分布式文件系统架构对比

由苏锐 (https://juicefs.io/blog/cn/author/rsu/)于2018年7月1日发表

本文源自 Juicedata 创始人 & CEO Davies 在上海 Linux 用户组（SHLUG）的月度分享（2018/6/23）。

什么是文件系统?

文件系统是计算机中一个非常重要的组件，为存储设备提供一致的访问和管理方式。在不同的操作系统中，文件系统会有一些差别，但也有一些共性几十年都没怎么变化：

1. 数据是以文件的形式存在，提供 Open、Read、Write、Seek、Close 等API 进行访问；
2. 文件以树形目录进行组织，提供原子的重命名（Rename）操作改变文件或者目录的位置。

文件系统提供的访问和管理方法支撑了绝大部分的计算机应用，Unix 的“万物皆文件”的理念更是凸显了它的重要地位。文件系统的复杂性使得它的可扩展性未能跟得上互联网的高速发展，极大简化了的对象存储及时填补了空缺得以快速发展起来。因为对象存储缺乏树状结构也不支持原子重命名操作，跟文件系统有很大的差别，本文暂不讨论。

单机文件系统的挑战

绝大多数文件系统都是单机的，在单机操作系统内为一个或者多个存储设备提供访问和管理。随着互联网的高速发展，单机文件系统面临很多的挑战：

1. 共享：无法同时为分布在多个机器中的应用提供访问，于是有了 NFS 协议，可以将单机文件系统通过网络的方式同时提供给多个机器访问。
2. 容量：无法提供足够空间来存储数据，数据只好分散在多个隔离的单机文件系统里。
3. 性能：无法满足某些应用需要非常高的读写性能要求，应用只好做逻辑拆分同时读写多个文件系统。
4. 可靠性：受限于单个机器的可靠性，机器故障可能导致数据丢失。
5. 可用性：受限于单个操作系统的可用性，故障或者重启等运维操作会导致不可用。

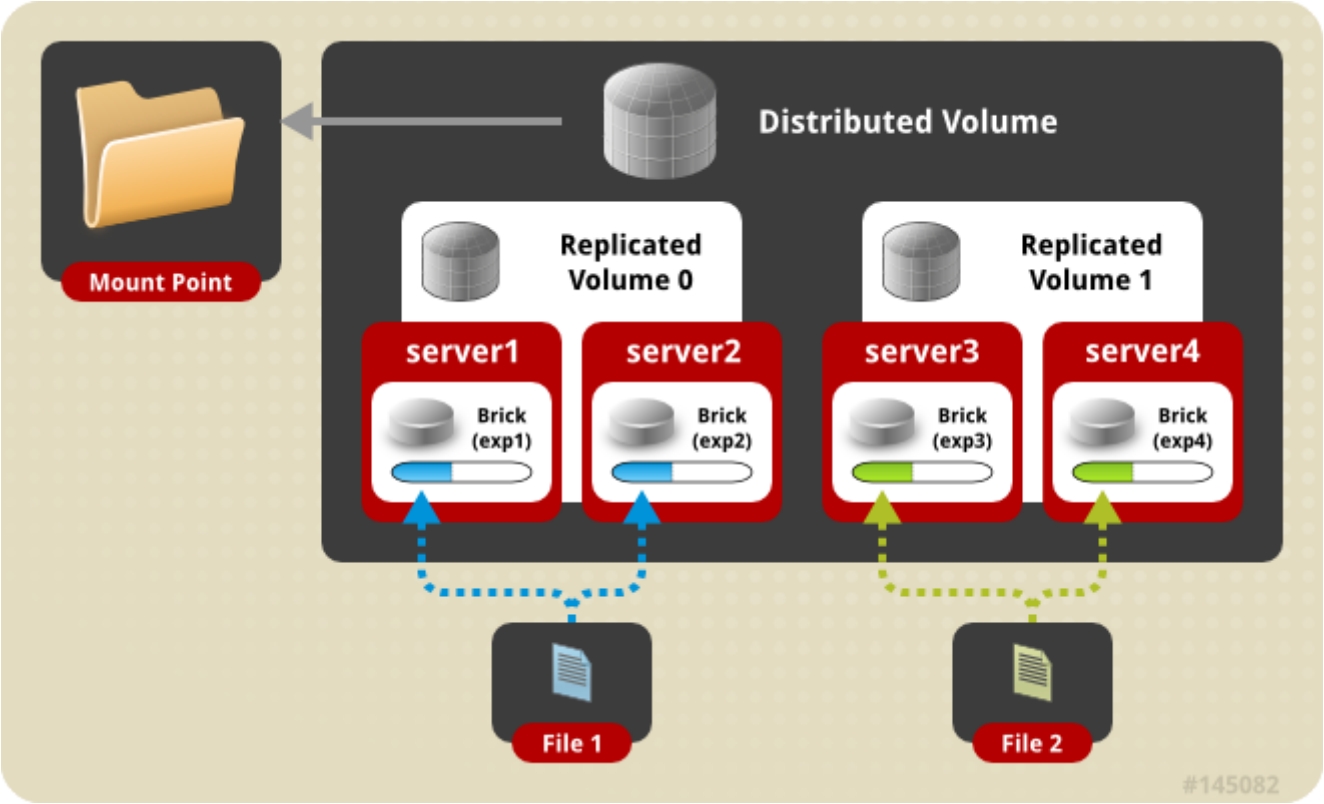
随着互联网的高速发展，这些问题变得日益突出，涌现出了一些分布式文件系统来应对这些挑战。

下面介绍几个我了解过的分布式文件系统的基本架构，并比较不同架构的优点和局限。

GlusterFS

GlusterFS 是由美国的 Gluster 公司开发的 POSIX 分布式文件系统（以 GPL 开源），2007年发布第一个公开版本，2011年被 Redhat 收购。

它的基本思路就是通过一个无状态的中间件把多个单机文件系统融合成统一的名字空间（namespace）提供给用户。这个中间件是由一系列可叠加的转换器（Translator）实现，每个转换器解决一个问题，比如数据分布、复制、拆分、缓存、锁等等，用户可以根据具体的应用场景需要灵活配置。比如一个典型的分布式卷如下图所示：



来源：https://docs.gluster.org/en/latest/Quick-Start-Guide/Architecture/ (https://docs.gluster.org/en/latest/Quick-Start-Guide/Architecture/)

Server1 和 Server2 构成有 2 副本的 Volume0，Server3 和 Server4 构成 Volume1，它们再融合成有更大空间的分布式卷。

优点：

1. 数据文件最终以相同的目录结构保存在单机文件系统上，不用担心 GlusterFS 的不可用导致数据丢失。
2. 没有明显的单点问题，可线性扩展。
3. 对大量小文件的支持估计还不错。

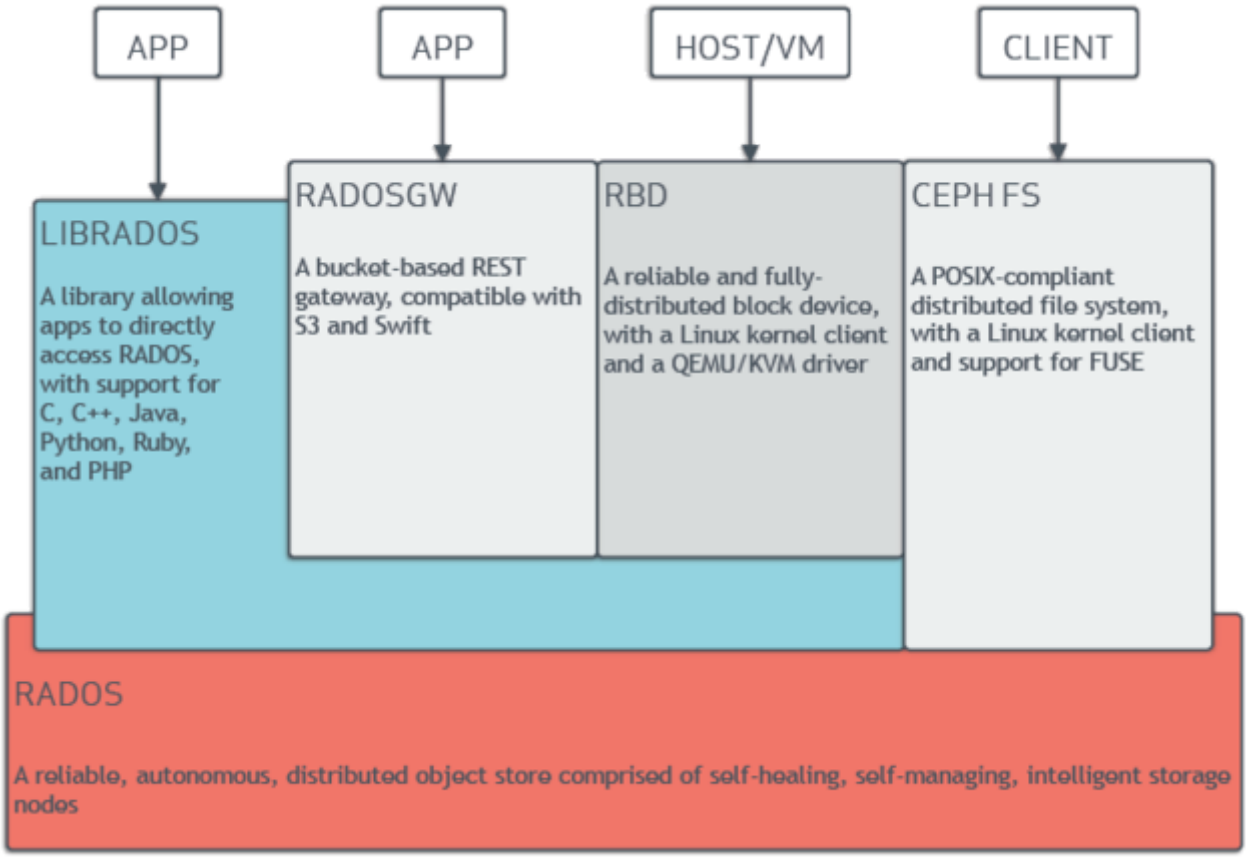
挑战：

1. 这种结构是相对静态的，不容易调整，也要求各个存储节点有相同的配置，当数据或者访问不均衡时没法进行空间或者负载调整。故障恢复能力也比较弱，比如 Server1 故障时，Server2 上的文件就没办法在健康的 3 或者 4上增加拷贝保障数据可靠。
2. 因为缺乏独立的元数据服务，要求所有存储节点都会有完整的数据目录结构，遍历目录或者做目录结构调整时需要访问所有节点才能得到正确结果，导致整个系统的可扩展能力有限，扩展到几十个节点时还行，很难有效地管理上百个节点。

CephFS

CephFS 始于 Sage Weil 的博士论文研究，目标是实现分布式的元数据管理以支持 EB 级别数据规模。2012 年，Sage Weil 成立了 InkTank 继续支持 CephFS 的开发，于 2014年被 Redhat 收购。直到 2016 年，CephFS 才发布可用于生产环境的稳定版（CephFS 的元数据部分仍然是单机的）。现在，CephFS 的分布式元数据仍然不成熟。

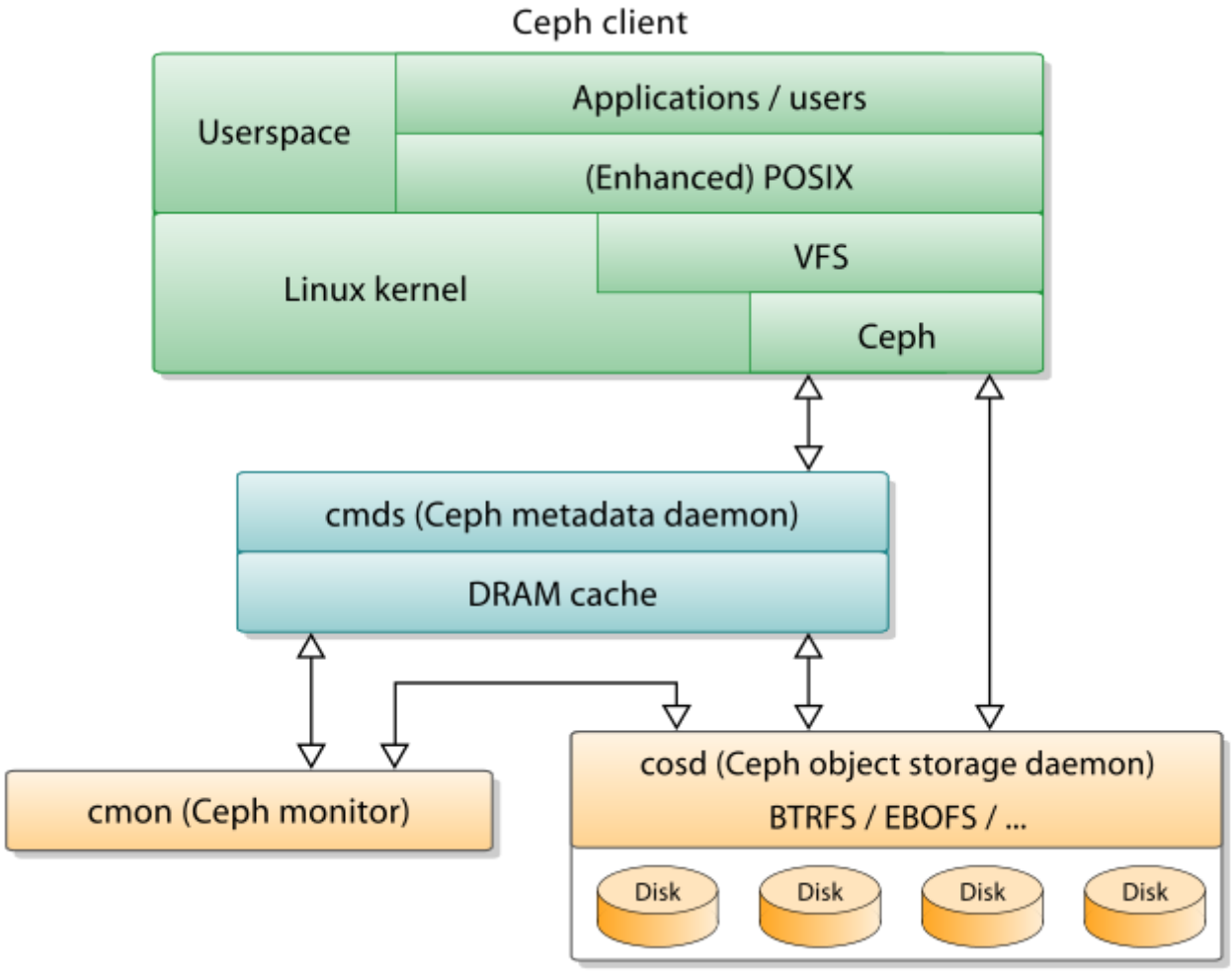
Ceph 是一个分层的架构，底层是一个基于 CRUSH（哈希）的分布式对象存储，上层提供对象存储（RADOSGW）、块存储（RDB）和文件系统（CephFS）三个API，如下图所示。



图片来源：https://en.wikipedia.org/wiki/Ceph_(software) (https://en.wikipedia.org/wiki/Ceph_(software))

用一套存储系统来满足多个不同场景的存储需求（虚拟机镜像、海量小文件和通用文件存储）还是非常吸引人的，但因为系统的复杂性需要很强的运维能力才能支撑，实际上目前只有块存储还是比较成熟应用得比较多，对象存储和文件系统都不太理想，听到不少负面的使用案例（他们用过一段时间Ceph 后就放弃了）。

CephFS 的架构如下图所示：



图片来源：https://en.wikipedia.org/wiki/Ceph_(software) (https://en.wikipedia.org/wiki/Ceph_(software))

CephFS 是由 MDS（Metadata Daemon) 实现的，它是一个或者多个无状态的元数据服务，从底层的 OSD 加载文件系统的元信息，并缓存到内存中以提高访问速度。因为 MDS 是无状态的，可以配置多个备用节点来实现 HA，相对比较容易。不过备份节点没有缓存，需要重新预热，有可能故障恢复时间会比较长。

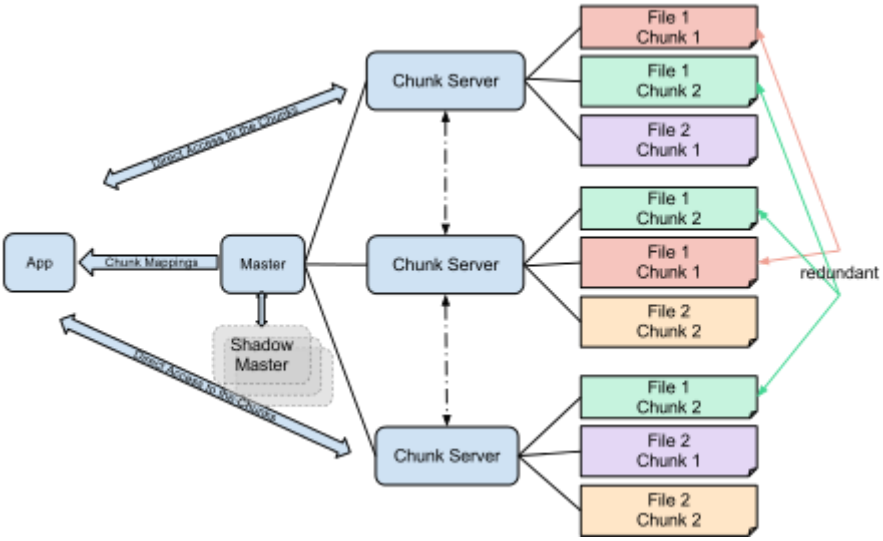
因为从存储层加载或者写入数据会比较慢，MDS 必须使用多线程来提高吞吐量，各种并发的文件系统操作导致复杂度大大上升，容易发生死锁，或者因为 IO 比较慢导致的性能大幅下降。为了获得比较好的性能，MDS 往往需要有足够多的内存来缓存大部分元数据，这也限制了它实际的支撑能力。

当有多个活跃的 MDS 时，目录结构中的一部分（子树）可以动态的分配到某个MDS并完全由它来处理相关请求，以达到水平扩展的目的。多个活跃之前，不可避免地需要各自锁机制来协商对子树的所有权，以及通过分布式事务来实现跨子树的原子重命名，这些实现起来都是非常复杂的。目前最新的官方文档仍然不推荐使用多个 MDS（作为备份是可以的）。

GFS

Google 的 GFS 是分布式文件系统中的先驱和典型代表，由早期的 BigFiles 发展而来。在 2003 年发表的论文中详细阐述了它的设计理念和细节，对业界影响非常大，后来很多分布式文件系统都是参照它的设计。

顾名思义，BigFiles/GFS 是为大文件优化设计的，并不适合平均文件大小在 1MB 以内的场景。GFS的架构入下图所示：



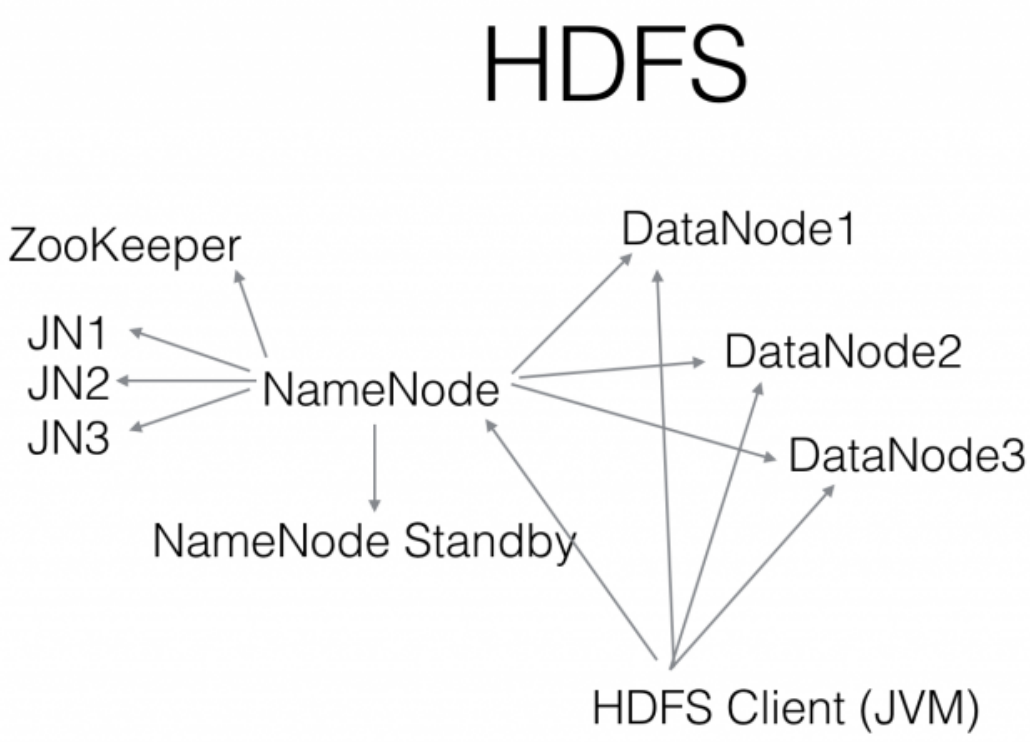
图片来源：https://en.wikipedia.org/wiki/Google_File_System
(https://en.wikipedia.org/wiki/Google_File_System)

GFS 有一个 Master 节点来管理元数据（全部加载到内存，快照和更新日志写到磁盘），文件划分成 64MB 的 Chunk 存储到几个 ChunkServer 上（直接使用单机文件系统）。文件只能追加写，不用担心 Chunk 的版本和一致性问题（可以用长度当做版本）。这个使用完全不同的技术来解决元数据和数据的设计使得系统的复杂度大大简化，也有足够的扩展能力（如果平均文件大小大于 256MB，Master 节点每 GB 内存可以支撑约 1PB 的数据量）。放弃支持 POSIX 文件系统的部分功能（比如随机写、扩展属性、硬链接等）也进一步简化了系统复杂度，以换取更好的系统性能、鲁棒性和可扩展性。

因为 GFS 的成熟稳定，使得 Google 可以更容易地构建上层应用（MapReduce、BigTable等）。后来，Google 开发了拥有更强可扩展能力的下一代存储系统 Colossus，把元数据和数据存储彻底分离，实现了元数据的分布式（自动 Sharding），以及使用Reed Solomon编码来降低存储空间占用从而降低成本。

HDFS

出自 Yahoo 的 Hadoop 算是 Google 的 GFS、MapReduce 等的开源Java实现版，HDFS 也是基本照搬 GFS 的设计，这里就不再重复了，下图是HDFS的架构图。



HDFS的可靠性和可扩展能力还是非常不错的，有不少几千节点和 100PB 级别的部署，支撑大数据应用表现还是很不错的，少有听说丢数据的案例（因为没有配置回收站导致数据被误删的除外）。

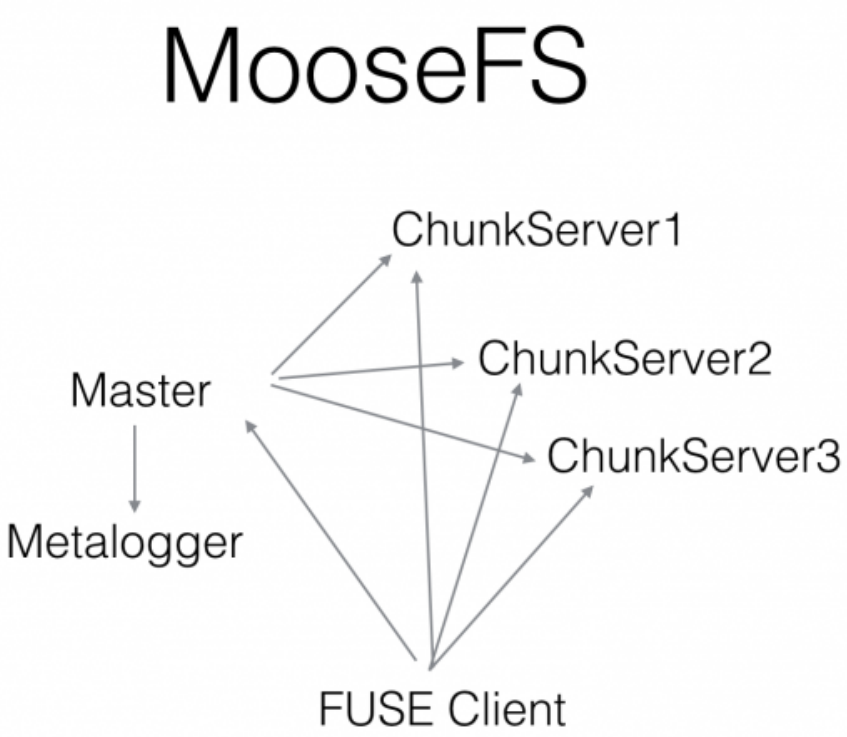
HDFS 的 HA 方案是后来补上的，做得比较复杂，以至于最早做这个 HA 方案的 Facebook 在很长一段时间（至少3年）内都是手动做故障切换（不信任自动故障切换）。

因为 NameNode 是 Java 实现的，依赖于预先分配的堆内存大小，分配不足容易触发 Full GC 而影响整个系统的性能。有一些团队尝试把它用 C++ 重写了，但还没看到有成熟的开源方案。

HDFS 也缺乏成熟的非 Java 客户端，使得大数据（Hadoop等工具）以外的场景（比如深度学习等）使用起来不太方便。

MooseFS

MooseFS 是来自波兰的开源分布式 POSIX 文件系统，也是参照了 GFS 的架构，实现了绝大部分 POSIX 语义和 API，通过一个非常成熟的 FUSE 客户端挂载后可以像本地文件系统一样访问。MooseFS 的架构如下图所示：



MooseFS 支持快照，用它来做数据备份或者备份恢复等还是恢复方便的。

MooseFS 是由 C 实现的，Master 是个异步事件驱动的单线程，类似于 Redis。不过网络部分使用的是 poll 而不是更高效的 epoll，导致并发到 1000 左右时 CPU 消耗非常厉害。

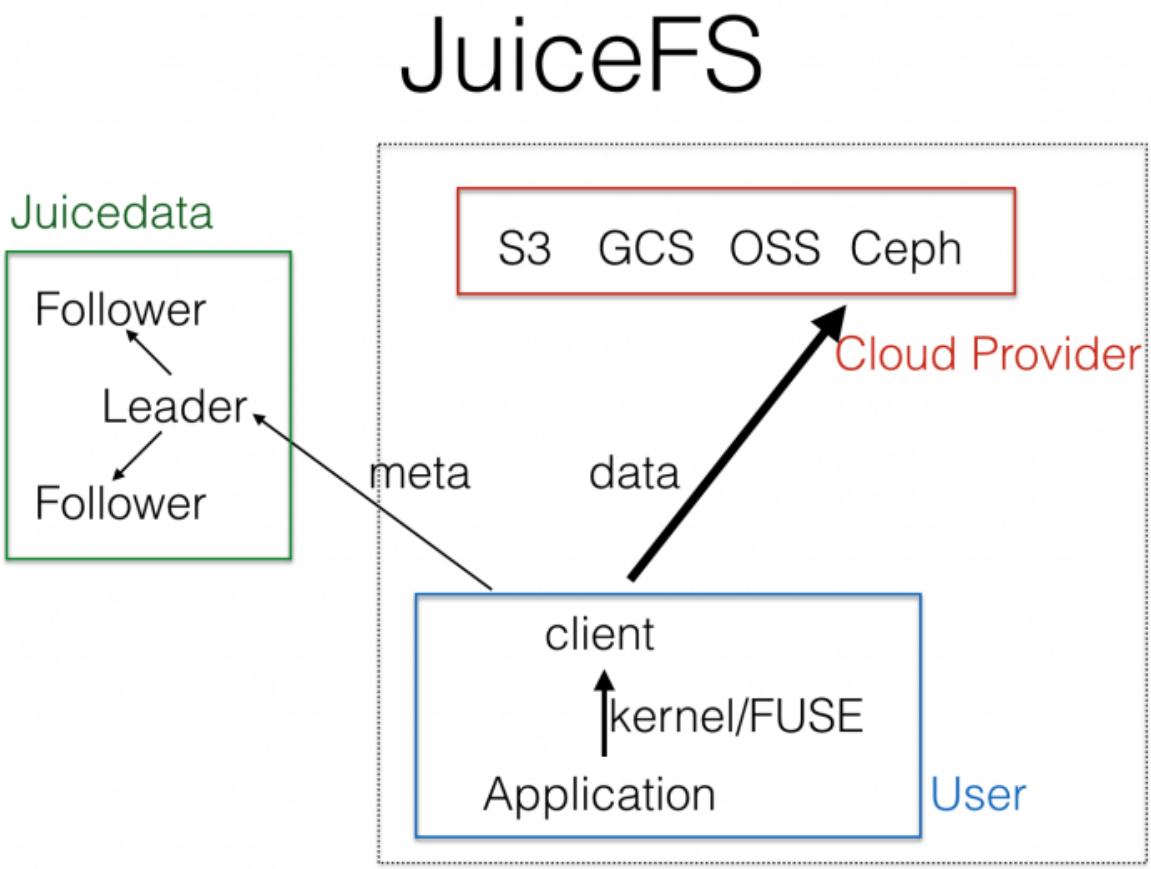
开源的社区版没有HA，是通过 metalogger 来实现异步冷备，闭源的收费版有 HA。

为了支持随机写操作，MooseFS 中的 chunk 是可以修改的，通过一套版本管理机制来保证数据一致性，这个机制比较复杂容易出现诡异问题（比如集群重启后可能会有少数 chunk 实际副本数低于预期）。

JuiceFS

上面说的 GFS、HDFS 和 MooseFS 都是针对自建机房这种软硬件环境设计的，将数据的可靠性和节点可用性合在一起用多机多副本的方式解决。但是在公有云或者私有云的虚拟机里，块设备已经是具有三副本可靠性设计的虚拟块设备，如果再通过多机多副本的方式来做，会导致数据的成本居高不下（实际上是 9 个拷贝）。

于是我们针对公有云，改进 HDFS 和 MooseFS 的架构，设计了 JuiceFS，架构如下图所示：

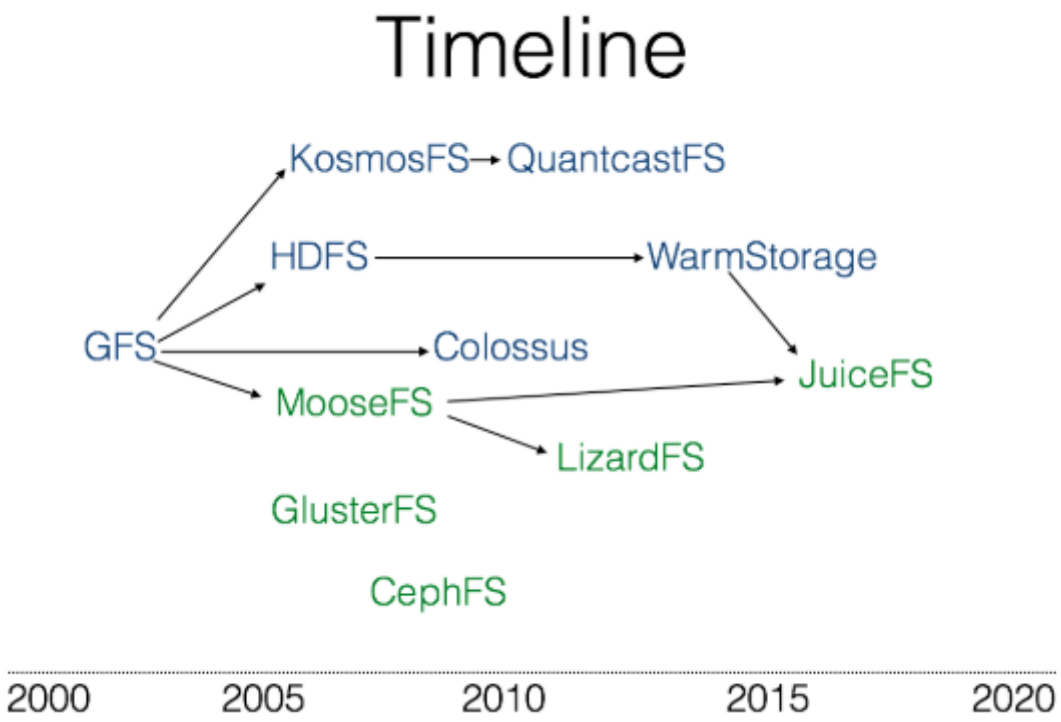


JuiceFS 使用公有云中已有的对象存储来替换 DataNode 和 Chunk Server，实现一个完全弹性的 Serverless 的存储系统。公有云的对象存储已经很好地解决了大规模数据的安全高效存储，JuiceFS 只需要专注元数据的管理，也大大降低了元数据服务的复杂度（GFS 和 MooseFS 的 master 要同时解决元数据的存储和数据块的健康管理）。我们也对元数据部分做了很多改进，从一开始就实现了基于 Raft 的高可用。要真正提供一个高可用高性能的服务，元数据的管理和运维仍然是很有挑战的，元数据是以服务的形式提供给用户。因为 POSIX 文件系统 API 是应用最最广泛的 API，我们基于 FUSE 实现了高度 POSIX 兼容的客户端，用户可以通过一个命令行工具把 JuiceFS 挂载到 Linux 或者 macOS 中，像本地文件系统一样快速访问。

上图中右边虚线部分是负责数据存储和访问的部分，涉及用户的数据隐私，它们是完全在客户自己的账号和网络环境中，不会跟元数据服务接触。我们（Juicedata）没有任何方法接触到客户的内容（元数据除外，请不要把敏感内容放到文件名里）。

小结

以上简要介绍了下我所了解的几个分布式文件系统的架构，把他们按照出现的时间顺序放在下面的图里（箭头表示后参考了前者或者是新一代版本）：



上图中上部分蓝色的几个文件下主要是给大数据场景使用的，实现的是 POSIX 的子集，而下面绿色的几个是 POSIX 兼容的文件系统。

他们中以 GFS 为代表的元数据和数据分离的系统设计能够有效平衡系统的复杂度，有效解决大规模数据的存储问题（通常也都是大文件），有更好的可扩展性。这个架构下支持元数据的分布式存储的 Colossus 和 WarmStorage 更是具有无限的可扩展能力。

JuiceFS 作为后来者，学习了 MooseFS 实现分布式 POSIX 文件系统的方式，也学习了 Facebook 的 WarmStorage 等元数据和数据彻底分开的思路，希望为公有云或者私有云等场景下提供最好的分布式存储体验。JuiceFS 通过将数据存储到对象存储的方式，有效避免了使用以上分布式文件系统时的双层冗余（块存储的冗余和分布式系统的多机冗余）导致的成本过高问题。JuiceFS 还支持所有的公有云，不用担心某个云服务锁定，还能平滑地在公有云或者区之间迁移数据。

最后，如果你手上有公有云账号，来 JuiceFS 注册一下 (https://juicefs.io/accounts/register)，5 分钟就可以给你的虚拟机（甚至自己的 macOS）挂载上一个 PB 级容量的文件系统了。

← 客户案例：下厨房 基于 JuiceFS 的 MySQL 备份实践 (https://juicefs.io/blog/cn/xiachufang-mysql-backup-practice-on-juicefs/)

为什么说存储和计算分离的架构才是未来？ → (https://juicefs.io/blog/cn/why-disaggregated-compute-and-storage-is-future/)