面向点对点的安全可靠存储系统*

陈 明+, 杨广文, 刘学铮, 史树明, 王鼎兴

(清华大学 计算机科学与技术系,北京 100084)

P2P-Oriented Secure Reliable Storage System

CHEN Ming+, YANG Guang-Wen, LIU Xue-Zheng, SHI Shu-Ming, WANG Ding-Xing

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

+ Corresponding author: Phn: +86-10-62799645, E-mail: cm01@mails.tsinghua.edu.cn, http://www.tsingha.edu.cn

Received 2004-02-10; Accepted 2004-12-08

Chen M, Yang GW, Liu XZ, Shi SM, Wang DX. P2P-Oriented secure reliable storage system. *Journal of Software*, 2005,16(10):1790−1797. DOI: 10.1360/jos161790

Abstract: This paper proposes a cooperative secure reliable storage system depending on pure P2P and supporting self-repair. Participant nodes of this system form an overlay by Paramecium protocol, which maintains the organization of system and provides routing service, or other compatible distributed hash table (DHT) protocols. The PKI authentication mechanism is introduced to ensure security in an open environment. Binding user data with replica identifiers supports secure auto-repair. The introduction of replica types provides secure sharing-write. Preliminary analysis and experiments demonstrate that it maintains a high reliability and read/write performance in real settings while keeping the cost of maintenance bandwidth low.

Key words: P2P (peer-to-peer); high salability; secure reliable storage system

摘 要: 利用 P2P 的方法实现了一个共享和合作的安全存储系统,其中参与节点运行 Paramecium 协议或其他兼容的 DHT(distributed hash table)协议形成自组织覆盖层,维护系统的组织结构和提供路由服务.由于该系统为开放式结构,引入了基于 PKI 的安全认证机制以确保用户数据的授权访问.用户数据和副本标示的绑定支持了安全的数据自修复;副本类型的引入提供了安全的共享写.初步的分析和实验表明,该 P2P 系统在现实条件下,在消耗较低的维护带宽的同时维持了较高的可靠性并提供了较好的读写性能.

关键词: 对等网络;高伸缩性;安全可靠存储系统

中图法分类号: TP393 文献标识码: A

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2002AA104580 (国家高技术研究发展计划(863)); the National Natural Science Foundation of China under Grant Nos.90412006, 60273007, 60373004, 60373005 (国家自然科学基金)

作者简介: 陈明(1978－),男,广西玉林人,博士生,主要研究领域为对等网络,网格,分布式系统;杨广文(1963－),男,博士,副教授,CCF 高级会员,主要研究领域为并行/高性能计算,网格;刘学铮(1978－),男,博士生,主要研究领域为对等网络,网格,模式识别;史树明(1976－),男,博士,主要研究领域为对等网络,网格;王鼎兴(1937－),男,博士,教授,博士生导师,主要研究领域为并行/分布处理计算机系统.

伴随计算机科学和技术的发展,个人计算机的性能依循莫尔定律(每隔 18 个月翻一番)快速地发展着.存储和网络带宽的发展速度超越了莫尔定律.现代个人计算机的能力已经大大超过了 20 年前的超级计算机.这两个趋势的发展使得构造基于 P2P 的存储系统成为可能.研究表明,大多数时候,个人 PC 的利用率不到 10%.目前,维护存储系统正常和持续性地工作成为存储的主要成本.基于 P2P 的存储系统有助于减少维护的开销.

P2P 强调的是对等服务,不区分服务器和客户端.每个节点在索取其他节点服务的同时,也与其他节点相配合提供相同的服务.它是一种扁平而不是层次化的结构,每个参与节点的地位都相等.

基于以上现状,我们提出了一个基于 P2P 的共享和合作的安全存储系统:CSS(corporative secure storage),尝试利用 P2P 优点解决现存的存储问题.CSS 由连接到 Internet 的节点构成.每个节点向系统贡献一部分存储空间,相互协作形成一个全局高可靠的完全平面的分布式存储系统.

本文第 1 节描述 CSS 的系统结构.第 2 节分析系统的可靠性和维护开销,实验讨论系统的性能.第 3 节讨论相关的工作.第 4 节总结全文.

1 系统架构

CSS 由参与到这个系统的并且运行 CSS 协议的联网节点构成.每个参与节点都向系统贡献一部分本地磁盘空间.在 CSS 协议的作用下,贡献出来的磁盘空间构成一个全局统一的存储系统.在 CSS 协议的上层看来,CSS 提供了类似本地文件系统的存储和目录服务.但在实现中,所有的持续性存储都由 CSS 协议保存到这个全局统一的存储系统中.从这一点上看,CSS 类似传统的分布式存储系统或文件系统.CSS 区别于这些传统系统之处在于:(1) CSS 中没有中央服务器,所有的参与节点都是平等的;(2) CSS 不需要管理员配置系统.在 CSS 系统中,我们不信赖除了本地节点外的其他任何节点,我们只信赖本地节点和整个系统.考虑到 P2P 环境高度的动态性,本系统不企图构造一个文件系统,而是试图设计一个与实际运行环境相适应的存储系统和弱目录系统.

我们首先简单介绍 CSS 的基础 Paramecium 覆盖层.需要指出的是,我们的系统也可以建立在其他 DHT(distributed hash table)之上,例如 CAN[1],Chord[2],Pastry[3]和 Tapastry[4]等,我们认为,建立在 Paramecium[5]会具有更好的稳定性和路由性能.Paramecium 简介之后是 CSS 的详细设计.

1.1 **Paramecium**:完全自组织、无中央服务器的**P2P**的自组织和路由覆盖层

Paramecium 是一个高效、可扩展、容错和自组织的 P2P 的覆盖层.Paramecium 的每个节点被赋予一个 160 位的数字 ID——NodeId.每个要存储的对象(可以是任何数据:文件、数字等)也有一个 160 位的数字 ID——Key.NodeId 空间和 Key 空间是同一个数字空间,并且都首尾相接(0 和 $2^{160}-1$)形成一个环.Paramecium 提出细胞的概念.每个细胞由二元组[左边界,右边界]来标识,称为细胞的疆界.例如:假定 ID 为 6 位二进制长,[111000,111100]和[110000,001000]都是细胞标识的例子,其中后者是一个首尾结合的例子.细胞相互不包含和不重叠;细胞疆界的并集覆盖整个 NodeId 空间.给定一个 Key,Paramecium 可以高效、分布式地找到这样一个节点:这个节点所在的细胞包含这个 Key,并且这个节点的 NodeId 和这个 Key 最接近.

Paramecium 向上层提供以下 API:

Join(**BootNodeIP**):通过 BootNode 将本节点带入 Paramecium 系统.

LookupNode(**ID**):在系统中查找一个节点,这个节点所在的细胞包含这个给定的 ID,并且这个节点的 ID 和这个给定的 ID 的绝对值之差在这个细胞里的所有的节点中是最小的.

Insert(**Key**,**Object**):将具有 Key 的 Object 存入系统中,保存这个对象的驻留节点是 LookupNode(Key)返回的结果.

LookupObject(**Key**):在系统中查找并返回与 Key 匹配的对象.

假定系统中有 N 个节点,每个细胞中包含 S 个节点,那么在正常情况下,整个路由的复杂度平均为 $O(1)+O(\log N/S)=O(\log N/S)$.

Paramecium 的一个关键设计就是在面临节点加入和离开的情况下维护系统的结构不变、路由性能不变.

1.2 CSS系统

回顾上一节我们知道,Paramecium 向上层提供了在全局中插入对象和查找对象这两个基本操作.其中,它的对象指的是可序列化的任意结构.在存储部分,我们把对象替换为“文件”.因此,每个文件都由一个 ID 作为标识符.文件是基本的存储单位.

1.2.1 CSS 安全协议

我们假定系统中的每个用户都有一个证书和对应的私钥.这个证书不必是受信赖的机构(CA)所颁发的.系统中的每个文件(本文中的“文件”泛指通常意义下的“文件”和“目录”)都有一个所有者.所有者使用私钥对文件进行签名,并把它附在文件后来表示对文件的所有权.所有者在取回文件时也可以用附在文件后的签名来验证其完整性.

在签署用户证书的时候,系统使用安全的随机数发生器生成 r 个随机数,将其作为证书的附加部分一起使用标准的 PKI[6]协议进行签署.这种证书和普通的证书完全兼容.系统在计算根证书的散列时,依次把这 r 个随机数附加到证书之后再计算散列值,由此可得到对应的 r 个散列值.下面提到的证书的散列值都是指这 r 个散列值的集合.

每个文件由两部分构成:用户数据和元数据.用户数据可以是明文也可以是密文.用户数据的类型不影响系统的运作.元数据部分包括文件所有者的证书、这个副本的随机数和对应的签名、这份用户数据的所有副本的随机数和对应签名的二元组的集合.元数据中包括所有者证书有助于文件的驻留节点验证文件的所有者,从而决定是否要覆盖原官方文件.同时,由于证书的内容很少,也不会造成过多的空间浪费.当用户向系统中插入一个文件的时候,系统也自动产生 r(系统要求的副本的个数)个随机数.系统依次将这 r 个随机数附加到用户数据和用户证书的后面,然后使用用户的私钥对此进行签名,从而等到对应的 r 个签名.随机数对应的签名构成二元组,所有二元组进而形成二元组向量.所有副本的二元组向量都相同.每个副本依次选取二元组向量中的一个元素作为该副本的随机数和签名二元组.对包括用户数据和元数据的整个副本计算其安全散列的值得到这个副本的 ID,称为副本 ID.一个文件所有副本 ID 的集合称为文件 IDs.同一个文件的所有副本相互间称为兄弟副本(如图 1 所示).

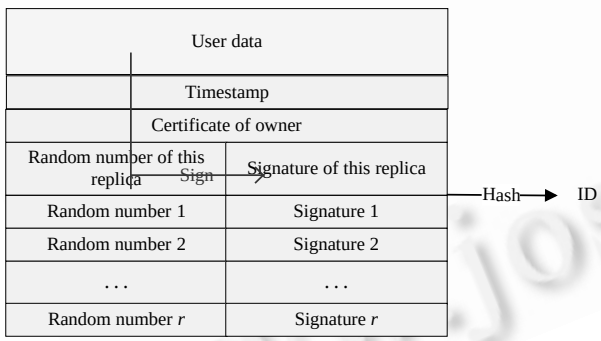

Fig.1 Composition of file and the generation of file ID

图 1 文件的构成和文件 ID 的生成

驻留节点在响应保存一个副本的请求时要求被保存的副本必须满足以下两个条件中的任意一个:(1) 副本的所有者证书的散列值中的任意一个和这个副本的 ID 相同;如果有同样 ID 的副本存在,新副本的时间戳必须晚于旧副本的时间戳.(2) 副本的 ID 是整个副本内容的散列值;如果有同样 ID 的副本存在,新的副本的时间戳必须晚于旧的副本的时间戳.对于不符合这两种规则的文件和 ID 对,节点参照以下协议处理.我们定义经过所有者签名被所有者认可的副本称为“官方副本”.CSS 参与节点遵循以下基本协议:只有副本的驻留节点和所有者才有权删除和覆盖(这里的覆盖指的是用一个新的文件代替原副本,当要求返回一个与原副本 ID 相同的副本时返回新的副本而不是原副本)官方副本.覆盖了原官方副本未经原所有者签名的文件称为“非官方副本”.驻留节点也可以不覆盖原官方副本而保存一份“影子副本”,进而形成影子副本向量,即依照时间序列排列的影子副本.影子副本与官方副本 ID 相同而内容不同.查找文件时,影子副本和官方副本同时返回给查询节点.这 3 种副本概括在表 1 中.

Erasure 这种线性编码方法提供了另外一种选择.考虑到使用 erasure 带来的网络和计算开销,系统采用副本而不是 erasure 方式来提高系统的可靠性.

Table1 Definitions and comparison of replica type

表 1 副本类型和比较

Replica type	Characteristics	Is its ID same with previous official replica?	Will previous official replica be replaced?	Which replicas will be returned upon query?
Official replica	Replica signed by its owner	Yes	Yes	Official replica
Non-Official replica	Replica having same ID with official replica, but without signature of owner of the official replica	Yes	Yes	Non-Official replica
Shadow replica	Replica having same ID with official replica, but without signature of owner of the official replica	Yes	No	Official and shadow replica

1.2.2 存储结构

系统中存在两类基本的文件:数据文件和目录文件.数据文件中存储用户数据,目录文件中存储本目录中的所有普通文件和目录文件的文件名及其对应的 IDs.每个用户都有一个属于他自己的根目录"/".根目录文件的 IDs 是其证书的散列值.由于用户的证书是公开且不变的,这样就提供了一个众所周知的"入口".目录系统的组织如图 2 所示.在这种组织方式下,任何一个文件的更新都会导致从这个文件向上到用户根目录所经过的所有目录的更新.这种方法尽管带来了更新的开销,但是只是安全而不是性能才是这部分的关键考虑因素.考虑到本系统主要面向存储应用,而存储系统一般具有写一次,读多次,很少更新,并且顺序读写的特点,因此性能不会受到太大的影响,而且这种方法还具有以下优点:

- 保证了副本 ID 与内容的关联性:在安全散列的作用下,攻击者无法制造出一份具有同样 ID 但内容不同的副本.
- 在系统中每个文件存在多个副本的情况下,除非所有副本的驻留节点合谋,否则总可以读出一个有效的副本.而在安全散列的作用下,副本的 ID 充分随机均匀分布,恶意攻击者能够控制所有驻留节点的概率很低.
- 保证了副本 ID 的唯一性:随机数导致散列内容的差异,而安全散列保证了高度的无冲突性.副本 ID 的唯一性防止出现文件竞争性覆盖和读出错误内容情形的发生.

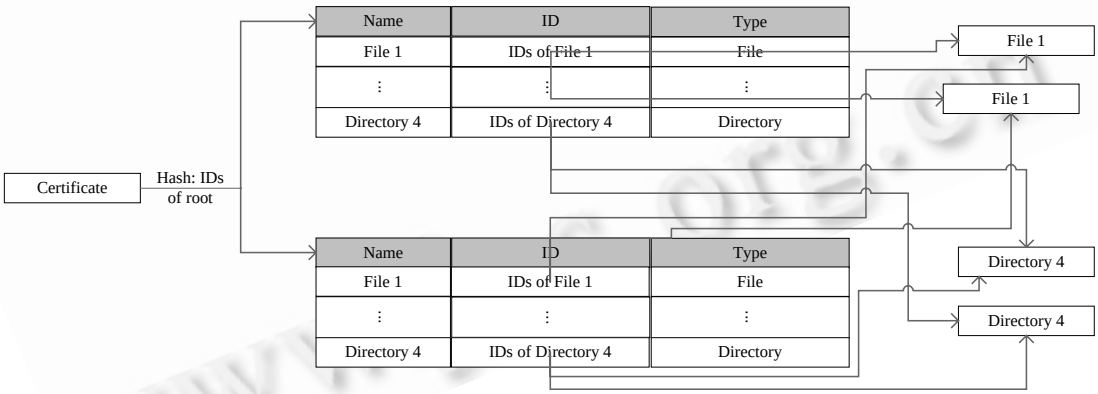

Fig.2 Directory structure of system

图 2 系统的目录结构

为了防止出现伪造节点 ID 类型的攻击,系统可以通过把节点 ID 和用户证书的散列绑定的方法来解决.

对于公开的目录和文件,系统可只签名而不加密,这样,其他节点就可以访问其内容.但如果某层目录被加密了,那么这个目录及其下的所有目录和文件即使不加密,其他节点也因为不知道对应文件的 ID 而难以访问.

1.2.3 周期性修复

参与节点的行为是高度动态并且不可预测的.底层的基本协议并不保证文件的可靠性.随着节点的加入和离开,部分文件可能丢失.为了保证每个文件始终存在一定的副本数目,系统将周期性地恢复丢失的副本.恢复过程很简单:每个节点定期地试图读取保存在本地的副本的兄弟副本,检测出丢失的副本,然后用本地的副本恢

复出丢失的副本,再把丢失的副本保存入系统中.根据前面的保存规则可知,所有的副本都可以由任意的节点来恢复而不存在被恶意覆盖的问题.为了减少网络流量,仅当副本数减少到一半时才进行恢复工作.当然,每个文件的所有者也可定期地进行主动的检测和修复.

1.2.4 共享写

影子文件的引入是为了实现安全的共享写操作.假设系统中有 *A* 和 *B* 两个用户.用户 *A* 有一个目录 Dir1,并且允诺 *B* 可以写目录 Dir1.用户 *B* 对目录 Dir1 进行写操作后使用 Dir1 原来的 IDs 将新的副本保存入系统.由于新的副本的 ID 不可能和原 ID 相同(内容的不同导致安全散列值不同),驻留节点将拒绝使用新的副本去覆盖旧的副本.驻留节点将新的副本保存成"影子副本".用户 *A* 检索目录 Dir1,取回包括官方副本和影子副本在内的所有副本.用户 *A* 可以用影子副本的签名验证这些副本的确是用户 *B* 生成的.假如用户 *A* 认可用户 *B* 的修改,用户 *A* 将根据影子副本中的内容生成官方文件并保存入系统.新的官方文件和旧的官方文件的 IDs 不同,因此,要从下到上更新父目录的内容.用户 *A* 随后删除旧的官方文件和影子文件.

由于每个用户都有自己一个独立的文件"根",同时一个用户几乎不涉及到频繁并发读写的问题,因此访问冲突的情况并不突出.我们将在进一步的研究中探索这个问题.

1.2.5 空间回收

用户离开系统而不清除文件和删除文件过程中由于部分节点不在线导致这些节点上的文件没有被删除,都会产生存储垃圾.驻留节点在每个文件附加一个最后访问时间属性的记录域.每个最后访问期限超过一定范围的文件,驻留节点有权删除.因此,随着时间的流逝,未被访问的文件逐渐消失,保护了系统资源.更为复杂的删除策略,例如支持用户策略的 Elephant file system[7],也可加入系统当中.我们计划在将来进一步研究这个问题.

2 初步的分析和实验

2.1 修复算法的可行性

我们先分析修复算法的可行性,包括带宽消耗和文件的可靠性.我们假定副本的死亡分布服从负指数分布,即: $P_{death}(t)=1-\mathrm{e}^{-\lambda t}$,其中 $1/\lambda$ 是副本的寿命期望.下面是推导过程中用到的其他符号的定义:

Bandwidth:联网节点的带宽.*N*:系统的节点数目.*F/N*:每个节点保存在系统中文件的平均数目.*FileSize*:系统中文件的平均大小.*Uptime*:每个节点每天的平均在线的时间.*T*:系统的修复周期.*R*:一个文件的副本数.

假如没有修复,一个文件经过 T 时间后它存活的可能性为 $P_{living}(1)=1-P_{death}^{R}(T)$.文件的修复是需要时间的,修复文件所需时间的上界是一个节点修复所有丢失的副本所需的时间: $T_{2R}=\dfrac{FileSize\times R}{Bandwidth}$.一个节点进行修复时可能由于下线等原因导致修复失败,可以假定修复在上线期间是均匀进行的,则修复失败的上限为 $\dfrac{T_{2r}}{Uptime}$.如果修复时文件还存活,从这一时刻往前看:那么如果修复成功,由于死亡分布的无记忆性,文件将以概率 1 存活下去,否则,文件以 $P_{living}(T)$ 的概率继续存活.由此可以得到文件存活的递推公式:

$$P_{living}(n)=\left(1.0\times\left(1-\frac{T_{2r}}{Uptime}\right)+P_{living}(1)\times\frac{T_{2r}}{Uptime}\right)\times P_{living}(n-1).$$

文件的可靠性为

$$P(t)=\left(1-p_{death}^{R}\left(t-T\times\left\lfloor\frac{t}{T}\right\rfloor\right)\right)\times P_{living}\left(\left\lfloor\frac{t}{T}\right\rfloor\right) \tag{1}$$

由于推导过程都是保守近似,因此给出的是可靠性的下界.

由可靠性图(如图 3 所示)可见,文件的可靠性下降很缓慢,如果文件的所有者节点主动修复,则文件的可靠性在相当长的时间内将几乎维持不变.我们假定文件的可靠性维持在一个较低的几乎稳定的下界,那么经过 T 时间后,一个文件存活的副本数为

$$Living(t)=\lfloor r\times(1-P_{death}(t))\rfloor.$$

丢失一半副本的概率为

$$P_{lost}(t)=\sum_{i=lLiving(t)-R/2+1}^{lLiving(t)} C_{Living(t)}^{i}(P_{death}(t))^{i}\times(1-P_{death}(t))^{Living(t)-i}:\quad \text{if}\quad Living(t)+1>R/2,$$

$$P_{lost}(t)=1:\quad \text{if}\quad Living(t)+1\le R/2.$$

恢复丢失的副本消耗的带宽为

$$Bandwidth_{repair}=(R-Living(t))\times P_{lost}(t)\times FileSize\times F/N/T\ .$$

检测副本丢失所需的带宽为

$$Bandwidth_{check}=\frac{R\times F/N\times 20}{T}\quad \text{(每个副本的 ID 是 160bits/8=20Bytes).}$$

总共消耗的带宽为

$$Bandwidth_{requirement}=Bandwidth_{repair}+Bandwidth_{check}\tag{2}$$

综合两幅图(图 3 和图 4)可见,即使每个节点向系统存入 10 000 个文件,每个文件平均 4M(每个节点平均有 40G 数据),在副本存活期望为 40 天的情况下,以 10 天为周期修复,可靠性和消耗的带宽是完全可以接受的.

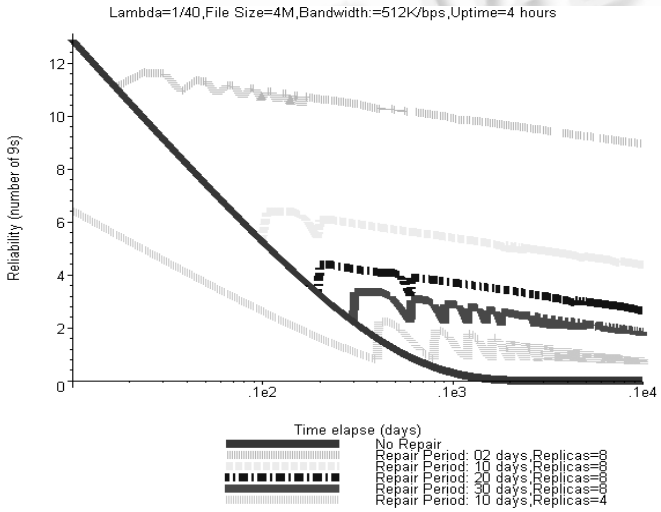

Fig.3 Reliability of file (Formula (1))

图 3 文件的可靠性(式(1))

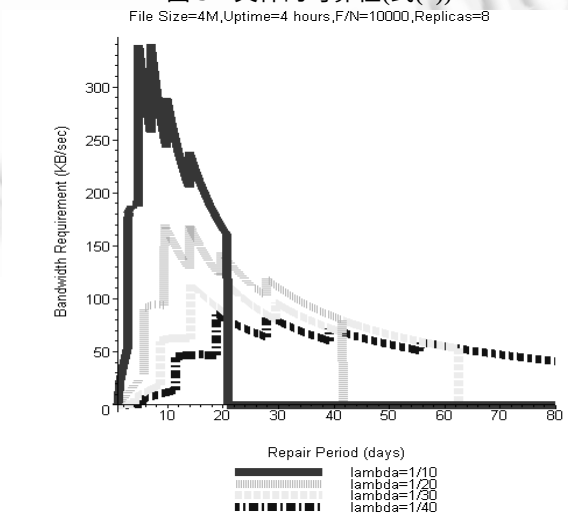

Fig.4 Repair cost (Formula (2))

图 4 修复开销(式(2))

2.2 初步的实验结果

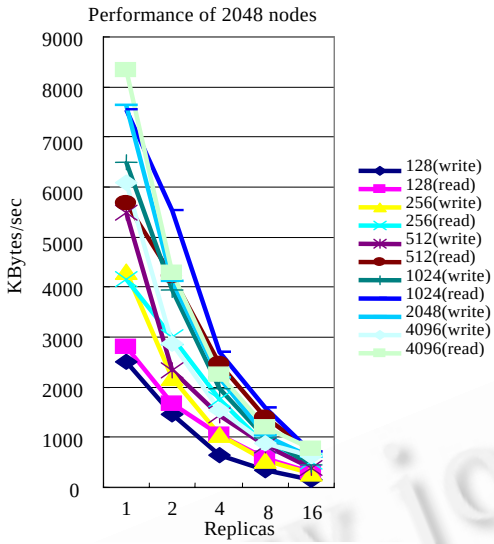

Fig.5 Read/write performance
图 5 系统的文件读/写性能

我们使用 Java1.4.2 平台初步实现了一个原形系统.由于受 Java 性能的限制,这里的结果应该看做是可行性而不是性能的证明.考虑到 Java 执行签名和验证的速度较慢,在签名和验证部分我们采用 openSSL 本地库.测试用的是 16 台使用 100M 以太网连接,安装了 Windows 2003 企业版操作系统,配备 512M 内存的 P4 1.4G 的机器.签名和验证使用了 512 位 RSA 算法.测试中所有的读写都是串行的,即先查找到驻留节点,串行读或写完毕后再进行下一步操作.为了避免 Java 的 JIT 带来的热身问题,所有的实验都先读写 3 次进行预热,经过 1 分钟后再进行测试.测试结果取 10 次测试的平均值.读写文件时都是随机选择一个节点作为发起节点.图 5 中的曲线分别表示了不同大小的文件在不同副本数目下的读写速度.从图 5 中可以看到,在文件同样大小的情况下,速度几乎和副本数成反比关系.这基本符合预期的结果.在文件大小为 128K,8 个副本的条件下,系统仍可达到约 200KB/s 的速度.

3 相关工作

FarSite[8]是一个面向 LAN 应用的分布式传统的文件系统,提供了一些文件系统的语义.FarSite 中的每个节点维护系统中一部分存活节点的信息,通过一个分布式目录服务来实现查找操作.在安全方面,它需要管理员配置一些“受信任”的节点.目前没有关于它的可扩展性信息.CFS[9]构建在 Chord 之上,是一个只读分块的存储共享系统.由于只有复制而没有周期性的修复,它只提供存储对象的弱持续性.Ivy[10]也建立在 Chord 的基础之上,是一个基于 log 结构的可读可写的文件系统.Ivy 提供了一些弱的文件语义和弱的持续性.Ivy 没有解决基于 log 带来的空间回收问题.OceanStore[11]的目标是在系统高度变化和不安全的前提下,构造一个完全分布式、安全、高可靠、高可用、负载平衡和提供非平凡服务的洋量存储系统.为了达到这些目的,OceanStore 最后实现的时候使用了根据节点能力分层的结构,引入了“受信任的团体”和“更新服务器环”,带来了管理和性能的问题,偏离了 P2P 完全平等的基本思想.Past[12]基于 Pastry.它很多方面与 OceanStore 类似,但更强调的是一个简洁和存储空间高利用率的存储系统.它和前面几个系统的一个区别是:Past 将整个文件不分块地保存,而前面几个系统都将文件分块后保存到系统中.在安全方面,Past 主要关注安全路由.Pastiche[13]是一个基于 P2P 的备份系统.它把参与节点上空闲的存储资源组织成一个共享的存储池,重点关注和备份相关的工作,包括减少相同副本的备份以及数据的安全与完整.CSS 强调在简单的同时引入足够的安全机制,通过周期性非信任修复提供高可靠性.与此同时,CSS 充分考虑到 P2P 环境的动态性和不稳定性,关注在现实的环境下构造与环境和需求相适应、不过分牺牲性能的安全存储.CSS 也是目前我们所知的支持任意节点发起安全非主动性修复的 P2P 存储系统.

4 总结及未来的工作

个人计算机性能和联网带宽的快速发展使得基于 P2P 的应用成为需要和可能.文中我们提出并初步实现了 CSS——一个具有高可扩展性和高可靠性的共享和合作的安全存储系统.

CSS 主要面向参与节点数在 10 000 及以上的中大规模的应用,具有如下特点:

- 对等性:CSS 在组织结构上为纯粹的 P2P 系统,不存在任何形式的中央服务器,所有参与节点地位相等;
- 易管理性:系统管理的复杂性与参与的节点及用户数无关,并在参与节点行为(加入与离开)流动性高且

不确定的情况下正常工作,无须人工干预;

- 可扩展性:Paramecium 组织和路由算法以及存储系统的非集中式管理和验证算法使其具有良好的可扩展性;
- 安全性:提供一定级别的安全机制,防止未授权的阅读和修改,并且对上层应用透明;
- 高可靠性:系统的自修复功能保证了高可靠性.

我们预备使用 Java 构造一个可以实际部署和模拟运行的系统,使之实现互通和互操作,并研究系统在实际环境的运行特征.

References:

[1] Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable content-addressable network. In: Proc. of the 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications. New York: ACM Press, 2001. 161−172.

[2] Stoica I, Morris R, Karger D, Kaashoek F, Balakrishnan H. Chord: A scalable P2P lookup service for Internet applications. IEEE/ACM Trans. on Networking (TON), 2003,11(1):17−32.

[3] Rowstron DP. Pastry: Scalable, distributed object location and routing for large-scale P2P system. In: Proc. of the 18th IFIP/ACM Int'l Conf. on Distributed Systems Platforms (Middleware 2001). London: Springer-Verlag, 2001. 329−350.

[4] Zhao BY, Kubiatowicz J, Joseph AD. Tapestry: An infrastructure for fault-toleratnt wide-area location and routing. Technical Report, UCB/CSD-01-1141, EECS of Berkeley, 2001.

[5] Chen M, Yang GW, Wu YW, Liu XZ. Paramecium: Assembling raw nodes into composite cells. In: Proc. of the IFIP Conf. on Network and Parallel Computing. London: Springer-Verlag, 2004. 481−484.

[6] PKI charter. 2003. http://www.ietf.org/html.charters/pkix-charter.html

[7] Santry D, Feeley M, Hutchinson N, Veitch A, Carton R, Ofir J. Deciding when to forget in the Elephant file system. In: Proc. of the 17th ACM Symp. on Operating System Principles. New York: ACM Press, 1999. 110−123.

[8] Adya A, Bolosky WJ, Castro M, Chaiken R, Cermak G, Douceur JR, Howell J, Lorch JR, Theimer M, Wattenhofer R. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In: Proc. of the 5th Symp. on Operating Systems Design and Implementation. New York: ACM Press, 2002. 1−12.

[9] Dabek F, Kaashoek MF, Karger D, Mor-Ris R, Stoica I. Wide-Area cooperative storage with CFS. In: Proc. of the 18th ACM Symp. on Operating System Principles. New York: ACM Press, 2001. 202−215.

[10] Muthitacharoen A, Morris R, Gil T, Chen IB. Ivy: A read/write P2P file system. In: Proc. of the 5th Symp. on Operating System Principles. New York: ACM Press, 2002. 31−44.

[11] Kubiatowicz J, Bindel D, Chen Y, Czerwinski S, Eaton P, Geels D, Gummadi R, Rhea S, Weath-Erspoon H, Weimer W, Wells C, Zhao B. Oceanstore: An architecture for global-scale persistent storage. ACM SIGPLAN Notices, 2000,35(11):190−201.

[12] Rowstron A, Druschel P. PAST: A large-scale, persistent P2P storage utility. In: Proc. of the 8th Workshop on Hot Topics in Operating Systems. New York: ACM Press, 2001. 75.

[13] Cox LP, Murray CD, Noble BD. Pastiche: Making backup cheap and easy. In: Proc. of the 5th Symp. on Operating Systems Design and Implementation. New York: ACM Press, 2002. 285−298.