计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(7):82−88 [doi: 10.15888/j.cnki.csa.007456]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

面向科研领域的分布式对象存储系统①

王锦涛[1,2], 张海明[1]

[1](中国科学院 计算机网络信息中心, 北京 100190)
[2](中国科学院大学, 北京 100049)
通讯作者: 张海明, E-mail: hai@cnic.cn

摘　要: 随着科研工作的推进, 科研数据出现了海量的增长, PB 级科研数据需要高效、稳定的存储系统. 传统的数据存储方案存在资源利用率差、集群扩展性能低以及用户界面操作不友好等问题, 严重限制了数据在科研场景下的有效利用. 依托中科院地球科学大数据专项, 本文设计并实现高效的存储系统 i-Harbor. 该系统以对象存储系统为核心架构, 以开源的 Ceph 分布式存储系统和 MongoDB 数据库作为对象数据和元数据的存储载体, 设计通用的基于 HTTP 和 FTP 协议的数据接口, 同时利用多副本和纠删码技术消除单点故障, 配合 Zabbix 集群监控系统, 实时定位平台参数以及故障, 提高平台容灾性和安全性. 此外, 基于底层分布式结构的特点, 集群可以随意添加存储节点, 提高了平台的扩展性.

关键词: 对象存储; 分布式; Ceph

引用格式: 王锦涛,张海明.面向科研领域的分布式对象存储系统.计算机系统应用,2020,29(7):82–88. http://www.c-s-a.org.cn/1003-3254/7456.html

Distributed Object Storage System for Scientific Research

WANG Jin-Tao[1,2], ZHANG Hai-Ming[1]

[1](Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: With the development of scientific research, there is a massive increase in scientific research data. PB-level scientific research data requires efficient and stable storage systems. The traditional data storage scheme has problems such as poor resource utilization, low cluster expansion performance, and unfriendly user interface operation, which seriously limit the effective use of data. Relying on the Big Data Project of the Chinese Academy of Sciences, we design and implement an efficient storage system i-Harbor. Its core architecture is based on object storage system, using open-sourced Ceph distributed system and MongoDB database as the storage carrier of object data and metadata. The data interface is designed on the basis of HTTP API and FTP. To improve the platform disaster tolerance and security, we use Multiple Copies and Erasure Coding technology to eliminate single node of failure. Meanwhile we locate the real-time platform parameters and faults by Zabbix cluster monitoring system. Based on the distribution characteristics, the cluster can add storage nodes at will, which improves the platform's scalability.

Key words: object storage; distributed storage; Ceph

1　概述

现代科学研究中, 数据密集型的科学研究产生越来越多的数据, 科学大数据对数据采集、数据处理、数据保存、数据共享等提出了巨大的挑战[1]. 在这种背景下, 传统的单节点数据存储显然无法满足需求, 其传输速度慢、存储周期短、灾备容错率低、传输共享性差等问题制约着海量数据的高效存储和应用. 对象存储系统以其在可伸缩性、易用性、可靠性和低成本等

① 基金项目: 中国科学院 A 类战略性先导科技专项 (XDA19000000)
Foundation item: Category A Strategy Priority Research Program of Chinese Academy of Sciences (XDA19000000)
收稿时间: 2019-11-20; 修改时间: 2019-12-16; 采用时间: 2019-12-20; csa 在线出版时间: 2020-07-03

方面的优势而成为应对这些挑战的很有前途的解决方案, 因而得到日益广泛的应用[2].

在中科院战略性先导科技专项地球大数据科学工程中, 提出了实现 200 台存储服务器节点总计 20 PB 存储容量的稳定、高效、安全和高可用的存储系统需求. 传统的数据存储主要依托于分布式的文件系统, 例如 Lusture[3]、GPFS[4]、PVFS[5]、Panasas[6]等, 其遵循 POSIX 存储语义进行存储空间的组织和数据访问, 功能完备, 系统设计复杂[7], 树状的目录便于文件的查看和管理, 但是当数据量不断增大时, 庞大的树状目录结构会给系统带来巨大的元数据管理的开销, 每次文件的读取和移动操作都需要与元数据服务器进行交互, 导致访问效率低下. 同时, 元数据与文件内容共存于同一存储物理结构内, 导致数据形式的耦合性高, 不利于存储节点的扩展. 此外, 文件系统的访问形式和传输工具也比较单一, 通过 NFS 或者 CIFS 网络协议挂载, 认证方式单一, 共享效率低下, 无法满足多种科研计算环境下的访问和获取需求.

对象存储技术自 2000 年左右被提出来以后[8], 随着大数据行业的蓬勃发展, 该技术被广泛应用到各种存储需求领域, 开源领域有 Ceph[9]、Gluster[10]和 OpenStack Swift[11]等及其成熟的对象存储底层存储项目, 这些开源项目在近些年持续高涨的火热, 也印证了该技术的可行性和科学性, 同时推动了对象存储技术的广泛应用. 在工业领域, 云计算鼻祖 Amazon 在 2006 年提出 S3[12]对象存储平台后, 彻底改变了 IT 存储领域. 但是在追求相对安全、扩展性和灵活性的科研存储领域, 单一地采用商业公有云存储或开源框架明显不能满足需求, 公有云对象存储服务存在安全性和扩展性方面的劣势, 开源软件则在功能服务性上过于基础化和单一化, 表 1 给出了两者的优缺点对比.

表 1 商业对象存储服务和开源对象存储框架对比

	商业化公有云对象存储服务	开源对象存储框架
优点	按需付费、运行稳定 API 接口丰富、服务性强	功能完善, 开源社区成熟 Lib 库丰富, 可按需灵活定制开发
缺点	必须按照厂商的规则进行调用和开发, 功能可扩展性差	没有上层服务, 基于各种命令行或 lib 库操作

在大型科研领域中, 考虑到数据的绝对的安全性和私有性, 项目组一般拥有一定规模的私有物理服务器集群, 因此选择共有云对象存储无法成为有意义的方案. 针对以上需求和现状, i-Harbor 结合两种存储方案的优势, 将灵活可扩展的开源对象存储框架作为底层存储引擎, 设计独立的元数据管理模块, 提供功能完善、可控性强的对象存储架构, 同时设计独立的业务管理模块、数据传输功能模块、权限认证模块等, 提供符合公有云对象存储业界标准的可定制化的服务, 增强功能可交互性, 提升产品用户体验.

综上, i-Harbor 拥有以几方面的下技术优势:

(1) 对象存储的核心思路是将文件中的内容和元数据进行分离, 分为控制节点+数据节点. 将数据内容以对象的形式存储到分布式的数据节点中, 此外搭建若干控制节点, 通过软件管理分布式的内容数据集群, 同时, 控制节点也存储元数据, 元数据主要负责存储对象的属性 (所属数据节点、创建时间、长度大小等). 元数据与对象数据存储资源实现逻辑隔离, 降低了存储形式的耦合性, 提高了物理存储资源的利用率;

(2) 对象存储结合块存储和文件存储各自的优势, 同时克服共同的缺陷. 块存储读写快, 不利于共享, 文件存储读写慢, 利于共享. 因此将数据切分为二, 元数据软件控制节点避免了树状目录在传输中的效率低的问题, 对象形式的扁平化数据内容存储提高了数据的读写效率;

(3) 科研数据不仅仅需要静态稳定的存储, 也需要动态的分享和下载, 通过元数据服务器中对每个对象位置和目录结构的记录, 分散地到分布式中读取数据内容, 直接将分片好的数据内容以字节流的形式通过 HTTP 或 FTP 网络传输协议等方式获取, 极大地提高了数据的共享效率和便捷性;

(4) 系统采用硬盘阵列的 raid 多副本技术[13]、Ceph 集群的纠删码[14]恢复技术和元数据数据库 MongoDB 的副本集[15]策略, 避免了集群单点故障, 提高了数据存储的安全性, 在实现分布式集群可扩展性的同时, 提高了集群的容灾性能.

2 系统设计

i-Harbor 是一个对象存储为核心功能的分布式存

储系统, 主要包含数据对象存储集群 Ceph、元数据存储集群 MongoDB、Web 数据管理系统、FTP 和 HTTP 传输接口、Zabbix[16]集群参数监控系统等模块, 图 1 为系统整体架构设计.

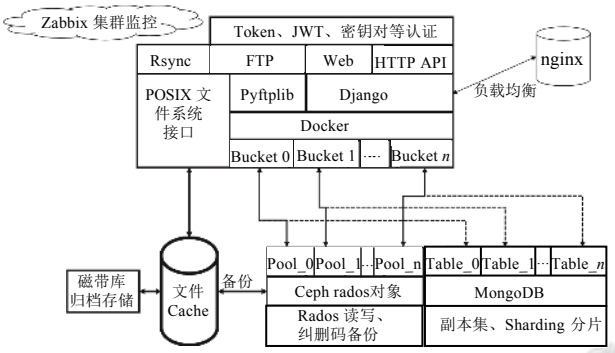

图 1 i--Harbor 系统架构图

2.1 业务管理系统

该模块主要负责为用户提供管理数据的主要入口, 包含用户管理、数据上传和下载、文件权限管理、数据共享等主要功能, 通过高效率的 MVC 开发框架 Django 实现. 功能逻辑上, 以 bucket 作为数据在物理资源的唯一划分单位, 同时也是数据上传、下载、共享的逻辑单元, 在每一个 bucket 内部, 用户可以创建不限量的文件夹和上传数据文件, 以此模拟出文件系统的树状目录结构, 方便用户管理和共享. 此外, 该模块实现了用户对 API 和 FTP 数据接口的权限认证和密钥获取, 方便用户在大规模批量地上传和获取数据场景下拥有稳定、高效的数据读写接口.

2.2 元数据管理

在 i-Harbor 系统中, 元数据负责记录用户存储的数据的具体信息, 包含名称、大小、创建时间、修改时间、所在目录等信息. 只有元数据分配更加平均, 才能使得集群资源得到充分利用, 元数据服务器之间的负载达到均衡, 从而提高系统的性能[17]. 因此采用开源数据库 MongoDB, 它是一个高性能、面向文档式的分布式架构数据库, 适用于大尺寸海量数据存储场景, 拥有高度扩展和伸缩、低容灾性的特点, 非常适合该系统.

在元数据管理模型中, 每个存储桶 bucket 作为一个 collection, 上述具体信息作为 field, 每一个 document 为用户上传的一个文件记录, 利用系统的 ObjectId 作为数据库主键, 也是该条数据在对象数据管理模型中的唯一标识, 通过它来寻找数据对象内容的具体存储地址.

i-Harbor 充分利用 MongoDB 原生的分布式性能, 搭建主+从+仲裁节点的副本集存储集群, 避免了存储集群的单点故障, 同时, 利用 sharding 分片机制, 根据 field 特点选择对 id 字段进行 Hash 分片, 使所有 document 随机、均匀地分布在所有的 mongod 存储节点, 实现了负载均衡, 同时在增加节点时通过内部的数据迁移机制, 实现了高度扩展的分布式特点, 大大提高了存储效率.

2.3 对象数据管理

作为 i-Harbor 的存储核心, 对象数据存储管理依赖于开源的分布式对象存储系统 Ceph. Ceph 是在存储领域应用最广泛的可持续化存储系统, 主要包含对象网关 RWG、块存储 RBD 和文件存储 CephFS 三大功能模块, 其底层依赖 rados 对象作为数据的基本存储单元, 通过每一个数据文件的唯一标识符 uuid, 经过原生的 crush 数据寻址算法计算出所在的存储池和存储设备, 充分利用了存储节点的计算能力, 实现了高度可扩展的对象数据存储策略.

i-Harbor 利用 Ceph 中的核心库 librados, 通过内存的字节流读写, 将数据写入 rados 对象且从中获取数据实现文件的上传和下载. 在该模块中, 用户通过业务管理模块中的数据目录和元数据记录获取数据的唯一标识符 uuid, 通过 Ceph 的 monitor 发送给 OSD, 经过 crush 的数据寻址算法找到对应的 rados 存储位置, 获取对应数据并经过对应接口返回给客户端, 以此来完成数据读写的主要 IO.

在该模块中, 为了保证对象数据存储的安全性, 设计了纠删码的冗余技术, 在测试实验环境中采取 16+4 的纠删码策略, 即在不超过五分之一的数据丢失损坏后可以利用其他完好的数据通过对应的矩阵算法恢复出丢失数据, 这种策略在保证数据安全性的同时极大的提高了资源的利用率, 同时配合实际物理环境中的 raid 多副本备份技术, 最大程度避免了单一故障, 提高了集群的容灾性.

2.4 数据接口管理

在业务管理系统中, 用户可以通过 Web 界面在线上传文件、删除文件、下载文件以及共享文件, 非常方便快捷, 但是在大规模的数据上传和下载场景下, 这种模式显然效率极低, 在科研环境中大部分的数据读写都是大批量进行的, 为此设计了 API 和 FTP 两种数据接口形式. API 为 RESTFul 形式接口, 遵循基于 HTTP 协议的接口标准形式, 用户通过 Token、JWT 或密钥

对等多种认证形式连接到 i-Harbor, 进行数据的上传下载. FTP 工具是基于开源的 Pyftplib 库实现, 用户以每个存储桶为操作单位, 通过对应的密码连接到 bucket 文件目录, 从而实现客户端与服务端文件同步显示的功能, 方便用户上传和下载.

2.5 其他模块

为满足整体系统健壮性, 配合以上主要模块稳定运行, 设计以下其他模块:

(1) Nginx 反向代理: 将用户通过 HTTP 的请求按照轮询机制反向代理到集群中的 django 业务管理系统节点, 实现访问负载均衡;

(2) Zabbix 集群监控体系: 监控所有物理节点 CPU、内存运行情况和网络负载等参数, 以及 MongoDB 和 Ceph 集群各节点数据读写速率, 保证系统平台高可用性;

(3) 数据归档备份: 对于超过一定年限的数据, 定时备份归档到存储成本更低的磁带库集群, 保证物理资源中高速读写硬盘的有效利用.

3 关键技术

3.1 MongoDB 副本+分片集群

在 2.2 元数据管理模块中, 利用 MongoDB 分布式数据库作为元数据的存储引擎, 并依赖副本集合分片机制保证数据的安全性和均匀分布. 在 i-Harbor 集群环境中, 搭建了一个 30 个物理节点的分片键+副本集的 MongoDB 集群. 本节将其约分为 3 个节点, 介绍其中关键技术.

共 3 个节点, 分为 3 个 shard, 3 个节点分别为 mongo01、mongo02 和 mongo03, mongo01 和 mongo02 节点上包含入口服务 mongos, 在 mongo03 节点上包含配置服务 config. 由于物理磁盘已做 raid 多副本备份, 考虑到存储效率, 主需要一个 MongoDB 备服务器, 所以采用 1 主+1 备+1 仲裁的集群节点分布策略 (仲裁节点负责在主服务器宕机情况下通过心跳机制将备服务器升级为主服务器), 表 2 给出了集群具体各个节点服务分布.

表 2 MongoDB 集群节点规划表

节点	Mongo01	Mongo02	Mongo03
入口和配置	Mongos	Mongos	Config
数据分布	Shard1 主 Shard3 副 Shard2 仲	Shard2 主 Shard1 副 Shard3 仲	Shard3 主 Shard2 副 Shard1 仲

表 3 给出了各服务端口分配情况.

表 3 MongoDB 集群各服务端口分配表

Mongos	Config	Shard1	Shard2	Shard3
20000	21000	27001	27002	27003

在 3 台节点的配置文件 conf 中配置好 3 台服务器的地址、数据目录等参数之后, 登录任意节点, 初始化副本集:

```
1. mongo --port 27000
2. #定义副本集配置 (键"_id"对应的值必须与配置文件中的 replicaction.replSetName 一致, priority 代表权重[1,100], 大的被分配为主服务器, 0 永久不会变为主服务器)
3. config = {
4. ···_id : "shard0",
5. ·····members : [
6. ·········{_id : 0, host : "mongo01:27001", priority : 1 },
7. ·········{_id : 1, host : "mongo02:27001", priority : 2 },
8. ·············{_id : 2, host : "mongo03:27001", arbiterOnly:true }
9. ·····]
10. ···}
11. #初始化副本集
12. rs.initiate(config)
13. #查看分区状态
14. rs.status();
```

初始化成功之后, 需要启用分片机制. MongoDB 的分片机制共有 Hash 和 range 两种, Hash 方法为随机分片, 按照分片键的值在副本集中随机分配, 利于数据均匀分布, 但是在按区间获取数据时会访问较多节点; range 方法根据分片键的值按照一定的范围分布, 在分区域获取数据时可以较少地访问不同节点, 但是无法做到数据足够均衡. 考虑到在 i-Harbor 中, 海量数据主要以批量上传为主, 较少有大规模按范围获取, 因此选择将主键 ObjectId 作为分片键, Hash 方法作为分片策略:

```
1. sh.addShard( "localhost:27 001")
2. sh.enableSharding("i-harbor")
3. sh.shardCollection("i-harbor.bucket01", { ObjectId : "hashed" } )
```

经过上述的关键步骤配置, 完成搭建一个副本集+分片键的 MongoDB 分布式高可用集群, 为元数据

存储模块提供稳定的数据存储引擎.

3.2 Rados 数据对象读写

在 2.3 对象数据管理模块中, 采用 Ceph 作为底层对象存储引擎, 此模块的关键在于数据从数据接口如何写入到 Ceph 中的 OSD (Object Storage Device), 因此需要设计合理的 rados 对象读写 IO 方法. 在 Ceph 中有原生的对象网关协议 RGW 支持 Amazon S3 和 Swift 的对象存储接口, 但是其认证方法和读写策略无法满足 i-Harbor 需求, 因此选择直接利用 librados 库实现对 rados 的读写的中间件.

在业务管理模块中, 数据接口和 Web 系统由 Python 开发实现, 但是在官方社区中对 librados 的 Python 版本支持较为滞后, 无法与当前版本兼容, 因此该中间件基于官方社区中的 go-ceph (对 librados 的 go 语言封装), 采用执行效率和开发效率更高的 go 语言, 实现对 rados 读写方法的封装, 编译为动态链接库.so, 为业务管理模块提供数据读写方法, 图 2 给出了该中间件结构设计.

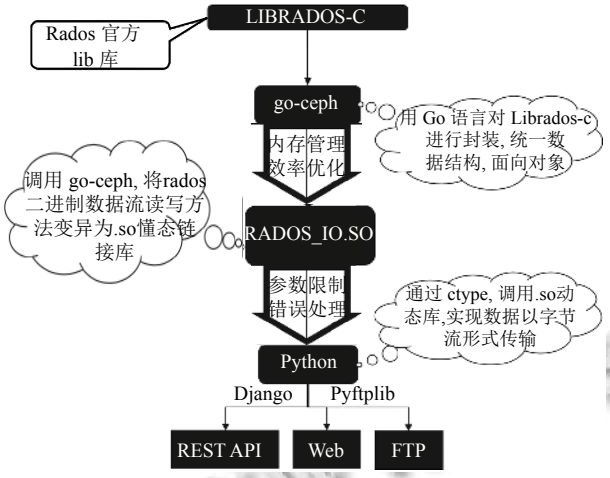

图 2 rados 读写中间件结构图

在该中间件中, 对 rados 的操作主要为 push、get、delete 和 list 四种方法, 主要功能为创建并写入对象、获取对象内容、删除对象、获取对象列表, 方法介绍如下:

(1) Push: 根据 oid 写入 rados 对象, 从 offset 位置起, 写入长度为 len, 数据内容为 data;

(2) Get: 获取 oid 的 rados 对象内容, 起始位置为 offset, 获取数据的长度为 block;

(3) Delete: 删除 oid 的 rados 对象;

(4) List: 获取指定 pool 中的所有 rados 对象 oid 列表.

表 4 给出了其所需参数.

表 4 Rados 读写参数表

	含义	类型	方法
cluster	Ceph 集群名	String	push、get、delete、list
user	用户名	String	
conf	Ceph 配置文件	String	
pool	存储池名	String	
oid	Rados 对象 id	String	push、get、delete
data	写入的数据内容 (push)	String	push
len	写入的数据长度 (push)	Int	push
offset	读/写的起始位置	Long int	push、get
block	读取数据的大小	Int	Get

经过上述设计, 实现 Ceph 底层对象到业务系统中的数据读写的中间件. 作为一个关键数据中转站, 其在 i-Harbor 中承担着非常重要的角色.

4 成果总结

4.1 系统环境

在地球科学大数据先导项目课题需求下, 搭建一套完整的计算+存储+网络系统环境. 共有 200 台对象存储服务器, 每台服务器 100 TB 硬盘, 部署 Ceph 分布式存储系统, 承担对象数据存储任务; 20 台元数据服务器, 每台服务器 10 TB 高速 SSD 固态硬盘, 部署分布式的副本集+分片制的 MongoDB 集群; 10 台业务系统服务器, 包含 Web 服务、API 接口服务、FTP 服务和 nginx 反向代理服务器; 3 台 Zabbix 监控服务. 节点间通信全部使用万兆以太网光纤.

4.2 系统成果

用户通过 Web 界面、API 或 FTP 接口, 创建存储桶, 在存储桶内创建任意数量的文件夹, 然后以存储桶为单位进行数据上传, 通过 Web 界面, 用户可以管理 FTP 和 API 数据接口认证. 图 3 给出了系统 Web 界面效果, 用户通过 Web 系统, 创建和删除存储桶, 实现文件夹和文件的创建、删除和下载, 同时管理 FTP 和 API 接口的认证方式.

用户通过 RESTful API 的调用接口, 将数据通过成熟的 HTTP 协议进行上传下载, 方便在各种在线数

据处理系统中调用存储桶中的数据, 图 4 给出 API 管理界面.

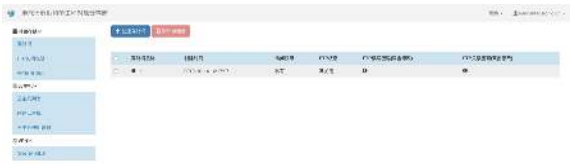

图 3 业务模块 Web 界面功能图

图 4 API 数据接口功能图

用户在业务管理系统中, 以存储桶为单位, 开启 FTP 连接, 管理 FTP 读写权限账户, 图 5 给出 Windows 下 FTP 数据接口映射说明, 用户通过网络驱动映射将某个存储桶通过 FTP 协议映射到本地, 实现本地浏览与线上系统同步, 方便用户进行数据上传下载.

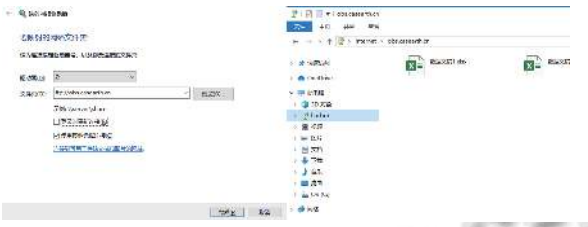

图 5 FTP 数据接口映射

5 结束语

针对在科研场景中海量级数据的高效率存储和数据共享需求, 本文设计并实现了 i-Harbor 分布式对象存储系统. 该系统以开源框架 Ceph 和 MongoDB 为对象数据和元数据存储引擎, 将整体底层架构合理解耦, 通过分布式技术实现弹性可扩展存储集群, 利用纠删码多分本等技术避免了单点故障, 保证了系统容灾性. 同时, 深度结合公有云对象存储服务, 提供丰富的多种数据读写接口, 方便用户在各种场景完成数据的批量上传下载, 同时设计合理数据共享策略, 方便用户在复杂的科研场景之下更高效的利用数据, 提升科研工作效率.

下一步将继续研究在科研场景中如何解决大数据量同时高并发上传瓶颈, 如何优化海量小文件上传这样的 IO 密集型任务[18]. 同时, 为更广泛利用 i-Harbor 存储系统的高效服务, 需要调研在数据处理场景下对于存储平台的需求, 实现对科研数据处理任务的支持.

参考文献

1 陈刚. 科学研究大数据挑战. 科学通报, 2015, 60(5): 439–444.
2 范中磊, 李小亮, 惠小红. 对象存储设备中对象的均匀分布方法. 微电子学与计算机, 2019, 36(6): 70–73.
3 Piernas J, Nieplocha J, Felix EJ. Evaluation of active storage strategies for the lustre parallel file system. Proceedings of the 2007 ACM/IEEE Conference on Supercomputing. Reno, NV, USA. 2007.
4 Schmuck FB, Haskin RL. GPFS: A shared-disk file system for large computing clusters. Proceedings of the Conference on File and Storage Technologies. Berkeley, CA, USA. 2002. 231–244.
5 Carns PH, Ligon III WB, Ross R, *et al*. PVFS: A parallel file system for linux clusters. Proceedings of the 4th Annual Linux Showcase & Conference. Vol 4. USENIX Association, 2000. 28.
6 Welch B, Unangst M, Abbasi Z, *et al*. Scalable performance of the panasas parallel file system. Proceedings of the 6th USENIX Conference on File and Storage Technologies. Berkeley, CA, USA. 2008. 1–17.
7 陈曦, 朱建涛, 何晓斌. 一种面向高性能计算的分布式对象存储系统. 计算机工程, 2017, 43(8): 69–73. [doi: 10.3969/j.issn.1000-3428.2017.08.012]
8 Factor M, Meth K, Naor D, *et al*. Object storage: The future building block for storage systems. Proceedings of 2005 IEEE International Symposium on Mass Storage Systems and Technology. Sardinia, Italy. 2005. 119–123.
9 Weil SA, Brandt SA, Miller EL, *et al*. Ceph: A scalable, high-performance distributed file system. Proceedings of the 7th Symposium on Operating Systems Design and Implementation. Berkeley, CA, USA. 2006.
10 Davies A, Orsaria A. Scale out with GlusterFS. Linux Journal, 2013, 2013(235): 1.
11 Arnold J. Openstack swift: Using, administering, and developing for swift object storage. Sebastopol, CA, USA: O'Reilly Media, Inc., 2014. 10–17.

12 Morris MR, Horvitz E. S^3: Storable, shareable search. In: Baranauskas C, Palanque P, Abascal J, *et al*.. eds. Human-Computer Interaction–INTERACT 2007. Berlin: Springer, 2007.

13 Friedman MB. RAID keeps going and going and…[magnetic disk storage]. IEEE Spectrum, 1996, 33(4): 73–79. [doi: 10.1109/6.486635]

14 Weatherspoon H, Kubiatowicz J D. Erasure coding Vs. Replication: A quantitative comparison. First International Workshop on Peer-to-Peer Systems. Cambridge, MA, USA. 2002. 328–337.

15 Hows D, Membrey P, Plugge E, *et al*. Replication. In: Hows D, Membrey P, Plugge E, *et al*., eds. The Definitive Guide to MongoDB. Berkeley: Apress, 2015.

16 郭晓慧, 李润知, 张茜, 等. 基于 Zabbix 的分布式服务器监控应用研究. 通信学报, 2013, 34(S2): 94–98.

17 方圆, 杜祝平, 周功业. 基于对象存储的新型元数据管理策略. 计算机工程, 2012, 38(3): 25–27. [doi: 10.3969/j.issn.1000-3428.2012.03.009]

18 屠雪真, 黄震江. 一种海量小文件对象存储优化方案. 计算机技术与发展, 2019, 29(8): 31–36. [doi: 10.3969/j.issn.1673-629X.2019.08.006]