基于云存储的分布式文件系统的同步与加密调研报告

指 导 教 师：邢 凯

小 组 成 员：朱一铭 PB15030773

王若晖 PB15000142

钟 立 PB15000037

韦 清 PB15000027

目　　录

项目背景及实践意义

1.1 背景

1.1.1 分布式存储

进入 21 世纪后，世界上的总信息量仍在经历爆发性的增长。据估计，2005 年，世界上总共有约 130EB 的信息，而到了 2010 年，这个数字增加到了 1,227EB，增长了约 10 倍【1】，这也证明大数据时代的到来。然而比起数据的增长，存储媒介的进步虽然很快，但难以直接满足快速增长的信息，以及其带来的各种需求，如当一个存储出现错误或遭到外来入侵的时候，怎样做能够使损失降到最低。在这种情况下，分布式存储应运而生，将要存储信息分散在不同的位置，通过网络连接互相传输，使整个存储系统的健壮性和安全性能够得到保证。
1984 年，Sun 公司首先开发了网络文件系统 NFS【2】。

1.1.2 MapReduce

MapReduce 是 Google 提出的一个软件架构，用于大规模数据集的并行运算。

如名字所提示的，MapReduce 主要分为 Map 和 Reduce 两部分，指定一个 Map 函数，用来把一组键值对映射成一组新的键值对，指定并发的 Reduce 函数，用来合并所有键值相等的元素，通过这种方式使得并行处理变得透明。

现在如 Apache 的 Hadoop 分布式文件系统(HDFS)，Google 的 Google 文件系统(GFS)都曾采用 MapReduce。

MapReduce 通过把对数据集的大规模操作分发给网络上的每个节点实现可靠性；每个节点会周期性的把完成的工作和状态的更新报告回来。如果一个节点保持沉默超过一个预设的时间间隔，主节点（类同 Google 档案系统中的主服务器）记录下这个节点状态为死亡，并把分配给这个节点的数据发到别的节点。每个操作使用命名文件的不可分割操作以确保不会发生并行线程间的冲突；当文件被改名的时候，系统可能会把他们复制到任务名以外的另一个名字上去。（避免副作用）【4】。

1.1.3 加密存储

毋庸置疑，随着网上数据量的增加，信息的保密与安全的问题也受到日益的重视。

中心化的加密存储面临一个困境，即整个存储系统的安全与否都取决于中心服务器的安全。一旦中心服务器遭到攻击，所有的数据都面临泄露或被篡改的危险。而如果中心服务器出现错误或者宕机，所有数据都将变得无法访问。

因此，采取与分布式存储相近的哲学，可以使用一种去中心化的加密存储，来保证在系统的一部分失败时仍能提供可靠安全的服务。

中本聪提出并实现的 BitCoin，采用点对点网络的区块链【3】，从算法上保证了比特币的安全和交易的可信性。

1.1.4 区块链

区块链（blockchain）是用分布式数据库识别、传播和记载信息的智能化对等网络，起源自比特币。区块链是一串使用密码学方法相关联产生的数据块，在比特币系统中，每一个数据块中包含了若干次比特币网络交易的信息，用于验证其信息的有效性（防伪）和生成下一个区块。

区块链在除去 Storj（发音 Storage）是采用区块链存储元数据的点对点存储，P2P 能够在增大访问带宽的同时，降低集中管理的人事费用等，从而使存储价格降至同类网盘的几分之一【5】。

1.2 实践意义

云储存已经成为互联网时代不可缺少的一项应用，国内外不少云储存服务提供商均提供了相当大的免费文件存储空间，如百度云提供 2000G、OneDrive 提供 7G、Google Drive 提供 15G 等。

现有的多种云存储的解决方案均有较好的用户体验，但是这些解决方案均存在着一些缺点：

（1）由于云服务商单一，容量有限，服务器连接较少，连接稳定性受限于用户网络和服务器所在网络；

（2）单个云储存空间对文件安全性有不利影响，服务器一旦宕机，数据将无法保全；

（3）根据 Google Drive 等多个云存储服务提供商的用户协议，云存储提供商拥有用户文件的同时也会获得对文件内容的使用和展示的权利，因此把用户文件完整地不加密地存储在单一的云存储空间上将带来文件保密的问题。

基于此，提出这样一种存储方案：

（1） 文件被拆分成多个碎片并加密，乱序储存在不同的网盘，并且存在冗余；

（2） 上传时根据网络情况并行地传输到多个云存储空间；

（3） 提供实现文件同步操作，一键化上传、下载；

（4） 提供前段用户界面以方便使用并保证跨平台兼容性。

立项依据

2.1 云存储的定义

云存储是在云计算概念上延伸和发展出的一个新概念，她可以将网络中大量各种不同类别的存储设备通过应用软件集合起来协同工作，共同对外提供数据存储和业务访问功能的一个系统。【1】

云存储是云计算的存储部分，本质上是为大数据量运算时提供存储和管理。关于云存储，全球网络存储工业协会给出的定义是：通过网络提供可配置的虚拟化的存储及相关数据的服务。当云计算系统运算和处理的核心是大量数据的存储和管理时，云计算系统中就需要配置

大量的存储设备，那么云计算系统就转变成为一个云存储系统，所以云存储是一个以数据存储和管理为核心的云计算系统。也就是说，云存储实际就是将数据资源放到网络上，随时供用户存取的一种新型存储模式。近年来，有多家商业软件公司推出了自己的云存储产品和服务，包括 Microsoft 的 Azure、Google 的 GFS、Amazon 的 S3 等等。

随着便携式移动设备的推广，生活的智能化，一个人拥有多台计算设备的情况越来越普遍，这使得用户产生、使用、修改的数据需要在多台计算设备上访问并及时同步，同时由于移动设备容易丢失，数据需要被及时进行备份。云存储提供了一种高效的解决方案。而且目前看来也是唯一运用较广的方案。

云存储还具有其他的一些优点。首先，云存储的管理非常方便，用户不需要去过多考虑云存储的容量，数据的安全，数据的时效性等等。这使得云存储拥有很好的用户体验。同时，云存储将存储模型高度抽象化，展现给用户的可以说只有数据，用户可以自由得使用数据而不用考虑数据的存放介质等其他因素，这为基于云存储的应用软件的开发提供了极大的便利。

2.2 提供跨平台的云存储服务的必要性

当今时代，越来越多的人同时拥有笔记本电脑、智能手机和平板电脑等电子设备。以前，我们采取数据线、蓝牙等设备实现数据传输，但是这些方法都受到了空间和时间的局限性，我们没有办法确保正在使用的某台设备上正好保存着自己需要查看的文件。另外，如果我们正在参与一项需要与多人协作的任务，那么使用 email 附件来交换不断修改中的文件也是一件非常令人头疼的事情。

跨平台的云存储服务是指云存储系统支持 Web、传统 PC 端、手机客户端等多个平台，进行的跨平台、跨终端的文件共享，用户上传的文件都会保存在云端。在访问文件时，无论登录哪个平台都可以访问到所有平台存储的文件，再进行浏览、下载、共享、编辑等操作。将文件保存到“云端”的意义就是为了“让文件跟着我们走”，不管人在何处，不管是使用什么样的终端设备，只要接入互联网，就可以随时获取所需的文件，并对文件进行编辑和处理。

如今，已经有很多公司推出了在线存储的云存储服务。在线存储的文件可以通过不同的设备获取，也可以方便地进行共享，甚至可以直接对文档进行协同编辑。对于苹果而言，目前的 iCloud 可以自动将 iPhone、iPad 以及 Mac 电脑上的照片、账户、文稿和设置备份到服务器上，在多台设备上同步邮件、通讯录、提醒事项等数据。同时它还可以将用户购买的音乐、应用和电子书等内容推送到 iOS 设备上。

而未来，跨平台云存储服务提供商 Dropbox 计划突破简单的文件存储行业，向程序数据同步领域扩张。例如，通过与开发者的合作，该公司可以将《愤怒的小鸟》等游戏的进度在不同平台之间同步，从而实现无缝衔接。这将为我们的生活提供更大的便利，前景也是十分诱人的。

2.3 项目实施需要解决的主要问题

a)修改原有文件系统的输入输出接口，使其与网盘进行数据交互
b)实现文件的分割与加密，上传至多个网盘账户，利用冗余保证数据丢失可能性最低
c)将切割后的文件整合，为用户提供集中的下载服务
d)提供文件同步服务，实现上传、下载一键化
e)管理云端网盘账户，实现存储空间无限扩容与充分利用

2.4 提供基于云储存的同步服务的必要性与可行性

1. 处于信息爆炸的时代，每个人需要处理、储存的数量都可能是海量的，基于云储存进行同步操作，自动备份数据对于任何一个人都是必要的。再通过适当的处理，使上传、下载变得更加简单易行也是必要的。
2. 网络普及率越来越高，价格也愈发廉价，实时实地进行同步服务的可行性也会越来越高。并且随着物联网技术的快速发展，基于云储存的同步服务的交互也会越来越方便可行。

2.5 提供基于云储存的加密服务的必要性

1. 在互联网上进行文件传输等活动时存在许多不安全因素，而且这种不安全性是互联网存在基础——TCP/IP 协议所固有的，包括一些基于 TCP/IP 的服务。所以为了保证安全，我们必须给文件加密。
2. 而在云端进行储存，若以明文储存，很可能泄露个人信息或造成机密文件外流。故而，进行基本的分块、加密后，可以保证文件的安全性与个人信息的安全性。
3. 加密在网络上的作用就是防止有用或私有化信息在网络上被拦截和窃取。一个简单的例子就是密码的传输，计算机密码极为重要，许多安全防护体系是基于密码的，密码的泄露在某种意义上来讲意味着其安全体系的全面崩溃。通过网络进行登录时，所键入的密码以明文的形式被传输到服务器，而网络上的窃听是一件极为容易的事情，所以很有可能黑客会窃取得用户的密码，如果用户是 Root 用户或 Administrator 用户，那后果将是极为严重的。【2】

相关调研

3.1 同步

3.1.1 基于分片的文件存储系统

随着互联网对海量数据存储要求的提高，基于单个计算机节点的存储模式的局限越来越明显。互联网上的数据量越来越大，单个文件的大小也在快速增加。一些文件在普通的计算机节点上根本无法存储，即使一些能存储在单个节点上的大文件，系统吞吐量也极其低下。基于分片的文件存储系统将一个大文件分割成很多小文件进行单独存储，这种技术使得一个文件各个部分的数据可以分散在多个计算机节点上。分片技术使得多用户可以并行地访问文件的各个部分，系统吞吐量得到提高，避免了性能瓶颈。

对于如何将文件的各个分片数据映射到存储节点，实现数据分片的均衡分布和系统的负载均衡时，主要存在两种技术：（1）基于文件映射表的分散存储。该方式为每个文件保存一个元数据表，元数据表的内容即为文件各个分片到存储节点的映射关系。（2）基于哈希函数的随机分散方法，该方式不需要保存元数据表，只要知道文件的相关信息即可计算出文件数据的所在位置，该技术完全避免了定位表的存在。据统计，元数据操作占据了文件系统一半的工作负载，特别是元数据存储在一个中心节点上时，基于定位表的寻找方式很容易成为系统性能的瓶颈。使用哈希函数映射的方式使得文件分片在概率上均匀分散开了，但由于这种映射关系是不可改变的，这使得系统在运行过程中可能出现热点问题。拥有多个存储节点的分布式文件系统中，存储节点出现故障的概率大大增加，一旦计算机节点失效就会导致一些数据不可用甚至导致系统不可用。 为了保证系统的可靠性，分布式文件系统使用了副本冗余技术，即将数据或分片冗余存储在多个节点上。一旦节点失效后，文件的各个部分和系统仍然可用。

3.1.2 各网盘提供 API

对个人用户而言，使用网盘提供的 API 进行对网盘的部分操作比较可行。

3.2 冗余

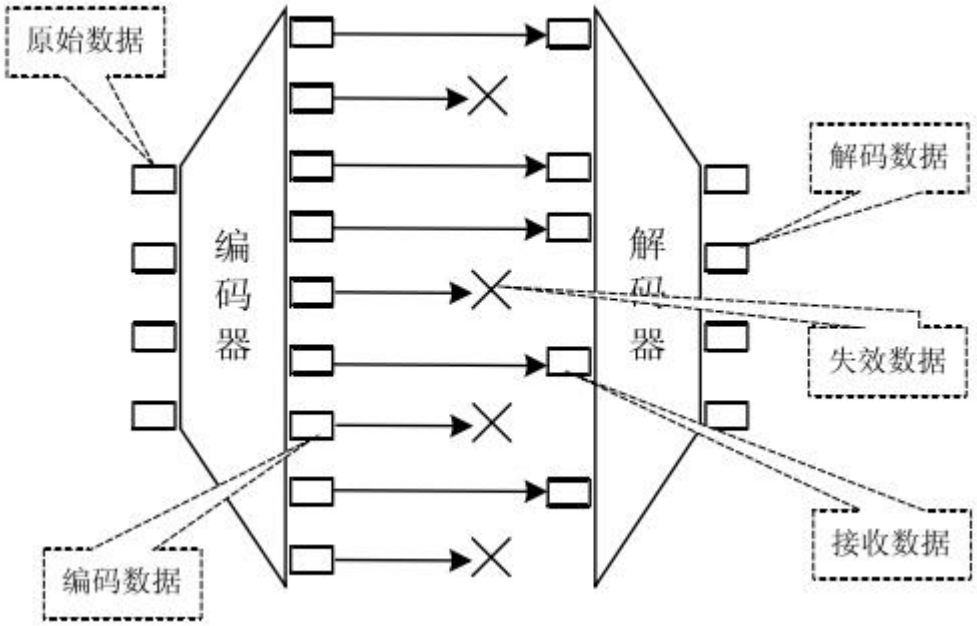
3.2.1 复制(Replication)

复制是最简单的数据冗余策略。原理很简单，通过将文件的多个副本分布到系统不同节点，只要这些节点中一个副本有效，就能获取该文件。文件的副本越多，数据的可用性越好，可靠性越高。由于复制不涉及编码运算，文件创建和读取不需要编解码操作，读写效率高分布式存储包括两级复制方式，即整个文件级（whole-file-level）和文件分块级（fragment-level）。前一种方式很简单，存储同一个文件的 N 个完整副本到 N 个不同节点，则系统能容忍存储

这些副本的 N-1 节点失效，采用完整复制策略的系统有 PAST,Freenet。一种复制方式则是把数据文件分布等长大小的 S 个分块，复制每个分块成 N 个副本，每个节点放置一个分块副本，例如 CFS。

3.2.2 纠删码(Erasure Coding)

纠删码是另一种重要的冗余策略，(n,k)纠删码将一个大小为 M 的原始文件分成 k 块，每块大小为 M/k；然后将这 k 块文件编码成 n 个编码块后分发到 n 个节点中去，其中每个存储节点存储一个编码块，且 n>k。（n,k）纠删码指的是 n 个编码块中的任意 k 个编码块就能重构原文件，其原理如下图所示。



3.3 修复

3.3.1 积极(Eager)修复

对于前者，只要系统检测到某个节点失效，马上启动修复进程，重建数据并保存到新节点。只有当节点失效速度高于节点失效检测与数据修复速度时，该数据才不可用。该策略实现简单，但缺陷在于没有区分节点暂时离开和永久离开（暂时离开无需启动修复进程）。再者，积极修复往往产生大量的重构进程，这可能导致某一时刻网络带宽突然提高。Glacier 和 CFS 系统都使用该策略。

3.3.2 延迟(Lazy)修复

与积极修复相对应，对于可能的节点暂时失效，系统往往使用延迟修复。只有当不可用的数据量达到或超过某一阈值时，系统才启动修复进程。延迟修复主要考虑系统尽可能的延迟修复进程，让冗余数据能够重返系统，从而减少因修复不必要的失效节点数据而造成的网络与存储开销。例如，Total Recall 是采用延迟修复的典型系统。

3.3.3 网络编码

2005 年，S.Acedanski 等引入网络编码的思想实现数据修复。通过比较不编码(即简单地对文件进行复制)、纠删码以及随机线性网络编码三种冗余方法，得出在具有同样冗余量的情况下，系统能够成功恢复数据的概率。他们得出随机线性网络编码不仅能达到与纠删码相同的高可靠性，并能拥有比纠删码更小的存储开销。 2007 年，A.G.Dimakis 等引入了修复带宽的概念，即新加入节点需要从系统中下载多少数据来产生冗余块，提出了两种网络编码：OMMDS 码以及 RC 码。分析得出两者拥有比一般纠删码更小的修复带宽。A.Kermarrec 引入了一种多节点的合作修复策略，并利用网络编码理论对基于多节点合作修复问题进行建模分析，给出了修复带宽耗费的最优下界。

3.3.4 拉格朗日插值法

基于容错的存储消息恢复。由于数据是分布式存储的，可能少部分服务器在传输数据过程中出现错误，哪怕这个现象出现的几率是比较小的。因此，新方案是利用构造多项式的方法生成具有容错能力的消息并将其加密分布存储于云中。当少部分服务器出错时，通过拉格朗日插值公式恢复出完整明文。

3.4 加密

Hsu 和 Chen 从 Web 应用程序提出了云服务的典范，但只能减轻第三方控制数据威胁，并且任何 Web 应用程序仍然获得用户的明文数据。Gold- reich，bellare，和 Goldwasser 引入增量对称加密，并给出了增量散列和签名的具体研究案例。赵一鸣，符旭朱洪提出了基于伪随机函数的增量加密方案。Wang 等人考虑允许第三方提供用户数据审计服务，不泄露用户的内容的隐私问题。iDataGuard 给出了中间件级的解决方案，外包用户数据文件给不可信的互联网数据服务。目前在线编辑应用程序得到很好的应用，如 Google Docs，Microsoft Office Live。Yan Huang and David Evans 提供用户一种方法，使他们受益于云应用程序在线编辑文件的优势，而不要求他们暴露敏感数据给服务提供商。

一些已有的具体方法如下：

3.4.1 密钥分层管理

对网盘中所有的密钥进行分层管理。密钥分为三层结构，第一层为口令密钥，第二层为主密钥和用户公私钥对，第三层为文件密钥。口令密钥加密主密钥和用户私钥，主密钥加密文件密钥，上层密钥加密下层密钥，保证了密钥的安全性，用户只需要牢记登录口令即可，主密钥，私钥，以及文件密钥在云端密文存储，并且做到了云服务提供商对用户密钥信息的零知晓。密钥分层管理方便、安全、高效。

用户所上传的文件都会在本地加密，加密完成后上传到云端存储。每一个文件对应一个文件密钥。文件密钥经主密钥加密后拼接到文件头部，作为一个整体存储，下载时先将文件下载到本地，在本地进行解密，保证了数据只有在用户本地呈现明文状态。用户数据以密文形式存储在云端，能够有效防止用户信息的泄露，防止非法用户及云服务提供商获取用户信息。通过公钥算法实现密文文件的共享，共享过程需要对下载文件密钥到本地解密，然后用对方公钥对文件密钥加密，加密后传至云端，即可完成密文文件的分享，分享过程不会泄漏

文件密钥信息。

3.4.2 AES 加密

AES 算法是美国国家标准技术研究所 NIST 于 2000 年确定采用 Rijndael 算法，2001 年作为加密标准的新一代对称加密算法。其全称为高级加密标准，作为 DES 替代品，具有安全、高效和适应不同软硬件的优点。作为分组密码算法，它的分组和密钥长度可以改变，一般满足安全性考虑，分组长度定在 128bit，密钥长度可以是 128、192、256bit，相应的轮数是 10、12、14，采用基于状态矩阵的变换。每轮由三层组成:第一层，确保多轮上高度扩散的线性混合层。第二层：由 16 个 S-盒共同起混合作用的非线性层。第三层：子密钥简单作异或处理到中间状态的密钥相加层。AES 的安全性假设，假如相同于每秒 552 个密钥进行 DES 密钥搜索，128bit 的 AES 密钥则要 149 年时间才能搜索完，如果攻击速度不比 AES 密钥生成速度快得多，未来 20 年 AES 是安全的。

3.4.3 不定长 AES 加密

采用 OPENSSL 里的 EVP 方法来实现一种不定长度的加密。然后通过基于位置的签名计算方法，来对文件进行不定长度的分割，这样分割就使得：当文件中改动只存在于少数地方的时候，相应的分块才会有影响，其他分块保持不变。这样就解决了安全性问题和传输速度（也就是性能）之间的矛盾。

3.4.4 密钥密文分散存储

用户的密钥和密文离散分布于云的多个站点中。即使攻击者入侵了部分云站点，他所获得的是无序、离散的密文数据，破译出的残缺明文也不具备完整性和可用性，从而大大增强云资料的安全性。

3.4.5 对称密码算法

对称密钥密码体制又称单钥密码体制或秘密密钥密码体制，指的是加密密钥和解密密钥相同，或者由其中的一方很容易地推导出另一方。按加密的方式不同来划分，对称密码算法可以分为分组密码和序列密码。分组密码是将明文表示的数字序列进行分组，每组分别在密钥的控制下变换成等长的密文数字序列。序列密码是利用密码序列对明文进行每比特加密从而产生密文。

对称密钥密码体制特点是计算开销小，算法简单，加密速度快，但密码通信的安全性依赖于密钥的保密，必须确保通信双方的密钥不会被泄密。

3.4.6 DES 加密算法

DES 是世界上最为广泛使用和流行的一种分组密码算法，他由美国国家标准与技术研究院（NIST）在 20 世纪 70 年代公布出来，分组密码算法 DES 的分组长度为 64bit，密钥长度为 56bit，明文处理过程分为 3 个阶段，第一阶段是重排明文分组 64bit，第二阶段是经过 16 轮相同功能的置换，每轮经过迭代运算，在最后一轮置换中输出左右对半，并且交换次序。第三阶段是一个逆置换。由于密钥会首先通过一个置换函数，对于其后的加密每轮，通过一个左循环移位和一个置换产生一个子密钥，其中由于每轮置换都一样而密钥重复被迭代，所以每轮产生的密钥不相同。为提供 DES 的安全性，可以实现二重 DES 和三重 DES。

3.4.7 RSA 算法

RSA 是现今为止最为成熟完善的公钥密码体制，这一算法被世界广泛应用。RSA 算法依赖于大整数的因式分解的困难性。RSA 算法的运算过程如下：(1)选择两个不公开的大素数 p、q，计算出 n=p •q 和欧拉函数 u(n)=(p-1)(q -1)。(2) 选取一个整数 e(1<e <u(n))，满足 gcd(u(n),e) =1，e 为私钥。令 d 满 e • d=1(mod(u(n)))，d 为公钥，然后 p、q 及 u(n) 要及时销毁。对信息加密时，将明文比特串 M 分组成 i 组，记明文组集合{m_1,m_2,m_3...,m_i}，m_i<n，也就是长度为 log2n，然后每 m 作加密运算:c_i=m^e mod n。信息解密时对密文 c_i 作解密运算：m_i=c^d mod n。

3.4.8 工业界的传统的云存储加密方案

(1)用户首次登录云存储系统，系统向密钥管理中心申请用户的 K。云存储系统验证用户身份信息无误后，密钥管理中心利用对称密码算法生成 K 保存于密钥库中。

(2)用户需要云系统为其提供安全存储服务。他将自己的明文数据 M 上传到云系统，密钥生成器为该用户生成 k。系统利用保存在密钥库的 K 对 k 加密生成密文 C_k=E_k(k) ,再利用 k 对 M 加密成密文集记为 C_M=E_K(M)保存于密文库中，而 C_k 保存在密钥库中。

(3)当用户要求下载云安全数据时，利用 K 对密文 C_k 进行解密得到数据密钥 k= D_K(k)（注：加密密钥和解密密钥相同），这时利用 k 分段解密密文集 C_M 后就可以恢复出明文 M=D_k(C_M)。

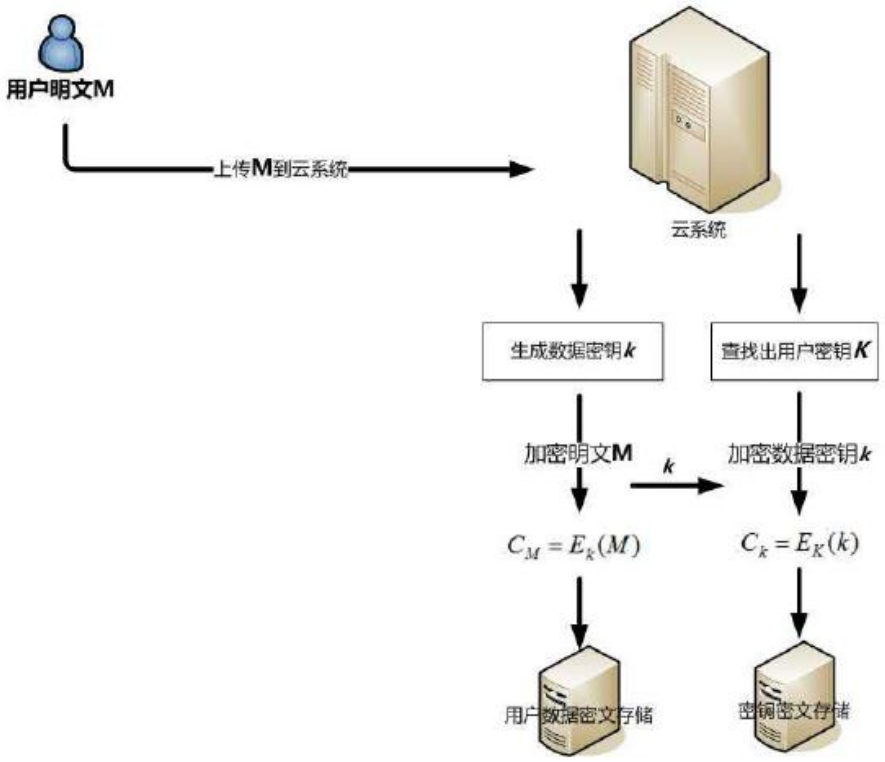


图 1 传统分布式存储加密方法

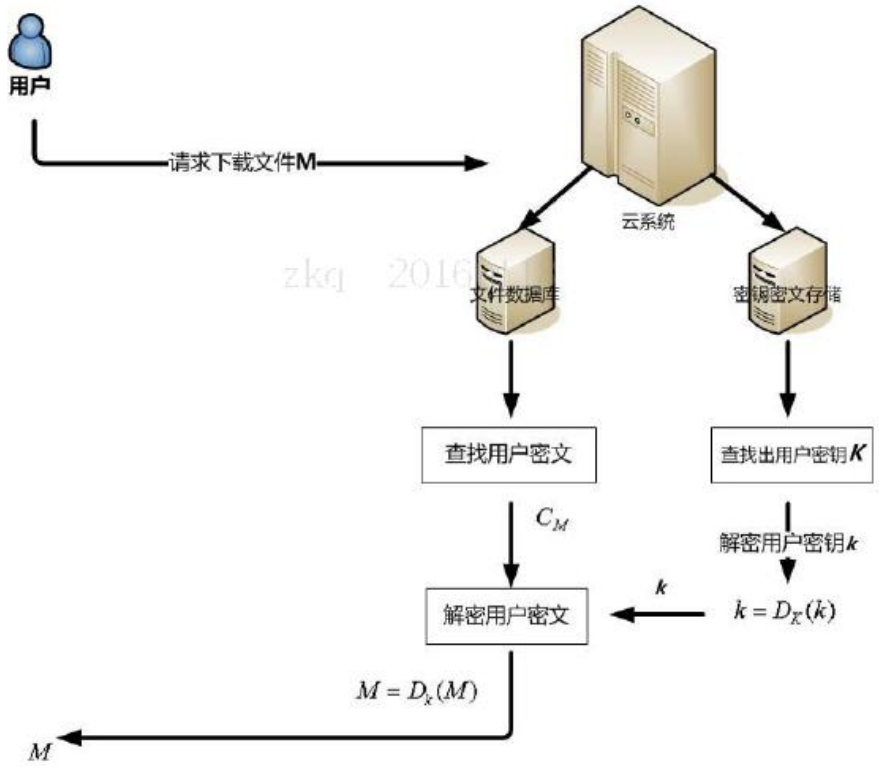


图 2 传统分布式存储解密方法

3. 4. 9 广泛使用的加密系统

3.4.9.1 EFS

EFS 是 Windows 提供的加密文件系统，旨在解决 NTFS 文件的安全存储问题。EFS 采用公钥密码机制和 Crypto API 体系，为每一个文件生成一个随机数作为其对称加密密钥 FEK，通过加强型的数据加密标准算 DESX 对文件进行加密，再利用用户公钥来保护文件密钥的安全性。EFS 和 Windows 操作系统紧密集成在一起，易于管理，不易遭受攻击，并且对用户是透明的。如果用户要访问一个加密的 NTFS 文件，并且有这个文件的私钥，那么用户能够打开这个文件，并透明地将该文件作为普通文档使用。没有该文件私钥的用户对文件的访问将被拒绝。

EFS 使用基于 RSA 的公共密钥加密算法对 FEK 进行加密，并把它和文件存储在一起，形成了文件的一个特殊属性字段：数据解密字段（Data Decryption Field，DDF），如下图所示。在解密时，用户用自己的私钥解密存储在文件 DDF 中的 FEK，然后再用解密后得到的 FEK 对文件数据进行解密，最后得到文件的原文。只有文件属主和管理员掌握解密的私钥。任何人都可以得到加密的公共密钥，但是即使他们能够登录到系统中，由于没有解密的私钥，也没有办法破解它。EFS 主要用于单机用户的文件加解密，支持文件的共享，不支持文件夹共享。

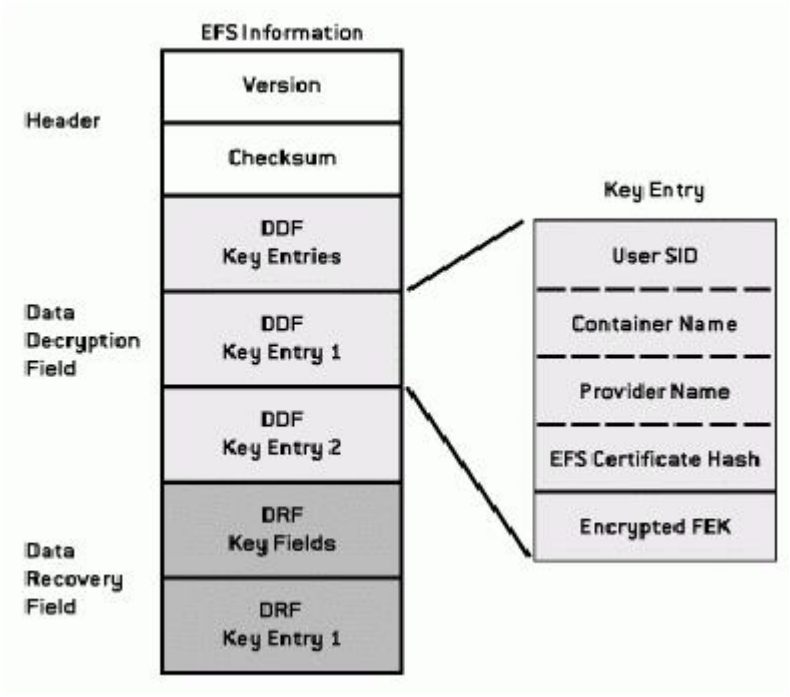


图 3 EFS 加密

3.4.9.2 e Cryptfs

e Cryptfs 是基于 Linux 实现企业级加密文件系统的第一次尝试。它以堆叠式加密文件系统的方式实现加解密过滤层，用户能够自由选择下层文件系统来存放加密文件（例如：Ext3，JFS 等）。由于不修改 VFS 层，e Cryptfs 通过挂载到一个已存在的目录之上的方式实现堆叠的功能。每次使用 e Cryptfs 前，用户只需执行 mount 命令，随后 e Cryptfs 自动完成相关的密钥产生或读取、文件的动态加解密和元数据保存等工作。e Cryptfs 实现文件级和用户级两级密钥。创建一个新文件时，系统利用内核提供的随机函数生成一个文件加密密钥 FEK，用来加密文件内容。新文件关闭时，系统使用用户口令或用户公钥加密 FEK，密钥密文和其他元数据一起存放在加密文件的头部中。打开一个文件时，系统通过下层文件系统读取该文件的头部元数据，解密得到 FEK。e Cryptfs 实现的安全性完全依赖于操作系统自身的安全。它允许多个用户加密同一个文件，但不支持目录级的加密共享。

3.4.9.3 TransCrypt

TransCrypt 也是 Linux 上实现的企业级加密文件系统。它基于系统的超级用户和系统服务不可信的前提，采用内核空间维护密码算法和密钥管理机制的方法，实现算法和密钥的安全，系统的超级用户和服务程序不能获得文件的密钥明文。在提供高安全性的同时，Trans Crypt 解决了多用户共享服务器的灵活性，以及对共享数据透明的远程访问等问题。

与 e Cryptfs 类似，TransCrypt 实现文件级和用户级两级密钥。创建文件时，系统随机生成文件加密密钥 FEK，并用用户公钥加密存储在文件的 ACL 属性中。用户将文件共享给其他用户时，系统首先解密得到 FEK，然后用相应用户的公钥加密 FEK。如下图所示，用户给一个目录增加共享用户时，需要解密得到其中所有文件的 FEK，然后用共享用户的公钥加密每一个文件密钥并存储在文件的 ACL 属性中。Trans Crypt 实现了文件级共享和目录级共享，但是给拥有数千个或数万个文件的目录增加共享用户会非常耗时甚至无法接受。

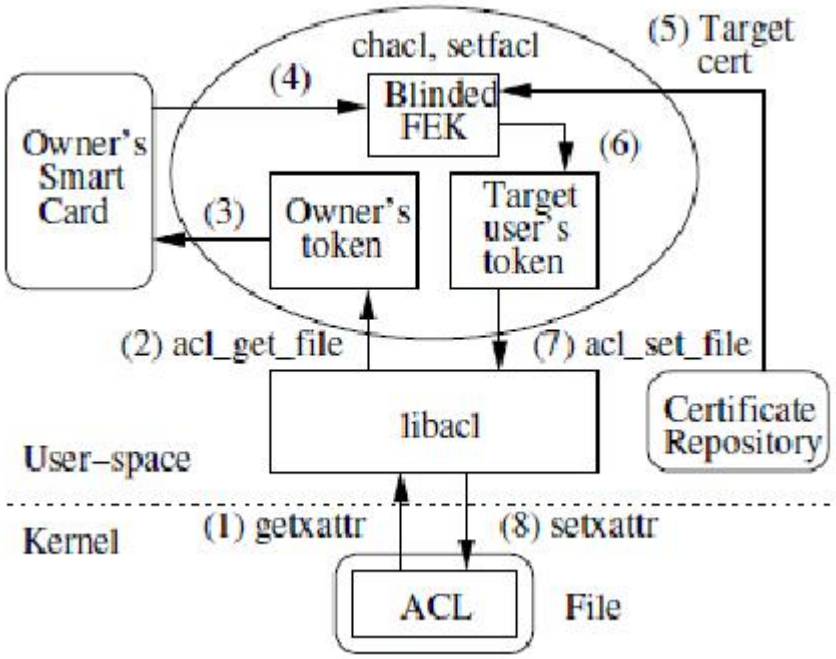


图 4 TransCrypt 加密

3.5 已有的分布式操作系统

3.5.1 GFS

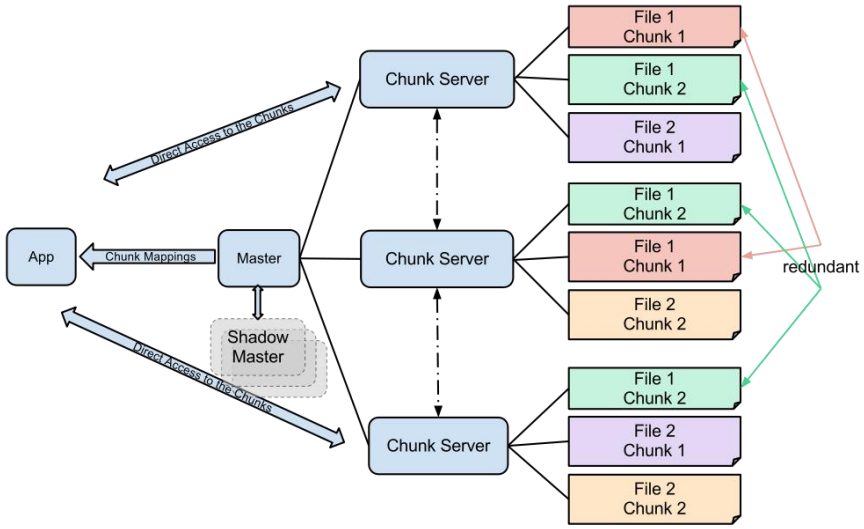
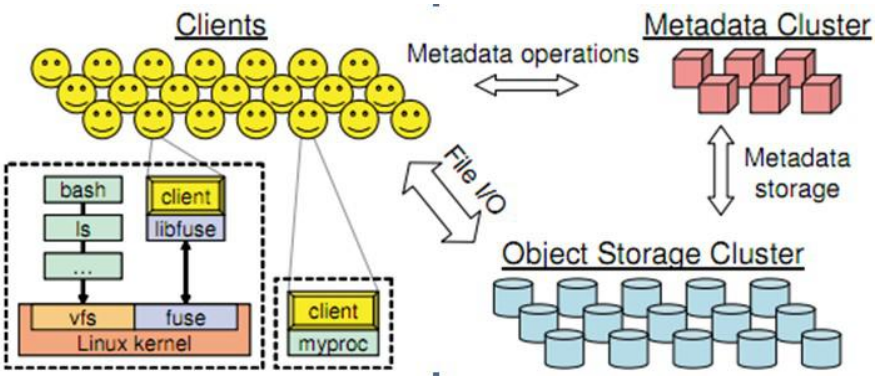


Figure 1

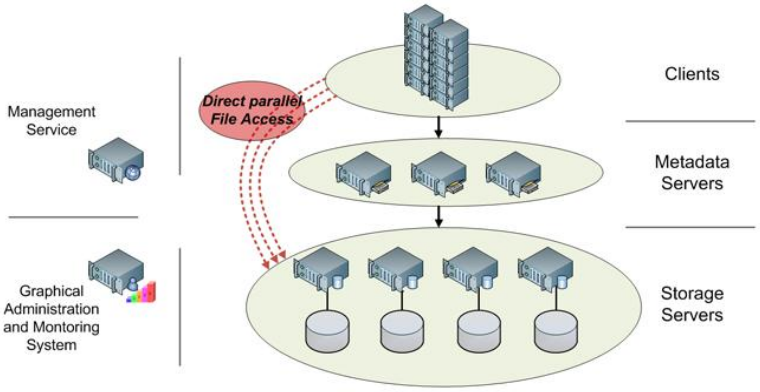
GFS 的架构如 Figure 1 所示：GFS(Google File System)是 Google 公司为了满足其需求而开发的基于 Linux 的专有分布式文件系统。

3.5.2 Ceph



Ceph 的架构如 Figure 2 所示：Ceph 是一个基于 Linux 的 Pb 级文件系统。它的设计 的设计包括保护单一点故障的容错功能，它假设大规模（PB 级存储）存储故障是常见现象而不是例外情况。

3.5.3 FhGFS



FhGFS 的架构如 Figure 3 所示：FhGFS 是由德国 ITWM (the Fraunhofer Institute for Industrial Mathematics)开发的并行文件系统，重点关注的是数据吞吐率。

3.5.4 GlusterFS

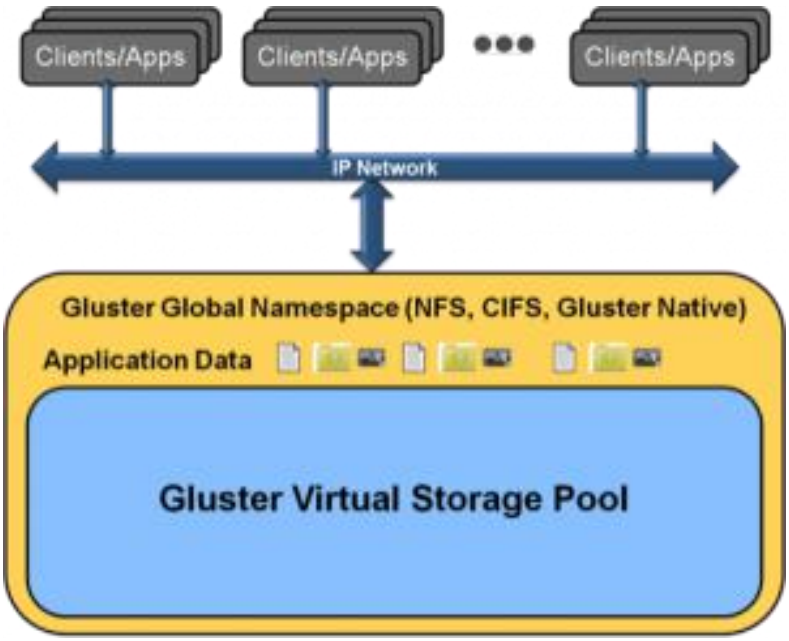


Figure 4

GlusterFS 的架构如 Figure 4 所示：Gluster 和 Ceph 很相似，都是开源系统，都是基于低配计算机搭建，节点间都可以相互备份，并且都是通过算法确定数据的存储位置。

3.5.5 Lutre

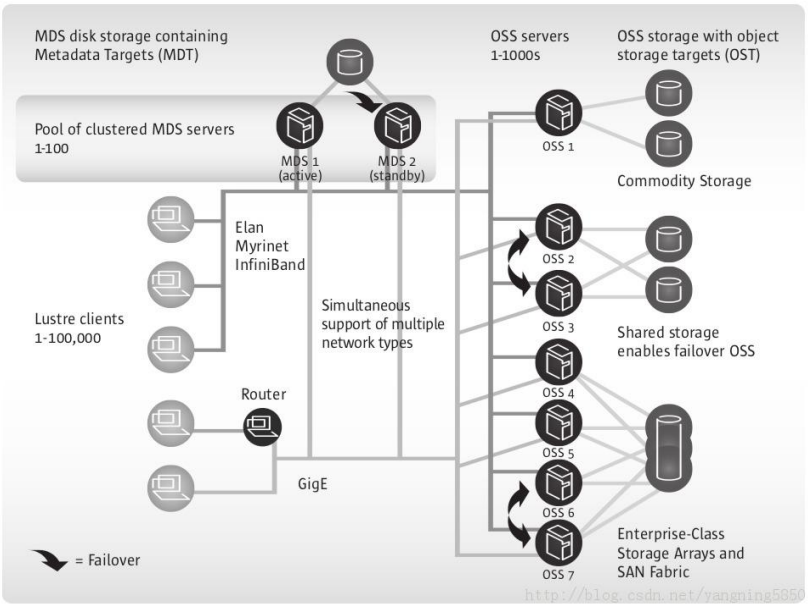


Figure 5

Lutre 的架构如 Figure 5 所示：Lutre 常用在超级计算机中，比如著名的泰坦（前不久刚被

中国的天河二号打败），它具有很高的扩展性，它可以由上万节点的多个集群构成，存储数据可以达到数十 PB 级，吞吐率可以达到每秒 TB 级。

3.5.6 HDFS

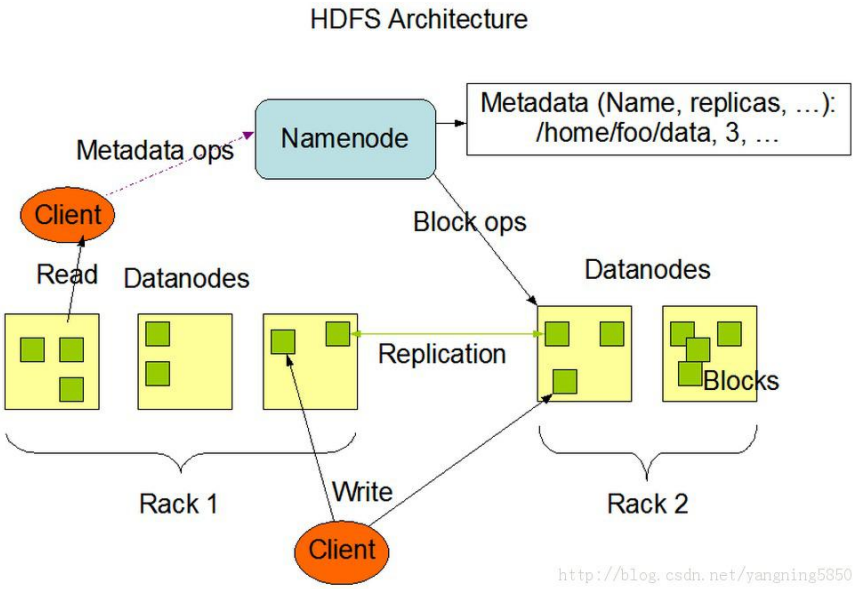


Figure 6

HDFS 的架构如 Figure 6 所示：Hadoop 分布式文件系统(HDFS)被设计成适合运行在通用硬件(commodity hardware)上的分布式文件系统。它和现有的分布式文件系统有很多共同点。但同时，它和其他的分布式文件系统的区别也是很明显的。HDFS 是一个高度容错性的系统，适合部署在廉价的机器上。HDFS 能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。HDFS 放宽了一部分 POSIX 约束，来实现流式读取文件系统数据的目的。HDFS 在最开始是作为 Apache Nutch 搜索引擎项目的基础架构而开发的。HDFS 是 Apache Hadoop Core 项目的一部分。

前瞻性与重要性分析

4.1 实时同步

1. 介绍

信息技术的高速发展改变了人们的生活方式，信息在人们的生活中扮演了越来越重要的角色，信息的存储，传递和展现是当今社会人们面临的巨大挑战。目前，随着数字化的发展，信息数据处于极速膨胀状态，并且信息量还在呈每年递增趋势。信息数据的快速增加，给传统的存储系统带来了巨大的挑战。大容量，高可靠性，高扩展性，高通用性是对存储系统的最新要求。随着对存储系统要求的提高，存储系统经历了从本地存储，分布式存储到云存储的发

展。

(1)本地存储

本地存储是指直接将存储设备连接到使用的主机上。这种存储方式是最直接的存储方式，中间环节少，存储成本低，存储速率快。但是这种存储方式的扩展性较低，但对存储容量有较高要求时，比较容易出现存储容量上的瓶颈。而且本地存储会导致数据备份麻烦，受硬件系统故障的影响也比较大，数据的容错率较低。

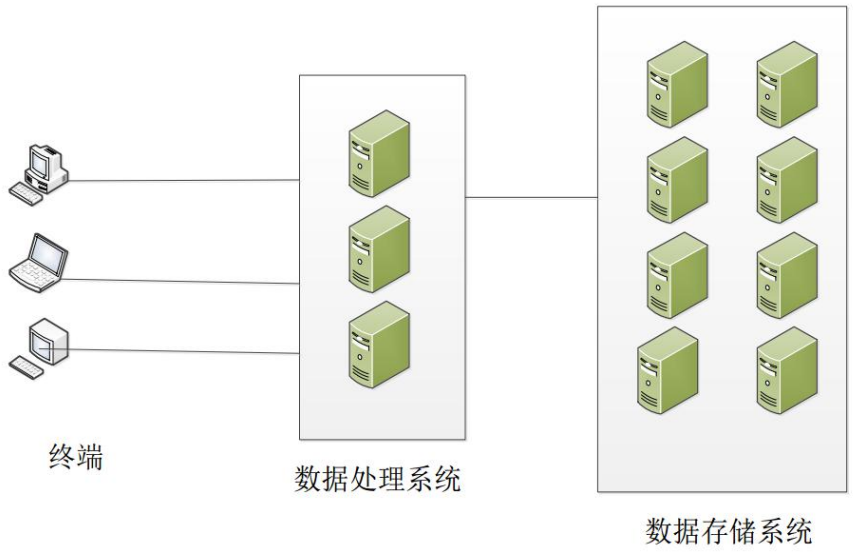
(2)分布式存储

与传统的集中式存储不同，分布式存储并不是将数据存储在某一台固定的存储节点上，而是通过网络使用网络中每一个存储节点中的磁盘空间,将这些分散的存储节点整合成一个虚拟的存储设备。由于每台存储节点都可以进行数据的存储，因此分布式存储的容量比较大。在分布式存储中，可以有多台服务器担任服务器的角色，协同工作，分担服务器压力，因此，并不要求每一台服务器拥有很高性能，在服务器的价格上具有优势。分布式系统不但提高了系统的可靠性、可用性和存取效率，还易于扩展。但是病毒容易在分布式系统中扩散，带来致命的后果，并且分布式存储的数据备份也存在一定困难。

(3)云存储

云存储是云计算范畴内的概念，采用了云计算技术，将物理上分散的存储介质整合在一起，为用户提供存储服务,允许用户将文件保存在云端并随时随地的访问云端中央服务的文件资源。云存储服务通常支持多智能终端的访问请求，例如 PC 客户端，手机客户端，Web 客户端等。云存储服务提供商在数据中心提供可靠安全的海量文件存储服务。

目前，云存储服务在国内外发展迅速，很多互联网企业搭建了自己的云存储服务。典型的代表有：国内的百度网盘，新浪微盘等；国外的 Googledrive，dropbox 等。为了方便用户，云存储服务往往允许用户通过多种客户端访问云端服务器，比如使用 PC 端，移动端和手机端等。



2. 前瞻性/重要性分析

作为当今人们最常使用的数据存储媒介，人们对于网盘所应实现的功能往往带着更多的期待，能够上传、下载和共享文件对于云存储来说是最为基本的功能，对于人们的需求却不不足够；能够实现服务器端与本地文件实行实时同步备份，而无需手动上传，无疑已经成为了云存储所迫切需要实现的功能之一。

我们所需实现的分布式实时同步文件系统应具有以下功能：

1） 文件多版本

服务器端中所备份的文件应按照修改时间具有多个历史版本，方便查看和恢复。

2） 多终端

服务器端可以响应多个终端（如 PC 端/移动端）的文件修改，并及时同步备份。

3） 高并发

一次本地操作的文件修改，将引起相关云存储服务器端的文件更新，并能够尽量做到实时并发传输。

4.2 加密

1. 重要性分析

互联网信息时代的到来使得信息在各方面起到的作用越来越大，而如何存储这样海量的信息也已经成为人们日益关注的问题。在这种大背景下，现代存储技术，尤其是云存储技术既受到了很大的挑战，也迎来了发展机遇。随着云存储技术的不断成熟，它的功能和适用领域越来越多，被更多的用户所接受和依赖，越来越多的人开始利用云存储来管理和备份自己的文件，但随着之而来的安全问题也变得不可忽略起来。近年来，用户资料泄露的例子时有发生，对企业和用户造成了巨大的损失。为了使用户能放心将自己的数据存储到云存储上，云存储服务提供商必须能提供非常安全的方案来保证用户的私密信息数据不能被他人轻易截获，即使截获也不能被他人所破译。因此，云存储的安全必须得到人们广泛的重视。

云存储系统中数据的安全性可分为存储安全性和传输安全性两部分，每部分又包含私密性、完整性、可靠性和不可否认性，前三者对于用户来说是最为重要的。

1)数据的私密性

云存储系统中的数据私密性要求数据不会被非法用户获取或查看。要求无论存储还是传输过程中，只有数据拥有者和授权用户能够访问数据明文，其他任何用户或云存储服务提供商都无法得到数据明文，从理论上杜绝一切泄漏数据的可能性。

2)数据的完整性

云存储系统中数据的完整性包含数据存储时和使用时的完整性两部分。数据存储时的完整性是指云存储服务提供商是按照用户的要求将数据完整地保存在云端，不能有丝毫的遗失或损坏；数据使用时的完整性是指当用户使用某个数据时，此数据没有被任何人伪造或篡改。

3)数据的可靠性

云存储的不可控制性滋生了云存储系统的可靠性研究。数据的可靠性要求数据不会丢失。与传统存储方式不同，云存储中所有硬件均非用户所能控制。因此，如何在存储介质不可控的情况下提高数据的可靠性是云存储系统的安全需求之一。[3]

2. 相关技术

从存储系统的技术支撑与发展来看，文件系统是构建云存储系统的重要部分. CFS 是最早的加密文件系统之一，它是一个用户态的虚拟加密文件系统，可以挂在在其他文件系统之上，为使用者提供文件/文件名加密保护的功能。此后，NCryptfs，ECFS，Cepheus，TCFS 等都

是在 CFS 的基础上研究开发的。NCryptfs 是一个内核态的加密文件系统，它将 CFS 的思想从用户态提升到内核态，同时为用户提供了方便的共享机制。ECFS 在加密数据的基础上提出了校验数据散列值(Hash value)的方式，提供了数据的完整性保护功能。Cepheus 提出了三方架构的模式，提出一个可信的第三方服务器进行用户密钥的管理，引入了锁盒子机制进行用户分组管理，同时提出了懒惰权限撤销的思想。TCFS 提出了多级密钥的加密方式，使用一个主密钥加密原来的文件密钥。

随着网络存储系统的发展，加密文件系统的理念也逐渐网络化、系统化，最终演变成安全网络存储系统。一般的安全网络存储系统至少包括客户端与服务器两部分，客户端由系统的使用者进行操作，为用户数据提供数据加解密、完整性校验以及访问权限控制等功能；服务器作为数据及元数据的存储介质，对数据没有任何的访问或使用权限。

安全网络存储系统中比较典型的有 Plutus，SAND 以及 Corslet 等。Plutus 是 Cepheus 思想在网络存储系统中的扩展。在 Plutus 系统中，客户端负责所有的密钥分发与管理，在共享过程中为用户数据与元数据提供端到端的机密性和完整性保护。SAND 提出了一种密钥对象的数据结构，为网络存储系统提供了端到端的安全解决方案。Corslet 是一个栈式文件系统，通过引入可信第三方服务器，消除了用户对底层存储系统的依赖，在不可信的网络环境下为用户提供端到端的数据私密性、完整性的保护以及区分读写的访问权限控制功能。

云存储的廉价、易扩展等特性使得它一出现就成为人们研究的热点，由于用户将数据存放在云存储中便意味着丧失了对数据的绝对控制权，云存储系统对安全性有着十分迫切的需求。在这种需求的驱使下，Microsoft 于 2009 年提出了 Cryptographic Cloud Storage。Cryptographic Cloud Storage 系统以加密的方式为数据提供机密性保护、以审计的方式为数据提供持有性保护，同时为系统提供了细粒度的访问控制功能，并在系统的原型设计中使用了可搜索的加密机制(searchable encryption)、基于属性的加密机制(attribute_based encryption)、数据持有性证明(probable of data possession)等技术，在提高系统整体性能的同时增强了用户的体验效果。

同业界一样，学术界也很重视云存储系统的安全问题。2010 年，Tang 等人在 FADE 系统中提出了一种解决云存储系统中数据可信删除(assured delete)的方法；Mahajan 等人在 Depot 系统中提出了一种最小化云存储中可信任(可用性方面)实体的方式，只要有一个正确(可访问)的客户端或服务器上有用户需要的数据，用户就可以通过网络获取到正确的数据；Shraer 等人在 VenusLz“系统中提出了基于一个核心集(core Set)的信任体系，通过三方架构的方式为用户提供安全功能. 2011 年，Bessani 等人在 DEPSKY 中提出了云中云的思想，在一定程度上减轻了数据机密性问题和运营商锁(vendordata lock—in)的问题。[3]

目前网络安全的研究已经发展到了很高的水平，出现了很多成熟的安全协议、安全标准和安全产品。而相对来说，存储的安全性研究特别是云存储的安全研究尚处于起步阶段。现在对于存储安全的研究一部分是考虑将现有的信息安全技术运用于特定的存储系统中，类似于搭配的办法，只能说对信息安全技术进行了移植。对存储系统的特性并没有进行深入的研究，因此很难发现真正建立在存储系统上的安全技术。还有一部分就是对已有的存储安全漏洞进行修补，这是一种被动的研究方式，很难取得很好的研究成果，而且依靠这种方式很难完整地构成一个安全体系。构建网络存储安全系统，除了要从网络的角度考虑安全性外，最根本的还是要从存储的角度来讨论和分析。相对来说，网络安全是动态的，而存储安全是静态的。目前的网络存储安全的研究主要表现在：

(1) 通过对存储进行加密来增强存储系统的安全性，可以保证数据的机密性和完整性。

(2) 使用一些防篡改技术，如文件效验码，可以保证数据的完整性。
(3) 使用备份和冗余技术，可以保证数据和系统的可用性。[1]

加密技术是保证数据私密的常用技术，可以分为对称加密和非对称加密。对称加密使用同一个密钥对数据进行加密和解密，常见对称加密算法有 DES、AES、twofish 和 IDEA 等，速度快于非对称加密。数字签名唯一标示相应数据文件，当该文件发生篡改时，其数字签名也就发生了变化，用户只通过检查数字签名即可判断数据完整性，常用的数字签名算法有 SHA-1，MD5 和 HMAC。
冗余编码是一种向前纠错码(FEC)，最先应用与通信领域，后来用于存储系统中保证数据的可靠性。RAID 码是一种常见的冗余编码，常用于磁盘整列中，RAID-5 和 RAID-6 分别能容单盘错和双盘错。Reed-Solomon 码(RS 码)能容更多磁盘失效，它保证 n 个磁盘中的 k 个可以恢复出原始数据(n>k)，即当不多于 n-k 个磁盘发生故障时，数据仍然可用。[2]

可行性分析

- Server-side 指主要操作在服务器上完成。
- Client-side 指主要操作在客户机器上完成。

下面从这两方面进行可行性分析。

5.1 Server-side

为了防止存储内容被利用，需要在客户端加密，而且，从个人用户角度，使用现有网盘存储则缺少服务器资源。因此使用 client-side。

使用 Python 现有的库（如 PyFilesystem）可以直接完成对文件的操作，但在命令行执行需要更多的程序。使用 FUSE 完成文件系统可以达到更底层的个性化，更为自由，但仍然面临程序接口的问题。

5.2 Client-side

5.2.1 使用已有的网盘

* 百度网盘

为了限速政策的进一步执行，百度于今年年初关闭了 API，现在事实上已没有提供给第三方的访问百度网盘的方法。

* Google Drive

Google 提供了 Drive 的 API，其中包含了上传，下载等必要的 API。

* Onedrive
同样提供 API。

优点：简单，易于管理，uptime 有保证。
缺点：成本过高，

5.2.3 在服务器上部署分布式系统

优点：自由，个性化。低成本，高存储量，可以达到很高速度。

5.3 前端

图形化客户端可用工具有.net(c#), QT(C++,python), GTK， Electron(JavaScript)；Web 客户端有 HTML， JavaScript, Canvas。

5.4 结论

最终目标是实现一个具有 Web 前端以及本地客户端的同步系统，借助 Python、FUSE 工具对本地文件操作，并应用 HASH 码实现数字签名，AES-CBC 算法在本地加密，确保上传的文件不会被损坏或篡改。最后，在存储时，文件按块分割(当作二进制数据流)，分别加密上传，考虑冗余以增加可靠性，另一方面，防止运营商对私人数据的泄露。

理论依据

6.1 SHA-1 码

在 1993 年，安全散列算法（SHA）由美国国家标准和技术协会(NIST)提出，并作为联邦信息处理标准（FIPS PUB 180）公布；1995 年又发布了一个修订版 FIPS PUB 180-1，通常称之为 SHA-1。SHA-1 是基于 MD4 算法的，并且它的设计在很大程度上是模仿 MD4 的。现在已成为公认的最安全的散列算法之一，并被广泛使用。

该算法的思想是接收一段明文，然后以一种不可逆的映射将它转换成一段（通常更小）密文，也可以简单的理解为取一串输入码（称为预映射或信息），并把它们转化为长度较短、位数固定的输出序列即散列值（也称为信息摘要或信息认证代码）的过程。

单向散列函数的安全性在于其产生散列值的操作过程具有较强的单向性。如果在输入序列中嵌入密码，那么任何人在不知道密码的情况下都不能产生正确的散列值，从而保证了其安全性。ＳＨＡ将输入流按照每块５１２位（６４个字节）进行分块，并产生２０个字节的被称为信息认证代码或信息摘要的输出。SHA-1 是不可逆的、防冲突，并具有良好的雪崩效应。1bit 数据的改变都会造成输出大大不同。

通过散列算法可实现数字签名实现，数字签名的原理是将要传送的明文通过一种函数运算（Hash）转换成报文摘要（不同的明文对应不同的报文摘要），报文摘要加密后与明文一起传送给接受方，接受方将接受的明文产生新的报文摘要与发送方的发来报文摘要解密比较，比较结果一致表示明文未被改动，如果不一致表示明文已被篡改。

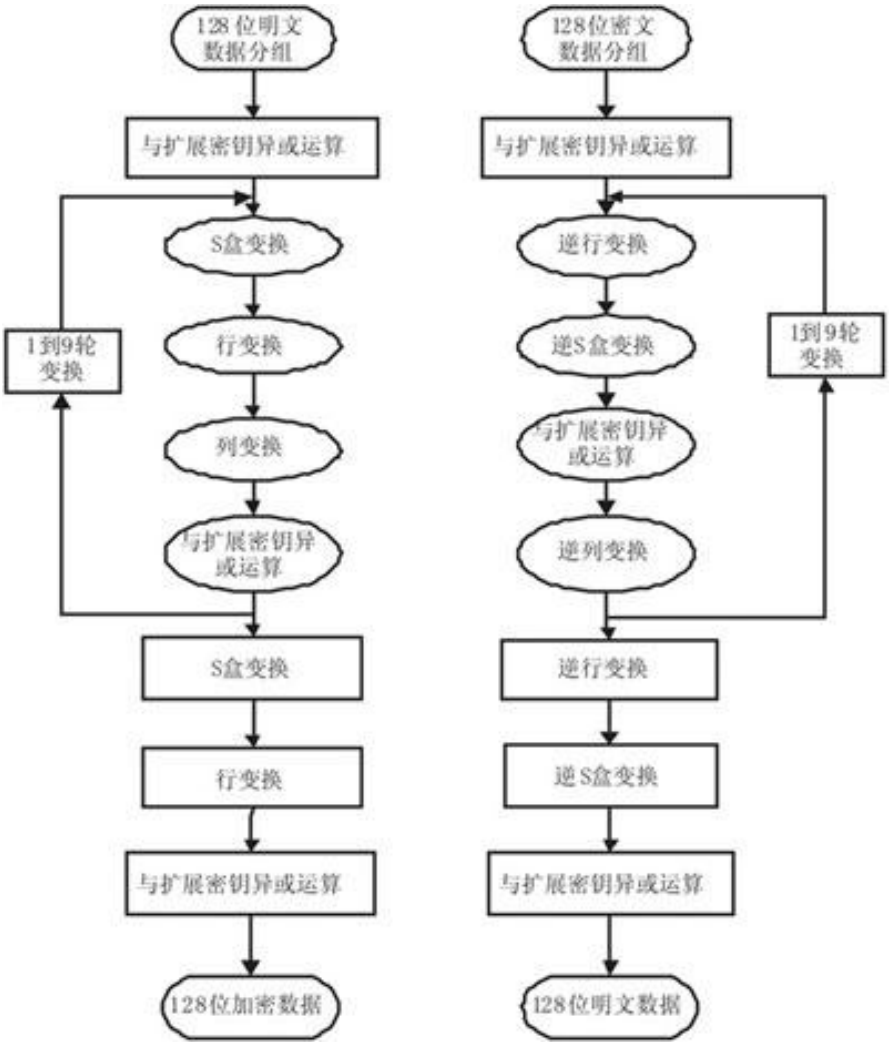
SHA1 产生相同消息摘要的可能性仅为 1/(1*10^48)，可以忽略这种可能性。

6.2 AES

在之前的相关工作中，已经提及 AES 加密算法的理论根据。在之前的相关工作中，已经提及 AES 加密算法，下面对其算法细节进行进一步阐述：

AES 算法（即 Rijndael 算法）是一个对称分组密码算法。数据分组长度必须是 128 bits，使用的密钥长度为 128，192 或 256 bits。对于三种不同密钥长度的 AES 算法，分别称为“AES-128”、“AES-192”、“AES-256”。

以 AES-128 为例，加密、解密流程图如下（事实上，迭代轮数与秘钥长度有关）



• 加密

伪代码实现如图：

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])

   for round = 1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

从伪代码描述中可以看出，AES 加密时涉及到的子模块有 SubBytes()、ShiftRows()、MixColumns()和 AddRoundKey()。这些子模块介绍如下：

① S 盒变换-SubBytes()

S 盒是一个 16 行 16 列的表，表中每个元素都是一个字节。函数 SubBytes()接受一个 4x4 的字节矩阵作为输入，对其中的每个字节，前四位组成十六进制数 x 作为行号，后四位组成的十六进制数 y 作为列号，查找 S 盒中对应的值替换原来位置上的字节。

② 行变换-ShiftRows()

行变换是将矩阵的每一行以字节为单位循环移位：第一行不变，第二行左移一位，第三行左移两位，第四行左移三位。右移的位数也与秘钥长度有关。

③ 列变换-MixColumns()

函数 MixColumns()同样接受一个 4x4 的字节矩阵作为输入，并对矩阵进行逐列变换，变换方式如下：

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad 0 \le c < 4$$

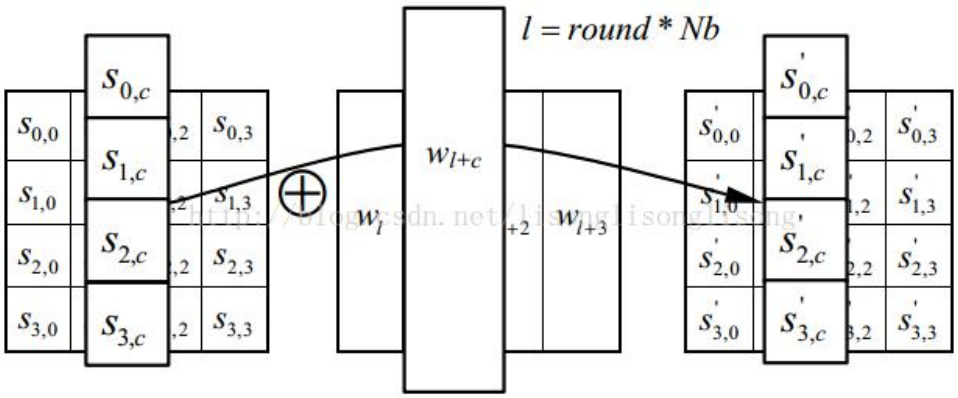
展开以后，得到如下计算式：

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$
$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$
$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c})$$

⊕表示异或，● 表示伽罗华域（有限域）上的乘法

④ 与扩展密钥的异或-AddRoundKey()

扩展密钥只参与了这一步。根据当前加密的轮数，用 w[]中的 4 个扩展密钥与矩阵的 4 个列进行按位异或。如下图：



• 解密

根据 AES 解密的整体流程图（本文开头），伪代码如下：

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]) // See Sec. 5.1.4

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)                    // See Sec. 5.3.1
      InvSubBytes(state)                     // See Sec. 5.3.2
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)                   // See Sec. 5.3.3
   end for

   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])

   out = state
end
```

从伪代码可以看出，我们需要分别实现 S 盒变换、行变换和列变换的逆变换 InvShiftRows()、InvSubBytes()和 InvMixColumns()。

① 逆行变换-InvShiftRows()

加密时 ShiftRows()是对矩阵的每一行进行循环左移，所以 InvShiftRows()是对矩阵每一行进行循环右移。

② 逆 S 盒变换-InvSubBytes()

与 S 盒变换一样，也是查表，查表的方式也一样，只不过查的是另外一个置换表（S-Box 的逆表）。

③ 逆列变换-InvMixColumns()

与列变换的方式一样，只不过计算公式的系数矩阵发生了变化。如下图：

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \qquad 0 \le c < 4$$

展开以后，得到如下计算式：

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$
$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$
$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$
$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

⊕表示异或，● 表示伽罗华域（有限域）上的乘法

由此，可以完整的实现 AES 算法。

在密码分组链接模式（Cipher Block Chaining (CBC)）下，先将明文切分成若干小段，然后每一小段与初始块或者上一段的密文段进行异或运算后，再与密钥进行加密。

技术依据

7.1 AES

事实上，使用一些工具，可以直接实现 AES，而不需要关注其实现细节。

• Crypto 库提供 AES:

示例代码：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from Crypto.Cipher import AES
import base64
PADDING = '\0'
#PADDING = ' '
pad_it = lambda s: s+(16 - len(s)%16)*PADDING
key = '1234567812345678'
iv = '1234567812345678'
source = 'Test String'
generator = AES.new(key, AES.MODE_CBC, iv)
crypt = generator.encrypt(pad_it(source))
cryptedStr = base64.b64encode(crypt)
print cryptedStr
generator = AES.new(key, AES.MODE_CBC, iv)
recovery = generator.decrypt(crypt)
print recovery.rstrip(PADDING)
```

• C#也在库中直接提供 AES 算法，对图形界面程序比较友好，但实现代码更长：

```
namespace test
{               class Class1
    {                       static void Main(string[] args)
        {                           Console.WriteLine("I am comming");
            String source = "Test String";
            String encryptData = Class1.Encrypt(source, "1234567812345678", "1234567812345678");
            Console.WriteLine("=1==");
            Console.WriteLine(encryptData);
            Console.WriteLine("=2==");
            String decryptData = Class1.Decrypt("2fbwW9+8vPId2/foafZq6Q==", "1234567812345678", "1234567812345678");
            Console.WriteLine(decryptData);
            Console.WriteLine("=3==");
            Console.WriteLine("I will go out");
        }
        public static string Encrypt(string toEncrypt, string key, string iv)
        {                           byte[] keyArray = UTF8Encoding.UTF8.GetBytes(key);
            byte[] ivArray = UTF8Encoding.UTF8.GetBytes(iv);
            byte[] toEncryptArray = UTF8Encoding.UTF8.GetBytes(toEncrypt);
            RijndaelManaged rDel = new RijndaelManaged();
            rDel.Key = keyArray;
            rDel.IV = ivArray;
            rDel.Mode = CipherMode.CBC;
            rDel.Padding = PaddingMode.Zeros;
            ICryptoTransform cTransform = rDel.CreateEncryptor();
            byte[] resultArray = cTransform.TransformFinalBlock(toEncryptArray, 0, toEncryptArray.Length);
            return Convert.ToBase64String(resultArray, 0, resultArray.Length);
        }
        public static string Decrypt(string toDecrypt, string key, string iv)
        {                           byte[] keyArray = UTF8Encoding.UTF8.GetBytes(key);
            byte[] ivArray = UTF8Encoding.UTF8.GetBytes(iv);
            byte[] toEncryptArray = Convert.FromBase64String(toDecrypt);
            RijndaelManaged rDel = new RijndaelManaged();
            rDel.Key = keyArray;
            rDel.IV = ivArray;
            rDel.Mode = CipherMode.CBC;
            rDel.Padding = PaddingMode.Zeros;
            ICryptoTransform cTransform = rDel.CreateDecryptor();
            byte[] resultArray = cTransform.TransformFinalBlock(toEncryptArray, 0, toEncryptArray.Length);
            return UTF8Encoding.UTF8.GetString(resultArray);
        }
    }
```

7.2 PyFilesystem

PyFilesystem 是文件系统的抽象层。实际上，任何包含文件和目录的东西（硬盘，压缩文件，FTP 服务器等等）都可以封装成一个共同的接口。使用这个模块，可以不需要知道文件确切的物理位置。FS 对象提供的抽象意味着可以编写与文件物理位置无关的代码。例如，编写了一个在目录中搜索重复文件的函数，则它将在硬盘驱动器上的目录，或 zip 文件，FTP 服务器上，Amazon S3 等上进行更改.

以下是可以使用 Pyfilesystem 访问的一些文件系统：

DavFS 访问 WebDAV 服务器上的文件和目录

FTPFS 访问 FTP 服务器上的文件和目录

MemoryFS 访问文件和存储在内存中的目录（非永久但非常快）

MountFS 创建一个从其他文件系统构建的虚拟目录结构

MultiFS 是将文件系统列表组合到一个的虚拟文件系统，并在打开文件时按顺序进行检查

OSFS 是本地文件系统存储在 Secure FTP 服务器上的 SFTPFS 访问文件和路由

S3FS 访问存储在 Amazon S3 存储上的文件和目录

TahoeLAFS 访问存储在 Tahoe 分布式文件系统上的文件和目录

ZipFS 访问文件和包含在 zip 文件中的目录

使用 PyFilesystem 要比操作底层的接口更加简单。只要选择的文件系统（或任何类似于文件系统的数据存储）中存在 FS 对象，就可以使用相同的 API。这意味着可以推迟将数据存储在以后的决定。如果决定在云中存储配置，则可能是单行更改，而不是主要重构。

PyFiles 系统也可用于单元测试; 通过交换内存中文件系统和操作系统文件系统，可以编写测试，而无需管理（或模拟）文件 IO。代码可以在 Linux，MacOS 和 Windows 上工作。

7.3 FUSE

文件系统是一种用来存储和组织计算机文件、目录及其包含的数据的方法，它使文件、目录以及数据的查找和访问得到简化。文件系统能提供丰富的扩展能力。它可以编写成底层文件系统的一个封装程序，从而对其中的数据进行管理，并提供一个增强的、具有丰富特性的文件系统（例如 cvsfs-fuse，它为 CVS 提供了一个文件系统的接口；或 Wayback 文件系统，它提供了一种用于保留原始数据文件的文件备份机制）。

FUSE 是 Linux 下同于支持用户空间文件系统(Filesystem in Userspace)的内核模块，可以开发功能完备的，在用户态实现的文件系统：其具有简单的 API 库，无需进行内核编程，可以被非特权用户访问，并可以安全的实施。更重要的是，FUSE 以往的表现充分证明了其稳定性。

就文件系统来说，用户空间的文件系统就不再是新奇的设计了。用户空间文件系统的商业实现与学术实现的实例包括：

1）LUFS 是一个混合用户空间的文件系统框架，它对用于任何应用程序无数的文件系统提供透明支持。大部分 LUFS 包括一个内核模块和一个用户空间的守护进程。从根本上来说，它将大部分 VFS 调用都委托给一个专用的守护进程来处理。

2）UserFS 让用户进程可以像普通的文件系统一样进行加载。这种概念性的原型提供了 ftpfs，这可以使用文件系统接口提供匿名 FTP 访问。

3）Ufo Project 是为 Solaris 提供的一个全局文件系统，它允许用户将远程文件真正当作本地文件一样对待。

4）OpenAFS 是 Andrew FileSystem 的一个开源版本。

5）CIFS 是 Common Internet FileSystem 的简称。

与这些商业实现和学术实现不同，FUSE 将这种文件系统的设计能力带到了 Linux 中来。由于 FUSE 使用的是可执行程序（而不像 LUFS 一样使用的是共享对象），因此可以简化程序的调试和开发。FUSE 可以在 2.4.x 和 2.6.x 的内核上使用，除 C 和 C++外，也可以支持 Java 绑定。

要使用 FUSE 来创建一个文件系统，需要声明一个 fuse_operations 类型的结构变量，并将其传递给 fuse_main 函数。fuse_operations 结构中有一个指针，指向在执行适当操作时需要调用的函数。下为 Fuse_operation 结构和其中需要的函数

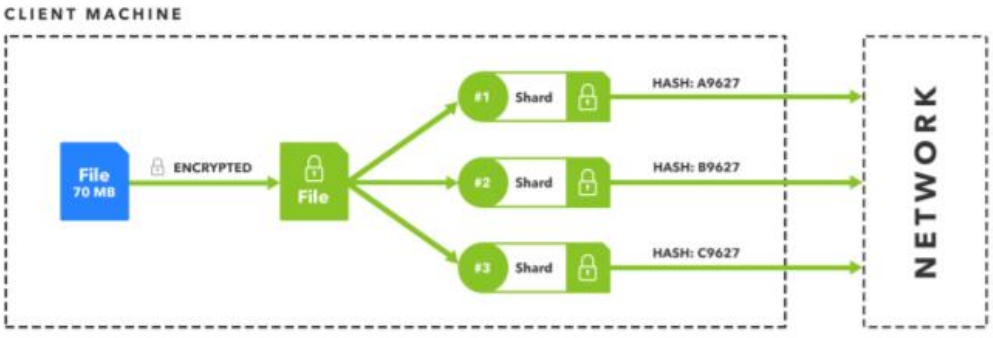
```
struct fuse_operations {
    int (*getattr) (const char *, struct stat *);
    int (*readlink) (const char *, char *, size_t);
    int (*getdir) (const char *, fuse_dirh_t, fuse_dirfil_t);
    int (*mknod) (const char *, mode_t, dev_t);
    int (*mkdir) (const char *, mode_t);
    int (*unlink) (const char *);
    int (*rmdir) (const char *);
    int (*symlink) (const char *, const char *);
    int (*rename) (const char *, const char *);
    int (*link) (const char *, const char *);
    int (*chmod) (const char *, mode_t);
    int (*chown) (const char *, uid_t, gid_t);
    int (*truncate) (const char *, off_t);
    int (*utime) (const char *, struct utimbuf *);
    int (*open) (const char *, struct fuse_file_info *);
    int (*read) (const char *, char *, size_t, off_t, struct fuse_file_info *);
    int (*write) (const char *, const char *, size_t, off_t,struct fuse_file_info *);
    int (*statfs) (const char *, struct statfs *);
    int (*flush) (const char *, struct fuse_file_info *);
    int (*release) (const char *, struct fuse_file_info *);
    int (*fsync) (const char *, int, struct fuse_file_info *);
    int (*setxattr) (const char *, const char *, const char *, size_t, int);
    int (*getxattr) (const char *, const char *, char *, size_t);
    int (*listxattr) (const char *, char *, size_t);
    int (*removexattr) (const char *, const char *);
};
```

7.4 Storj

Storj 是基于 blockchain 技术和点对点(peer to peer)协议的开源、去中心化存储平台，提供最安全，私有和加密的云存储服务。Storj 打算使用多重区块链存储元数据，作为记录文件存储位置和冗余度的目录。一旦区块链变得特别庞大，他们将使用 Merkle 树加速进程，类似于比特币 SPV 钱包工作原理。

实行客户端加密的点对点云存储网络可以使用户不依赖第三方存储供应方来传输和共享数据。去中心存储可以减少大多数传统存储带来的数据错误与断供，并显著提高数据安全、隐私和数据可靠性。Storj 可用于网络上的点间协同传输数据、检验数据的完整性和可用性，恢复数据。支持端对端加密(end-to-end encryption)，采用点对点传输技术(peer to peer)，快速高效，并具有高度可用性。

切分(shard)是加密文件在分布式网路上存储的分配，在安全、隐私、性能和可用性方面有重要作用。文件在被切分前必须先经过加密。这种策略可以保护数据内容不被存储提供方看到，以保护隐私。只有数据拥有者持有对密钥的完整控制，知道文件的切分方式和每个切分的存储位置，进而可以查看和使用数据。当网络中的切分变多，如果不提前知道切分的位置，会使定位给定切分集变得格外困难，这就意味着文件的安全性和网络大小的平方是成比例的。



The sharding process

1. Files are encrypted.
2. Encrypted files are split into shards, or multiple files are combined to form a shard.

3. Audit pre-processing is performed for each shard (see Section 2.3). 4. Shards may be transmitted to the network.

7.5 Sync：Rsync

1. sync 命令是一个远程数据同步工具，可通过 LAN/WAN 快速同步多台主机间的文件。rsync 全称 remote sync，是一种更高效、可以本地或远程同步的命令，之所以高效是因为 rsync 使用所谓的“rsync 算法”，会对需要同步的源和目的进度行对比，只同步有改变的部分，而不是每次都整份传送，因此速度相当快，比 scp 命令高效得多。但是 rsync 本身是一种非加密的传输，可以借助-e 选项来设置具备加密功能的承载工具进行加密传输。
2. rsync 的工作模式：

1) shell 模式，也称作本地模式
2) 远程 shell 模式，此时可以利用 ssh 协议承载其数据传输过程
3) 列表模式，其工作方式与 ls 相似，仅列出源的内容：-nv
4) 服务器模式，此时，rsync 可以工作在守护进程，能够接收客户端的数据请求；在使用时，可以在客户端使用 rsync 命令把文件发送到守护进程，也可以像服务器请求获取文件

3. rsync 命令选项

-v, --verbose 详细模式输出。

-q, --quiet 精简输出模式。

-c, --checksum 打开校验开关，强制对文件传输进行校验。

-a, --archive 归档模式，表示以递归方式传输文件，并保持所有文件属性，等于-rlptgoD。

-r, --recursive 对子目录以递归模式处理。

-R, --relative 使用相对路径信息。

-b, --backup 创建备份，也就是对于目的已经存在有同样的文件名时，将老的文件重新命名为~filename。可以使用--suffix 选项来指定不同的备份文件前缀。

--backup-dir 将备份文件(如~filename)存放在在目录下。

-suffix=SUFFIX 定义备份文件前缀。

-u, --update 仅仅进行更新，也就是跳过所有已经存在于 DST，并且文件时间晚于要备份的文件，不覆盖更新的文件。

-l, --links 保留软链结。

-L, --copy-links 想对待常规文件一样处理软链结。

--copy-unsafe-links 仅仅拷贝指向 SRC 路径目录树以外的链结。

--safe-links 忽略指向 SRC 路径目录树以外的链结。

-H, --hard-links 保留硬链结。

-p, --perms 保持文件权限。

-o, --owner 保持文件属主信息。

-g, --group 保持文件属组信息。

-D, --devices 保持设备文件信息。

-t, --times 保持文件时间信息。

-S, --sparse 对稀疏文件进行特殊处理以节省 DST 的空间。

-n, --dry-run 现实哪些文件将被传输。

-w, --whole-file 拷贝文件，不进行增量检测。

-x, --one-file-system 不要跨越文件系统边界。
-B, --block-size=SIZE 检验算法使用的块尺寸，默认是 700 字节。
-e, --rsh=command 指定使用 rsh、ssh 方式进行数据同步。
--rsync-path=PATH 指定远程服务器上的 rsync 命令所在路径信息。
-C, --cvs-exclude 使用和 CVS 一样的方法自动忽略文件，用来排除那些不希望传输的文件。
--existing 仅仅更新那些已经存在于 DST 的文件，而不备份那些新创建的文件。
--delete 删除那些 DST 中 SRC 没有的文件。
--delete-excluded 同样删除接收端那些被该选项指定排除的文件。
--delete-after 传输结束以后再删除。
--ignore-errors 及时出现 IO 错误也进行删除。
--max-delete=NUM 最多删除 NUM 个文件。
--partial 保留那些因故没有完全传输的文件，以是加快随后的再次传输。
--force 强制删除目录，即使不为空。
--numeric-ids 不将数字的用户和组 id 匹配为用户名和组名。
--timeout=time ip 超时时间，单位为秒。
-I, --ignore-times 不跳过那些有同样的时间和长度的文件。
--size-only 当决定是否要备份文件时，仅仅察看文件大小而不考虑文件时间。
--modify-window=NUM 决定文件是否时间相同时使用的时间戳窗口，默认为 0。
-T --temp-dir=DIR 在 DIR 中创建临时文件。
--compare-dest=DIR 同样比较 DIR 中的文件来决定是否需要备份。
-P 等同于 --partial。
--progress 显示备份过程。
-z, --compress 对备份的文件在传输时进行压缩处理。
--exclude=PATTERN 指定排除不需要传输的文件模式。
--include=PATTERN 指定不排除而需要传输的文件模式。
--exclude-from=FILE 排除 FILE 中指定模式的文件。
--include-from=FILE 不排除 FILE 指定模式匹配的文件。
--version 打印版本信息。
--address 绑定到特定的地址。
--config=FILE 指定其他的配置文件，不使用默认的 rsyncd.conf 文件。
--port=PORT 指定其他的 rsync 服务端口。
--blocking-io 对远程 shell 使用阻塞 IO。
-stats 给出某些文件的传输状态。
--progress 在传输时现实传输过程。
--log-format=formAT 指定日志文件格式。
--password-file=FILE 从 FILE 中得到密码。
--bwlimit=KBPS 限制 I/O 带宽，KBytes per second。
-h, --help 显示帮助信息。

7.6 Server：SFTP

1. SFTP，即安全文件传送协议(Secure File Transfer Protocol)，是一个交互式文件传输程式，

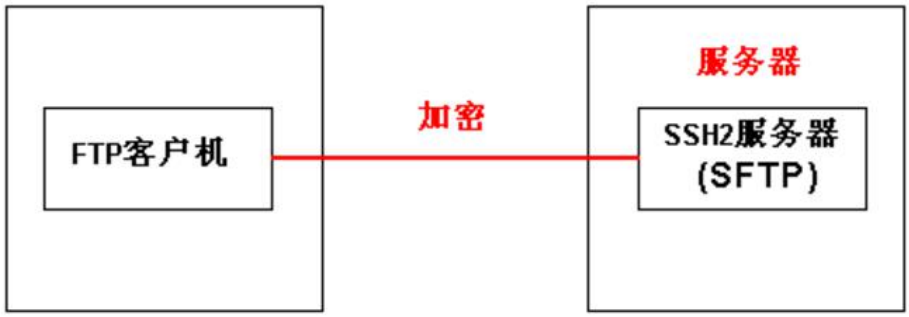
可以为传输文件提供一种安全的加密方法。SFTP 与 FTP 有着几乎一样的语法和功能，但它进行加密传输，比 FTP 有更高的安全性。SFTP 为 SSH 的一部分，是一种传输档案至 Blogger 伺服器的安全方式。其实在 SSH 软件包中，已经包含了一个叫作 SFTP(Secure File Transfer Protocol)的安全文件传输子系统，SFTP 本身没有单独的守护进程，它必须使用 sshd 守护进程（端口号默认是 22）来完成相应的连接操作，所以从某种意义上来说，SFTP 并不像一个服务器程序，而更像是一个客户端程序。SFTP 同样是使用加密传输认证信息和传输的数据，所以使用 SFTP 是非常安全的。但是，由于这种传输方式使用了加密/解密技术，所以传输效率比普通的 FTP 要低得多。

2. 常用登陆方式：

格式：sftp <user>@<host>

通过 sftp 连接<host>，端口为默认的 22，指定用户<user>。

3. 工作模式



SFTP 的工作模式，它是作为 SSH2 的一个子服务工作的。

3.基本使用

(1)文件下载

get [-Ppr] remote [local]

如：get test.cpp ./Project/

将远程当前目录下的文件 test.cpp 下载到本地当前目录的 Project 文件夹中。

(2)文件上传

put [-Ppr] local [remote]

如：put /home/liu/Software/RHEL_5.5\ x86_64.iso /home/xudong/Blog/

将本地/home/liu/Software/目录下的 ios 文件传送到远程登陆主机的/home/xudong/Blog/目录下。

(3)其他命令

-> Help：建立连接后，查看 sftp 支持的命令选项。

-> pwd 和 lpwd：pwd 是看远端服务器的目录，即 sftp 服务器默认的当前目录。
lpwd 是看 linux 本地目录。

-> ls 和 lls：ls 查看 sftp 服务器下当前目录下的文件，lls 查看 linux 当前目录下的文件。

-> ![command]：在 linux 上执行 command 所列命令。例如!ls 是列举 linux 当前目录下的文件， !rm a.txt 是删除 linux 当前目录下的 a.txt 文件。

-> exit 和 quit：退出。

7.7 已有的云运营商提供 API 接口

以 Google Drive 为例。Google Drive 提供其 API 接口，对其用户提供一系列语言中的库以供调用。其中包括 Google APIs Client Library for Python。使用 .json 文件存储用户 ID 以及用户密码以及其他 OAuth 2.0 参数。

API 又分为：

1）简单 API 访问

不获取用户私人数据，调用的程序只需要证明其为 Google API 项目即可。

2）授权 API 访问

获取用户私人数据。因此，在调用该 API 之前必须获得用户授权。且只能进行允许范围内的一些操作，并在用户许可下进行。用户许可后，会提供给应用访问码以及更新码。访问码可以授权 API 调用，但具有时限。访问码一旦过期，可以用更新码来获取新的访问码。

获得相应权限后，通过创建对象并发起请求，即可对云上文件进行操作。

创新点

通常的分布式存储加密都是在服务端进行，不适用于私人用户。而采取在客户端的加密以及签名，实现方式简洁，且易于用户管理。通过 Python 的 PyFilesystem 直接操作本地程序，达到一体化。Javascript 可直接嵌入 HTML，不需要服务器本地程序，可以有更好的界面。

参考文献

【1】Gantz J, Reinsel D. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east[J]. IDC iView: IDC Analyze the future, 2012, 2007(2012): 1-16.

【2】Russel Sandberg; David Goldberg; Steve Kleiman; Dan Walsh; Bob Lyon (1985). "Design and Implementation of the Sun Network Filesystem". USENIX.

【3】http://www.newyorker.com/tech/elements/the-mission-to-decentralize-the-internet.
Joshua Kopstein (12 December 2013). "The Mission to Decentralize the Internet", The New Yorker.

【4】https://zh.wikipedia.org/wiki/MapReduce. Retrieved on Mar 17,2017

【5】http://www.8btc.com/storj-vs-dropbox-why-decentralized-storage-is-the-future. Storj VS. Dropbox：为什么去中心化存储是未来 | 巴比特. Retrieved on Mar 17,2017

【6】云储存 – 百度百科 accessed :2017.3.18

【7】浅谈文件加密的重要性和加密方法-百度文库 accessed:2017.3.18
【8】云同步系统的设计与实现 王少奎
【9】基于操作日志的云存储服务多终端同步算法 张晓杰
【10】实时同步云存储客户端的设计与实现 刘光亚
【11】一种云存储服务客户端增量同步算法 吕瀛
【12】云存储中的数据安全与保密 胡凯
【13】一种基于云存储的安全存储方法 徐玉兰
【14】安全云存储系统与关键技术综述 傅颖勋
【15】https://www.ibm.com/developerworks/cn/linux/l-fuse/#resources
【16】https://github.com/libfuse/libfuse
【17】https://storj.io/storj.pdf
【18】https://storj.io/
【19】https://github.com/Storj/
【20】http://www.linuxidc.com/Linux/2016-10/136143.html
【21】http://man.linuxde.net/rsync
【22】https://developers.google.com/drive/v3/web/about-sdk
【23】http://www.blogjava.net/yxhxj2006/archive/2012/10/15/389547.html
【24】http://docs.pyfilesystem.org/en/latest/index.html