分布式文件系统调研报告

一、项目背景

1. 分布式文件系统概念
2. 分布式文件系统发展历史
3. 分布式文件系统的性能评测
4. FastDFS简介

二、立项依据

1. FastDFS 的优点
2. 去中心化
3. 动态负载均衡

三、前瞻性/重要性分析

四、相关工作

1. 去中心化
2. 动态负载均衡
3. 其他一些分布式文件系统

一、项目背景

1. 分布式文件系统概念

- 相对于本机端的文件系统而言,分布式文件系统(Distributed File System，DFS)是一种允许文件通过网络在多台主机上分享的文件系统。在这样的文件系统中,客户端并非直接访问底层的数据存储区块,而是通过网络,以特定的通信协议和服务器沟通。借由通信协议的设计,可以让客户端和服务器端都能根据访问控制清单或是授权,来限制对于文件系统的访问。相对地,在一个分享的磁盘文件系统中,所有节点对数据存储区块都有相同的访问权,在这样的系统中,访问权限就必须由客户端程序来控制。

2. 分布式文件系统发展历史

- 第一代分布式文件系统（1980年代）
  - 早期的分布式文档系统一般以提供标准接口的远程文档访问为目的，更多地关注访问的性能和数据的可靠性，以NFS和AFS(Andrew File System)最具代表性，他们对以后的文档系统设计也具备十分重要的影响。
  - NFS从1985年出现至今，已经历了四个版本的更新，被移植到了几乎任何主流的操作系统中，成为分布式文档系统事实上的标准。NFS利用Unix系统中的虚拟文档系统（Virtual File System，VFS）机制，将客户机对文档系统的请求，通过规范的文档访问协议和远程过程调用，转发到服务器端进行处理；服务器端在VFS之上，通过本地文档系统完成文档的处理，实现了全局的分布式文档系统。Sun公司公开了NFS的实施规范，互连网工程任务组（The Internet Engineering Task Force，IETF）将其列为征求意见稿（RFC-Request for Comments），这很大程度上促使NFS的很多设计实现方法成为标准，也促进了NFS的流行。NFS不断发展，在第四版中提供了基于租赁（Lease）的同步锁和基于会话（Session）语义的一致性等。
  - Carnegie Mellon大学在1983年设计研发的AFS将分布式文档系统的可扩展性放在了设计和实现的首要位置，并且着重考虑了在不安全的网络中实现安全访问的需求。因此，他在位置透明、用户迁移、和已有系统的兼容性等方面进行了特别设计。AFS具备很好的扩展性，能够很容易地支持数百个节点，甚至数千个节点的分布式环境。同时，在大规模的分布式文档系统中，AFS利用本地存储作为分布式文档的缓存，在远程文档无法访问时，依然能够部分工作，提高了系统可用性。后来的Coda File System、Inter-mezzo File System都受到AFS的影响，更加注重文档系统的高可用性（High Availability）和安全性，特别是Coda，在支持移动计算方面做了很多的研究工作。
  - 早期的分布式文档系统一般以提供标准接口的远程文档访问为目的，在受网络环境、本地磁盘、处理器速度等方面限制的情况下，更多地关注访问的性能和数据的可靠性。AFS在系统结构方面进行了有意义的探索。他们所采用的协议和相关技术，为后来的分布式文档系统设计提供了很多借鉴。
- 第二代分布式文档系统（1990～1995）
  - 20世纪90年代初，面对广域网和大容量存储应用的需求，借鉴当时先进的高性能对称多处理器的设计思想，加利福尼亚大学设计研发的xFS，克服了以前的分布式文档系统一般都运行在局域网（LAN）上的弱点，很好地解决了在广域网上进行缓存，以减少网

络流量的难题。他所采用的多层次结构很好地利用了文档系统的局部访问的特性，无效写回（Invalidation-based Write Back）缓存一致性协议，减少了网络负载。对本地主机和本地存储空间的有效利用，使他具备较好的性能。

  - Tiger Shark并行文档系统是针对大规模实时多媒体应用设计的。他采用了多种技术策略确保多媒体传输的实时性和稳定性：采用资源预留和优化的调度手段，确保数据实时访问性能；通过加大文档系统数据块的大小，最大限度地发挥磁盘的传输效率；通过将大文档分片存储在多个存储设备中，取得尽量大的并行吞吐率；通过复制文档系统元数据和文档数据，克服单点故障，提高系统可用性。
  - 基于虚拟共享磁盘Petal的Frangipani分布式文档系统，采用了一种新颖的系统结构—分层次的存储系统。Petal提供一个能够全局统一访问的磁盘空间。Frangipani基于Petal的特性提供文档系统的服务。这种分层结构使两者的设计实现都得到了简化。在Frangipani中，每个客户端也是文档系统服务器，参和文档系统的管理，能够平等地访问Petal提供的虚拟磁盘系统，并通过分布式锁实现同步访问控制。分层结构使系统具备很好的扩展性，能够在线动态地添加存储设备，增加新用户、备份等，同时系统具备很好的机制来处理节点失效、网络失效等故障，提高了系统的可用性。
  - Slice File System（SFS）考虑标准的NFS在容量、性能方面存在的限制，采用在客户机和服务器之间架设一个μproxy中间转发器，以提高性能和可扩展性。他将客户端的访问分为小文档、元数据服务、大文档数据三类请求。通过μproxy将前两种请求转发到不同的文档服务器上，将后者直接发送到存储服务器上。这样SFS系统就能够支持多个存储服务器，提高整个系统的容量和性能。μproxy根据请求内容的转发是静态的，对于整个系统中负载的变化难以做出及时反应。

- 第三代分布式文档系统（1995～2000）
  - 网络技术的发展和普及应用极大地推动了网络存储技术的发展，基于光纤通道的SAN、NAS得到了广泛应用。这也推动了分布式文档系统的研究。 在这个阶段，电脑技术和网络技术有了突飞猛进的发展，单位存储的成本大幅降低。而数据总线带宽、磁盘速度的增长无法满足应用对数据带宽的需求，存储子系统成为电脑系统发展的瓶颈。这个阶段，出现了多种体系结构，充分利用了网络技术。
  - 出现了多种分布式文档系统体系结构，如Global File System（GFS）、General Parallel File System （GPFS）、惠普的DiFFS、SGI公司的CXFS、EMC的HighRoad、Sun的qFS、XNFS等。数据容量、性能和共享的需求使得这一时期的分布式文档系统管理的系统规模更大、系统更复杂，对物理设备的直接访问、磁盘布局和检索效率的优化、元数据的集中管理等都反映了对性能和容量的追求。规模的扩展使得系统的动态性，如在线增减设备、缓存的一致性、系统可靠性的需求逐渐增强，更多的先进技术应用到系统实现中，如分布式锁、缓存管理技术、SoftUpdates技术、文档级的负载平衡等。
- 第四代分布式文档系统（2000年以后）
  - 随着SAN和NAS两种结构逐渐成熟，研究人员开始考虑如何将两种结构结合起来。网格的研究成果等也推动了分布式文档系统体系结构的发展。随着SAN和NAS两种体系结构逐渐成熟，研究人员开始考虑如何将两种体系结构结合起来，以充分利用两者的优势。另一方面，基于多种分布式文档系统的研究成果，人们对体系结构的认识不断深入，网格的研究成果等也推动了分布式文档系统体系结构的发展。这一时期，IBM的StorageTank、Cluster的Lustre、Panasas的PanFS、蓝鲸文档系统（BWFS）等是这种体系结构的代表。各种应用对存储系统提出了更多的需求：

        - 大容量：现在的数据量比以前任何时期更多，生成的速度更快；
        - 高性能：数据访问需要更高的带宽；
        - 高可用性：不但要确保数据的高可用性，还要确保服务的高可用性；
        - 可扩展性：应用在不断变化，系统规模也在不断变化，这就需要系统提供很好的扩展性，并在容量、性能、管理等方面都能适应应用的变化；
        - 可管理性：随着数据量的飞速增长，存储的规模越来越庞大，存储系统本身也越来越复杂，这给系统的管理、运行带来了很高的维护成本；
        - 按需服务：能够按照应用需求的不同提供不同的服务，如不同的应用、不同的客户端环境、不同的性能等。
    - 处于这个阶段的系统都在研究中，但从中也能够看出一些发展趋势：体系结构的研究逐渐成熟，表现在不同文档系统的体系结构趋于一致；系统设计的策略基本一致，如采用专用服务器方式等；每个系统在设计的细节上各自采用了很多特有的先进技术，也都取得了很好的性能和扩展性。另外，在协议方面的探索也是研究的热点之一，如Direct Access File System利用了远程内存直接访问的特性，借鉴了NFS第四版本和Common Internet File System等协议，设计了一套新的网络文档访问协议。

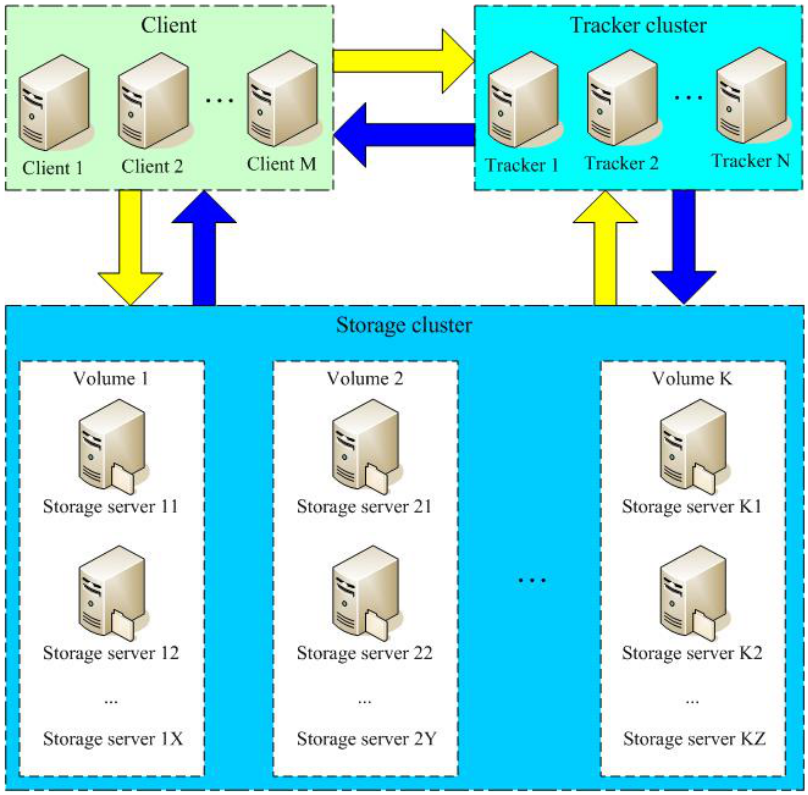
3. 分布式文件系统的性能评测

- 一个衡量分布式文件系统性能的方式是：它需要用多少时间来完成服务请求。在传统的系统中,完成请求所需要的时间包括了实际的硬盘访问时间，和一小部分的中央处理器处理时间。但在一个网络文件系统中，由于分布式架构的关系，远程访问动作会产生额外的经常性负担,包括:把请求从客户端送到服务器端的时间、把回应从服务器端传回客户端的时间、以及这两个传输过程中用来运行网络传输协议的中央处理处时间。

4. FastDFS简介

- FastDFS是一款开源的、分布式文件系统(Distributed File System)，由淘宝开发平台部资深架构师余庆开发。FastDFS由纯 C 实现,支持 Linux、 FreeBSD 等 UNIX 类 GoogleFS, 提供了 C、Java 和 PHP API 为互联网应用量身定做。作为一个分布式文件系统，它对文件进行管理，功能包括:文件存储、文件同步、文件访问(文件上传、文件下载)等，解决了大容量存储和负载均衡的问题，特别适合中小文件(建议范围:4KB < file_size <500MB)，对以文件为载体的在线服务，如相册网站、视频网站等等具有显著的效果。

图一:FastDFS架构



- FastDFS的架构设计: 由client、 storage server、tracker 三部分组成
  - Storage server:
    - Storage server以组(卷,group或volume)为单位组织,一个group内包含多台storage机器,数据互为备份,存储空间以group内容量最小的storage为准,所以建议group内的多个storage尽量配置相同,以免造成存储空间的浪费。
    - 以group为单位组织存储能方便的进行应用隔离、负载均衡、副本数定制(group内storage server数量即为该group的副本数,(即负载是静态的)),比如将不同应用数据存到不同的group就能隔离应用数据,同时还可根据应用的访问特性来将应用分配到不同的group来做负载均衡;缺点是group的容量受单机存储容量的限制,同时当group内有机器坏掉时,数据恢复只能依赖group内地其他机器,使得恢复时间会很长。
    - group内每个storage的存储依赖于本地文件系统,storage可配置多个数据存储目录,比如有10块磁盘,分别挂载在/data/disk1-/data/disk10,则可将这10个目录都配置为storage的数据存储目录。
    - storage接受到写文件请求时,会根据配置好的规则,选择其中一个存储目录来存储文件。为了避免单个目录下的文件数太多,在storage第一次启动时,会在每个数据存储目录里创建2级子目录,每级256个,总共65536个文件,新写的文件会以hash的方式被路由到其中某个子目录下,然后将文件数据直接作为一个本地文件存储到该目录中。
  - Tracker Server:
    - 跟踪服务器,主要做调度工作,起负载均衡的作用。在内存中记录集群中所有存储组和存储服务器的状态信息,是客户端和数据服务器交互的枢纽。Tracker 是 FastDFS 的协调者,负责管理所有的 storage server 和 group,每个 storage在启动

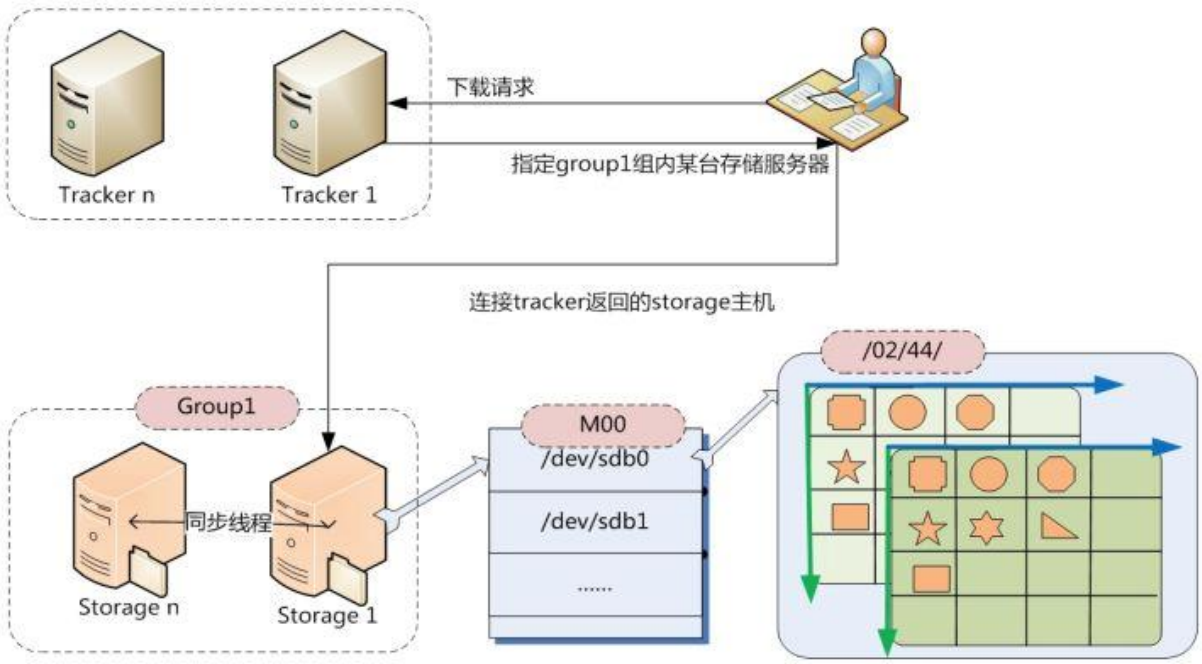
后会连接 Tracker,告知自己所属的 group 等信息,并保持周期性的心跳, tracker根据 storage 的信息,建立 group==>[storage server list]的映射表。

- Tracker 需要管理的元信息很少,会全部存储在内存中;另外 tracker 上的元信息都是由 storage 汇报的信息生成的,本身不需要持久化任何数据,这样使得 tracker非常容易扩展,直接增加 tracker 机器即可扩展为 tracker cluster 来服务,cluster 里每个tracker 之间是完全对等的,所有的 tracker 都接受 stroage 的信息,生成元数据信息来提供读写服务。

- Client:
  - 客户端,作为业务请求的发起方,通过专有接口,使用 TCP/IP 协议与跟踪器服务器或存储节点进行数据交互。FastDFS 向使用者提供基本文件访问接口,比如 upload、download、append、delete 等,以客户端库的方式提供给用户使用。

- FastDFS的上传过程:
  - 当Tracker收到客户端上传文件的请求时,会为该文件分配一个可以存储文件的group,当选定了group后就要决定给客户端分配group中的哪一个storage server。当分配好storage server后,客户端向storage发送写文件请求,storage将会为文件分配一个数据存储目录。然后为文件分配一个fileid,最后根据以上的信息生成文件名存储文件。

图二：FastDFS数据传输流程



- FastDFS的文件同步
  - 写文件时,客户端将文件写至group内一个storage server即认为写文件成功,storage server写完文件后,会由后台线程将文件同步至同group内其他的storage server。
  - 每个storage写文件后,同时会写一份binlog,binlog里不包含文件数据,只包含文件名等元信息,这份binlog用于后台同步,storage会记录向group内其他storage同步的进度,以便重启后能接上次的进度继续同步;进度以时间戳的方式进行记录,所以最好能保证集群内所有server的时钟保持同步。
  - storage的同步进度会作为元数据的一部分汇报到tracker上,tracke在选择读storage的时

候会以同步进度作为参考。
    - 比如一个group内有A、B、C三个storage server,A向C同步到进度为T1 (T1以前写的文件都已经同步到B上了),B向C同步到时间戳为T2(T2 > T1),tracker接收到这些同步进度信息时,就会进行整理,将最小的那个做为C的同步时间戳,本例中T1即为C的同步时间戳为T1(即所有T1以前写的数据都已经同步到C上了);同理,根据上述规则,tracker会为A、B生成一个同步时间戳。

二、立项依据

1. FastDFS 的优点

- 对机器的配置要求低

  FastDFS无论是上传下载还是同步恢复,对机器的配置要求都是非常低的,这就使得使用FastDFS来提供分布式文件存储的成本很低。

- 上传速度快

  由于使用的是弱一致性模型,上传后不需要强写副本再返回,故上传速度相对于采用强一致性模型的分布式文件系统要快得多。

- 高可靠性

  FastDFS作为分布式文件系统,其冗余为文件系统提供了极高的可靠度,即使有单个节点故障,整个系统仍能正常工作,并对故障节点的数据进行恢复。

- 应用场景

  FastDFS是为互联网应用量身定做的一套分布式文件存储系统，非常适合用来存储用户图片、视频、文档等文件。对于互联网应用，和其他分布式文件系统相比，优势非常明显。

- 用户量

  截止2012年底至少有25家公司在使用FastDFS，其中有好几家是做网盘的公司。其中存储量最大的一家，集群中存储group数有400个，存储服务器超过800台，存储容量达到6PB，文件数超过1亿。以下是部分使用Fast DFS的用户列表：

    - 支付宝（http://www.alipay.com/）
    - 京东商城（http://www.360buy.com/）
    - 淘淘搜（http://www.taotaosou.com/）
    - 飞信（http://feixin.1008**/）
    - 赶集网（http://www.ganji.com/）
    - 淘米网（http://www.61.com/）
    - 迅雷（http://www.xunlei.com/）
    - 蚂蜂窝（http://www.mafengwo.cn/）

- 易于修改

  FastDFS由纯C语言实现，在阅读其源码时，在语言层面上不存在太大的障碍。相比于其他分布式文件系统，其5.2万行的代码量减轻了在阅读源码时的负担

2. 去中心化

- FastDFS可以通过增加冗余的tracker server解决单点故障的问题，但是多个tracker server就意味着storage server需要同时向每个tracker server报告自身的磁盘剩余空间、文件同步状况、文件上传下载次数等状态信息；而这些任务的完成就要求每个storage server都要在后台运行着一个单独的线程来完成对tracker server的连接和定时报告。这种机制必定意味着资源的浪费。
- 对于分布式文件系统,去中心化的存储可以使网络变得稳健且有弹性。将数据集中或者把文件系统的核心集中在某一部位会使得系统安全性降低。

3. 动态负载均衡

- 负载均衡算法对分布式文件系统的高伸缩性和高容错性起到至关重要的作用，在日益增加的数据量面前，设计一个适用的负载均衡算法显得尤为重要。负载均衡调度的主要依据是各个服务器自身的实时负载和当前任务所带来的负载量，同时还要考虑策略本身对服务器造成的额外开销。目前使用较多的经典负载均衡算法根据调度策略的不同可以分成静态和动态两种。静态负载均衡算法实现简单、使用方便、占用较少的系统资源，但是没有能够实时的反映出节点服务器的真实负载情况。动态负载均衡算法虽然能够有效的反映出节点服务器的真实负载状况，但其实现较复杂、占用较大的系统资源、节点间通信开销较大。用最小的开销换取节点服务器的最真实负载状态，是负载均衡算法的理想化状态，采用何种方式让服务器用较小的代价换来节点服务器较真实的负载，作为节点服务器负载分配的依据，让服务器能够长时间运行且不出现负载倾斜的现象，正是负载均衡技术研究的方向所在。
- Fast DFS现有的负载均衡算法属于静态负载算法，该算法为每个存储组定义了一个评价值Vi，Vi的计算方法是：Vi = Vi0 + Ti / Tmax (Vi0是上一层该组的Vi值，Ti为该组内各storage server的存储空间之和，Tmax是所有组中Ti的最大值)。Tracker server在选择存储组的时，总是选择Vi最高的两个存储组。而Vi显然与该存储组当前的空间利用率、性能、连接数无关，这就意味着总存储空间大但是当前负载很重，连接数很大的storage server可能被选中。
- 随着互联网的蓬勃发展，需要的存储空间越来越多，并发量越来越大，FastDFS原有的静态负载均衡策略难以适应高并发下的线性扩容。目前的FastDFS发布的各个版本的负载均衡策略，均采用的是静态的负载均衡策略，在扩容时会因为没有考虑节点的实时负载，使存储服务器在一段时间内出现负载倾斜的情况，为了适应数据量急剧增加的互联网应用，寻找更适合的负载均衡策略迫在眉睫。

三、前瞻性/重要性分析

- 在长远的发展来看，文件系统使用的集群会变得非常庞大，这个集群它不再是某个大型数据中心，也可能会是很多个数据中心的集合体，而这些数据中心也会分布在不同的区域范围内。未来集群不能再是外部数据中心那么简单唯一，甚至最终有可能的是通过互联网来建立

“唯一”的一个集群。而当下使用一台服务器是不行的，会阻碍发展的需求。如果集群的范围变小，结果将会是集群特别不稳定，影响用户对数据的需求，遏制了集群的快速发展。所以分布式文件系统的发展前景很广阔。在大规模使用分布式文件系统后，它的安全性、性能等因素将会是研究的重点。故我们对以FastDFS为代表的分布式文件系统的负载均衡和去中心化研究很有意义。

四、相关工作

1. 去中心化

- DHT技术：
  - DHT全称叫分布式哈希表(Distributed Hash Table)，是一种分布式存储方法。它提供的服务类似于hash表，键值对存储在DHT中，任何参与该结构的节点能高效的由键查询到数据值。这种键值对的匹配是分布式的分配到各个节点的，这样节点的增加与删减只会带来比较小的系统扰动。这样也可以有效地避免“中央集权式”的服务器（比如：tracker）的单一故障而带来的整个网络瘫痪。
  - DHT技术本质上强调以下特性：
    - 离散性：构成系统的节点并没有任何中央式的协调机制。
    - 伸缩性：即使有成千上万个节点，系统仍然应该十分有效率。
    - 容错性：即使节点不断地加入、离开或是停止工作，系统仍然必须达到一定的可靠度。
  - 其关键技术为：任一个节点只需要与系统中的部分节点（通常为O(logN)个）沟通，当成员改变的时候，只有一部分的工作（例如数据或键的发送，哈希表的改变等）必须要完成。基本上，DHT技术就是一种映射key和节点的算法以及路由的算法。
  - DHT的结构：关键值空间分区(keyspace partitioning)和延展网络(overlay network)
    - 关键值空间分区是指每一个节点掌管部分键空间。
    - 延展网络是指一个连接各个节点的抽象网络，它能使每个节点找到拥有特定键的节点。每个节点或者存储了该键，或者储存有离这个键更近（这个距离由具体算法定义）的节点链接。
  - 当这些组件都准备好后,一般使用分布式散列表来存储与读取的方式如下所述。假设关键值空间是一个 160 位长的字符串集合。为了在分布式散列表中存储一个文件,名称为 filename 且内容 data,我们计算 filename 的 SHA1 散列值（一个 160 位的关键值k）并将消息 put(k,data)送给分布式散列表中的任意参与节点。此消息在延展网络中被转送,直到抵达在关键值空间分区中被指定负责存储关键值 k 的节点。而 (k,data)即存储在该节点。其他的节点只需要重新计算 filename 的散列值 k,然后提交消息 get(k)给分布式散列表中的任意参与节点,以此来找与 k 相关的数据。此消息也会在延展网络中被转送到负责存储 k 的节点。而此节点则会负责传回存储的数据 data。
  - 优点
    - 平衡性：平衡性是指哈希的结果能够尽可能分布到所有的缓冲中去,这样可以使得所有的缓冲空间都得到利用。很多哈希算法都能够满足这一条件。
    - 单调性：单调性是指如果已经有一些内容通过哈希分派到了相应的缓冲中,又有新的缓冲加入到系统中。哈希的结果应能够保证原有已分配的内容可以被映射到

原有的或者新的缓冲中去,而不会被映射到旧的缓冲集合中的其他缓冲区。
        - 分散性：在分布式环境中,终端有可能看不到所有的缓冲,而是只能看到其中的一部分。当终端希望通过哈希过程将内容映射到缓冲上时,由于不同终端所见的缓冲范围有可能不同,从而导致哈希的结果不一致,最终的结果是相同的内容被不同的终端映射到不同的缓冲区中。这种情况显然是应该避免的,因为它导致相同内容被存储到不同缓冲中去,降低了系统存储的效率。分散性的定义就是上述情况发生的严重程度。好的哈希算法应能够尽量避免不一致的情况发生,也就是尽量降低分散性。
        - 负载：负载问题实际上是从另一个角度看待分散性问题。既然不同的终端可能将相同的内容映射到不同的缓冲区中,那么对于一个特定的缓冲区而言,也可能被不同的用户映射为不同 的内容。与分散性一样,这种情况也是应当避免的,因此好的哈希算法应能够尽量降低缓冲的负荷。
        - 平滑性：平滑性是指缓存服务器的数目平滑改变和缓存对象的平滑改变是一致的。
    - DHT实现算法有Chord, Pastry, Kademlia等。

2. 动态负载均衡

- 在P2P网络中进行负载均衡的技术。
    - 动态副本法：主要针对由于资源访问热度不均而引起的网络负载不均问题。算法的核心思想是通过增加“热”文件副本数量，将客户端连接分散到多个节点中，实现 “热”点降温。不同动态副本算法之间的主要区别在于如何选择副本放置节点。
    - 虚拟节点算法：主要针对因服务器配置不同而引起的存储负载不均问题。设计思想是通过虚拟节点而不是实际物理节点进行数据的储存和路由路径上的文件查询转发。算法将配置不同的服务器虚拟成配置一致的虚拟节点，并随机分布在哈希环上。每一个实际的服务器负责多个虚拟节点。当有节点负载过重时，将负载过重的物理节点的虚拟节点迁移到负载较轻的物理节点上。
    - 动态路由表：针对资源查找路由产生的负载。在P2P系统中，拥有热点文件的节点会承担大量的文件查询请求，以及查询路由路径上的节点也会承担相应的请求消息路由负载。
- 一些基本的动态负载均衡算法包括: 最少连接数,最快响应速度,观察方法,预测法,动态性能分配,动态服务器补充,服务质量,服务类型,规则模式等。
    - 最少的连接方式(Least Connection):传递新的连接给那些进行最少连接处理的服务器。当其中某个服务器发生第二到第 7 层的故障, BIG-IP 就把其从服务器队列中拿出,不参加下一次的用户请求的分配, 直到其恢复正常。
    - 最快模式(Fastest):传递连接给那些响应最快的服务器。当其中某个服务器发生第二到第 7 层的故障,BIG-IP 就把其从服务器队列中拿出,不参加下一次的用户请求的分配,直到其恢复正常。
    - 观察模式(Observed):连接数目和响应时间以这两项的最佳平衡为依据为新的请求选择服务器。当其中某个服务器发生第二到第 7 层的故障,BIG-IP 就把其从服务器队列中拿出,不参加下一次的用户请求的分配,直到其恢复正常。
    - 预测模式(Predictive): BIG-IP 利用收集到的服务器当前的性能指标,进行预测分析,选择一

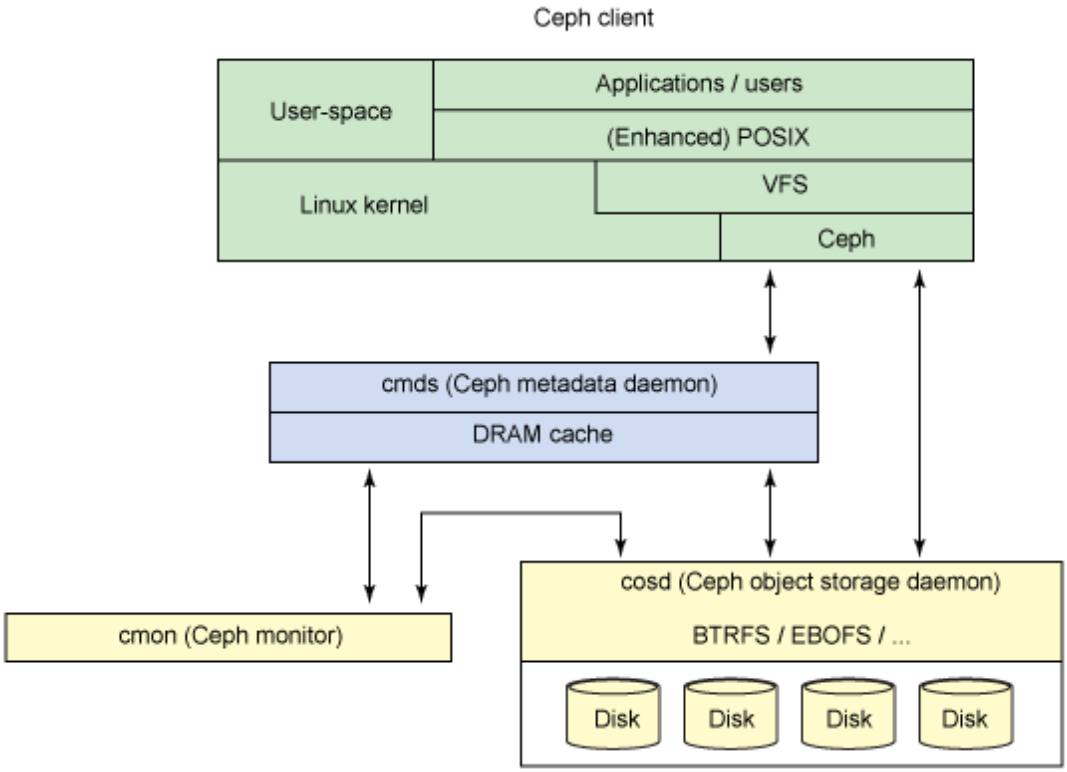
台服务器在下一个时间片内,其性能将达到最佳的服务器相应用户的请求。(被BIG-IP 进行检测)

- 动态性能分配(Dynamic Ratio-APM):BIG-IP 收集到的应用程序和应用服务器的各项性能参数,动态调整流量分配。
- 动态服务器补充(Dynamic Server Act.):当主服务器群中因故障导致数量减少时,动态地将备份服务器补充至主服务器群。
- 服务质量(QoS):按不同的优先级对数据流进行分配。
- 服务类型(ToS): 按不同的服务类型(在 Type of Field 中标识)负载均衡对数据流进行分配。
- 规则模式:针对不同的数据流设置导向规则,用户可自行。

3. 其他一些分布式文件系统

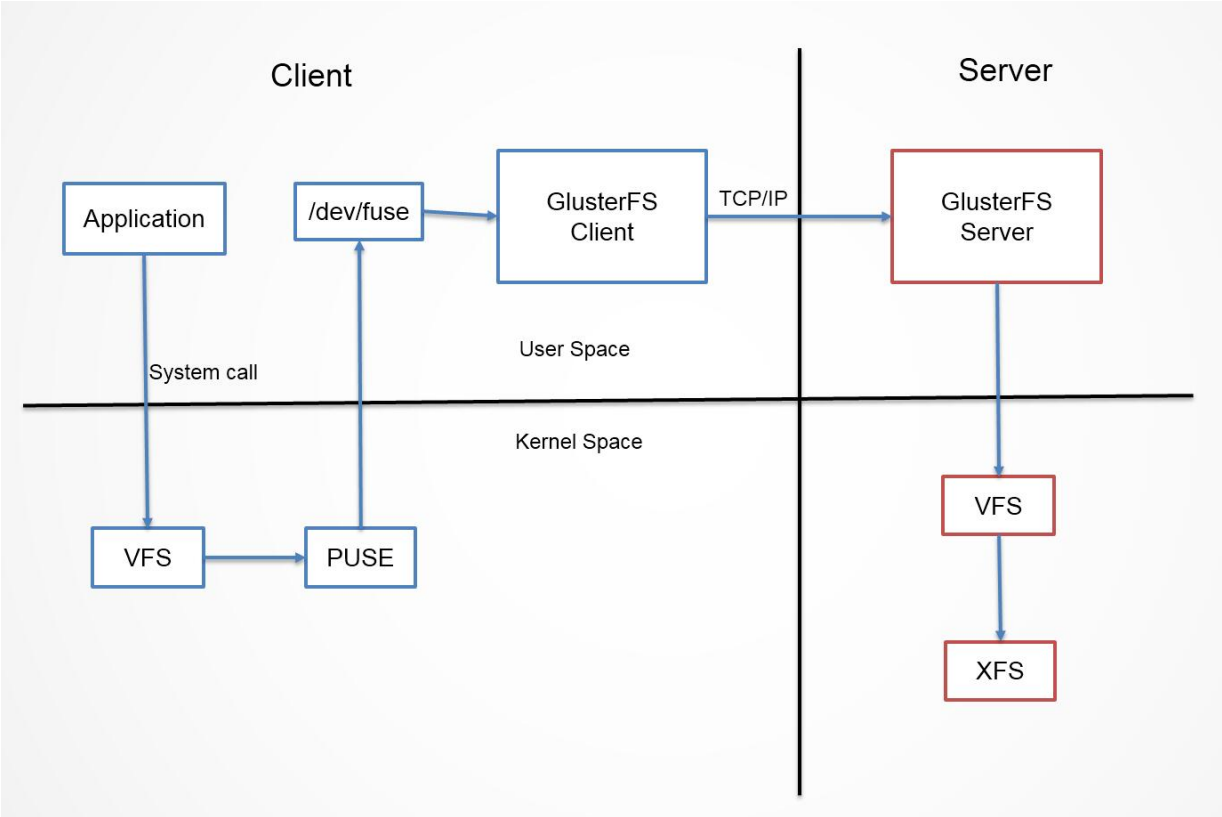
- ceph：
  - Ceph分布数据的过程：首先计算数据x的Hash值并将结果和PG（分区）数目取余，以得到数据x对应的PG编号。然后，通过CRUSH算法将PG映射到一组OSD中。最后把数据x存放到PG对应的OSD中。这个过程中包含了两次映射，第一次是数据x到PG的映射。如果把PG当作存储节点，那么这和文章开头提到的普通Hash算法一样。不同的是，PG是抽象的存储节点，它不会随着物理节点的加入或则离开而增加或减少，因此数据到PG的映射是稳定的。
  - Ceph文件系统拥有优秀的性能、高可靠性和高可扩展性这三大特性。秀的性能是指数据能够在各个节点上进行均衡地分布；高可靠性表示在Ceph文件系统中没有单点故障，并且存储在系统中的数据能够保证尽可能的不丢失；高可扩展性即Ceph系统易于扩展，能够很容易实现从TB到PB级别的扩展。

图三：简单的 Ceph 生态系统



- glusterfs：
  - GlusterFS采用独特的无中心对称式架构，与其他有中心的分布式文件系统相比，它没有专用的元数据服务集群。在文件定位的问题上，GlusterFS使用DHT算法进行文件定位，集群中的任何服务器和客户端只需根据路径和文件名就可以对数据进行定位和读写访问。换句话说，GlusterFS不需要将元数据与数据进行分离，因为文件定位可独立并行化进行。
  - 数据访问流程：
    1. 使用Davies-Meyer算法计算32位hash值，输入参数为文件名
    2. 根据hash值在集群中选择子卷（存储服务器），进行文件定位
    3. 对所选择的子卷进行数据访问。
  - 存在的问题：GlusterFS目前主要适用大文件存储场景，对于小文件尤其是海量小文件，存储效率和访问性能都表现不佳。
  - 优势：使用弹性哈希算法，从而获得了接近线性的高扩展性，同时也提高了系统性能和可靠性。

图四：glusterfs客户端访问流程



- FastDHT
  - FastDHT 是一个高性能的分布式哈希系统 (DHT) ,使用 Berkeley DB 做数据存储,使用libevent做网络IO处理,提供 Java 版的客户端接口包。适合用来存储用户在线、会话等小数据量信息。
  - FastDHT存储Key Value Pair支持两种存储方式:缓存方式的MPOOL和持久存储方式的BDB。Key包括三部分:Namespace,ObjectID和Key。Key可设置过期时间,自动清除过期数据。 Server端划分group,同group数据互相备份,并且可自动压缩binlog.服务端可使用单线程,多线程模式。
  - FastDHT一些特性:
    - 虚拟farm,便于扩容;
    - 分布式算法client端实现,不需要中心服务器;
    - 二进制通信协议,支持Proxy;
    - 使用libevent,异步IO方式,支持大并发;
    - 自动failover;
    - 支持长连接。
  - FastDHT集群由一个或多个组(group)组成,每个组由一台或多台服务器组成,同组服务器上存储的数据是相同的,数据同步只在同组的服务器之间进行。组内各个服务器是对等的,对数据进行存取时,可以根据key的hash值来决定使用哪台服务器。数据同步采用推(Push)的方式,由源服务器主动将数据同步到本组的其他服务器。FastDHT集群由一个或多个组(group)组成,每个组由一台或多台服务器组成,同组服务器上存储的数据是相同的,数据同步只在同组的服务器之间进行。组内各个服务器是对等的,对数据进行存取时,可以根据key的hash值来决定使用哪台服务器。数据同步采用推(Push)的方式,由源服务器

主动将数据同步到本组的其他服务器。

- Lustre：
    - Lustre分布式文件系统是1999年卡内基梅隆大学的Peter J.Braam所发起的研究项目，Lustre是典型的元数据中心式的分布式文件系统，MDS（元数据服务器集群）用于存储文件的元数据信息，OSS（对象存储服务器）用于存储实际的文件数据。
    - Luster支持文件整体存储和文件分片两种存储方式。整体存储就是将文件整体存储在OST（对象存储目的地）中；而分片存储就是将文件分块存储在子目录Stripe下，Stripe有size（文件分片大小），ost（起始存储位置），和count（分片数量）三个参数。
    - Lustre的优点是可以设置具体子目录Stripe的参数，从而灵活的存储不同大小的文件，相比于单一的整体存储方式更为高效。但是Lustre不提供数据保护，也存在着元数据存储上单点问题，这两点决定了Lustre在安全性和容错性上的短板。
- HDFS（Hadoop Distributed File System）
    - HDFS是为Hadoop架构所设计的分布可扩展轻量级分布式文件系统，其由负责元数据存储的Namenode和负责存储实际数据的Datanodes组成。Namenode是一个中心服务器，负责管理文件系统的名称空间、文件数据快的组织以及客户端对数据的访问；而Datanodes就是一个个存储实际数据的节点。
    - HDFS只支持文件的分块存储，其将每个文件默认分为若干个64MB的数据块，集中存储在一个Datanode上。在存储某个文件的时，HDFS会为其创建另一个Datanode以存储其副本，从而在一定程度上保证了数据的安全性。
    - HDFS本身在性能等方面没有很大的优势，其优势在于以其为基础的一系列衍生架构，配合其衍生架构，大多数人将其用于数据挖掘数据分析等领域。
- GFS（Google File System）
    - GFS是Google公司为了存储海量搜索数据而设计的专用文件系统，其最为鲜明的特点是独特的容错机制以及自动负载均衡技术。GFS系统中包括一个独立的GFS Master控制服务器，多台GFS Chunkserver数据块服务器以及多个GFS Client客户端。
    - 和HDFS一样，上传到GFS中的文件都被分割成固定大小的Chunk，每一个Chunk在不同的Chunkserver中保存多份。Chunk数据块用以存储Chunk数据，并根据指定的Chunk标识符和字节范围来读写块数据。
    - GFS Master存储Chunk的元数据，接受来自各个Chunkserver的周期性信息，并向各个Chunkserver发送指令
    - 在GFS中，GFS服务器生成诊断日志用于记录许多关键事件（如Chunkserver的启动和关闭）以及所有的远程调用的请求和相应。在空间允许的情况下，GFS会尽量保存这些日志。在系统发生故障后，通过对远程调用日志的分析，可以重建交互历史从而诊断出错误。