#### 一、数据模型

##### ColumnFamily

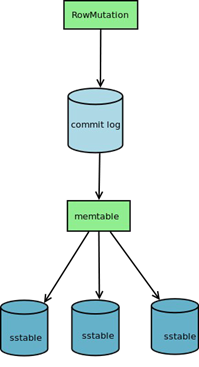
大体上，列族是为了将相关的列群组在一起，以便一次性读出相关数据，而不是单个分散的数据。

Key 决定从哪里找，怎么找到对应的列族

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ColumnFamily |  |  |  |  |  |  |  |
| key | Column List (1 .. \* Columns) |  |  |  |  |  |  |
|  | Name | Value | Timestamp | Name | Value | Timestamp | ... |

##### 存储体系

沿用 BigTable 的术语，内存中的表示用 memtable 表示，数据文件为 SSTable (Sorted String Table)。Write Path 需要通过以下几个层次：



#### 二、系统架构

##### 概述

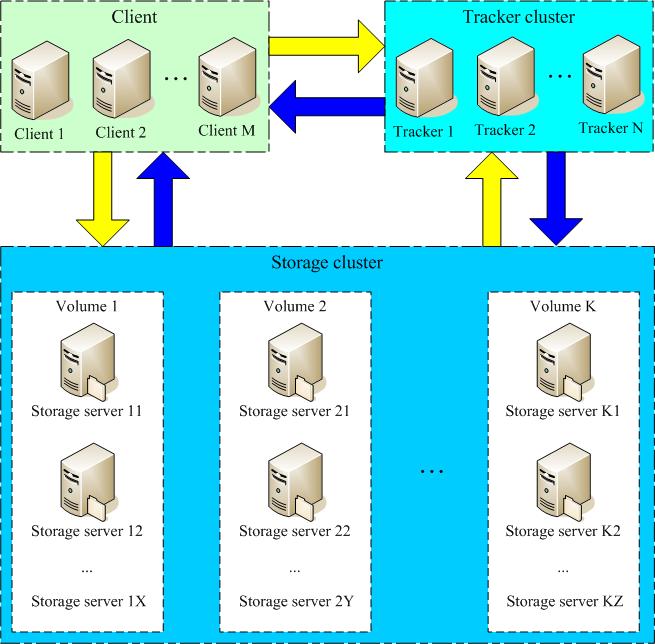
DFS只有两个角色：Tracker和Storage。Tracker 作为中心结点，其主要作用是负载均衡和调度。Tracker 在内存中记录分组和Storage的状态等信息，不记录文件索引信息，占用的内存量很少。另外，客户端（应用）和Storage节点访问Tracker时，Tracker 扫描内存中的分组和Storage节点信息，然后给出应答。由此可以看出Tracker 非常轻量化，不会成为系统瓶颈。

Storage节点 直接利用OS的文件系统存储文件。DFS不会对文件进行分块存储，客户端上传的文件和Storage节点 上的文件一一对应。

众所周知，大多数网站都需要存储用户上传的文件，如图片、视频、电子文档等。出于降低带宽和存储成本的考虑，网站通常都会限制用户上传的文件大小，例如图片文件不能超过5MB、视频文件不能超过100MB等。文件分块存储没有多大的必要。它既没有带来多大的好处，又增加了系统的复杂性。DFS不对文件进行分块存储，与支持文件分块存储的DFS相比，更加简洁高效，并且完全能满足绝大多数互联网应用的实际需要。

在客户端上传文件时，文件ID不是由客户端指定，而是由Storage生成后返回给客户端的。文件ID中包含了组名、文件相对路径和文件名，Storage节点可以根据文件ID直接定位到文件。因此集群中根本不需要存储文件索引信息。

系统结构如下图所示：



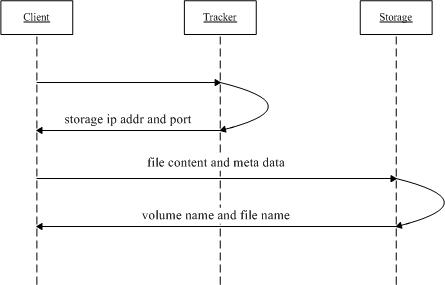
Tracker和Storage都可以由一台多台服务器构成。Tracker和Storage节点中的服务器均可以随时增加或下线而不会影响线上服务。其中Tracker中的所有服务器都是对等的，可以根据服务器的压力情况随时增加或减少。

为了支持大容量，存储节点（服务器）采用了分卷（或分组）的组织方式。存储系统由一个或多个卷组成，卷与卷之间的文件是相互独立的，所有卷的文件容量累加就是整个存储系统中的文件容量。一个卷可以由一台或多台存储服务器组成，一个卷下的存储服务器中的文件都是相同的，卷中的多台存储服务器起到了冗余备份和负载均衡的作用。

在卷中增加服务器时，同步已有的文件由系统自动完成，同步完成后，系统自动将新增服务器切换到线上提供服务。

当存储空间不足或即将耗尽时，可以动态添加卷。只需要增加一台或多台服务器，并将它们配置为一个新的卷，这样就扩大了存储系统的容量。

DFS中的文件标识分为两个部分：卷名和文件名，二者缺一不可。

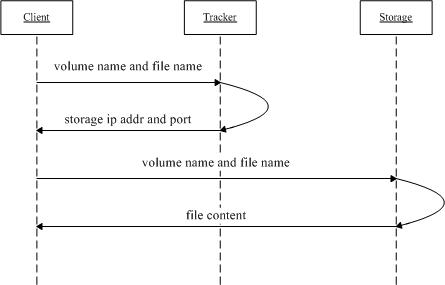


上传文件交互过程：

1. client询问tracker上传到的storage，不需要附加参数；

2. tracker返回一台可用的storage；

3. client直接和storage通讯完成文件上传。



下载文件交互过程：

1. client询问tracker下载文件的storage，参数为文件标识（卷名和文件名）；

2. tracker返回一台可用的storage；

3. client直接和storage通讯完成文件下载。