基于 Erasure Code 与 P2P 的分布式文件系统

可行性报告

王珺 夏昊珺 滕思洁 郑值

2017 年 4 月 2 日星期日

摘要

本报告首先提出了一个基于 Erasure Code 与对等网络的分布式文件系统设计并讨论了其可行性；接着介绍了上述分布式文件系统的设计中用到的范德蒙编码、对等网络等技术与理论，分析了基于这些技术设计的分布式文件系统在特定的场景下相较于传统的分布式文件系统的创新与优势。最后，本文调研了文件索引服务器和浏览器对 DFS 的支持作为对此前的调研报告的补充。

目录

一、可行性分析

在这一部分中，我们将就利用 Erasure Code 进行数据保护、基于 P2P 进行网络传输、跨平台的实现、前端访问接口的实现、客户端可设置的共享模式、分布式调度管理、客户节点的设计几个方面讨论我们设计的分布式文件系统及其可行性。

利用 Erasure Code 进行数据保护

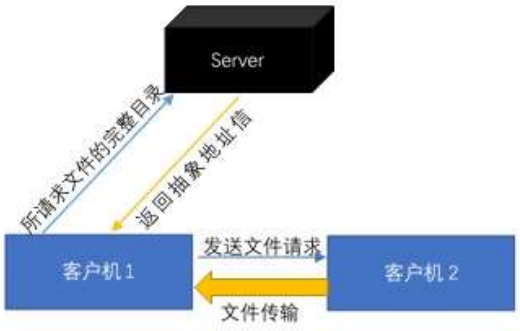
我们设计的分布式文件系统利用Erasure Code（纠删码）进行副本管理以实现数据保护。

纠删码本身目前已经是一种比较成熟的算法，且其中的reed solomon算法是比较早并且已经有开源实现的一种算法，相对引入系统的难度较低。

基于 P2P 进行网络传输

我们设计的分布式文件系统利用了P2P技术，在中心服务器的调度下通过客户端间的直接联系完成数据的存储与传输。有了中心服务器负责调度，维护各节点的网络位置（ip）和文件位置，只要各节点运行了相应的服务进程，一个节点便可以主动和另一个节点直接建立连接，进行所需的文件交互。

其中的文件传输协议可以基于简单的FPT，或者是私人定制的传输协议。
云盘逻辑上等同于一般的客户节点，只不过本身不会主动发起文件请求且需要根据云盘维护方提供的特殊协议进行访问。



文件下载请求模型（其他请求类似）

跨平台的实现

通过使用Java语言编写的客户端，只要在各个节点上搭建好JVM环境，java代码便可以忽略操作系统和硬件平台的差异，实现跨平台。

前端访问接口的实现

关于前端，我们目前尚考虑有两种思路可行。

基于网页：
网页的优势在于更加美观，可以基于js实现

基于java客户端：
使用java内置的GUI库Swing 和JavaFX 2。
客户端的优势在于更符合人们管理文件的习惯（类似于windows文件管理器），但是实现起来时间成本和难度更大。

客户端可设置的共享模式

我们设计的客户端具有一下几种共享方式：

本地私有目录
不进行共享的本地目录（未添加到分享目录列表的目录）；

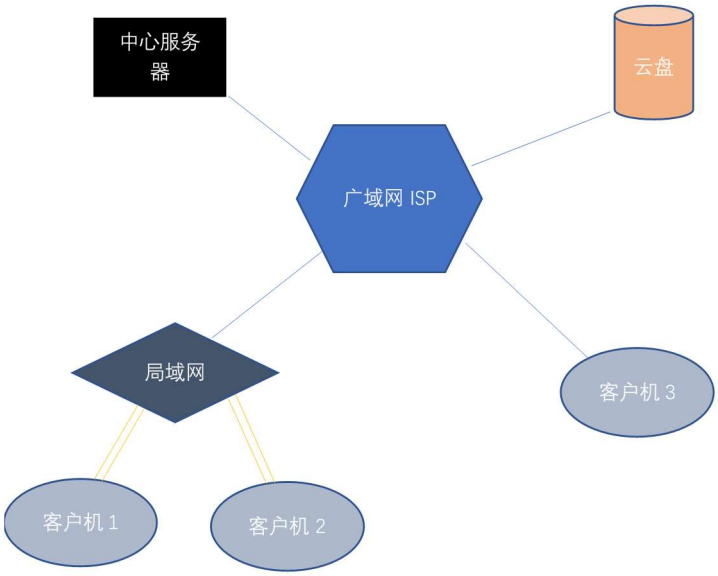
本地分享目录
当客户机在线时，其他节点可以分享的文件目录（即上述所言的本地分享）；

云共享目录
当客户机不在线时，其他节点可以通过访问其文件分布在整个系统上的碎片进行访问的文件目录（即云共享）。在这种共享模式下进行云共享的本机将保留一份原来的文件副本。

未来实现云共享，每台客户机上都会存在一个本地EC文件存储，负责存放整个系统上分配到本机负责存储的EC冗余备份文件。

分布式调度管理

通过增加一个中心服务器，我们大大地减少了P2P网络中的各种问题，提供了集中化的管理。由于服务器运行在确定的机器上，其可以不基于java程序，使用C或C++保证高效性。



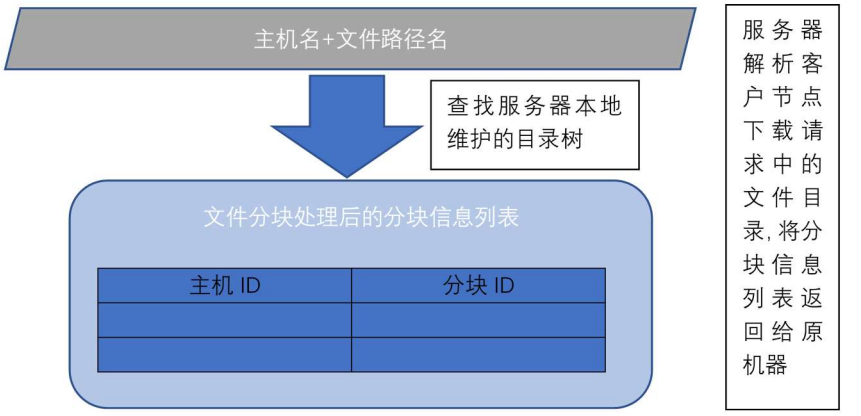
中心服务器的内部可以通过以下数据结构实现预定功能：

P2P节点信息管理：（中心服务器存储并维护下表）

客户编号（Int 变量存储）	IP 地址	EC 文件本地备份空间占用&EC 本地存储允许的最大容量
0		
1		
2		
++		

采用客户机报告信息的方式：客户机节点主动报告上线和下线
同时采用服务器定期广播核实

中心服务器对请求的分析和处理



文件下载时的目录解析模型

其中，分块ID=文件全路径名hash值+分块序号（拼接）
分块序号为0，1，2……

客户节点的设计

客户节点根据从中心服务器获得的列表，优先选择带宽高的对等机进行文件块获取。
客户节点通过主机ID确定文件所在网络位置
客户节点通过分块ID在客户机EC文件夹进行分块的查找（分块EC文件命名为分块ID）

二、理论依据与技术依据

本报告设计的分布式文件系统中主要涉及到了以下的理论与技术：

网络数据传输：为了使分布在不同地点的各种设备可以共同维护、使用我们设计的分布式文件系统，基于网络的数据交换是不可避免的。目前在世界范围内应用最为广泛的网络是因特网，其使用TCP/IP协议互联了世界上数以亿计的各类设备。因此，我们设计的分布式文件系统将使用因特网进行数据的传输。

P2P：我们设计的分布式文件系统利用了P2P的原理，在服务器的调度下通过客户端间的直接联系完成数据的存储与传输。P2P又称对等互联网络技术，是一种利用网络中参与者的计算能力和带宽，而不是把依赖都聚集在较少的几台服务器上的网络技术。P2P网络在文件共享与下载、在线视频、VOIP等领域已经得到了广泛的使用。

JVM：我们设计的分布式文件系统使用Java编程，因此其将运行在Java虚拟机（JVM）上。JVM是一种用于计算设备的规范，它是一个虚构出来的计算机，是通过在实际的计算机上仿真模拟各种计算机功能来实现的。JVM是Java语言实现与平台的无关性的关键：一般的高级语言如果要在不同的平台上运行需要编译成不同的目标代码，而JVM屏蔽了与具体平台相关的信息，使得Java语言只要编译为字节码就可以在多种平台上不加修改地运行。

Erasure Code：我们设计的分布式文件系统使用Erasure Code来实现副本管理。Erasure Code（又称纠删码）是一种编码技术，它可以将n份原始数据，增加m份数据，并能通过n+m份中的任意n份数据，还原为原始数据。

网盘：我们设计的分布式文件系统使用商业网盘来增强可用性。网盘是由互联网公司推出的在线存储服务，利用这些公司的服务器为用户提供文件的存储、访问、备份、共享等功能。由于目前的商业网盘都由专业人员维护并采用了复杂的副本机制，其可靠性与稳定性均有较好的保证。同时，大多数网盘也都提供了API接口和SDK的支持，这使得利用网盘服务来实现新服务变得非常简单。

对于上述大多数内容，我们在之前的调研报告中已经进行了介绍。下面，我们将着重补充如何使用范德蒙码（Erasure Code的一种）进行文件的编码与解码并简要讨论网络数据传输的原理。

用范德蒙码编码及解码文件

编码：

若用d来表示data块，c来表示code块，则采用下图所示范德蒙生成矩阵由data块生成code块：

$$\begin{bmatrix} 1 & 0 & 0\cdots & 0 \\ 0 & 1 & 0\cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0\cdots & 1 \\ 1 & 1 & 1\cdots & 1 \\ 1 & 2 & 3\cdots & n \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 2^{m-1} & 3^{m-1}\cdots & n^{m-1} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}$$

4个data块，2个Code块情况下，编码过程如下：

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \end{matrix} \cdot \begin{matrix} D_0 \\ D_1 \\ D_2 \\ D_3 \end{matrix} = \begin{matrix} D_0 \\ D_1 \\ D_2 \\ D_3 \\ C_0 \\ C_1 \end{matrix}$$

其中Code块是：

$$C_0 = D_0 + D_1 + D_2 + D_3$$
$$C_1 = D_0 + 2 \times D_1 + 4 \times D_2 + 8 \times D_3$$

解码：

解码过程原理是：未损失的数据块和校验块乘以编码矩阵的逆矩阵可以得到原来的数据。

以4个data块，2个Code块的情况的解码来解释，当code的块数为2时，最多坏掉两块数据块（按照解方程就是四元一次方程，至少4个才能解出来四个元的值）。此处假设一个数据块D1和一个code块C0丢失。

解码过程分为两步：

1）顺序遍历编码矩阵的前n行，顺序选取没有损坏的前k行（意思是该行对应的数据块或者校验块没有损坏）。生成k*k的矩阵M。本例中M矩阵如下：

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 4 & 8 \end{matrix}$$

编码encode的时候这几行发生了下面的事情：

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 4 & 8 \end{matrix} * \begin{matrix} D_0 \\ D_1 \\ D_2 \\ D_3 \end{matrix} = \begin{matrix} D_0 \\ D_1 \\ D_2 \\ C_1 \end{matrix}$$

所以解码的时候，有 C_1, D_0, D_1, D_2 以及M，很显然可以通过求M的逆矩阵来求出 D_0, D_1, D_2, D_3：

$$M^{-1} * \begin{matrix} D_0 \\ D_1 \\ D_2 \\ C_1 \end{matrix} = \begin{matrix} D_0 \\ D_1 \\ D_2 \\ D_3 \end{matrix}$$

2）求出损失的数据。由于（1）中已经求出来了所有的数据块的内容，而且编码矩阵是知道的，因此可以求出所有的数据，对于本例子，其实是在第三个式子两边同时乘以一个矩阵来求出C_0，得到的就是相应的编码矩阵的部分，于是就求出来了丢失的数据D_1和 Code C_0。

网络数据传输

互联网连接了世界上不同国家与地区无数不同硬件、不同操作系统与不同软件的计算机，为了保证这些计算机之间能够畅通无阻地交换信息，必须拥有统一的通信协议。而作为一个通信协议，要提供数据传输目的地址和保证数据迅速可靠传输的措施，这是因为数据在传输过程中很容易丢失或传错，所以互联网上就使用TCP/IP作为一个标准的通信协议。

TCP/IP协议所采用的通信方式是分组交换方式。就是数据在传输时分成若干段，每个数据段称为一个数据包，TCP/IP协议的基本传输单位是数据包，TCP/IP协议主要包括两个主要的协议，即TCP协议和IP协议，这两个协议可以联合使用，也可以与其他协议联合使用，它们在数据传输过程中主要完成以下功能：

（1）首先由TCP协议把数据分成若干数据包，给每个数据包写上序号，以便接收端把数据还原成原来的格式。
（2）IP协议给每个数据包写上发送主机和接收主机的地址，一旦写上源地址和目的地址，数据包就可以在网上传送数据了。
（3）这些数据包可以通过不同的传输途径(路由)进行传输，由于路径不同，加上其它的原因，可能出现顺序颠倒、数据丢失、数据失真甚至重复的现象。这些问题都由 TCP协议来处理，它具有检查和处理错误的功能，必要时还可以请求发送端重发。

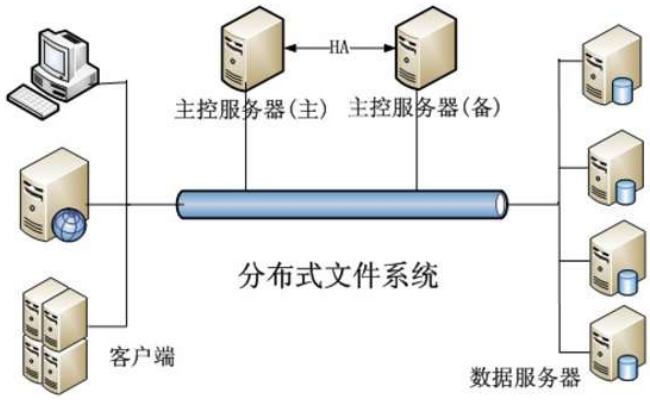
换句话说，IP协议负责数据的传输，而TCP协议负责数据的可靠传输。TCP/IP协议参考模型如下图：

三、创新点

在这一部分中，我们会从体系结构、备份方式、可用性、访问方式和部署难度五个方面将本报告中设计的分布式文件系统与目前已有的分布式文件系统进行对比并分析其具有的优势与创新。

体系结构：借鉴 P2P 机制但保留了名字服务器

目前主流的分布式文件系统常常采用下图[3.1]所示的结构：



其中主控服务器（也称名字服务器）主要负责响应用户的请求、维护系统的状态（如命名空间、各文件的物理位置、各数据服务器的状态等）并根据系统状态进行适当的调度；数据服务器则主要负责文件的接收、存储和发送。这种架构的分布式文件系统明确地区分了各个服务器的功能与任务，具有状态易于收集维护、扩展性好的优点，但为了保证服务的稳定与高效，其往往要求数据服务器是专用且同构的：对于家庭、小微企业而言，往往缺乏能力或意愿去购买、维护这些专用的数据服务器。

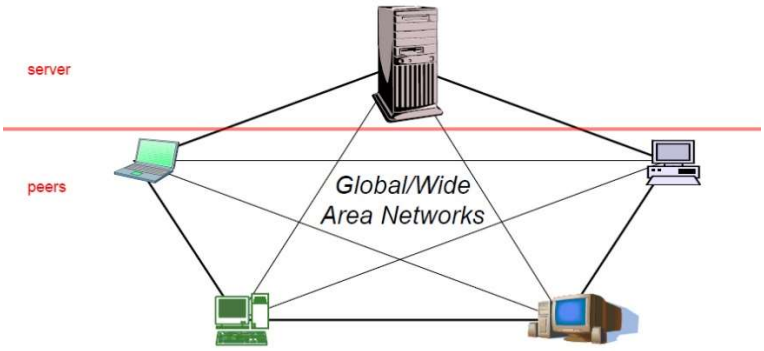
目前也有一些基于 P2P 的分布式文件系统，下表[3.2]展示了其中的一些例子：

System	Scheme	Lookup Time	Unit	Persist-ence	Read/ Write
Freenet	Depth-First Flooding	Variable	File	No	Read Only
CFS	Chord	O(log N)	Block	Yes	Read Only
PAST	Pastry	O(log N)	File	Yes	Read Only
Ocean Store	Bloom Filters + Tapestry	O(log N)	Fragment	Yes	Read/ Write
Ivy	Chord	O(log N)	Log-based	Yes	Read/ Write

这些分布式文件系统往往完整地使用了 P2P 的结构，没有保留任何中心服务器。这使得它们维护系统状态、查找并存储需要的文件的工作变得非常困难，工作效率大大低于上述主流的分布式文件系统，甚至有些无法完整的实现文件系统应有的功能。

我们设计的分布式文件系统创新性地结合了上述两种结构的优点。如下图所示，我们使用少量（往往是一个）专用服务器维护系统状态，调度大量对等方并响应请求；对等方们则在使用分布式文件系统的服务的同时共享自己的存储空间组成系统的存储空间。与大多数

分布式文件系统的实现相同，在收到服务请求时，专用服务器只是查询其文件目录并向请求方返回文件具体的物理位置，而具体的数据传输工作则在对等方之间完成，这使得我们少量的专用服务器不至于成为系统性能的瓶颈。

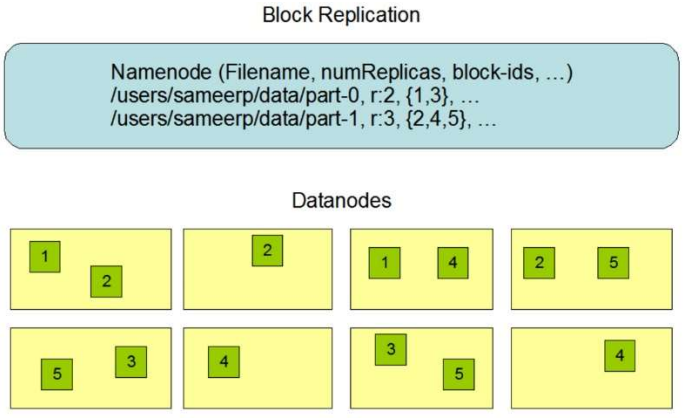


在我们的设计中，通过有限的专用服务器的介入，系统状态的维护较上述纯 P2P 实现大为简化，查询文件的操作也可以在专用服务器内完成而无需轮询各对等方，这使得我们的分布式文件系统可以获得近似于主流分布式文件系统的性能；同时，与一般的 P2P 系统相同，我们既不要求各对等方长期保持在线，也不要求各对等方具有使用同样的硬件平台或操作系统，这使得我们的分布式文件系统可以方便的运行在一般的办公电脑甚至是闲置设备上。

备份机制：Erasure code

为了提供容错功能并保证在数据节点（无论是传统的数据服务器还是P2P机制中的对等方）崩溃或离线时数据的可用性，大多数分布式文件系统都实现了备份机制，即通过保存文件的多个副本保证文件的安全性。

目前包括GFS、HDFS在内的主流分布式文件系统往往采用了完全复制副本[3.3][3.4]，即将一个文件（文件块）完整地复制为多份完全相同的副本并保存在不同的位置。以HDFS为例，HDFS的备份机制如下图[3.3]所示:



HDFS允许用户为不同的文件设置不同的复制级别并根据文件的复制级别将文件复制为相同的多份并保存在不同的服务器上。HDFS的默认复制级别为3，这时其会将一个副本存放在本地机架的节点上，另一个副本放在同一机架的另一个节点上，最后一个副本放在不同机架的节点上。

完全复制副本的方式无疑能提供文件的安全性，同时其也允许客户端获取离其最近的副本以加快访问速度，但完全复制副本的缺点也是明显的：其消耗了过多的存储空间，以HDFS默认的3备份为例，这意味着系统只能提供其物理存储空间1/3容量的存储服务。此外，在采用P2P机制的分布式文件系统中，由于对等方的服务较专用的服务器更不稳定，其往往需要设置更大的复制级别，也就意味着可用的存储空间更小。

Erasure Code（中文名为抹除码或纠删码）是一种编码技术，它可以将n个原始数据块编码为n+m个数据块，并能通过在n+m个数据块任取n个进行解码还原出原始数据。Erasure Code有多种实现方式，Reed-Solomon Code[3.5]和 Tonardo Code 是最常见的 2 种编码方式。Reed-Solomon Code的编码方式如下图示，对于原始数据$d_1 \sim d_n$我们可以左乘一个编码矩阵（由单位矩阵与范德蒙矩阵组合而成）进行编码；而解码时只需在编码矩阵中删去未获得的m个数据对应的行，将剩余的n*n 阶矩阵求逆并左乘当前拥有的数据组成的列向量即可。

$$\begin{bmatrix} 1 & 0 & 0\cdots & 0 \\ 0 & 1 & 0\cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0\cdots & 1 \\ 1 & 1 & 1\cdots & 1 \\ 1 & 2 & 3\cdots & n \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 2^{m-1} & 3^{m-1}\cdots & n^{m-1} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}$$

一般的，记编码效率$\mathrm{r} = \frac{n+m}{n}$,表示编码后总占用空间与原始数据大小的比。对于常见的应用，一般把 m 设置为$\frac{n}{2}$左右，即 r 约为 1.5，显然这个结果要远好于完全复制副本的实现方式。

我们计划使用 Erasure Code 作为我们的数据备份方式。目前已经有了一些基于 Erasure Code 的存储系统设计[3.6][3.7]，但正如前文所述，主流的分布式文件系统大多现在仍使用完全复制副本的方式。

可用性：结合对等方主机与网盘服务

除了备份机制，为了进一步提高文件的可用性，我们创新性的将目前已十分成熟的网盘资源整合进了本文设计的分布式文件系统，这种实现方式在目前常见的分布式文件系统中尚未出现。

当一个对等方需要上传云共享文件时，中心服务器会向其返回一个主机列表指示每个块要保存的位置，而主机列表中即包括其他对等方，也包括网盘；类似的，当一个对等方需要下载云共享文件时，中心服务器也会向其返回一个主机列表指示每个块当前保存的位置，对等方可以根据当前各主机的在线情况与网络情况自行决定在哪些对等方/网盘中下载块以组成完整的文件。

需要注意的是，由于访问网盘的协议是由网盘的提供方设计的，其往往与我们设计的对等方间的通讯协议不同。故在本文设计的分布式文件系统中，网盘与对等方间的差异是显式的，即请求块的对等方必须根据主机列表中的主机类型选择合适的通讯协议。

由于网盘往往由专业的数据存储企业运营、维护，其本身也使用了一套数据备份机制，故其几乎可以保证总是可用。因此，网盘的加入能够大大地提高分布式文件系统中文件的可用性。同时，与单纯依靠网盘进行存储相比，我们的设计只需在网盘中存储文件的部分块（而非整个文件），这意味着我们的设计可以利用相同空间的网盘资源存储更多的文件，进而减少了租赁网盘空间的成本。

访问方式：从 API、客户端到网页

以GFS为代表的主流分布式文件系统往往面向专业用户或程序设计者，其提供的访问接口往往以库函数或POSIX标准的API为主。由于缺少专业的技术人员，一般的家庭、非IT领域的小微企业应用这类分布式文件系统是非常困难的。

我们设计的分布式文件系统提供了包括传统的API、客户端软件在内的多种访问方式。我们创新性的提供了网页访问方式，允许用户使用任何一台连接互联网的主机如同打开商业网盘一样打开我们的分布式文件系统。

部署难度：纯 Java 开发的客户端，可在任何 Java 虚拟机上运行

为了尽可能追求效率，包括GFS、NFS、FastDFS在内的大多数分布式文件系统往往选择使用C/C++进行实现，这意味着在各个服务器/对等方不同构时（如使用了不同的处理器、操作系统）往往需要多次编译以使分布式文件系统可以在所有机器上运行。

随着JIT技术的发展，目前Java的执行效率已得到了巨大的提升，同时，Java基于JVM的运行模式使其可以兼容各种体系结构的主机。我们的客户端完全使用Java编写，这使得任何可以允许Java虚拟机的主机理论上都可以运行我们的客户端。

附录：相关调研

在这一节中我们调研了DFS的浏览器支持和目录索引服务器作为对调研报告的补充。

目录索引服务器

文件结构 ：

目录项：一个文件系统维护的一个索引节点的数组。目录项中的每一项包括文件索引节点 i 的节点号和文件名，目录只是将文件的名称和它的索引节点号结合在一起的一张表。

索引节点 ： 又称 I 节点，在文件系统结构中，包含有关相应文件的信息的一个记录，这些信息包括文件权限、文件名、文件大小、存放位置、建立日期等。文件系统中所有文件的索引节点保存在索引节点表中。

数据 ： 文件的实际内容。（在服务器中不保存文件数据）

文件的记录形式：

文件系统使用索引节点(inode)来记录文件信息。一个文件系统维护了一个索引节点的数组，每个文件或目录都与索引节点数组中的唯一的元素对应。每个索引节点在数组中的索引号，称为索引节点号。文件系统将文件索引节点号和文件名同时保存在目录中，目录中每一对文件名称和索引节点号称为一个连接。

文件在服务器中的索引：

在我们的 dfs 设计中文件数据不会被保存到服务器中，也就是在服务器中只会储存目录项和索引节点。另外文件目录的部分会增加文件所在的主机以确定文件所在位置。文件索引服务器目的：通过文件名确定文件所在目录（及主机），已经在机器中具体储存位置。

索引文件：

索引文件由主文件和索引表构成。

主文件：文件本身；

索引表：在文件本身外建立的一张表，它指明逻辑记录和物理记录之间的一一对应关系；

索引文件的建立：按输入记录的先后次序建立数据区和索引表。其中索引表中关键字是无序的。待全部记录输入完毕后对索引表进行排序，排序后的索引表和主文件一起就形成了索引文件。

索引表结构选择：

由于文件系统经常需要更新或发生变动，以及考虑到访问的性能。选取 B 树作为其结构。理由如下：

① 插入、删除方便

② 本身是层次结构，无须建立多级索引

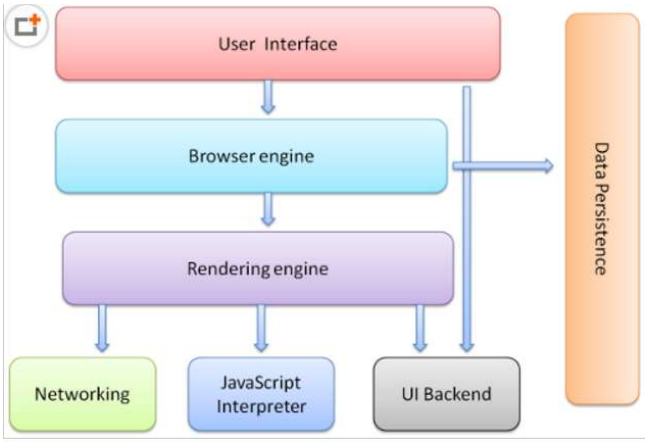
③ 建立索引表的过程即为排序过程。

④ B 树的查找效率高。

浏览器对 DFS 的支持

浏览器组件构成：

大多数浏览器的组件构成如图：



在最底层的三个组件分别是网络，UI后端和js解释器。作用如下：

（1）网络：用来完成网络调用，例如http请求，它具有平台无关的接口，可以在不同平台上工作。
（2）UI后端：用来绘制类似组合选择框及对话框等基本组件，具有不特定于某个平台的通用接口，底层使用操作系统的用户接口。
（3）JS解释器：用来解释执行JS代码。

Javaspcript执行原理：

JavaScript 是一种动态、弱类型、基于原型的语言，可以通过浏览器可以直接执行。如下图示，JavaScript 由核心（ECMAScript）、文档对象模型（DOM）和浏览器对象模型（BOM）三部分组成。

当浏览器遇到<script> 标记的时候，浏览器会执行之间的 javascript 代码。嵌入的 js 代码是顺序执行的，每个脚本定义的全局变量和函数，都可以被后面执行的脚本所调用。
ECMAScript 是一个描述，定义了脚本语言的所有属性、方法和对象。其他语言也可以实现 ECMAScript 来作为功能的基准。
DOM（文档对象模型）是 HTML 和 XML 的应用程序接口（API）。DOM 将把整个页面规划成由节点层级构成的文档 IE 3.0 和 Netscape Navigator 3.0 提供了一种特性 - BOM（浏览器对象模型），可以对浏览器窗口进行访问和操作。使用 BOM，开发者可以移动窗口、改变状态栏中的文本以及执行其他与页面内容不直接相关的动作。。HTML 或 XML 页面的每个部分都是一个节点的衍生物。
文件操作支持：
Html 定义了<input type="file" /> 用于文件上传。因此可在网页上实现上传文件。
Javascript 很多对文件的操作：

CopyFile() 复制文件
CopyFolder() 复制目录
CreateFolder() 创建新目录
CreateTextFile() 生成一个文件
DeleteFile() 删除一个文件
DeleteFolder() 删除一个目录
DriveExists() 检验盘符是否存在
Drives 返回盘符的集合
FileExists() 检验文件是否存在
FolderExists 检验一个目录是否存在
GetAbsolutePathName() 取得一个文件的绝对路径
GetBaseName() 取得文件名
GetDrive() 取得盘符名
GetDriveName() 取得盘符名
GetExtensionName() 取得文件的后缀
GetFile() 生成文件对象
GetFileName() 取得文件名
GetFolder() 取得目录对象
GetParentFolderName 取得文件或目录的父目录名
GetSpecialFolder() 取得特殊的目录名
GetTempName() 生成一个临时文件对象
MoveFile() 移动文件
MoveFolder() 移动目录

参考文献

[3.1] zyd_cu（ChinaUnix博客）. *分布式文件系统：原理、问题与方法*

[3.2] Chun-Hsin Wu. *P2P Storage Systems*

[3.3] HADOOP官方文档. Hadoop分布式文件系统：架构和设计

[3.4] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. *The Google File System*

[3.5] Reed I S, Solomon G. *Polynomial Codes Over Certain Finite Fields[J]*

[3.6] Alexandros G. Dimakis, Vinod Prabhakaran, Vinod Prabhakaran, *Decentralized erasure codes for distributed networked storage*

[3.7] 贾军博, 谷建华, 朱靖飞, 等. *网络环境下基于Erasure Code的高可靠性存储体系设计[J]*

[7] Kevin Fu, M. Frans Kaashoek, David Mazieres. *Fast and secure distributed read-only file system*

[8]Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. *The Andrew File System (AFS)*