好,但是截止目前,运用 FastDFS 进行存储,并在其基础上管理媒体资产的系统还未出现典型代表。

基于此,文中提出了一种利用容器化技术在给定时间进行数据同步的机制,保证整个存储集群满足高性能存储的数据可靠性要求,同时由于容器的多租户技术^[4],具备了很好的隔离性。最后对其进行实验验证和测试。

1 系统基本结构

1.1 系统结构

海量媒体数据管理功能负责存储传输海量场景数据,由 tracker 集群和存储集群组成。tracker 集群负责管理数据的元数据信息,具有文件分块/多副本/版本控制等功能,媒体作品创作端读写文件时首先和 tracker 集群通信完成元数据操作,随后将数据通过 restful API 对象接口传输到存储集群。具体使用场景如图 1 所示。

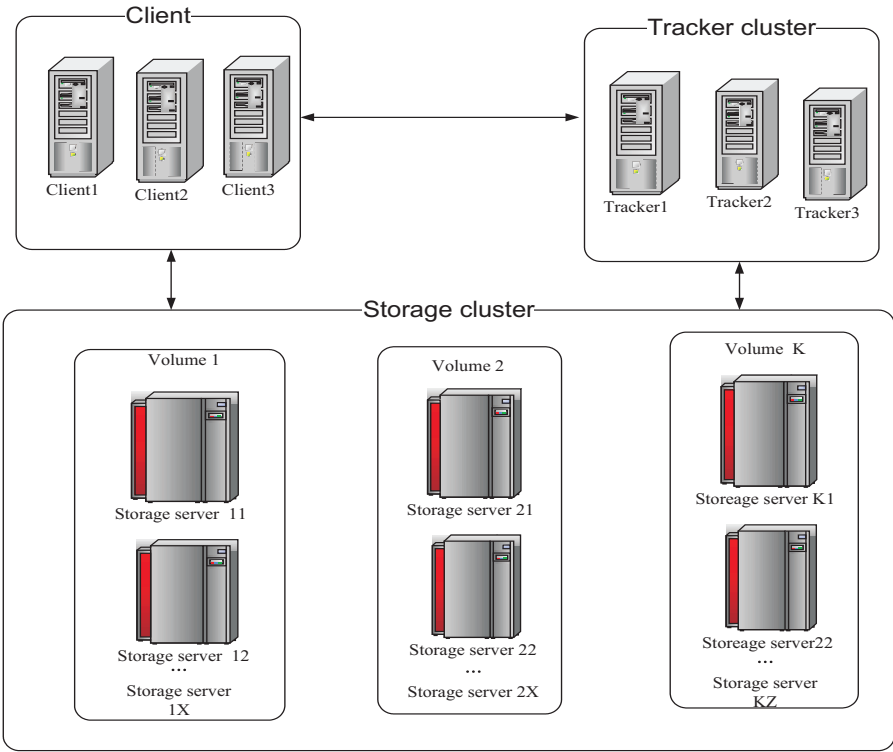


图 1 数字媒体资产在线管理系统结构

1.2 FastDFS 分布式文件系统

1.2.1 分布式文件系统概念

Distributed file system (DFS) 是指多台计算机协同

合作,从而完成文件存储、大数据计算等目标。互联网上的所有资源,最终都会以文件的形式存放在具体的物理机器的存储设备上。存储、读取和管理这些海量

的文件依靠单一的机器无疑是不行的,所以分布式文件系统进入人们的视野。

1.2.2 FastDFS 的系统架构

FastDFS 组成包括跟踪服务器(tracker server)、存储服务器(storage server)和客户端(client)^[5]。tracker 的工作职责首要是调度,在整个过程中维持 storage 均衡;第二,作为所有 storage server 和 group 的管理者,需要连接每个 storage,并得到它们所属 group 信息,同时获取其周期性心跳。得到心跳信息后,tracker 负责建立 group 与 storage serverlist 的映射。

storage 的责任是提供容量信息和冗余备份;storage server 按 group 分组,同组内可以有多台 storage,其中存储的文件有同步备份。按 group 为单位存储,可以进行隔离应用、服务器负载均衡、获取到副本数等。

client 主要是客户端上传下载文件等数据的服务器,即在 client 上部署自身开发的项目。安装 Nginx 后,用户发送的请求可以被分发到不同的 tracker 上。

(1) FastDFS 的存储策略。

FastDFS 具有大容量存储的特性,其原因是存储节点采用了分组的组织方式。存储系统中包含一个或多个组,组与组之间的文件独立,整个存储系统中的文件容量是将所有组的文件容量累加起来得到的。一个组可以由一台或多台 storage 组成^[6]。当组中增加 storage 时,系统自动完成同步组内已有的文件,同步结束后,新增服务器会被系统切换上线,并对外提供服务。如果发生存储空间不足或快要使用完的情况,动态添加组即可扩大存储系统的容量^[7]。

(2) FastDFS 的上传过程。

使用者可以通过 upload、download、appendfile、deletfile 等基本文件访问接口结合 FastDFS 进行操作,这些接口是以客户端库的方式提供给用户的。首先客户端发送上传文件的请求给 tracker,tracker 会分配一个可以存储文件的 group,之后再继续分配合适的 storage server。确定 storage server 后,client 继而发送写文件请求,storage 得到该消息后,将会为文件分配一个数据存储目录^[8]。同时文件将得到 fileid 以作区分,最后生成存储文件的文件名信息。

(3) FastDFS 的文件下载。

client 端上传文件成功后,通过 storage 生成唯一文件名,调用 download 方法访问到该文件并下载。在下载时,client 可以选择任意 tracker server。client 发送 download 请求给某个 tracker,请求信息中包含文件全名,tracker 从文件全名中解析出文件的 Id、所属 group、扩展名等信息,之后选择一个 storage 读请求,提供下载服务^[9]。万方数据

2 容器环境下的 FastDFS 器群组织与配置

2.1 环境的配置

```
docker pull zxj2017/fastdfs:v1
docker network create --subnet=172.18.0.0/16 fastDfsnet
docker run -d --name trackerA -h Server-A --net=fastDfsnet --ip 172.18.0.10 --add-host="storageA:172.18.0.12" --add-host="storageB:172.18.0.13" zxj2017/fastdfs:v1 sh tracker.sh
docker run -d --name trackerB -h Server-B --net=fastDfsnet --ip 172.18.0.11 --add-host="storageA:172.18.0.12" --add-host="storageB:172.18.0.13" zxj2017/fastdfs:v1 sh tracker.sh
docker run -d --name storageA --net=fastDfsnet --ip 172.18.0.12 -h storageA -e TRACKER_IP=172.18.0.10:22122 -e GROUP_NAME=group1 zxj2017/fastdfs:v1 sh storage.sh
docker run -d --name storageB --net=fastDfsnet --ip 172.18.0.13 -h storageB -e TRACKER_IP=172.18.0.11:202:22122 -e GROUP_NAME=group2 zxj2017/fastdfs:v1 sh storage.sh
```

环境测试:

进入 trackerA,上传文件 查看结果:docker exec -it trackerA /bin/bash

```
vi /etc/fdfs/client.conf
```

```
tracker_server= Server-A:22122
```

```
vi test.txt
```

```
hellofastDFS
```

```
/usr/bin/fdfs_upload_file /etc/fdfs/client.conf test.txt
```

进入 storageA 查看结果:docker exec -it storageA/bin/bash

此时表明 FastDFS 的容器已经启动成功。

2.2 FastDFS 的可靠性解决方案

环境说明:工作服务器 A:IP 地址 172.18.0.10,操作系统 Ubuntu,已建立用户 tom;备份服务器 B:IP 地址 172.18.0.11,操作系统 Ubuntu,已建立用户 jack (uid 503,gid 503)。

实现目的:每天早上 3 点,将 A 服务器上的用户目录/home,自动备份到 B 服务器的/home/jack/backup-A 下,备份增量进行^[10],不需要任何用户交互。

配置步骤:

配置备份服务器 B。

(1)[root@Server-B ~]# rpm -qa | grep rsync # 查看是否有 rsync 包

```
rsync-2.6.8-3.1
```

以上输出说明 rsync 已经装好了,保证/etc/services 有下面的行。

```
rsync      873/tcp      *rsync
```

```
rsync      873/udp      *rsync
```

(2)rsync 的 rpm 包本身没有附带 rsyncd 的配置文件,需要手动创建(/etc/rsyncd.conf)。

```
[root@ Server-B ~]#vi /etc/rsyncd.conf
Log file=/var/log/rsyncd.log
[local0]
Comment=back from Server-A
Path=/home/jack/backup-A
Hosts allow = 172.18.0.10 127.0.0.1
Read only=false
uid=503 #保证在服务器 B 的备份操作以 jack 这个用户
进行
gid=503
```

(3) 修改/etc/xinetd.d/rsync,打开 rsync 服务。

```
[root@ Server-B ~]# vi /etc/xinetd.d/rsync
# default: off
# description: The rsync server is a good add
# allows crc checksumming etc.
service rsync
{
    disable = no
    socket_type      = stream
    wait            = no
    user            = root
    server          = /usr/bin/rsync
    server_args     = --daemon
    log_on_failure += USERID
}
```

(4) 开启 rsyncd 服务,并设置系统启动时加载 rsync 服务。

```
[root@ Server-B ~]# /usr/bin/rsync --daemon
```

(5) 检验 rsync 服务是否启动成功。

```
[root@ Server-B ~]#rsync localhost::
```

```
[root@Server-B ~]# rsync localhost::
local0      backup from Server-A
```

有如下内容表示已经成功启动。

```
[root@Server-B ~]# lsof -i :873
COMMAND  PID USER  FD  TYPE DEVICE SIZE NODE NAME
rsync    6197 root   4u  IPv6  35252      TCP *:rsync (LISTEN)
rsync    6197 root   5u  IPv4  35253      TCP *:rsync (LISTEN)
```

(6) 配置 ssh 的非交互式登录。

```
>>在服务器 A 上生成一对密钥(以 root 的身份执行)[root
@ Server-B ~]# ssh-keygen -t rsa
>>远程登录到备份服务器 B 上并且创建.ssh 目录
[root@ Server-A ~]#ssh jack@ 172.18.0.11
[jack@ Server-B ~] $ mkdir .ssh;chmod 0700 .ssh
>>在 A 机上执行远程拷贝公钥到 B 机:
[root@ Server-A ~]#scp .ssh/id-rsa.pub root@ 172.18.0.
```

11: /home/jack/.ssh/authorized_keys
这样,无交互的 ssh 登录就完成了。

(7) 编制备份脚本。

在服务器 A 上编写一个备份脚本,放置在/home/tom/public_scripts 下,名为 backup.sh。

```
#!/bin/sh
TARGET_DIR=backup-A
for SOURCE_DIR in "/home"
do
    echo "Backing up $ SOURCE_DIR ..."
    rsyn-au -delete $ SOURCE_DIR jack@ 192.168.1.87:/
home/jack/ $ TARGET_DIR
done
```

[root@ Server-A public_scripts]# chmod 755 backup.sh
该脚本权限设置为 755,以便其他用户可访问到。

(8) 修改计划任务。

在服务器 A 上,用 root 身份执行以下命令,可以设定什么时候执行脚本

```
[root@ Server-A ~]# crontab -e
3 * * * * /home/tom/public_scripts/backup.sh
```

同样的操作在 storageA 和 storageB 之间将它们
的目录数据采用定时同步,然后对节点的同步进行备份。

3 实验测试

3.1 测试环境

测试节点配置见表 1。

表 1 测试节点配置

名称	操作系统	配置	IP 地址
Tracker 跟踪服务器	Ubuntu server 16.04LTS amd64	CPU 型号: Intel(R)	172.18.0.10
		Core(TM) i7-4790	
		CPU @ 3.60 GHz	
		硬盘大小: 1 T	
内存大小: 32 G		172.18.0.12	
CPU 型号: Intel(R)			
Core(TM) i7-4790			
CPU @ 3.60 GHz			
StorageA 存储服务器	硬盘大小: 1 T	172.18.0.13	
StorageB 存储服务器	内存大小: 32 G		
StorageC 存储服务器	172.18.0.14		
StorageD 存储服务器		CPU 型号: Intel(R)	
		Core(TM) i7-4790	
		CPU @ 3.60 GHz	
	硬盘大小: 1 T	172.18.0.15	
内存大小: 32 G			

续表 1

名称	操作系统	配置	IP 地址
数据库管理节点		CPU 型号: Intel(R)	192.168.0.243
		Core(TM) i7-4790	192.168.0.241
数据节点、访问节点		CPU @ 3.60 GHz	
		硬盘大小: 1 T	192.168.0.242
数据节点、访问节点		内存大小: 32 G	

该系统的测试使用 FastDFS tracker 服务器, FastDFS storage 服务器, Mysql 服务器与一台终端进行,安装完成后,可通过上传、下载的命令进行是否安装成功的测试。图 2 是测试环境的网络拓扑。

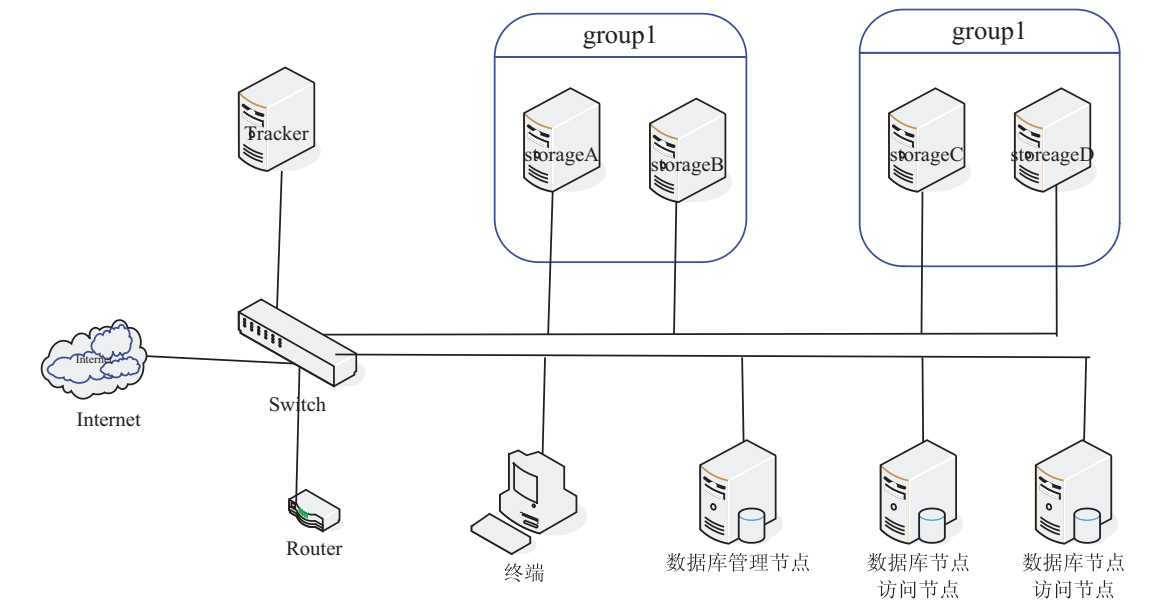


图 2 测试环境网络拓扑

由图 2 可知,因为服务器在 FastDFS 中担任的角色不同,所以配置启动的服务也不同。首先,在服务器 172.18.0.10 上对应配置 tracker,而在存储服务器的 group1 对应的 172.18.0.12 (storageA) 和 172.18.0.13 (storageB)、group2 对应的 172.18.0.14 (storageC) 和 172.18.0.15 (storageD) 上配置 storage,分成 2 组。按要求,tracker、storage 服务器的配置文件都需要修改后方可运行。根据上节提出的同步目录组问题的设计^[11],可知 172.18.0.14 (storageC) 和 172.18.0.15 (storageD) 中还必须配置 rsync 服务。最后配置启动数据库,包含 3 台服务器。按照测试环境配置中划分

的角色,修改配置文件/var/lib/mysql-cluster/config.ini 和/etc/my.cnf。由于条件限制,数据节点和访问节点安装在同一服务器上,先启动管理节点 192.168.0.243,再启动数据节点 192.168.0.241,192.168.0.242。

3.2 可靠性测试

由部署的测试环境可知,FastDFS 组内一般有多台存储服务器,组内设计在 group2 对应的 172.18.0.14 和 172.18.0.15 上配置 storage,通过将其中一台存储服务器 172.18.0.14 以及数据库集群节点 172.19.0.12 关闭,模拟出分组中一台存储服务器及数据库集群宕机的情况,如图 3 所示。

```
ndb_mgm> show
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=2 @192.168.0.241 (mysql-5.6.21 ndb-7.3.7, starting, Nodegroup: 0)
id=3 (not connected, accepting connect from 192.168.0.242)
[ndb_mgmd(MGM)] 1 node(s)
id=1 @192.168.0.243 (mysql-5.6.21 ndb-7.3.7)
```

图 3 关闭集群节点

第二步登入系统上传一个文件,文件上传成功后保存在 FastDFS 中的 key 是:

group1/M00/00/rBIADFsMB9-AQUvBAAAA

DT_u2ZM252.txt。由图 4、图 5 可知,当同组内的某台存储服务器宕机时,集群对外提供的服务稳定,并且可将上传的文件同步到在分组中另外一台存储服务器

172.18.0.15 上。

```
root@storageA:/data/fast_data/data/00/00# ll
total 16
drwxr-xr-x. 2 root root 48 May 29 09:33 ./
drwxr-xr-x. 258 root root 8192 May 29 09:28 ../
-rw-r--r--. 1 root root 13 May 29 09:33 rBIADFsNHlmaQVbQAAADT_u2ZM646.txt
root@storageA:/data/fast_data/data/00/00# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:12:00:0c
          inet addr:172.18.0.12  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe12:c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:142 errors:0 dropped:0 overruns:0 frame:0
          TX packets:156 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:185875 (185.8 KB)  TX bytes:11589 (11.5 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:18 errors:0 dropped:0 overruns:0 frame:0
          TX packets:18 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1391 (1.3 KB)  TX bytes:1391 (1.3 KB)
```

图4 上传到 172.18.0.12 上的文件

```
root@storageB:/data/fast_data/data/00/00# ll
total 4
drwxr-xr-x. 2 root root 48 May 29 09:50 ./
drwxr-xr-x. 3 root root 16 May 29 09:49 ../
-rw-r--r--. 1 root root 13 May 29 09:50 rBIADFsNHlmaQVbQAAADT_u2ZM646.txt
root@storageB:/data/fast_data/data/00/00# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:12:00:0d
          inet addr:172.18.0.13  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe12:d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:117 errors:0 dropped:0 overruns:0 frame:0
          TX packets:120 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:182321 (182.3 KB)  TX bytes:7308 (7.3 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:643 (643.0 B)  TX bytes:643 (643.0 B)
```

图5 上传到 172.18.0.13 上的文件

由测试结果可知,当 172.18.0.12 宕机时,开启 172.18.0.13 服务器,可查找到已经同步的文件,集群在某些节点宕机的情况仍可工作,因此系统具备较高的可靠性。

4 结束语

文中提出一种在 FastDFS 基础上增加新功能的解决方案,以满足平台对海量媒体存储要求的技术,通过容器技术对存储组镜像进行隔离^[12-14],结合 nginx 模块的无缝衔接,跟踪节点和存储节点可以灵活分配。针对 FastDFS 在数据恢复方面的缺点,提出一种在不同节点上的同步策略,保证了存储系统的可靠性和高效性。实验结果证明了该方法在高性能计算环境下的可靠性和隔离性^[15],该技术提高了数字媒体系统在存储方面的效率和灵活性。但是依然有一些不足,比如对于上层的应用程序还可以结合容器编排技术对其进行微服务式的管理,这样可以大大提高整个系统的可靠性。另外,同步策略在同步数据时会给系统的性能

带来一些影响,这些问题还需要在今后的研究中继续解决。

参考文献:

[1] 张威,鲍丽娜.可快速定位的视频流媒体大数据存储系统[J].科学技术与工程,2014,14(1):239-243.

[2] 操顺德,华宇,冯丹,等.面向海量高清视频数据的高性能分布式存储系统[J].软件学报,2017,28(8):1999-2009.

[3] 王鲁俊,龙翔,吴兴博,等.SFFS:低延迟的面向小文件的分布式文件系统[J].计算机科学与探索,2014,8(4):438-445.

[4] 李森.浅析基于 SaaS 架构的多租户技术[J].电子设计工程,2013,21(20):41-44.

[5] 尹向东,杨杰,屈长青.云计算环境下分布式文件系统的负载均衡研究[J].计算机科学,2014,41(3):141-144.

[6] LIN C C, CHIN H H, DENG D J. Dynamic multiservice load balancing in cloud-based multimedia system[J]. IEEE Systems Journal, 2014, 8(1):225-234.

[7] 周江,王伟平,孟丹,等.面向大数据分析的分布式文件系统关键技术[J].计算机研究与发展,2014,51(2):382-394.

[8] 周国安,李强,陈新,等.云环境下海量小文件存储技术研究综述[J].信息安全,2014(6):11-17.

[9] 王宝会,邢景轩,高远.运用 FastDFS 和 Drill 构建海量 BIM 族数据存储和查询平台[J].土木建筑工程信息技术,2016,8(6):23-28.

[10] 王志华,吴丙涛,徐艳秋.一种基于 rsync 的文件系统实时同步方法:CN,CN105893633A[P].2016-08-24.

[11] WANG X, LIU J. The application of media asset management system in the research and development of TV programs[J]. Journal of Head Trauma Rehabilitation, 2015, 25(5):362-365.

[12] ANDERSON C. Docker [software engineering][J]. IEEE Software, 2015, 32(3):102-c3.

[13] 刘思尧,李强,李斌.基于 Docker 技术的容器隔离性研究[J].软件,2015,36(4):110-113.

[14] PEINL R, HOLZSCHUHER F, PFITZER F, et al. Docker cluster management for the cloud-survey results and own solution[J]. Journal of Grid Computing, 2016, 14(2):265-282.

[15] 张松,疏官胜,李京.容器微云监控系统的设计和实现[J].中国科学技术大学学报,2017,47(8):627-634.