第 8 卷 第 5 期
2019 年 9 月
集 成 技 术
JOURNAL OF INTEGRATION TECHNOLOGY
Vol. 8 No. 5
Sep. 2019

引文格式：
李诗逸, 古亮, 喻之斌. 基于堆叠式分布式文件系统的端到端校验 [J]. 集成技术, 2019, 8(5): 13-25.
Li SY, Gu L, Yu ZB. End-to-end data integrity for stacked distributed file system [J]. Journal of Integration Technology, 2019, 8(5): 13-25.

基于堆叠式分布式文件系统的端到端校验

李诗逸[1,2] 古 亮[2] 喻之斌[1]

[1](中国科学院深圳先进技术研究院 深圳 518055)
[2](深信服科技股份有限公司 深圳 518071)

摘 要 端到端校验是一种有效的数据完整性检测手段，可为分布式存储系统提供基本的可靠性保证。Glusterfs 是一种常用的堆叠式分布式文件系统，但缺乏有效的数据完整性检测机制，存在用户数据遭受破坏而无法被发现的风险，即返回错误数据给用户。这种风险在某些情况还会扩散，造成多副本或灾备、双活情况下的数据丢失。针对这一问题，该文提出了一种高性价比的基于 Glusterfs 的端到端校验方案(命名为 Glusterfs-E2E)，可以有效解决 Glusterfs 文件系统中存在的数据完整性风险。该方案不但可以提供全路径的保护，具备 2%～8% 的高性能开销，而且还可以提供软件故障的定位功能。

关键词 静默错误；数据完整性；端到端校验；分布式文件系统
中图分类号 TP 338.8 **文献标志码** A doi: 10.12146/j.issn.2095-3135.20190729002

End-to-end Data Integrity for Stacked Distributed File System

LI Shiyi[1,2] GU Liang[2] YU Zhibin[1]

[1](*Shenzhen Institutes of Advanced Technology*, *Chinese Academy of Sciences*, *Shenzhen* 518055, *China*)
[2](*Sangfor Technologies Inc.*, *Shenzhen* 518071, *China*)

Abstract End-to-end checksum is an effective means of data integrity detection, which can provide basic reliability guarantee for the distributed storage systems. Glusterfs is a popular stacked distributed file system, but it lacks an effective data integrity detection mechanism. User data storage in the Glusterfs have a risk of being damaged and not being discovered. Moreover, this kind of risk can spread in some cases, causing data loss even with the protection of multiple copies or disaster recovery. This paper proposes a cost-effective Glusterfs-based end-to-end checksum scheme called Glusterfs-E2E, which can effectively solve the data integrity risk of Glusterfs. The proposed solution can not only provide full path protection, 2% to 8% performance overhead, but also can locate software bugs.

Keywords silent corruption; data integrity; end-to-end checksum; distributed file system

收稿日期：2019-07-29 **修回日期**：2019-08-18

作者简介：李诗逸，博士，研究方向为存储系统和分布式系统的可靠性研究；古亮，博士，研究方向为安全攻防、漏洞/文件检测、威胁情报、云计算、大数据及 AI 平台构建和优化；喻之斌(通讯作者)，博士，研究员，研究方向为计算机体系结构、计算系统虚拟化、云计算及大数据分析平台构建与优化，E-mail：zb.yu@siat.ac.cn。

1 引 言

随着数据规模的增加和数据存储需求的增大，大规模的分布式系统和数据中心存储系统被用来存储用户数据并提供数据存储服务。分布式存储系统为各种云应用提供数据管理服务，如互联网搜索、照片和视频服务[1-3]、社交网络[4-5]、运输服务[6-7]和电子商务[8]。但应用程序的多样性、复杂冗长的输入/输出(Input/Output，I/O)路径、大量使用廉价而低可靠的硬件，这些给系统带来严峻的可靠性问题。

现代存储系统通常以副本[9]或纠删码[10]的方式来存储数据以提供高可靠性和容错能力。副本和纠删码以分条冗余的方式分布在系统的各个服务器上，从而可以解决诸如系统崩溃、断电、磁盘和网络故障等大多数错误[11-13]，但无法解决一些微妙的错误，如静默错误[14-17]。其中，很多因素会造成静默错误，如丢弃写入或写入错误位置等罕见事件会在磁盘上留下过时或损坏的数据[16,18-19]。由于芯片缺陷[20-22]或辐射[23-24]，存储器中的位会被翻转。软件错误也是数据损坏的根源，这源于低级设备驱动程序[25]、系统内核[26-27]和文件系统[28-29]。更糟糕的是，设计缺陷并不罕见，并可能导致严重的数据丢失或损坏[30]。这些静默错误都会给数据完整性带来威胁。

现代存储系统最重要的职责之一是保持数据完整性，因此多年来已经开发并应用了许多技术来改善其完整性。例如，各种校验和广泛应用于多种计算机组件，包括磁盘[31]、系统总线[32]和网络协议[33]。冗余，特别是以副本、纠删码的形式，通常用于提供恢复。当数据跨组件传输时，数据仍然可能不受保护。更全面的数据保护方法应该采用“端到端”的理念[34]。如动态文件系统(Zettabyte File System，ZFS)[35]中就提供了端到端数据完整性保护，然而仍然有一些分布式文件系统，如 Ceph[36]、Glusterfs[37]，并没有提供端到端数据完整性保护。因此，Glusterfs 和 Ceph 系统中存储的数据存在遭受破坏而无法发现的风险，并在用户读取时可能返回错误数据给用户。在有其他故障并进行恢复的情况下，这些错误数据还会被当作“正确”的数据去恢复，从而造成扩散。最终，即使在有多副本的情况下仍会造成数据丢失[38]。

虽然 Glusterfs 和 Ceph 都是分布式文件系统，但它们之间还是有比较大的差别。Glusterfs 是堆叠式和无元数据中心架构设计。基于 Glusterfs 的云存储产品有深信服的 aSAN[39]、TaoCloud 的 XDFS[40]。本文以 Glusterfs 为例，分析研究基于分布式文件系统的端到端校验方案。Glusterfs 原生的 BitRot 模块，被认为具备磁盘错误检测功能，但该方案粒度大、非实时，且非端到端。

因此，本文重点研究和设计基于 Glusterfs 的端到端校验，提出了一种高性价比的端到端校验方案，即 Glusterfs-E2E(End-to-end Checksum for Glusterfs)，可以有效解决 Glusterfs 的数据完整性风险。该方案不但可以提供全路径的保护，性能开销低(2%～8%)，而且还可以帮助协助定位各种软件问题(如软件 bug)。

2 背景知识

本节主要对静默错误、软件校验和与分布式文件系统三个方面的背景知识进行介绍。

(1)静默错误：由于源自存储堆栈的不同层的许多原因导致磁盘损坏。

①硬件问题。磁介质中的错误导致“比特翻转”的问题，其中某个位或几个位的磁特性被损坏。电源波动、不稳定的手臂运动和介质划痕也会导致磁盘块损坏[41-42]。

②固件问题。由于复杂驱动器固件中的错误(现代驱动器包含数十万行固件代码)，也会导致错误。一些报告的固件问题包括错误的写入、固

件意外写入错误的位置[43]或丢失的写入，磁盘会报告写入已完成，而实际上从未到达磁盘[44]。另外，还发现总线控制器错误地将磁盘请求报告为完整或损坏数据[45]。

③软件问题。操作系统中的软件错误也是潜在的静默错误来源。有问题的设备驱动程序可以发出带有错误参数或数据的磁盘请求[25-27]。文件系统本身的软件错误可能导致将错误的数据写入磁盘。

(2)软件校验和：校验和是使用抗冲突哈希函数计算的块哈希，用于检测在其存储或传输期间可能引入的错误。校验和通常用于验证数据完整性，但不依赖于验证数据的真实性。从数据输入产生校验和的实际过程称为校验和算法。校验和技术经过几十年的发展不断成熟，越来越多高效的校验和算法被提出，如 Fletcher、循环冗余码校验(Cyclic Redundancy Check，CRC)、Adler、XXhash 等算法[46-48]。

根据其设计目标，良好的校验和算法通常会输出显著不同的值，即使对输入进行小的更改也是如此，从而可用于检测许多数据损坏错误并验证整体数据完整性。如果当前数据输入的计算校验和与先前计算的校验和的存储值相匹配，那么数据未被意外更改或损坏的可能性非常高。一些纠错码基于特殊的校验和，不仅可以检测常见错误，而且可以在某些情况下恢复原始数据。

(3)基于堆叠式模块设计的分布式文件系统：分布式文件系统是指文件系统管理的物理存储资源不一定直接连接在本地节点上，而是通过计算机网络与节点相连。分布式文件系统的发展十分迅速，且是云计算的基础架构和核心系统。从最初谷歌公司提出的谷歌文件系统，后续的各大分布式文件系统都在其基础上进行优化和改进。现在最常见的三大主流开源分布式文件系统，分别是 Hadoop[49]、Ceph 和 Glusterfs。

分布式文件系统中细分成不同的子类，因此整个分布式文件系统可以划分为多个模块，而对这些模块采用堆叠式的方式进行组织，从而形成从上到下一层层的堆叠式结构。这种文件系统被称为基于堆叠式模块设计的分布式文件系统，其典型代表是 Glusterfs。以 Glusterfs 为代表的基于堆叠式模块设计的分布式文件系统具有很好的可扩展性、良好的软件结构设计、易于扩展和配置、各个模块灵活搭配的特点。

Glusterfs 中进行数据完整性保护的是其原生的 BitRot 模块。该模块以文件粒度生成校验和，为了保证提供足够的检测能力，使用了强检测能力的 SHA-256 算法。但该算法执行效率低，对小文件显得十分低效。此外，只有在一个文件被认为稳定后才生成校验和，即当一个文件被经常访问时，无法生成校验和。但事实上，经常被访问的文件更应该提供数据完整性保护以检测数据错误，避免返回错误数据给用户，造成严重的损失和后果。

因此，本文的目标是为以 Glusterfs 为代表的堆叠式分布式文件系统重新设计端到端校验方案，并具备低开销、全路径、端到端的特性。接下来，将介绍基于 Glusterfs 的端到端校验的设计和架构。

3　设计和实现

3.1　系统架构

首先，介绍基于 Glusterfs 的端到端校验的系统架构，如图 1 所示。整个架构引入 3 个大模块：Checksum 引擎、Verify 引擎和 Checksum 管理模块。

(1)Checksum 引擎：主要是为各种数据与校验和(自验证)提供生成校验和服务。为了保证通用性，多个模块可调用，Checksum 引擎是以 LIB 库的形式提供的。LIB 库的形式也方便后续更新，如果有更高效、更符合需求的校验和算法，可以很容易合入。Checksum 引擎中集成

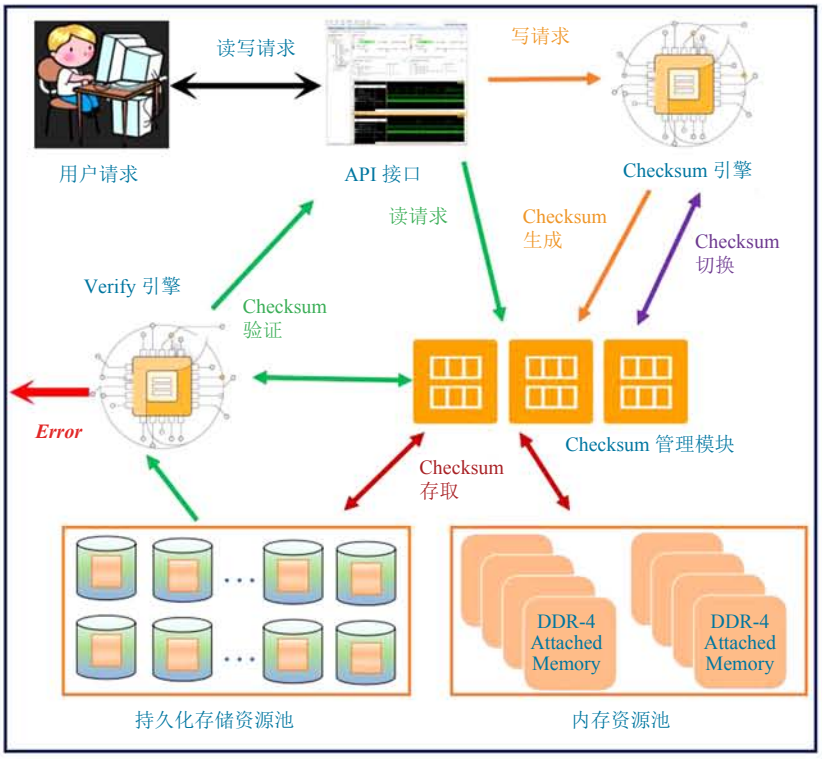


图 1 Glusterfs-E2E 的架构

Fig. 1 The architecture of Glusterfs-E2E

了多种校验和算法(如 CRC、Fletcher、Adler 和 XXhash 等)，并提供默认推荐配置。

(2) Verify 引擎：主要是提供校验和验证服务，类似 Checksum 引擎。Verify 引擎是以 LIB 库的形式提供的，以保证通用性和多个模块可调用。此外，Verify 引擎还要在校验和验证不通过时进行错误告警，触发 Glusterfs 的错误处理机制。

(3) Checksum 管理模块：Checksum 管理模块的功能比较多。首先，要维持数据与校验和的一一映射，同时提供索引服务，可以很容易地通过数据查找对应的校验和。在校验和持久化存储时要保持与数据的一致性。典型的不一致情况就是数据已经更新，但是校验和仍未更新。此时若使用旧校验和去验证新数据，系统会显示验证不通过并报错，而实际上这是误报，将会带来不好的后果。由于内存是有限的，因此，Checksum 管理模块要对校验和进行管理，以控制校验和的内存开销。另外，Checksum 管理模块还需提供当前校验和算法与配置的相关信息。

然后，分析引入端到端校验后 Glusterfs 的 I/O 流程。用户的读写请求都是通过前端应用程序编程接口发送给后端 Glusterfs 存储。

(1)针对写请求

①一进入 Glusterfs，就把写请求携带的待写入数据输入给 Checksum 引擎生成校验和。

②然后，校验和存储在校验和管理模块中，而写入数据再存储到磁盘、固态盘(Solid State Disk，SSD)等持久化存储之前会经过多个 Glusterfs 模块(如路由选择模块 afr、dht 等)。这些模块会进行各种处理，可能会对写入数据引入错误，常见的如软件踩内存问题、远程过程调用通信干扰造成的比特翻转等。因此，在此期间校验和管理模块要保证写入数据与其校验和的一一对应及随时可查找，以便及时发现

数据错误。

③当写入数据存储到磁盘、SSD 等持久化存储时，校验和也要保存到持久化存储中。该过程要注意断电、系统崩溃带来的一致性问题。

④在整个过程中，校验和本身也可能因为各种原因被损坏或发生错误，因此校验和本身会有自校验。

(2)针对读请求

①读请求在访问磁盘、SSD 等持久化存储，并读取所需数据之前，没有任何改变。

②从读取数据开始有区别，需要在读取数据时，提高 Checksum 管理模型去读取其之前写入时对应生成并存储的校验和。

③一旦从磁盘、SSD 等持久化存储返回所读数据到 Glusterfs 中，就把所读数据输入给 Checksum 引擎重新生成新校验和。

④把新校验和与原校验和都输入给 Verify 引擎，Verify 引擎进行比对验证。验证不通过的会进行错误报警，同时触发 Glusterfs 的错误处理机制。

由此，写请求只需要和 Checksum 引擎、Checksum 管理模块交互，而读请求需要和 Checksum 引擎、Verify 引擎、Checksum 管理模块交互。

如上即是基于 Glusterfs 的端到端校验的整体设计和架构，以及相应的 I/O 流程。接下来将从实现的角度来描述具体实现上的一些细节。

3.2　设计和实现

在设计和实现过程中有如下几个主要的注意点：

(1)可以看到写请求一进入系统就需要对待写入数据生成校验和，此过程放在 Glusterfs 的最上层模块(在本文设计中，放在网络文件系统(Network File System，NFS)模块中)。

(2)类似地，读请求一旦从磁盘、SSD 等持久化存储读取数据返回 Glusterfs 后就需要重新生成校验和并输入给 Verify 引擎进行对比验证，此过程放在 Glusterfs 的最底层模块（在本文设计中，放在块设备(Block Device，BD)模块中)。

(3)Checksum 引擎和 Verify 引擎是通过 LIB 库提供的，因此每个模块都可以调用这两个服务，同时引入新的校验和算法也更容易。

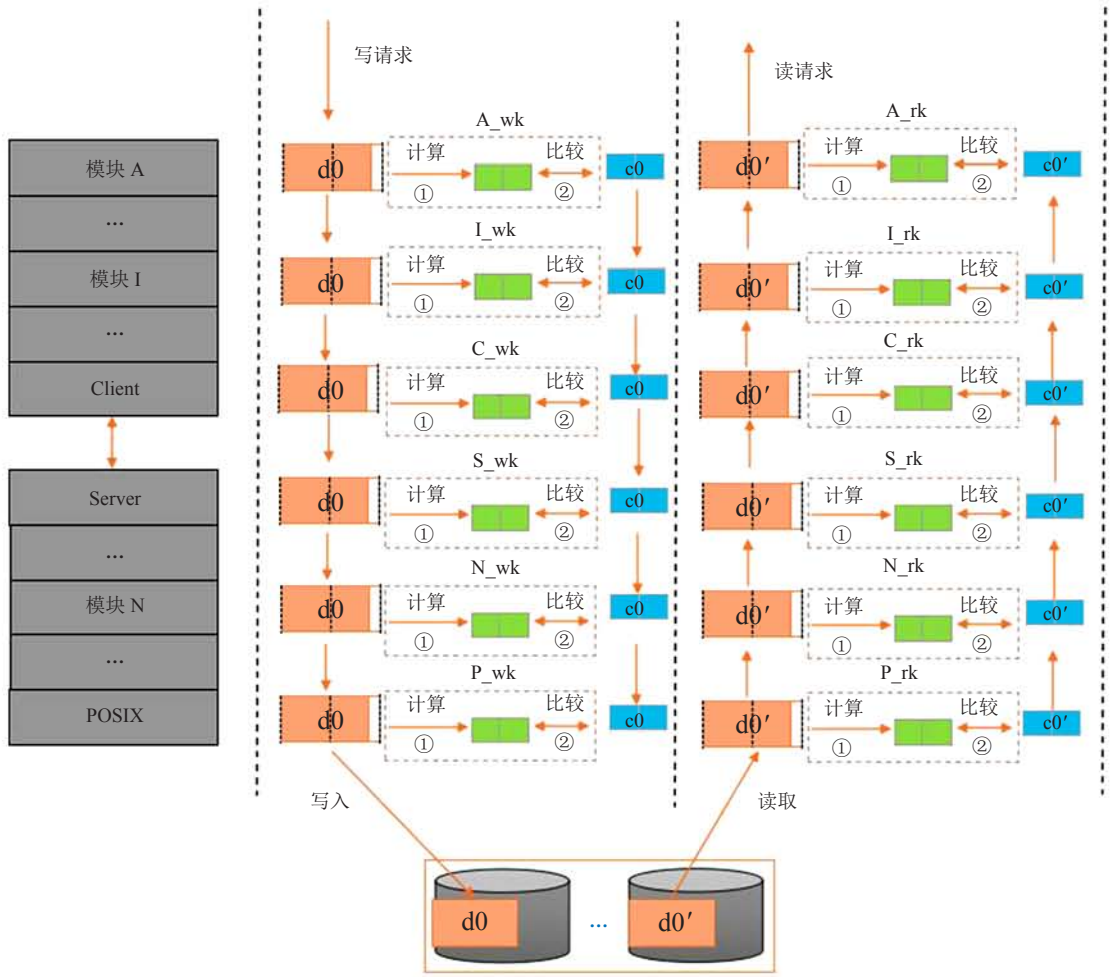
(4)当写入数据存储到磁盘、SSD 等持久化存储时，校验和也要保存到持久化存储中。同时还要保证一致性。

(5)为了对 Glusterfs 这种基于堆叠式模块设计的分布式文件系统的各个模块进行校验和保护及验证，本文做了独特设计，并把该设计命名为基于 Glusterfs 的模块间校验。该设计可以有效地发现各模块可能存在的数据损坏问题，特别是新模块开发时，及时发现错误，帮助快速定位，防止错误扩散。

图 2 为基于 Glusterfs 的模块间校验的整体过程。读写数据在经过每一个模块时都会重新生成校验和并与原始校验和进行验证。

(1)模块开关。我们发现没有必要让每个模块一直都保持校验和验证，很多模块经过持续观察已经很稳定后，就无需继续保持校验和验证。因此，本文提供动态开关，可以根据需要来开关每个模块的校验和验证。在我们的经验中，开发中的模块以及和开发中模块强关联的模块保持常开，即使在开发完成后也要在生产环境中继续打开一段时间，这样才能有效地发现错误，取得更好的效果。而其他模块并不需要打开，如有需要可以偶尔打开。

(2)无需每个模块改动。我们发现 Glusterfs 有提供 STACK_WIND 和 STACK_UNWIND 宏在各模块间进行交互时被调用。因此，本文的实现就嵌入在 STACK_WIND 和 STACK_UNWIND 宏中，这样无需改动每个模块就能实现模块间校验。



注：Client 和 Server 分别表示 Glusterfs 的客户端和服务端；POSIX 表示可移植操作系统接口

图 2 基于 Glusterfs 模块间校验的整体过程

Fig. 2 The overall process of Glusterfs-based inter-module verification

4 挑 战

在实际系统中，要实现一个高效的端到端校验存在多项挑战：使用什么样的校验和算法来生成校验和？校验和的存放位置，怎么保证校验和的正确性？

4.1 校验和算法

首先来看第一个问题，现在业界使用的校验和算法主要有：CRC-32（华为[50]、VMware vSAN[51]、Ceph[36]），MD5，Adler-32（Nutanix[52]），SHA-1，SHA-256（Glusterfs[37]）等。然而，本文基于 Glusterfs 的端到端校验方案并没有像其他厂商一样使用上述算法，而是使用 XXhash 算法。

XXhash 是一种极速的哈希算法，以内存速度限制运行。该算法成功完成了 SMHasher 测试套件，该套件用以评估哈希算法的冲突、随机性等。XXhash 具有高度可移植性，并且所有平台上的哈希都是相同的（小/大端）。关于 XXhash 算法的详细描述请参考文献[48]。

校验和算法有两个主要的评价标准：

（1）生成校验的速度，一般以 MB/s 或 GB/s 为单位。

(2)冲突率和均匀性。冲突率是指两个数据不一样，但生成一样的校验和的概率。均匀性是指每个数据的冲突率都差不多，不会出现一些数据比较容易出现冲突的情况。

一般而言，速度快、冲突率低、均匀性好的校验和算法更受欢迎。各类算法的对比结果如表 1 和表 2 所示。

表 1　各校验和算法生成校验的速度

Table 1　The speed of checksum calculation

算法类型	生成校验的速度（MB/s）
XXhash64	～13 000
ISA-L(CRC32)	～10 000
Adler-32	～4 000
CRC32	～700
MD5	～500
SHA-1	～400
SHA-256	～200

注：“～”表示量级

表 2　各校验和算法的冲突率

Table 2　The collision rate of various checksum algorithm

算法类型	冲突率
XXhash64	$\sim 2^{-64}$
ISA-L(CRC32)	$\sim 2^{-32}$
Adler-32	$\sim 2^{-32}$
CRC32	$\sim 2^{-32}$
MD5	$\sim 2^{-128}$
SHA-1	$\sim 2^{-160}$

注：“～”表示量级

由此可见，XXhash64 不但速度最快，可达 13 GB/s，而且冲突率低(2^{-64})。本文在真实环境中进一步测试，采用中央处理器(Central Processing Unit，CPU)：Intel(R) Core(TM) i7-4770K CPU @3.50 GHz。同时，采用 6 类负载来测试 XXhash 的适用性。其中，exe 是执行程序数文件；mp4 是视频类文件；vms 是 Qcow2 镜像文件；rar 是压缩类文件；syn 是合成的随机数组成的文件(类似哈希化后的数据库数据镜像负载)；src 是 gcc 源码数据文件。这 6 类负载是目前云计算环境中比较常见的负载。

测试方法如下：

(1)速度测试。主要测试校验算法的哈希速度(MB/s)的差异，每个数据都测试 3 次，再取平均值，以减少 CPU 状态不稳定或者其他计算的干扰。

(2)冲突和均匀性测试。通过测试校验算法的均匀性，统计分布在每个哈希值上的累计次数，从而得出哈希值分布的均匀性。

表 3 是在真实环境下各校验算法的速度测试结果。可以看到，在 6 种负载下 XXhash 均能稳定地提供＞12 GB/s 的速度。而在图 3 的冲突均匀性测试中，以 exe 和 src 负载为例，XXhash 的均衡性十分好。真实环境中的测试验证了 XXhash64 在 6 种常见的云计算负载下，不但速度快，而且冲突率低、均匀性好。

4.2　校验和存储

校验和在内存中生成，并可跟随数据一起传输、存储。当数据存储到磁盘、SSD 等非易丢性存储中，校验和也需要进行存储。这样会带来额外的写开销，最初始的方法就是把校验和与数据一起存储。基于此，衍生出了 T10-PI[53](华为等厂商使用)。但该方法存在两个缺陷：

表 3　在真实环境下各校验算法的速度测试结果

Table 3　The speed of checksum calculation in the experiment

算法类型	哈希速度（MB/s）					
	exe	mp4	vms	rar	syn	src
XXhash64	12 827.67	12 892.15	12 715.00	12 789.93	12 731.73	12 529.10
Fletcher	1 785.73	1 797.23	1 786.16	1 748.07	1 777.50	1 779.08
CRC32	481.79	482.51	481.57	482.42	475.57	474.88

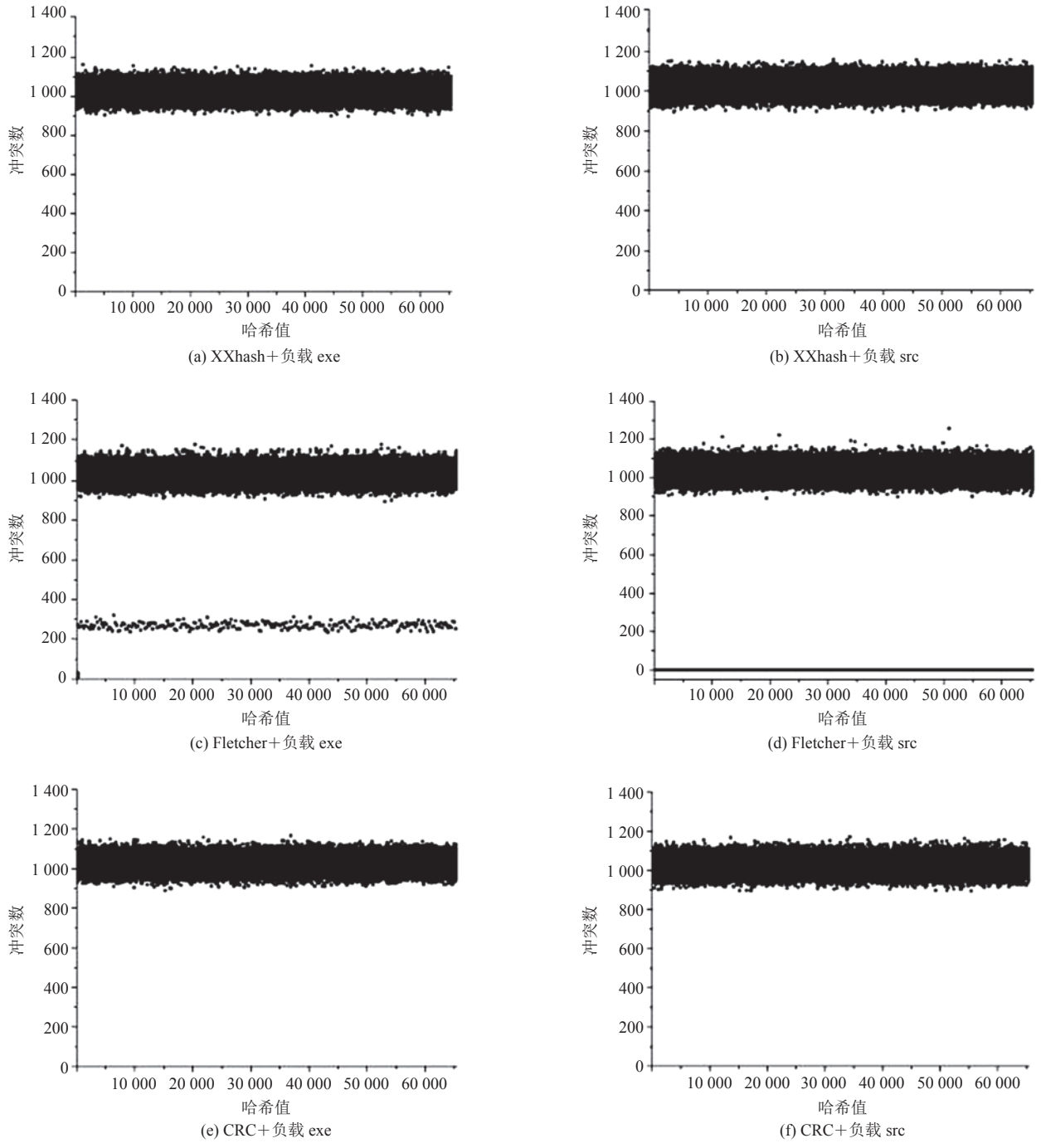


图 3 三种校验算法在 exe 和 src 负载上的均匀性测试结果

Fig. 3 Uniformity test results for the three checksum algorithms above the exe and src workloads

一是存储时存在数据与校验和一起丢失的情况(即 lost write)；二是无法发现同时写错误的情况(即 misdirect write)。为了解决上述两个问题，Prabhakaran 等[19]引入了数据完整性段(Data Integrity Segment，DIS)来解决。

另一种方法就是分离存储，把校验和与数据分开存储。该方法最大的问题就是额外的写开销问题。2004 年诞生的 ZFS 使用父子校验[35]来解

决额外的写开销，即把数据的校验和与父目录存放在一起，在写数据时肯定会更新父目录，这样就可以一同把校验和与父目录进行更新，从而解决写开销问题。该方法衍生到分布式系统，就是把校验和放在元数据中心，将校验和与元数据一起更新(Nutanix 使用该方法，把校验和存储在 Cassandra 分布式数据库中[52])。但该方法只适合有元数据中心的分布式系统，并且会带来额外的网络开销。

因 Glusterfs 是无元数据中心架构，所以本文设计的基于 Glusterfs 的端到端校验采用校验和与数据一起存储，并引入 DIS 来解决 lost write 和 misdirect write 的问题。相较于校验和与数据分离存储的方式，校验和与数据一起存储的一致性设计要简单很多，只要保证数据与校验和的单个写的原子性即可。这也是本文选择校验和与数据一起存储的另一个问题。

后面的实验证明了该方案可以在保证正确性、有效性的基础上提供高性能。Glusterfs 自带的 BitRot 功能，是以文件为单位，并且文件稳定时(所有文件描述符关闭且一定时间内没有更改操作)才生成校验和。该方法变长粗粒度，并且对不稳定的文件无法提供校验功能。

5　实　验

本节对本文设计的基于 Glusterfs 的端到端校验的整体性能进行评估。所有实验是在 3 台服务器搭建的 Glusterfs 集群上执行的。每台服务器配备有 Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40 GHz 处理器，128 GB 内存，3 块 SSD(其中 1 块 128 GB SSD 用作系统盘、2 块 480 GB SSD 用作缓存盘)，6 块 2 TB 西数磁盘。Glusterfs 为 3.5.3 版本，操作系统使用 linux 3.10.0 内核。另外，本文使用 iometer 作为测试基准。

首先，测试所有模块都打开校验开关的情况下，基于 Glusterfs 的端到端校验的性能开销。作为对比，我们还实现了 Fletcher 算法和 CRC 算法。此外，将最开始的实现称为 orig-xxhash。我们在实现过程中发现锁机制、内存拷贝可以进一步进行优化。所以，可以通过锁优化，减少锁冲突；同时，通过结构体复用减少内存拷贝来进行优化，优化后的称为 opti-xxhash。作为对比的 Fletcher 和 CRC 也都采用了这些优化，对应图 4 中的 fletcher 和 crc。如图 4 所示，纵坐标是有端到端校验相对于没有端到端校验的归一化性能；横坐标是 iometer 的深度参数。四个子图分别对应 4 KB 大小随机全读(4 KB rand 0w)，4 KB 大小随机 30% 写(4 KB rand 30%w，相当于 70% 是读、30% 是写)，4 KB 大小随机全写(4 KB rand 100%w)，1 MB 大小随机全写(1 MB rand 100%w)。可以看到，opti-xxhash 在各种深度、负载、I/O 大小下都提供高性能(始终＞92%，说明只有不到 8% 的性能影响)。特别在大块 I/O 和多深度时发现，锁和拷贝的优化效果十分明显。opti-xxhash 相对于 orig-xxhash 在 4 KB rand 30%w、32 深度时约有 26% 的优化，在 1 MB rand 100%w、32 深度时约有 220% 的优化。这是因为大块和高深度时竞争更激烈，所以对性能影响也越大。通过细粒度的优化锁机制和内存拷贝十分重要，可以有效地减少 Glusterfs-E2E 的性能开销，特别是在大块和多深度下。同时，可以看到 fletcher 和 crc32 在大块 I/O 和多深度时性能越来越差。fletcher 和 crc32 在 4 KB rand 0w、32 深度时分别大约只有 0.49 和 0.25 的原生性能，在 1 MB rand 100%w、32 深度时分别大约只有 0.26 和 0.1 的原生性能。这是因为计算开销大，计算速度慢，CPU 资源消高，在大块 I/O 和多深度时对 CPU 资源的竞争更激烈。而 opti-xxhash 并没有这个现象，无论是大块还是多深度，对 opti-xxhash 的影响都不大。这是由

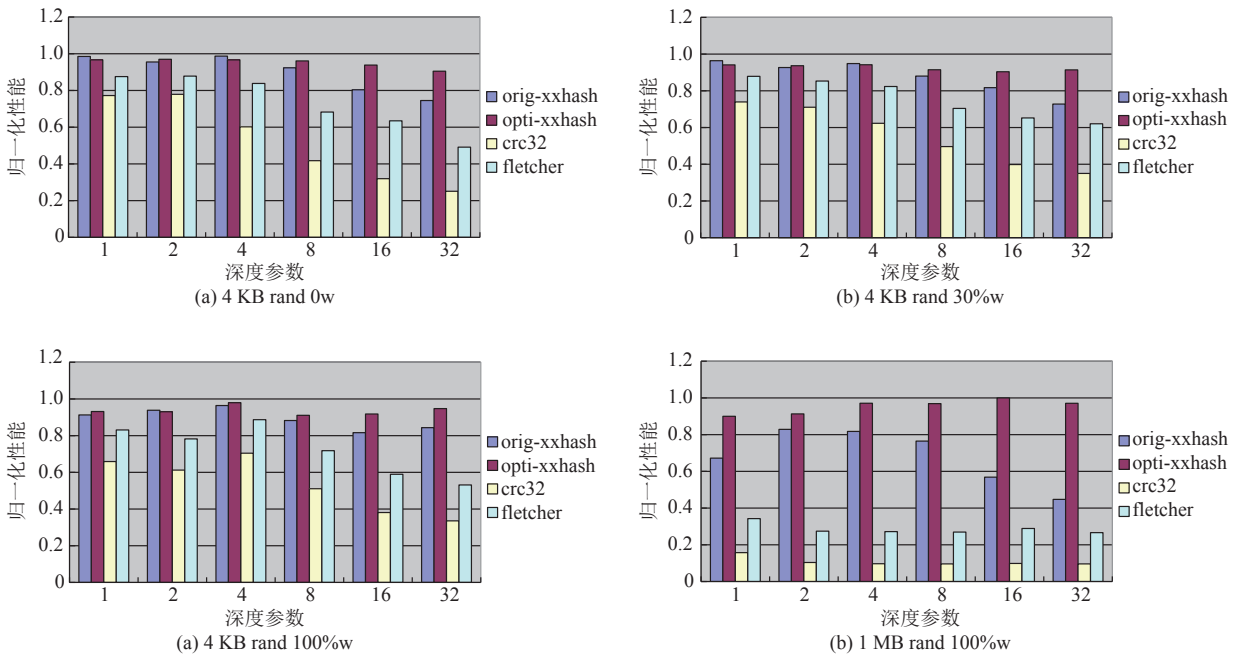


图 4　所有模块都打开校验开关时 Glusterfs-E2E 的性能开销

Fig. 4　Performance overhead of Glusterfs-E2E with all modules' verification turned on

于 XXhash 算法本身的开销比较小，计算速度快，没有成为瓶颈；另外，细粒度的优化锁机制和内存拷贝减少了加锁和内存拷贝的开销。

所有模块都打开校验开关是最极端的情况，不过这种极端情况可以使我们发现每个模块因为软件开发、网络通信、内存使用过程中的各种异常造成的数据错误。

然后，测试打开模块数最小的情况(只打开最上层的 NFS 模块和最底层的 BD 模块来维持端到端校验)下，基于 Glusterfs 的端到端校验的开销。如图 5 所示，纵坐标是有端到端校验相对于没有端到端校验的归一化性能；横坐标是 iometer 的深度参数。由图 5 可以看到，Opti-xxhash 在各种深度、负载、I/O 大小下都提供极高的性能(始终＞98%，说明只有不到 2% 的性能影响)。由于实验有一定的测试误差，有时 Glusterfs-E2E 比原生 Glusterfs 的性能还略好，而不到 2% 的性能影响也包含了测试误差。

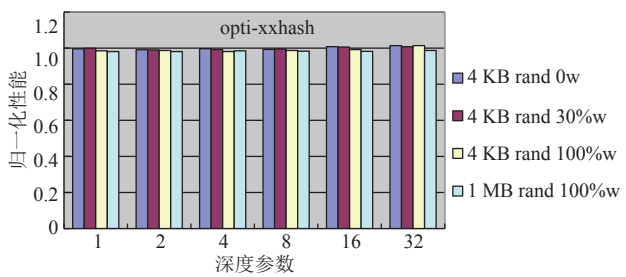


图 5　只打开最上层的 NFS 模块和最底层的 BD 模块开关时 Glusterfs-E2E 的开销

Fig. 5　Performance overhead of Glusterfs-E2E with only the top-level NFS module and the lowest-level BD module's verification turned on

因为只有不到 2% 的性能影响就可以给 Glusterfs 提供端到端校验，而如果还需要提供模块间校验功能，即使是给所有模块提供校验功能的最极端情况，最多也只有不到 8% 的性能影响。所以，Glusterfs-E2E 可以较小的性能开销，提供全路径的数据保护并协助定位软件故障。

6　相关工作

在传统的系统中，已经有很多关于端到端数据完整性的工作，并取得了很好的效果，如 T10-PI、ZFS[53]、DIS。但在开源分布式系统中，如 Ceph、Glusterfs，仍没有提供端到端校验。Ceph 是利用对象的方式，把校验和与元数据存放在一起，但校验和是在对象存储设备端写操作时才生成，所以 Ceph 暂不支持端到端校验。而在 Glusterfs 中，BitRot 功能以文件为单位，且文件稳定时才生成校验和，该方法也是非端到端。因此，本文有借鉴传统系统中的一些方法，如引入 DIS，同时在分布式系统(Glusterfs)上实现端到端校验。

现有的企业分布式系统中，Nutanix 公司把校验和存储在 Cassandra 分布式数据库中[52]，华为公司采用父子校验[54]。这些方法都是采用把数据与校验和分离存储的方式，而本文的实现采用的是把数据与校验和一起存储的方式。

此外，在校验和的选取上，华为[50]、vSAN[51]等公司的产品采用了 CRC32 算法，Nutanix 公司的产品则采用了 Adler 算法[52]。而本文的实现采用具有更快速度、更低冲突率且均匀的 XXhash 算法。

本文所描述的数据完整性检测都是针对强信任环境，而在弱信任环境中，数据完整性的检测通常会采用数据持有性证明技术[55-56]。弱信任环境的数据完整性检测超出了本文的讨论范围，因此不做讨论和分析。

7　总　结

针对分布式系统中(如 Glusterfs)无法提供端到端校验的不足，本文设计和实现了一种具有高性能(2%～8% 的性能开销)、全路径保护的基于 Glusterfs 的端到端校验。为了减少端到端校验的性能开销，我们做了很多工作。首先，通过测试选用速度快、冲突率低、均匀性好的 XXhash 算法；其次，选择校验和与数据一起存储的方式；最后，在实现中进行锁优化和内存拷贝优化。此外，该方法还设计实现了 Glusterfs 的模块间校验来最大化帮助协助定位软件故障(如软件 bug)。总之，基于 Glusterfs 的端到端校验可以显著改善 Glusterfs 中数据存在遭受破坏而无法发现的不足，提高可靠性。

参 考 文 献

[1] Datastax. Netflix cassandra usecase [EB/OL]. 2019[2019-07-26]. http://www.datastax.com/resources/casestudies/netflix.

[2] Redis. Instagram architecture [EB/OL]. 2018[2019-07-26]. http://highscalability.com/blog/2012/4/9/the-instagram-architecture-facebookbought-for-a-cool-billio.html.

[3] Redis. Redis at flickr [EB/OL]. 2018[2019-07-26]. http://code.flickr.net/2014/07/31/redis-sentinel-atflickr/.

[4] Twitter. Kafka at twitter [EB/OL]. 2017[2019-07-26]. https://blog.twitter.com/2015/handling-fivebillion-sessions-a-day-in-real-time.

[5] Voldemort. Project voldemort [EB/OL]. 2018[2019-07-26]. http://www.project-voldemort.com/voldemort/.

[6] Uber. The uber engineering tech stack, part I: the foundation [EB/OL]. 2019[2019-07-26]. https://eng.uber.com/tech-stack-part-one/.

[7] Uber. The uber engineering tech stack, part II: the edge and beyond [EB/OL]. 2019[2019-07-26]. https://eng.uber.com/tech-stack-part-two/.

[8] MongoDB. MongoDB at eBay [EB/OL]. 2019[2019-07-26]. https://www.mongodb.com/presentations/mongodbebay.

[9] Ghemawat S, Gobioff H, Leung ST. The google file system [C] // Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, 2003.

[10] Patterson D, Gibson G, Katz R. A case for

redundant arrays of inexpensive disks (RAID) [C] // Proceedings of ACM SIGMOD Conference, 1988: 109-116.

[11] Dean J. Building large-scale internet services [EB/OL]. 2017[2019-07-26]. http://static.googleusercontent.com/media/research.google.com/en//people/jeff/SOCC2010-keynoteslides.pdf.

[12] Gill P, Jain N, Nagappan N. Understanding network failures in data centers: measurement, analysis, and implications [J]. ACM SIGCOMM Computer Communication Review, 2011, 41(4): 350-361.

[13] Keeton K, Santos C, Beyer D, et al. Designing for disasters [C] // The USENIX Conference on File & Storage Technologies, 2004.

[14] Bairavasundaram LN, Goodson GR, Schroeder B, et al. An analysis of data corruption in the storage stack [J]. ACM Transactions on Storage, 2008, 4(3): 1-28.

[15] Sridharan V, DeBardeleben N, Blanchard S, et al. Memory errors in modern systems: the good, the bad, and the ugly [J]. ACM SIGPLAN Notice, 2015, 50(4): 297-310.

[16] Jaffer S, Maneas S, Hwang A, et al. Evaluating file system reliability on solid state drives [C] // The Nineteenth USENIX Annual Technical Conference, 2019.

[17] Tai A, Kryczka A, Kanaujia SO, et al. Who's afraid of uncorrectable bit errors? Online recovery of flash errors with distributed redundancy [C] // The 19th USENIX Annual Technical Conference, 2019.

[18] Panzer-Steindel B. Data integrity [EB/OL]. 2007[2019-07-26]. http://indico.cern.ch/getFile.py/access?contribId=3&sessionId=0&resId=1&materialId=paper&confId=13797.

[19] Prabhakaran V, Bairavasundaram LN, Agrawal N, et al. IRON file systems [J]. ACM SIGOPS Operating Systems Review, 2005, 39(5): 206-220.

[20] Hwang AA, Stefanovici IA, Schroeder B. Cosmic rays don't strike twice: understanding the nature of dram errors and the implications for system design [C] // The Seventeenth International Conference on Architectural Support for Programming Languages & Operating Systems, 2012.

[21] Kim Bryan S, Choi J, Min SL. Design tradeoffs for SSD reliability [C] // The 19th USENIX Conference on File and Storage Technologies, 2019.

[22] Xu E, Zheng M, Qin F, et al. Lessons and actions what we learned from 10K SSD-related storage system failures [C] // The 19th USENIX Annual Technical Conference, 2019.

[23] May TC, Woods MH. Alpha-particle-induced soft errors in dynamic memories [J]. IEEE Transactions on Electron Devices, 1979, 26(1): 2-9.

[24] Ziegler JF, Lanford WA. Effect of cosmic rays on computer memories [J]. Science, 1979, 206(4420): 776-788.

[25] Swift MM, Bershad BN, Levy HM. Improving the reliability of commodity operating systems [J]. ACM Transactions on Computer Systems, 2003, 37(5): 207-215.

[26] Chou A, Yang J, Chelf B, et al. An empirical study of operating system errors [C] // Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, 2001: 73-88.

[27] Engler D, Chen DY, Hallem S, et al. Bugs as deviant behavior: a general approach to inferring errors in systems code [C] // Proceedings of the eighteenth ACM Symposium on Operating Systems Principles, 2001: 57-72.

[28] Yang J, Sar C, Engler DR. EXPLODE: a lightweight, general system for finding serious storage system errors [C] // The 7th Symposium on Operating Systems Design and Implementation, 2006.

[29] Yang J, Twohey P, Engler D, et al. Using model checking to find serious file system errors [J]. ACM Transactions on Computer Systems, 2006, 24(4): 393-423.

[30] Krioukov A, Bairavasundaram LN, Goodson GR, et al. Parity lost and parity regained [C] // The 6th USENIX Conference on File and Storage Technologies, 2008: 127-141.

[31] Bairavasundaram LN, Goodson GR, Pasupathy S, et al. An analysis of latent sector errors in disk drives [C] // Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and

Modeling of Computer Systems, 2007.

[32] Satran J, Meth K, Sapuntzakis C, et al. Internet small computer systems interface [EB/OL]. 2004[2019-07-26]. http://www.ietf.org/rfc/rfc3720.txt.

[33] Postel J. Transmission control protocol [EB/OL]. 1981[2019-07-26]. http://www.ietf.org/rfc/rfc793.txt.

[34] Saltzer JH, Reed DP, Clark DD. End-to-end arguments in system design [J]. ACM Transactions on Computer Systems, 1984, 2(4): 277-288.

[35] Bonwick J, Moore B. The last word in file systems [EB/OL]. 2018[2019-07-26]. http://opensolaris.org/os/community/zfs/docs/zfs last.pdf.

[36] Weil SA, Brandt SA, Miller EL, et al. Ceph: a scalable, high-performance distributed file system [C] // The 7th Symposium on Operating Systems Design and Implementation, 2006.

[37] Red Hat. Glusterfs [EB/OL]. 2019[2019-07-26]. https://redhatstorage.redhat.com/products/glusterfs/.

[38] Ganesan A, Alagappan R, Arpaci-Dusseau AC, et al. Redundancy does not imply fault tolerance: analysis of distributed storage reactions to single errors and corruptions [J]. ACM Transactions on Storage, 2017, 13(3): 1-33.

[39] Lybbn. 深信服超融合 HCI 桌面虚拟化 VMP 虚拟存储 aSAN 技术介绍 [EB/OL]. 2017-04-21[2019-07-26]. http://www.lybbn.cn/data/datas.php?yw=167.

[40] 大道云行. GlusterFS 系统快速管理 [EB/OL]. 2016-03-27[2019-07-26]. http://www.taocloudx.com/index.php?a=shows&catid=4&id=17.

[41] Anderson D, Dykes J, Riedel E. More than an interface-SCSI vs. ATA [C] // Usenix Conference on File and Storage Technologies, 2003.

[42] The Data Clinic. Hard disk failure [EB/OL]. 2019[2019-07-26]. http://www.dataclinic.co.uk/hard-disk-failures.htm.

[43] Weinberg G. The solaris dynamic file system [EB/OL]. 2017[2019-07-26]. http://members.visi.net/thedave/sun/DynFS.pdf.

[44] Sundaram R. The private lives of disk drives [EB/OL]. 2019[2019-07-26]. http://partners.netapp.com/go/techontap/matl/sample/0206tot_resiliency.html.

[45] Wehman J, Den Haan P. The enhanced IDE/fast-ATA FAQ [EB/OL]. 2018[2019-07-26]. http://thef-nym.sci.kun.nl/cgi-pieterh/atazip/atafq.html.

[46] Stack Overflow. Horrific collisions of adler32 hash [EB/OL]. 2019[2019-07-26]. http://stackoverflow.com/questions/13455067/horrific-collisions-ofadler32-hash.

[47] Sheinwald D, Satran J, Thaler P, et al. Internet protocol small computer system interface cyclic redundancy check/checksum considerations [EB/OL]. 2002[2019-07-26]. http://www.ietf.org/rfc/rfc3385.txt.

[48] GitHub. XXhash [EB/OL]. 2019[2019-07-26]. https://github.com/Cyan4973/xxHash.

[49] Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system [C] // International Conference on Massive Storage Systems and Technology, 2010.

[50] Huawei Technologies Co., Ltd. FusionStorage intelligent distributed storage [EB/OL]. 2019[2019-07-26]. https://e.huawei.com/en/products/cloud-computing-dc/storage/cloud-storage/fusionstorage.

[51] VMware, Inc. vSAN [EB/OL]. 2019[2019-07-26]. https://www.vmware.com/products/vsan.html.

[52] Poitras S. Nutanix [EB/OL]. 2019[2019-07-26]. https://nutanixbible.com/.

[53] T10 Technical Committee. SCSI block commands-3 [EB/OL]. 2019[2019-07-26]. http://www.t10.org/members/w sbc3.htm.

[54] Huawei Technologies Co., Ltd. OceanStor dorada V3 all-flash storage [EB/OL]. 2019[2019-07-26]. https://e.huawei.com/en/products/cloud-computing-dc/storage/unified-storage/dorado-v3.

[55] Juels A, Kaliski B. Pors: proofs of retrievability for large files [C] // ACM Conference on Computer and Communications Security, 2007: 584-597.

[56] Shen J, Shen J, Chen X, et al. An efficient public auditing protocol with novel dynamic structure for cloud data [J]. IEEE Transactions on Information Forensics and Security, 2017, 12(10): 2402-2415.