核心：亿级甚至十亿小文件存储。

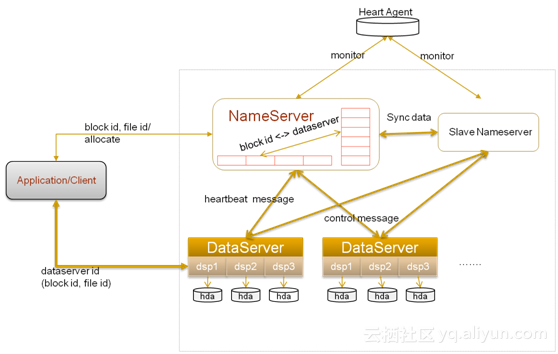
调研的文件系统：**TFS，Ceph，haystack，Seaweed-fs**

# TFS

**TFS为淘宝提供海量小文件存储，通常文件大小不超过1M，满足了淘宝对小文件存储的需求，被广泛地应用在淘宝各项应用中。**

## TFS架构设计

TFS集群由NamServer和DataServer组成，以block（通常为64M，可配置）为单位存储和组织数据。



图表 1 TFS架构图

### 1）NameServer

NameServer主要管理维护Block和DataServer相关信息,包括DataServer加入，退出, 心跳信息,block和DataServer的对应关系建立，解除。  
正常情况下，一个块会在DataServer上存在， 主NameServer负责Block的创建，删除，复制，均衡，整理， NameServer不负责实际数据的读写，实际数据的读写由DataServer完成。

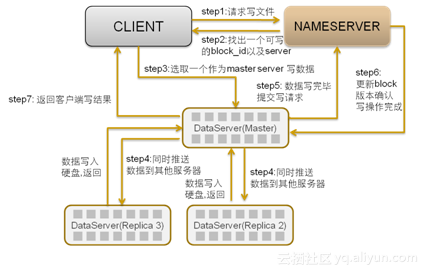
NameServer采用了HA结构，即两台机器互为热备，同时运行，一台为主，一台为备，主机绑定到对外vip，提供服务；当主机器宕机后，迅速将vip绑定至备份!NameServer，将其切换为主机，对外提供服务。

### 2）DataServer

DataServer主要负责实际数据的存储和读写。

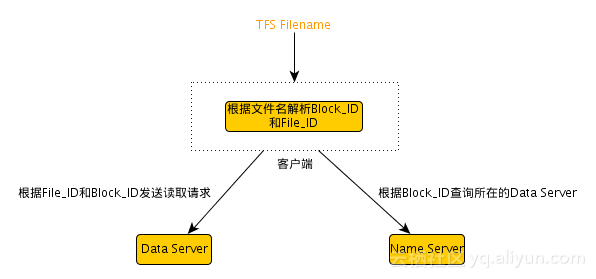
#### 1.TFS写操作数据流

TFS会将多个小文件存储在同一个block中，并为block建立索引，以便快速在block中定位文件；每个block会存储多个副本到不同的机架上，以保证数据的高可靠性。



图表 2 TFS的写操作数据流图

#### 2.TFS读操作数据流

****

图表 3 TFS读操作数据流

1）获得Block ID和File ID

根据TFS文件名解析出Block ID和block中的File ID。

2）获取dataserver地址

向nameserver发送查询请求得到Block ID所在的dataserver地址。  
由于nameserver中维护了block和dataserver的对应关系，所以nameserver能够提供相应的信息。  
Note: 由于TFS是把大量小文件放在一个block里面，  
所以TFS的文件复制是基于block的，而且复制出来的block的block id应该是一致的。

### 3）请求文件

通过发送Block\_ID、File\_ID和offset为参数的读请求到对应的dataserver，得到文件内容。  
dataserver会根据本地记录的信息来得到File ID所在block的偏移量，从而读取到正确的文件内容。

## 优点

1）针对小文件量身定做，随机IO性能比较高；

2）支持在线扩容机制，增强系统的可扩展性；

3）实现了软RAID，增强系统的并发处理能力及数据容错恢复能力；

4）支持主备热倒换，提升系统的可用性；

5）支持主从集群部署，其中从集群主要提供读/备功能；

## 缺点

1）TFS只对小文件做优化，不适合大文件的存储；

2）不支持POSIX通用接口访问，通用性较低；

3）不支持自定义目录结构，及文件权限控制；

4）通过API下载，存在单点的性能瓶颈；

5）官方文档非常少，学习成本高；

## 应用场景

1）多集群部署的应用

2）存储后基本不做改动

3）海量小型文件

## 需求实现不太确定的：

**权限控制（10年有一篇帖子说是后期要加权限控制，不知道做没做这件事），**

**并发机制（并发机制，对于同一个文件来说，多个用户可以并发读；现有TFS并不支持并发写一个文件。一个文件只会有一个用户在写。这在TFS的设计里面对应着是一个block同时只能有一个写或者更新操作。）**

**外部接口设计，感觉是需要代理，TFS提供了Nginx模块，通过Nginx作为HTTP客户端可以实现对TFS的访问。另外，TFS还提供了C++和Java库，允许程序员利用这些库开发自己的TFS客户端。**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TFS** | 成本控制方式设计 | 文件聚合技术设计 | 总体结构设计，集群 | 容错机制设计 | 并发机制设计 | 文件管理设计，错误检查，错误修正 | 外部接口设计 | 权限控制 |
| **描述** | **文档少，学习成本高** | **支持** | **支持** | **支持** | **支持并发读，对一个块只允许一个写** | **存疑，没有找到相关内容，这个感觉文件系统应该有这方面的考虑** | **专用的API接口去访问，需要代理，Nignx模块作为客服端访问** | **有一篇博客说是要做，不知道后来做没做** |

# Ceph

Cephfs提供了一个符合posix标准的文件系统。

## Ceph文件系统简介

## Ceph:a-scalable-high-performance-distributed-file-system // OSDI’06

Ceph是加州大学Santa Cruz分校的Sage Weil（ DreamHost的联合创始人） 专为博士论文设计的新一代自由软件分布式文件系统。

2004年， Ceph项目开始， 提交了第一行代码。

2006年， OSDI学术会议上， Sage发表了介绍Ceph的论文， 并在该篇论文的末尾提供了Ceph项目的下载链接。

2010年， Linus Torvalds将CephClient合并到内核2.6.34中，使Linux与Ceph磨合度更高。

2012年， 拥抱OpenStack， 进入Cinder项目， 成为重要的存储驱动。

2014年， Ceph正赶上OpenStack大热， 吸引来自不同厂商越来越多的开发者加入， Intel、 SanDisk等公司都参与其中， 同时Inktank公司被Red Hat公司1.75亿美元收购。

2015年， Red Hat宣布成立Ceph顾问委员会， 成员包括Canonical、 CERN、 Cisco、 Fujitsu、Intel、 SanDisk和SUSE。 Ceph顾问委员会将负责Ceph软件定义存储项目的广泛议题， 目标是使Ceph成为云存储系统。

2016年， OpenStack社区调查报告公布， Ceph仍为存储首选， 这已经是Ceph第5次位居调查的首位了。

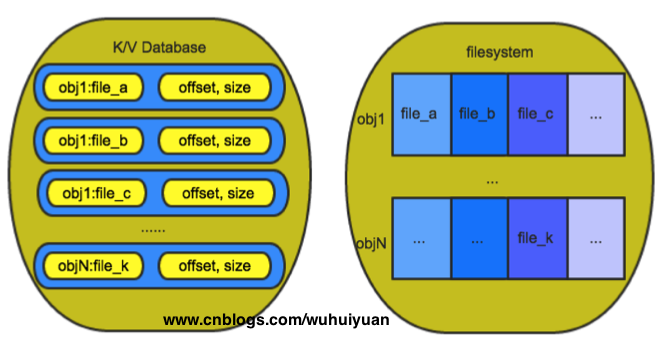
## 针对海量小文件的优化

**有博客介绍如何优化Ceph文件系统的小文件存储。如这篇博客提到的**

**小文件合并操作**：

海量小文件存储与Ceph实践<https://www.cnblogs.com/wuhuiyuan/p/4651698.html>

实现了对小文件进行合并将**若干小文件合并存储在RADOS（**Ceph的基础组件**）系统的一个对象**(object)中，<小文件的名字、小文件在对象中的offset及小文件size>组成kv对，作为相应对象的扩展属性(或者omap，本文以扩展属性表述，ceph都使用kv数据库实现，如leveldb)进行存储，如下图所示，对象的扩展属性数据与对象数据存储在同一块盘上；

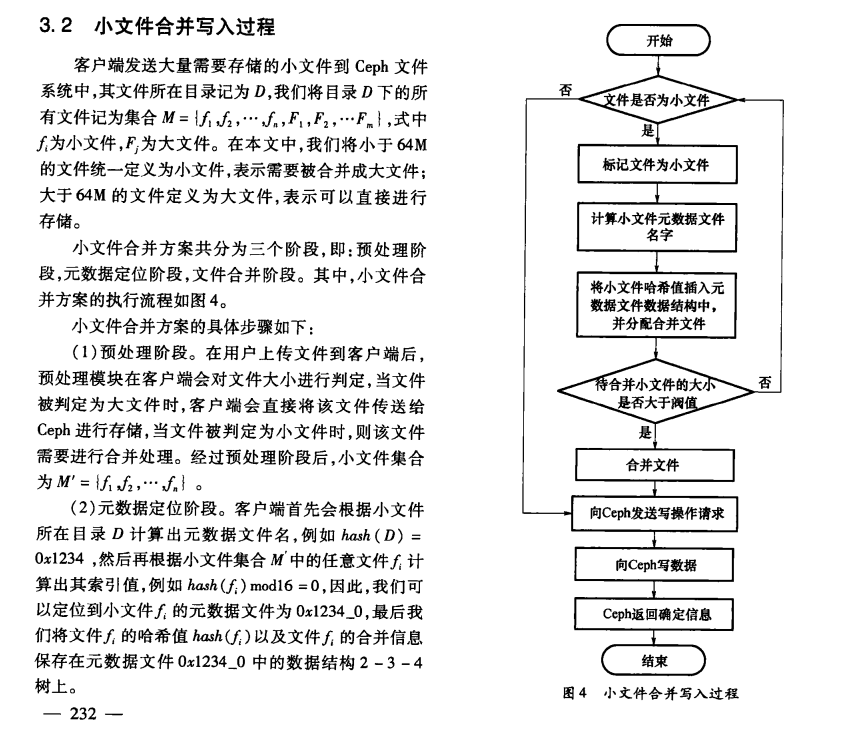


图表 4 <小文件的名字、小文件在对象中的offset及小文件size>组成kv对 作为扩展属性

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Ceph** | 成本控制方式设计 | 文件聚合技术设计 | 总体结构设计，集群 | 容错机制设计 | 并发机制设计 | 文件管理设计，错误检查，错误修正 | 外部接口设计 | 权限控制 |
| **描述** | **开源，流行** | **有北邮使用2-3-4树来维护小文件的索引的文章实现** | **支持** | **支持** | **有锁的实现** | **感觉文件系统应该有这方面的考虑，不太确定** | **符合posix标准** | **可以添加用户，为用户分配相应读写权限** |

## 北邮论文海量小文件存储的优化

**基于Ceph的海量小文件存储的优化方法\_张毕涛**



图表 5 小文件合并写入\_北邮

# haystack

Facebook 发表的论文 haystack 特意为大量的小图片存储提出的文件系统

**Finding a needle in Haystack: Facebook’s photo storage // OSDI’10 提出haystack的概念**

**Facebook’s Warm BLOB Storage // OSDI’14 对haystack架构进行优化，提出冷热数据的概念，优化相应存储**

## 一些数字

* 6500亿张照片
* 2.6万亿 张图片
* 20PB
* 每周10亿张新增照片
* 每周新增存储60TB
* 峰值请求100万/s

## 业务特征

一次写,频繁的读取,不修改,很少删除。

## 过去的经验

使用posix文件系统存储文件

问题: 当文件数量增加时获取一个图片需要多次io,磁盘io有限,所以会导致获取图片效率降低。

* **原因**
  + **读取文件的过程是**
    - **读取目录元数据获取文件inode**
    - **读取文件inode**
    - **读取真实文件。**
  + **上面的步骤据需要将元数据缓存到内存当中**
    - **元数据中有大量对于业务场景不需要的数据**
      * **创建时间**
      * **修改时间**
      * **权限**
      * **组**
      * **etc…**
  + **当文件过多时目录的元数据会变得越来越大,读取目录时就需要多次IO,同时由于目录元数据太大能cache能缓存的目录元数据会下降，频繁的被淘汰，导致经常要重新从磁盘读取目录元数据,导致io增加。**

## haystack

**Facebook设计了haystack为了解决上述问题**

* **高吞吐低延时: 减少图片元数据并且保存在内存当中**
* **容错：haystack复制到了多地,一台机器挂了备用的可以顶上来。**
* **性价比: 成本下降了28%,提供的请求提升了4倍。**
* **简洁：刻意保持简单。**

**haystack设计和实现**

**haystack的设计就是为了用来解决系统之前的瓶颈的: 磁盘IO操作。接受长尾请求带来的磁盘IO。**

## 核心思想

1. **一个图片存为单独一个文件会导致太多的元数据**
   * **haystack维护一个大文件,小文件分布在大文件中,haystack自行维护文件的offset,控制文件数量.**
2. **元数据太大无法缓存**
   * **haystack删除了无用的元数据,仅保留图片相关的基本元数据**
3. **减少除了访问真实文件之外的IO操作**
   * **使用单独的大文件,不需要每个文件都要重新去加载数据,在内存保存了所有的元数据**

**TIPS: 元数据分为两种需要注意区分 一种是应用元数据，一种是文件系统元数据，前者用来构建浏览器访问的url，后者用于在磁盘上进行文件检索。**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Haystack** | 成本控制方式设计 | 文件聚合技术设计 | 总体结构设计，集群 | 容错机制设计 | 并发机制设计 | 文件管理设计，错误检查，错误修正 | 外部接口设计 | 权限控制 |
| **描述** | **开源** | **支持** | **支持** | **支持** | **不太确定，论文没提并发** | **不太确定，论文没提，只提了从崩溃中恢复，容错的概念** | **通过URL上传，读取** | **论文没提** |

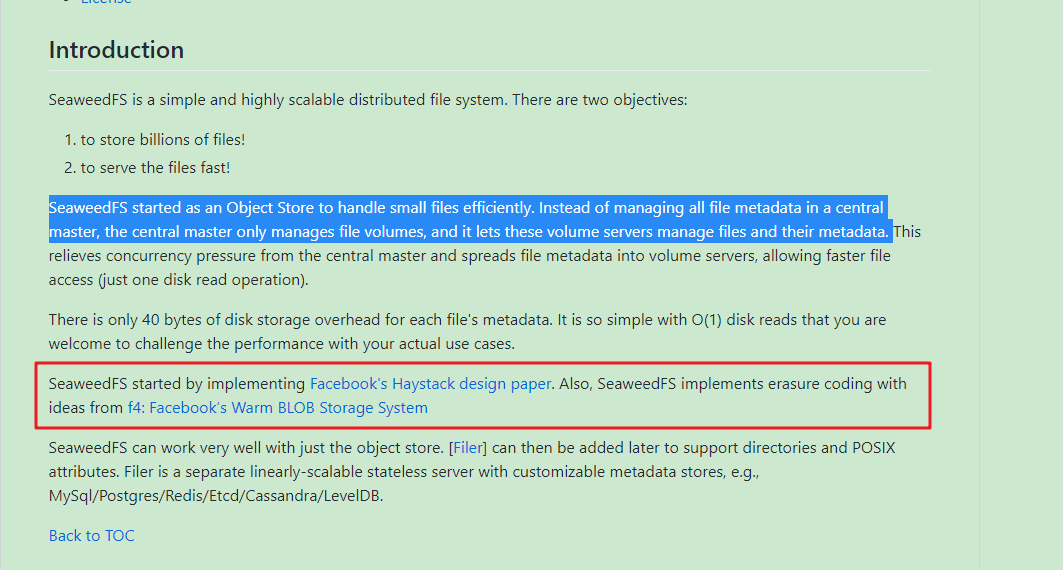
# seaweedfs

## weed-fs文件系统简介

[weed-fs](https://github.com/chrislusf/seaweedfs)，全名Seaweed-fs，是一种用[go语言](http://tonybai.com/tag/golang)实现的简单且高可用的分布式文件系统。该系统的目标有二：

- 存储billions of files  
- serve the files fast

**weed-fs起初是为了搞一个基于Fackbook的**[**Haystack论文**](http://www.usenix.org/event/osdi10/tech/full_papers/Beaver.pdf)**的实现，Haystack旨在优化Fackbook内部图片存储和获取。**后在这个基础上，weed-fs作者又增加了若干feature，形成了目前的weed-fs。后在facebook第二篇论文的基础上增加了纠删码的功能。下图为官网的截图：https://github.com/chrislusf/seaweedfs



图表 6 seaweedfs github截图

## 基本功能

分布式集群、一致性保证、目录支持

# 附录目前主流系统整体对比

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 对比说明  /文件系统 | TFS | FastDFS | MooseFS | GlusterFS | Ceph |
| 开发语言 | C++ | C | C | C | C++ |
| 开源协议 | GPL V2 | GPL V3 | GPL V3 | GPL V3 | LGPL |
| 数据存储方式 | 块 | 文件/Trunk | 块 | 文件/块 | 对象/文件/块 |
| 集群节点通信协议 | 私有协议（TCP） | 私有协议（TCP） | 私有协议（TCP） | 私有协议（TCP）/ RDAM(远程直接访问内存) | 私有协议（TCP） |
| 专用元数据存储点 | 占用NS | 无 | 占用MFS | 无 | 占用MDS |
| 在线扩容 | 支持 | 支持 | 支持 | 支持 | 支持 |
| 冗余备份 | 支持 | 支持 | 支持 | 支持 | 支持 |
| 单点故障 | 存在 | 不存在 | 存在 | 不存在 | 存在 |
| 跨集群同步 | 支持 | 部分支持 | - | 支持 | 不适用 |
| 易用性 | 安装复杂，官方文档少 | 安装简单，社区相对活跃 | 安装简单，官方文档多 | 安装简单，官方文档专业化 | 安装简单，官方文档专业化 |
| 适用场景 | 跨集群的小文件 | 单集群的中小文件 | 单集群的大中文件 | 跨集群云存储 | 单集群的大中小文件 |
| 中国流行程度 | 小 | 流行 | 中 | 流行 | 流行 |

**可以深入了解Ceph+优化合并小文件，基于haystack论文用Go实现的seaweedfs等文件系统。**