参考：<http://blog.chinaunix.net/uid-20196318-id-3076416.html>

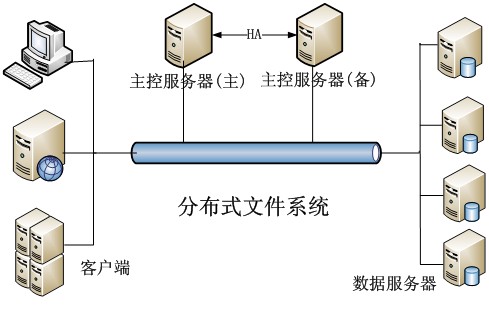
[分布式文件系统：原理、问题与方法](http://blog.chinaunix.net/uid-20196318-id-3076416.html) 2012-02-18 14:26:35

分类： 服务器与存储

本地文件系统如ext3，reiserfs等（这里不讨论基于内存的文件系统），它们管理本地的磁盘存储资源、提供文件到存储位置的映射，并抽象出一套文件访问接口供用户使用。但随着互联网企业的高速发展，这些企业对数据存储的要求越来越高，而且模式各异，如淘宝主站的大量商品图片，其特点是文件较小，但数量巨大；而类似于youtube，优酷这样的视频服务网站，其后台存储着大量的视频文件，尺寸大多在数十兆到数吉字节不等。这些应用场景都是传统文件系统不能解决的。分布式文件系统将数据存储在物理上分散的多个存储节点上，对这些节点的资源进行统一的管理与分配，并向用户提供文件系统访问接口，其主要解决了本地文件系统在文件大小、文件数量、打开文件数等的限制问题。

**典型架构**

目前比较主流的一种分布式文件系统架构，如下图所示，通常包括主控服务器（或称元数据服务器、名字服务器等（me：类似nameserver），通常会配置备用主控服务器以便在故障时接管服务，也可以两个都为主的模式），多个数据服务器（或称存储服务器，存储节点等），以及多个客户端，客户端可以是各种应用服务器，也可以是终端用户。



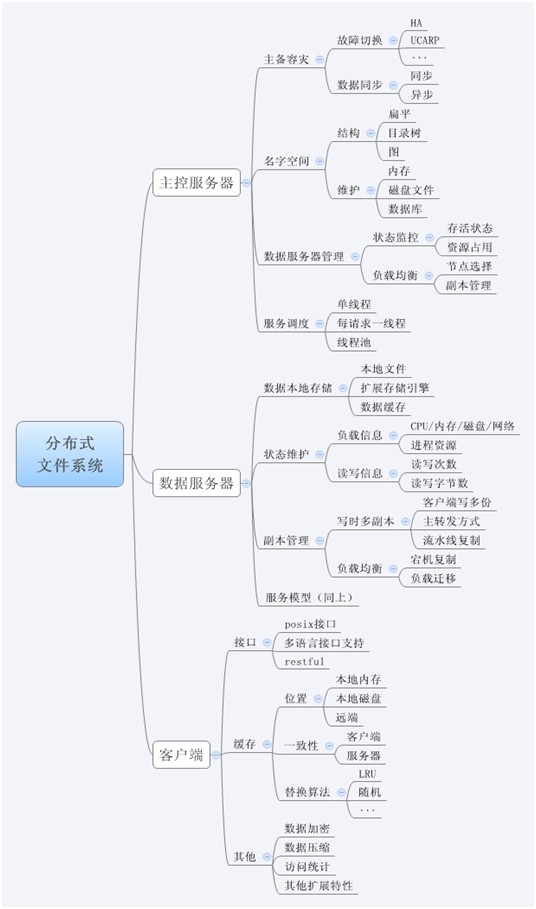
分布式文件系统的数据存储解决方案，归根结底是将将大问题划分为小问题（me：分而治之）。大量的文件，均匀分布到多个数据服务器上后，每个数据服务器存储的文件数量就少了，另外通过使用大文件存储多个小文件的方式，总能把单个数据服务器上存储的文件数降到单机能解决的规模；对于很大的文件，将大文件划分成多个相对较小的片段，存储在多个数据服务器上（me：大变小，小变大）（目前，很多本地文件系统对超大文件的支持已经不存在问题了，如ext3文件系统使用4k块时，文件最大能到4T，ext4则能支持更大的文件，只是受限于磁盘的存储空间）。

理论上，分布式文件系统可以只有客户端和多个数据服务器组成，客户端根据文件名决定将文件存储到哪个数据服务器，但一旦有数据服务器失效时，问题就变得复杂，客户端并不知道数据服务器宕机的消息，仍然连接它进行数据存取，导致整个系统的可靠性极大的降低，而且完全有客户端决定数据分配时非常不灵活的，其不能根据文件特性制定不同的分布策略。

于是，我们迫切的需要能知道各个数据服务器的服务状态，数据服务器的状态管理可分为分散式和集中式两种方式，前者是让多个数据服务器相互管理，如每个服务器向其他所有的服务器发送心跳信息，但这种方式开销较大，控制不好容易影响到正常的数据服务，而且工程实现较为复杂；后者是指通过一个独立的服务器（如上图中的主控服务器）来管理数据服务器，每个服务器向其汇报服务状态来达到集中管理的目的，这种方式简单易实现，目前很多分布式文件系统都采用这种方式如GFS、TFS(<http://code.taobao.org/p/tfs/wiki/index/> )、MooseFS (<http://www.moosefs.org/> )等。主控服务器在负载较大时会出现单点，较多的解决方案是配置备用服务器，以便在故障时接管服务，如果需要，主备之间需要进行数据的同步。

**问题及解决方法**

本文主要讨论基于上图架构的分布式文件系统的相关原理，工程实现时需要解决的问题和解决问题的基本方法，分布式文件系统涉及的主要问题及解决方法如下图所示。为方便描述以下主控服务器简称Master，数据服务器简称DS（DataServer）。



**主控服务器**

**命名空间的维护**

Master负责维护整个文件系统的命名空间，并暴露给用户使用，命名空间的结构主要有典型目录树结构（me:即有数据节点文件的存储目录一致）如MooseFS等，扁平化结构（me：即k-v的格式存储）如淘宝TFS（目前已提供目录树结构支持），图结构（主要面向终端用户，方便用户根据文件关联性组织文件，只在论文中看到过）。

为了维护名字空间，需要存储一些辅助的元数据如文件（块）到数据服务器的映射关系，文件之间的关系等，为了提升效率，很多文件系统采取将元数据全部内存化（元数据通常较小）的方式如GFS, TFS；有些系统则借助数据库来存储元数据如DBFS，还有些系统则采用本地文件来存储元数据如MooseFS。

一种简单的实现目录树结构的方式是，在Master上存储与客户端完全一样的命名空间，对应的文件内容为该文件的元数据，并通过在Master上采用ReiserFS来进行小文件存储优化，对于大文件的存储（文件数量不会成为Master的瓶颈），这种方式简单易实现。曾经参与的DNFS系统的开发就是使用这种方式，DNFS主要用于存储视频文件，视频数量在百万级别，Master采用这种方式文件数量上不会成为瓶颈。

**数据服务器管理**

除了维护文件系统的命名空间，Master还需要集中管理数据DS，可通过轮询DS或由DS报告心跳的方式实现。在接收到客户端写请求时，Master需要根据各个DS的负载等信息选择一组（根据系统配置的副本数）DS为其服务；当Master发现有DS宕机时，需要对一些副本数不足的文件（块）执行复制计划；当有新的DS加入集群或是某个DS上负载过高，Master也可根据需要执行一些副本迁移计划。

如果Master的元数据存储是非持久化的，则在DS启动时还需要把自己的文件（块）信息汇报给Master。在分配DS时，基本的分配方法有随机选取，RR轮转、低负载优先等，还可以将服务器的部署作为参考（如HDFS分配的策略），也可以根据客户端的信息，将分配的DS按照与客户端的远近排序，使得客户端优先选取离自己近的DS进行数据存取.

**服务调度**

Master最终的目的还是要服务好客户端的请求，除了一些周期性线程任务外，Master需要服务来自客户端和DS的请求，通常的服务模型包括单线程、每请求一线程、线程池（通常配合任务队列）。单线程模型下，Master只能顺序的服务请求，该方式效率低，不能充分利用好系统资源；每请求一线程的方式虽能并发的处理请求，但由于系统资源的限制，导致创建线程数存在限制，从而限制同时服务的请求数量，另外，线程太多，线程间的调度效率也是个大问题；线程池的方式目前使用较多，通常由单独的线程接受请求，并将其加入到任务队列中，而线程池中的线程则从任务队列中不断的取出任务进行处理。

**主备（主）容灾**

Master在整个分布式文件系统中的作用非常重要，其维护文件（块）到DS的映射、管理所有的DS状态并在某些条件触发时执行负载均衡计划等。为了避免Master的单点问题，通常会为其配置备用服务器，以保证在主控服务器节点失效时接管其工作。通常的实现方式是通过HA、UCARP等软件为主备服务器提供一个虚拟IP提供服务，当备用服务器检测到主宕机时，会接管主的资源及服务。

如果Master需要持久化一些数据，则需要将数据同步到备用Master，对于元数据内存化的情况，为了加速元数据的构建，有时也需将主上的操作同步到备Master。处理方式可分为同步和异步两种。同步方式将每次请求同步转发至备Master，这样理论上主备时刻保持一致的状态，但这种方式会增加客户端的响应延迟（在客户端对响应延迟要求不高时可使用这种方式），当备Master宕机时，可采取不做任何处理，等备Master起来后再同步数据，或是暂时停止写服务，管理员介入启动备Master再正常服务（需业务能容忍）；异步方式则是先暂存客户端的请求信息（如追加至操作日志），后台线程重放日志到备Master，这种方式会使得主备的数据存在不一致的情况，具体策略需针对需求制定。

**数据服务器**

**数据本地存储**

数据服务器负责文件数据在本地的持久化存储，最简单的方式是将客户每个文件数据分配到一个单独的DS上作为一个本地文件存储，但这种方式并不能很好的利用分布式文件系统的特性，很多文件系统使用固定大小的块来存储数据如GFS, TFS, HDFS，典型的块大小为64M。

对于小文件的存储，可以将多个文件的数据存储在一个块中，并为块内的文件建立索引，这样可以极大的提高存储空间利用率。Facebook用于存储照片的HayStack系统的本地存储方式为，将多个图片对象存储在一个大文件中，并为每个文件的存储位置建立索引，其支持文件的创建和删除，不支持更新（通过删除和创建完成），新创建的图片追加到大文件的末尾并更新索引，文件删除时，简单的设置文件头的删除标记，系统在空闲时会对大文件进行compact把设置删除标记且超过一定时限的文件存储空间回收（延迟删除策略）。淘宝的TFS系统采用了类似的方式，对小文件的存储进行了优化，TFS使用扩展块的方式支持文件的更新。对小文件的存储也可直接借助一些开源的KV存储解决方案，如Tokyo Cabinet（HDB, FDB, BDB, TDB）、Redis等。

对于大文件的存储，则可将文件存储到多个块上，多个块所在的DS可以并行服务，这种需求通常不需要对本地存储做太多优化。

**状态维护**

DS除了简单的存储数据外，还需要维护一些状态，首先它需要将自己的状态以心跳包的方式周期性的报告给Master，使得Master知道自己是否正常工作，通常心跳包中还会包含DS当前的负载状况（CPU、内存、磁盘IO、磁盘存储空间、网络IO等、进程资源，视具体需求而定），这些信息可以帮助Master更好的制定负载均衡策略。

很多分布式文件系统如HDFS在外围提供一套监控系统，可以实时的获取DS或Master的负载状况，管理员可根据监控信息进行故障预防。

**副本管理**

为了保证数据的安全性，分布式文件系统中的文件会存储多个副本到DS上，写多个副本的方式，主要分为3种。最简单的方式是客户端分别向多个DS写同一份数据，如DNFS采用这种方式；第2种方式是客户端向主DS写数据，主DS向其他DS转发数据，如TFS采用这种方式；第三种方式采用流水复制的方式，client向某个DS写数据，该DS向副本链中下一个DS转发数据，依次类推，如HDFS、GFS采取这种方式。

当有节点宕机或节点间负载极不均匀的情况下，Master会制定一些副本复制或迁移计划，而DS实际执行这些计划，将副本转发或迁移至其他的DS。DS也可提供管理工具，在需要的情况下由管理员手动的执行一些复制或迁移计划。

**服务模型**

参考**主控服务器::服务模型**一节

**客户端**

**接口**

用户最终通过文件系统提供的接口来存取数据，linux环境下，最好莫过于能提供POSIX接口的支持，这样很多应用（各种语言皆可，最终都是系统调用）能不加修改的将本地文件存储替换为分布式文件存储。

要想文件系统支持POSIX接口，一种方式时按照VFS接口规范实现文件系统，这种方式需要文件系统开发者对内核有一定的了解；另一种方式是借助FUSE([http://fuse.sourceforge.net](http://fuse.sourceforge.net/))软件，在用户态实现文件系统并能支持POSIX接口，但是用该软件包开发的文件系统会有额外的用户态内核态的切换、数据拷贝过程，从而导致其效率不高。很多文件系统的开发借助了fuse，参考<http://sourceforge.net/apps/mediawiki/fuse/index.php?title=FileSystems>。

如果不能支持POSIX接口，则为了支持不同语言的开发者，需要提供多种语言的客户端支持，如常用的C/C++、java、php、python客户端。使用客户端的方式较难处理的一种情况时，当客户端升级时，使用客户端接口的应用要使用新的功能，也需要进行升级，当应用较多时，升级过程非常麻烦。目前一种趋势是提供Restful接口的支持，使用http协议的方式给应用（用户）访问文件资源，这样就避免功能升级带来的问题。

另外，在客户端接口的支持上，也需根据系统需求权衡，比如write接口，在分布式实现上较麻烦，很难解决数据一致性的问题，应该考虑能否只支持create（update通过delete和create组合实现），或折中支持append，以降低系统的复杂性。

**缓存**

分布式文件系统的文件存取，要求客户端先连接Master获取一些用于文件访问的元信息，这一过程一方面加重了Master的负担，一方面增加了客户端的请求的响应延迟。为了加速该过程，同时减小Master的负担，可将元信息进行缓存，数据可根据业务特性缓存在本地内存或磁盘，也可缓存在远端的cache系统上如淘宝的TFS可利用tair作为缓存（减小Master负担、降低客户端资源占用）。

维护缓存需考虑如何解决一致性问题及缓存替换算法，一致性的维护可由客户端也可由服务器完成，一种方式是客户端周期性的使cache失效或检查cache有效性（需业务上能容忍），或由服务器在元数据更新后通知客户端使cache失效（需维护客户端状态）。使用得较多的替换算法如LRU、随机替换等。

**其他**

客户端还可以根据需要支持一些扩展特性，如将数据进行加密保证数据的安全性、将数据进行压缩后存储降低存储空间使用，或是在接口中封装一些访问统计行为，以支持系统对应用的行为进行监控和统计。

**总结**

本文主要从典型分布式文件系统架构出发，讨论了分布式文件系统的基本原理，工程实现时需要解决的问题、以及解决问题的基本方法，真正在系统工程实现时，要考虑的问题会更多。如有问题，欢迎拍砖。