MinIO创始者是Anand Babu Periasamy, Harshavardhana（戒日王）等人， Anand是GlusterFS的初始开发者、Gluster公司的创始人与CTO，Harshavardhana曾经是GlusterFS的开发人员，直到2011年红帽收购了Gluster公司。MinIO在设计上汲取了GlusterFS的相关经验与教训，系统复杂度上作了大量简化。

一、MinIO简介

01.概述

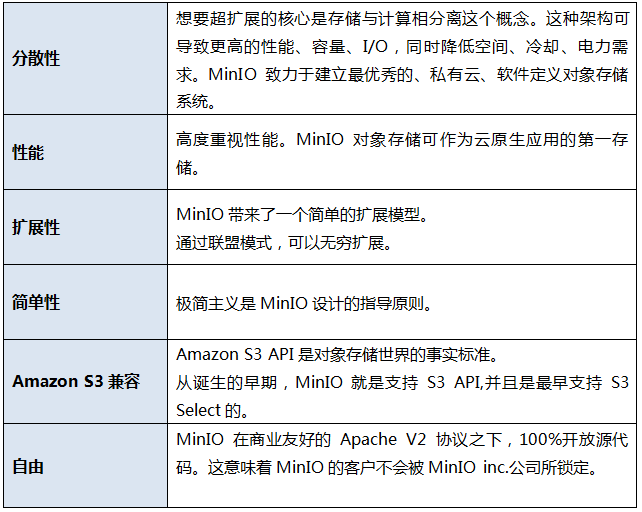
MinIO对象存储系统是为海量数据存储、人工智能、大数据分析而设计，基于Apache License v2.0开源协议的对象存储系统，它完全兼容Amazon S3接口，单个对象最大可达5TB，适合存储海量图片、视频、日志文件、备份数据和容器/虚拟机镜像等。MinIO主要采用Golang语言实现，整个系统都运行在操作系统的用户态空间，客户端与存储服务器之间采用http/https通信协议。

02.设计哲学

极简理念——采用尽可以简单可靠的集群管理方案，摒弃复杂的大规模集群调度管理，减少风险因素与性能瓶颈，聚焦产品的核心功能，打造高可靠的集群、灵活的扩展能力以及超高的性能；

积木式扩展——建立众多的中小规模、易管理的集群，支持跨数据中心将多个集群聚合成超大资源池，而非直接采用大规模、统一管理的分布式集群。

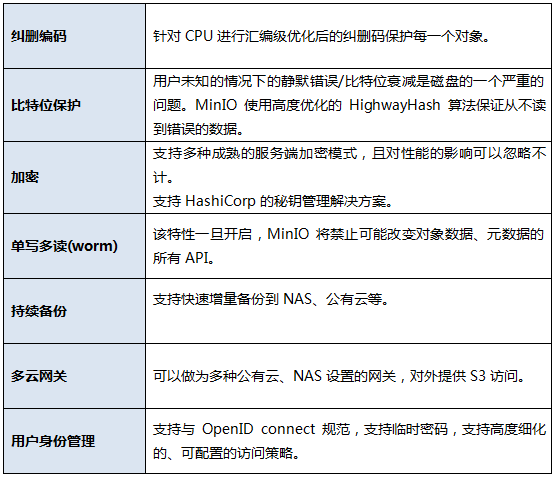
03.设计原则



04.产品特点



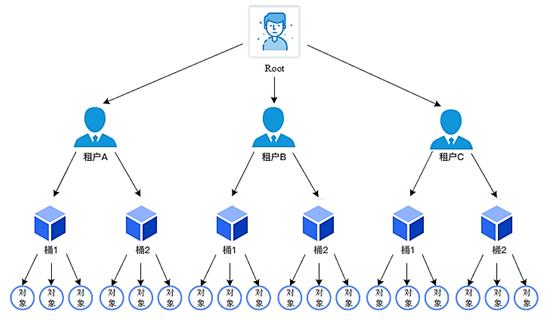
05.高级特性



二、技术架构

01 数据组织结构

NAS系统把整个存储资源组织为目录树的形式。与此不同，对象存储系统把存储资源组织为租户-桶-对象的形式。数据结构组织见下图：



对象：类似于hash表中的表项：它的名字相当于关键字，它的内容相当于“值”。

桶：是若干个对象的逻辑抽象，是盛装对象的容器。

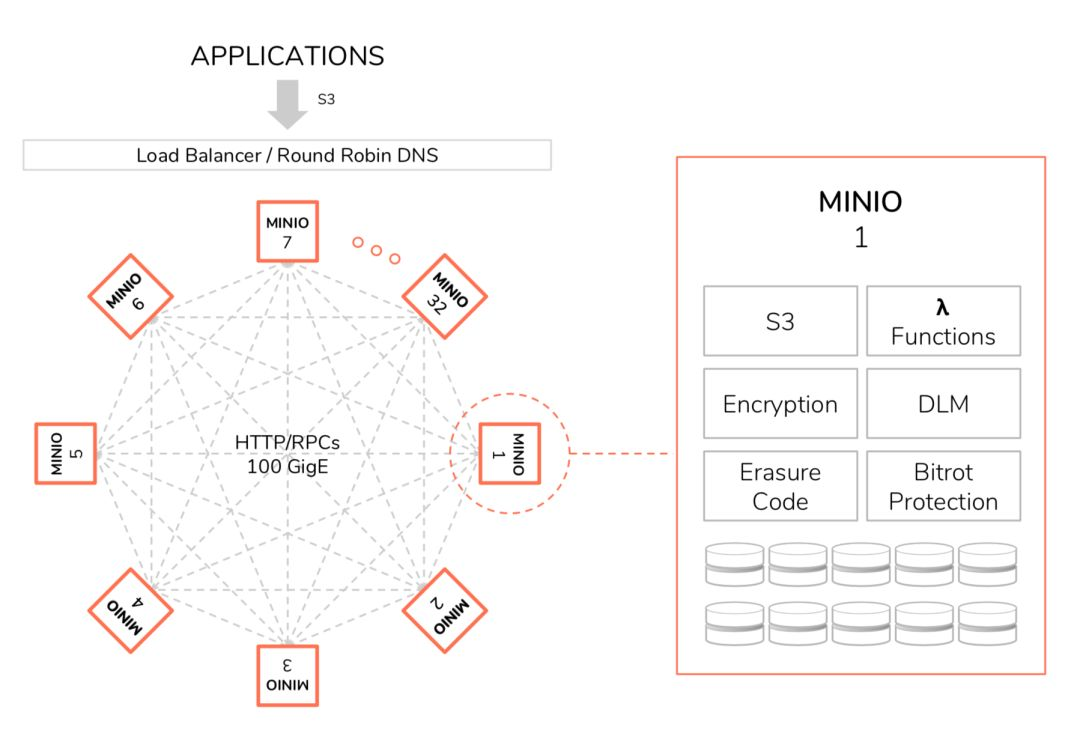
租户：用于隔离存储资源。在租户之下可以建立桶、存储对象。

用户：在租户下面创建的用于访问不同桶的账号。可以使用MinIO提供的mc命令设置不用用户访问各个桶的权限。

02 数据分布与均衡

1 去中心化架构

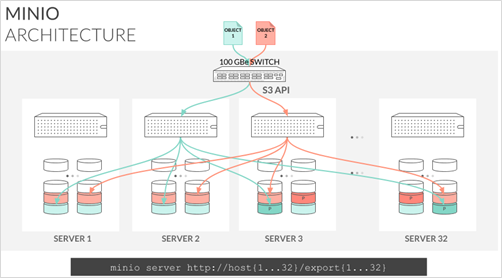
MinIO采用去中心化的无共享架构，对象数据被打散存放在不同节点的多块硬盘，对外提供统一命名空间访问，并通过Web负载均衡器或DNS轮询（DNS round-robin）在各服务器之间实现负载均衡。



2 统一命名空间

MinIO对象存储系统主要有两种部署方式，一种是常见的本地分布式集群部署，一种是联盟模式部署。本地分布式集群部署方式即在多个本地服务器节点部署MinIO软件，并将其组件成单套分布式存储集群，并提供统一命名空间和标准S3访问接口。联盟部署模式即将多个MinIO集群在逻辑上组成了统一命名空间，实现近乎无限的扩展与海量的数据规模管理，这些集群可以都在本地，或分布在不同地域的数据中心。

如下图所示，4个服务器节点组成一个MinIO集群，每个服务器节点中会选择相同数据的硬盘创建一个纠删组，某个桶的数据会根据MinIO的分布式算法，切片分散存储到对应的纠删组中（详见纠删码相关内容）。



3 分布式锁管理

与分布式数据库相类似，MinIO对象存储系统也面临数据一致性问题：一个客户端程序在读取一个对象的同时，另一个客户端程序可能正在修改或者删除这个对象。为了避免出现数据不一致情况，MinIO相关开发人员为MinIO对象存储专门设计并实现了dsync分布式锁管理器。它采用如下分布式锁管理机制：

l 任何一个节点的锁请求都会广播给集群内所有在线节点；

l 如果n/2 + 1个节点回应“是”，则成功获得锁；

l 客户端获得锁以后可保留任意时间，不需要时自己释放即可。释放操作也会广播给所有的节点，从而恢复锁的可用状态。写锁仅能被一个写入者获得。

设计目标

•要求设计简单，因为简单的设计，可以避免程序中很多非常棘手的条件分支的支持。

•不存在主节点，因为一旦在设计上引入主节点，那么如果主节点宕机，整个锁管理器机制即将失效，这对MinIO对象存储系统影响非常严重，是不可接受的。

•系统必须是弹性的，即使存在多个失效的节点，只要它们的个数小于n/2, 整个锁管理系统是可以正常工作的。

•完全可以替代Golang标准库中的sync.RWMutex互斥锁。这样可以简化MinIO对象存储系统的编程。

•当失效节点重启以后，其它节点重新连接。

不使用zookeeper/raft等技术的原因

zookeeper/raft功能丰富，而MinIO对象储存的使用用例其实很有限。在MinIO中使用zookeeper/raft，会使整个系统增加不必要的复杂性。

优势

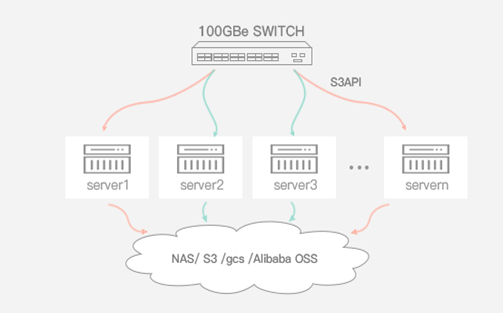
•实际操作极其简单，有效代码不足一千行，易理解，易维护。

•超高的性能。详细数据请参考文献[12]

4 云网关模式

MinIO存储系统的后端可以是磁盘，也可以作为云网关，对接第三方的NAS系统、分布式文件系统或公有云存储资源，并为业务系统转换提供标准的对象访问接口。

目前MinIO支持Google 云存储、HDFS、阿里巴巴OSS、亚马逊S3, 微软Azure Blob 存储等第三方存储资源。



03 元数据

1 架构

MinIO对象存储系统无元数据数据库，所有的操作都是对象级别的粒度的。这种做法的优势是:

• 个别对象的失效，不会溢出为更大级别的系统失效。

•便于实现“强一致性”这个特性。此特性对于机器学习与大数据处理非常重要。

2 管理

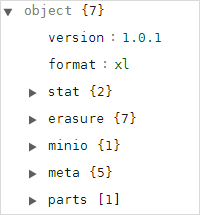
元数据与数据一起存放在磁盘上：数据部分纠删分片以后存储在磁盘上，元数据以明文形式存放在元数据文件里（xl.json）。假定对象名字为obj-with-metadata, 它所在的桶的名字是bucket\_name, disk是该对象所在纠删组的任一个磁盘的路径，如下目录：

disk/bucket\_name/obj-with-metadata

记录了这个对象在此磁盘上的信息。其中的内容如下：



其中的xl.json即是此对象的元数据文件。part.1 即此对象的第一个数据分片。对象的元数据文件xl.json的内容是如下这种形式的json字符串：



字段说明

1 format字段

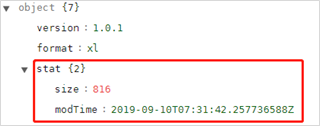
该字段指明了这个对象的格式是xl。MinIO内部存储数据主要有两种数据格式:xl与fs。使用如下命令启动的MinIO使用的存储格式是fs:



这种模式主要用于测试， 对象存储很多API都是并没有真正实现的桩函数。在生产环境所用的部署方式（本地分布式集群部署、联盟模式部署、云网关模式部署）中，存储格式都是xl。

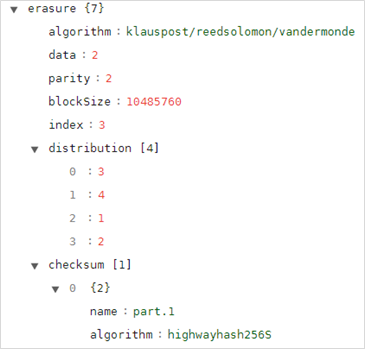
2 stat字段

记录了此对象的状态，包括大小与修改时间，如下图：



3 erasure字段

这个字段记录此对象与纠删码有关的信息，如下图：



• 其中的algorithm指明了此对象采用的是Klaus Post实现的纠删码，生成矩阵是范德蒙矩阵。

• data，parity指明了纠删组中数据盘、校验盘的个数。

• blockSize 指明了对象被分块的大小，默认是5M（请参见上一节“数据分布与均衡”）。

•index指明了当前磁盘在纠删组中的序号。

• distribution:每个纠删组的数据盘、校验盘的个数是固定的，但是不同的对象的分片写入这个纠删组的不同磁盘的顺序是不同的。这里记录了分布顺序。

• checksum:它下面的字段个数跟此对象的分片数量有关。在旧版本的MinIO对象存储系统，每一个分片经过hash函数计算出的checksum会记录在元数据文件的这个位置。最新版的MinIO会把checksum直接计入分片文件（即part.1等文件）的前32个字节。

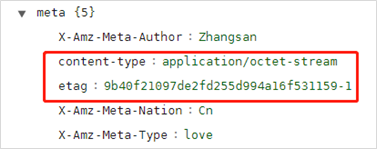
此字段之下algorithm的值是”highwayhash256S”表明checksum值是写入分片文件的。

4 minio字段

这个字段记录了存储此对象的minio的版本。



5 meta字段



Content-type, etag两个字段是MinIO对象存储系统自动生成的。

用户在使用Python等语言的写作的访问MinIO的程序中，如果上传对象时候指定了几个自定义属性，比如：

author属性值为Zhangsan

Nation属性值为Cn

Type属性值为love

那么对象元数据文件的meta字段就会出现如下几个子字段：

X-Amz-Meta-Author

X-Amz-Meta-Nation

X-Amz-Meta-Type

6 parts字段

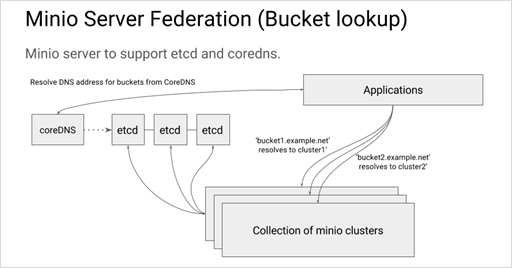
记录各个分片的信息：



04 集群扩展

1 扩展方式

MinIO支持联盟部署模式，即将多个MinIO集群组成一个统一命名空间（一个ETCD集群，若干个CoreDNS服务）。其中ETCD集群相当于整个对象存储系统的配置数据库，很多重要的信息，如桶IP地址等存储于其中。这种模式的MinIO的架构如下图：

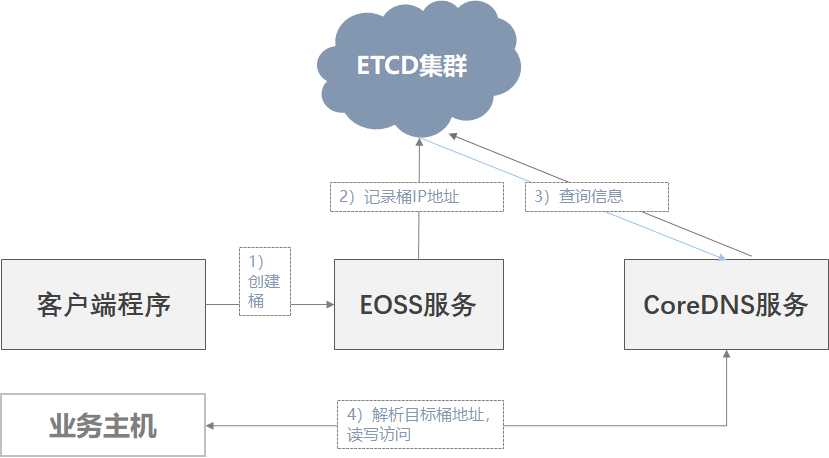


联盟模式多集群部署

同样，MinIO在扩展时也采用相同机制，而不是传统分布式存储的添加节点方式。MinIO主要通过添加新的集群来扩大整个系统，可用空间大幅增加且仍然保持统一命名空间。通过这种方式，MinIO对象存储系统几乎可以无限的扩展总体性能和容量。

2 统一域名访问

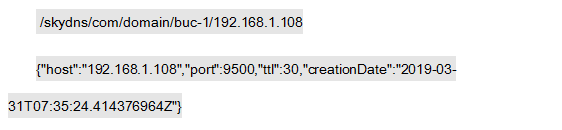
MinIO集群扩展加入新了集群或桶后，对象存储的客户端程序需要通过统一域名/url（如bucket1.domain.com）来访问数据对象，这个过程涉及到了CoreDNS系统。



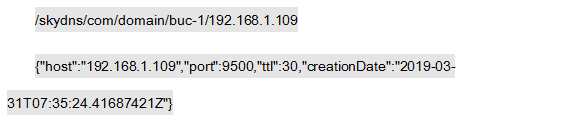
CoreDNS实现单一域名/URL访问

MinIO对象存储的某个客户端（比如mc），首先向某个MinIO服务发送创建桶的请求。MinIO服务把这个桶所在的MinIO集群的外部网址（一般为一个Nginx的IP地址，或者MinIO集群的每一台服务器的IP地址）,写入到etcd集群中。

假定域名为domain.com,桶名为buc-1，集群的服务器IP地址为192.168.1.108、192.168.1.109，那么写入etcd集群的共有两条数据.第一条数据的key，value二元组为：



第二条数据的key，value二元组为：



CoreDNS通过etcd系统获知”bucket1.domain.com”这个url所对应的两个IP地址为192.168.1.108, 192.168.1.109。对象存储的客户端主机设置如上所配置的CoreDNS服务之后，客户端程序就可以通过域名”bucket1.domain.com”来找到访问这个桶。

3 优势特性

单一的、超大的命名空间需要花费大量的创建、维护与停机时间，复杂的部署管理，进而带来更严重的次生故障。MinIO的设计理念就是化整为零，简化集群扩展，减小单个集群的体量，轻量化单个集群的运维，从而使得超大规模的存储管理与维护变得更加容易。

• 集群的节点完全对等，没有主节点，多个节点可以并发提供对象访问服务;

• 创建桶的时候，可以指定数据中心/地域，以匹配对应的业务访问;

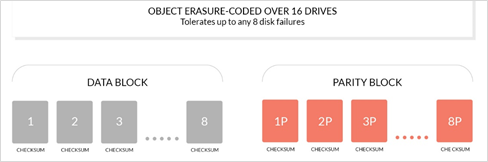
• 无论添加多少个集群,原有集群的性能几乎是不变的;

• 集群不会过大（32个节点），可实现可靠的分布式锁管理器，进而保证更新、删除等操作的强一致性。传统的架构允许集群扩容到数百上千节点，此情况下的强一致性容易产生性能问题;

• 故障的影响范围小，限制在单个集群内部。

05 纠删码

在同一集群内，MinIO会自动生成若干纠删组，用于存放桶数据。一个纠删组中的一定数量的磁盘发生的故障（故障磁盘的数量小于等于校验盘的数量），通过纠删码算法可以恢复出正确的数据。MinIO集成了Reed-Solomon纠删码库，MinIO存储对象数据时，首先把它生成若干等长的片段（对于大对象，默认按5MB切片)，然后每一个片段会纠删算法分成若干分片，包括数据分片与校验分片，每个分片放置在一个纠删组的某个节点上。对象的每一个数据分片、校验分片都被“防比特位衰减”算法所保护。



对于一个对象，MinIO是如何定位它所在的纠删组呢？

假定所有的纠删组都有一个序号（从0开始，直至纠删组个数减1）。MinIO会根据对象名（类似于文件系统的全路径名)，使用crc32哈希算法计算出一个整数。然后使用这个整数除以纠删组的个数，得到一个余数。这个余数，可以作为纠删组的序号，这样就确定了这个对象所在的纠删组。MinIO采用CRC32哈希算法，与GlusterFs的Davies-Meyer哈希算法（性能、冲突概率与md4, md5相近）不一样的是， CRC32算法的哈希值分布较不均匀，但运算速度极快，高出md4数倍。相对于容量均衡，MinIO更看重数据的写入速度。

06 数据修复

比特位衰减（Bitrot）是指存在存储介质中的数据发生了缓慢的变化，如向存储介质写入一段比特流，一段时间后再读出来，二者并不一致。比特位衰减的原因大致有：磁记录磨损、磁盘幻象写(phantom writes)、磁盘指向错误(misdirectedreads/writes)、宇宙射线的辐射等。MinIO对象存储系统从设计之初即考虑到修复静默错误，从被修复的目标来说，按照大小可以分为以下三种类型的修复：某个对象、某个桶、整个集群。

在控制台上执行mc命令即开始进行数据修复。该命令一方面向minio发送数据修复的HTTP请求，另一方面不断地接收minio服务进程返回的修复进度信息，而后输出到控制台，直到修复工作完毕。

如前文所述，每个对象都被分成多个分片，然后存储于多台主机的磁盘上。数据修复可以分为正常、深度两种模式，正常模式下只是简单地检查分片状态信息，深度模式下会使用hash算法来校验分片的内容，找出比特位错误，同时也更耗费资源。

MinIO具体修复流程如下：

• mc命令作为MinIO对象存储的客户端软件、管理工具，它内部链接了minio软件（代码网址：https://github.com/minio/minio/）的madmin软件模块，通过调用madmin中的修复函数，mc包装了mc命令的命令行参数，然后向minio服务进程发送HTTP消息。

•mc发送一个修复请求，在minio中被类healSequence所描述。每一个healSequence可以启动、停止、查询状态。minio服务程序收到新的任务的时候，会检查是否跟原有的healSequence有重叠的任务，如果有重叠，则启动的修复任务失败。如果minio服务没有发现错误，则使用深度优先搜索的算法，按照磁盘元数据信息、桶、对象的顺序，不断地给后台修复线程推送任务。

•minio后台修复线程修复对象的流程算法：对于对象的每一个block（默认大小为5M），从纠删组的各个主机读取各个分片，如果有错误的分片，就需要修复，有两种可能：校验分片错误——minio使用各个数据分片重新计算缺失的校验片。数据分片错误——使用纠删算法恢复数据（需要计算逆矩阵）。

07 lambda计算

MinIO对象存储软件支持lambda计算通知机制，即桶中的对象支持事件通知机制。MinIO当前支持的事件类型有：对象上传、对象下载、对象删除、对象复制等。MinIO对象存储软件当前支持的事件接受系统有：Redis,NATS, AMQP, MQTT,Apache Kafka, MySql, PostgreSQL, Elasticsearch等。

对象通知机制，极大地增强了MinIO对象存储的扩展性，可以让用户通过自行开发来实现某些MinIO对象存储不便实现的功能，比如基于元数据进行的各种检索、各种跟用户的业务有关的计算。既方便了用户，又有助于MinIO对象存储的生态建设。

对象通知机制，使用极为简单，用户只需在MinIO进行少许配置即可。请参考文献[15]。

08 持续备份

传统的复制的一大问题是不能有效地扩展，很难超过几百TB。在当今的时代，为了支持灾难恢复，任何单位都需要一个备份策略。而且这个备份策略需要跨越不同的地理位置、不同的数据中心、多种多样的云环境。

MinIO的持续备份是为跨数据中心的大型部署而设计的。通过使用lambda计算通知机制，它可以快速、有效地计算处需要增量备份的内容，这远比传统的批处理模式优秀。持续备份使得未备份的数据尽可能的少，这意味着发生灾难或者严重错误时候，丢失的数据尽可能的少，很好地保护了用户的数据资产。

9 软件模块

MinIO对象存储系统主要由以下软件模块部分组成：存储服务器软件minio，存储客户端软件mc，多种语言的客户端SDK。minio分为上下两层，上层负责minio的系统管理与对外接口，下层实现具体的逻辑。

1 cmd模块

这是minio的上层，也就是源代码中的cmd子目录，参见： https://github.com/minio/minio/tree/master/cmd。这一部分主要负责minio的命令行参数解析、初始化系统、格式化磁盘、管理内嵌的web服务器、S3 API的解析与逻辑处理。

2 各个软件包

这个是minio底层逻辑实现，也就是源代码目录中的pkg子目录。其中一些软件包（比如madmin）, 可被其它组织（或个人）在编写辅助minio的软件的时候所重复使用。

• madmin：使用这个软件包可以自己使用Golang语言撰写MinIO集群的管理程序，比如获取服务的状态（磁盘、cpu等信息）、重启某个机器服务、启动修复某个桶的任务、重新配置系统、获取剖析信息等等。

• S3 select：如果对象存储系统中有很多超大型的对象，比如大小是几个GB甚至几个TB的对象。如果应用程序（比如spark分析程序），要把符合条件的若干个对象都读过去，然后再做分析，会及其的慢，浪费很多带宽(毕竟对象中可能只有很少的一部分是对某个分析程序有用的)。因此Amazon引入了S3 Select 的功能。通俗地说，就是把select 类型的sql语句在某个对象上执行，从对象中取出一部分内容返回给应用。MinIO提供了S3 Select 功能。相对于S3 Select, MinIO要求对象的内容必须是CSV、 JSON,或者 Parquet格式。S3Select API实现中使用的语法分析器是 Alec Thomas写的如下项目：

https://github.com/alecthomas/participle

这个实现的分析算法是带有栈的ll(k)分析算法。

三 性能测试

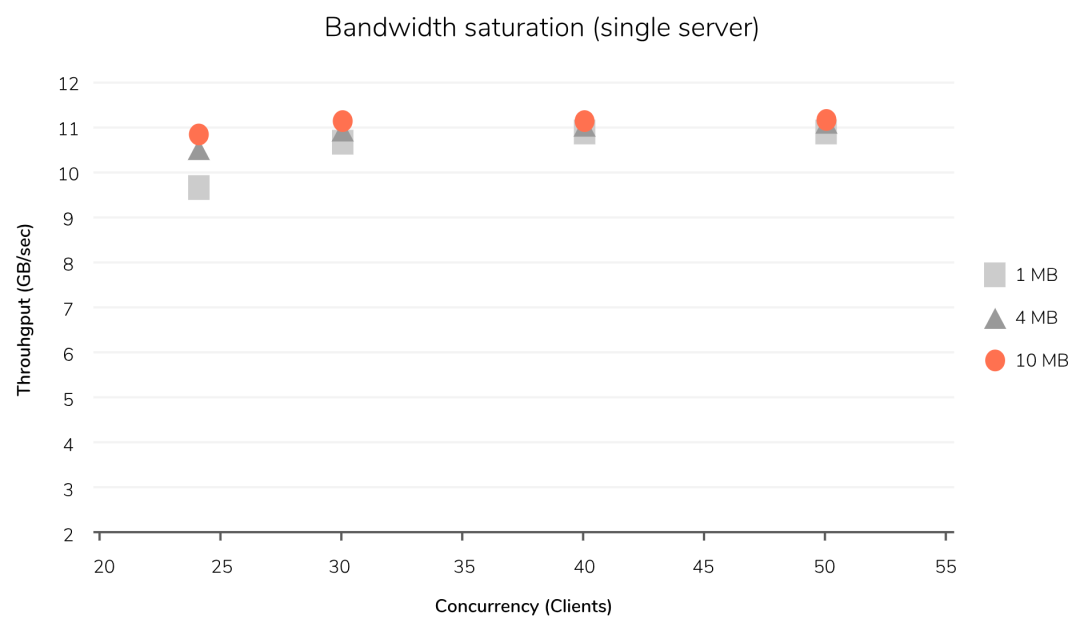
MinIO已经为高性能做过高度优化，尤其是部分关键的算法已经使用SIMD指令对Intel(AVX2/AVX512)、Arm(NEON)的cpu做过特殊优化，主要包括：

1) 纠删码部分用到的伽罗瓦域的运算：加法、乘法、乘方等等；

2) 监测比特位衰减(bitrot)的哈希函数,如HighwayHash。

另外每一个MinIO集群都是无中心的，其中的每一个节点都是对等的，从而在性能上，不会存在单点瓶颈，也不会有单点故障。

如下的硬件配置之下：Intel Skylake CPU, NVMe磁盘，以及Mellanox CX5 dual 100-GbE网卡。下图是MinIO inc的测试结果：



四 设计讨论

为什么MinIO单集群不支持扩展？

•传统的扩展方式的劣势

通过增加节点来扩展单集群，一般需要进行数据均衡，否则群集内各存储节点会因负载不均而出现新的瓶颈。除了数据均衡操作的时机这个问题以外，在均衡过程中一般需要从存储使用率高的节点向使用率低的节点迁移数据。当集群扩容之后，大量已经写入的文件落点会出现改变，文件需要迁移到真实的落点。当存储系统容量比较大时，则会发生大量的文件/对象进行迁移,迁移过程可能由于占用大量资源而导致上层应用性能下降。而且当文件/对象迁移过程中，机器故障可能会导致一些意想不到的情况,尤其是有大量业务的时候。当然针对此类问题，Gluterfs之类的文件系统有一些比较复杂的处理办法。

•使用场景

人工智能、大数据分析、视频监控等典型使用场景中，对象存储系统中存储的数据往往写入以后一般不再修改。如果现有MinIO集群存储空间使用完毕，重新添加新集群，然后继续写入新集群即可。MinIO对象存储的客户端应用，从业务层面自行决定那些对象存在于哪个集群里面，使用起来并不麻烦。

单集群不可扩展，也就是说系统不需要处理扩展和数据均衡，不仅有效降低系统复杂性，而且可以使得系统部署规划具有很好的可预测性。

对于海量对象存储应用场景，数据通常具有典型的生命周期特征，根据实际需求设计好单集群规模，按联合方式扩展，整个系统具有非常好的可维护性。

•MinIO方案的优势

不支持对单个集群进行扩展，MinIO对象存储系统的这种设计，使得系统的很多模块更加简单（比如从一个对象转换到它所在的纠删组，只用简单的哈希即可。）降低了整个系统出错的概率，使得MinIO对象存储系统更加可靠、稳定。

详细的讨论参见文献[14]

MinIO是否有类似于GlusterFs 的translator类机制？

没有，GlusterFs是使用c语言实现的，而c语言是比较低级的语言，本身没有模块机制。Golang语言自身有强大的模块机制，所以也就不需要类似于translator之类的机制。

MinIO的纠删码机制，为何没有采用柯西矩阵？

就Reed-Solomon纠删码的生成矩阵来说，Klaus的纠删码库里面可以选择柯西生成矩阵。不过当前MinIO软件使用的仍然是范德蒙矩阵的Reed-Solomon纠删算法。这是因为:虽然柯西矩阵的生成相比范德蒙矩阵更快，不过MinIO编码矩阵的生成是只进行一次的操作（程序运行中，生成的这个矩阵会被保存起来）。使用柯西矩阵对数据的吞吐量并没有什么影响。

五 对象存储产品选型讨论

开源对象存储软件以MinIO,Ceph为典型代表。为帮助相关人员在选择对象存储系统之时选择合适的产品，此处对二者的特点、特性做一定讨论。

01 MinIO优势

1 部署极其简单

MinIO系统的服务程序仅有minio一个可执行文件，基本不依赖其它共享库或者rpm/apt包。minio的配置项很少（大部分都是内核之类系统级的设置），甚至不配置也可以正常运行起来。百度、google、bing等搜索引擎上基本没有关于MinIO部署问题的网页，可见在实践中，很少有使用者遇到这方面的问题。

相比之下，Ceph系统的模块，相关的rpm、apt包众多，配置项非常多，难以部署，难调优。某些Linux发行版的Ceph安装包甚至有bug，需要使用者手动改动Ceph的python脚本，才能安装完毕。

2 二次开发容易

MinIO对象存储系统除了极少数代码使用汇编实现以外，全部使用Golang语言实现。Ceph系统是使用业界闻名的难学难用的c++语言编写的。Golang语言由于产生较晚，吸收了很多语言尤其是c++的教训，语言特性比较现代化。相对而言，MinIO系统的维护、二次开发比较容易。

3 网管模式支持多种其他存储

通过网关模式，MinIO对象存储后端，可以对接各种现有的常见其它存储类型，比如的NAS系统，微软Azure Blob 存储、Google 云存储、HDFS、阿里巴巴OSS、亚马逊S3等，非常有利于企业复用现有资源，有利于企业低成本（硬件成本约等于零，部署MinIO对象存储软件即可）地从现有系统平滑升级到对象存储。

02 Ceph优势

•数据冗余策略更加丰富

Ceph同时支持副本、纠删码，而MinIO只支持纠删码。对于个别的对于数据可靠性要求极高的单位，Ceph对象存储更加合适。

•社区目前更成熟

03 其他对比

1 厂商支持

国内使用Ceph的厂商、基于Ceph进行自研的存储厂商都比较多，在使用过程中遇到的问题（有些时候，甚至需要修改、增强乃至重新实现Ceph本身的功能），可以向相关厂商寻求支持。国际方面，Ceph早已被红帽收购，而红帽近期又被IBM收购。

MinIO开发与支持的厂商只有MinIO公司。由于架构比较先进，语言高级，MinIO本身的程序比较容易读懂、修改。招聘Golang程序员来 维护MinIO所花费的成本，显然低于招聘c++程序员来维护Ceph。

2 多语言客户端SDK

二者均有常见编程语言的客户端，比如：python, java等。MinIO对象存储软件的开发SDK另外支持纯函数式的语言Haskell。

3 技术文档

内部实现的文档MinIO基本不存在。想要了解内部实现乃至参与开发的技术人员，只能到如下社区：

http://minio.slack.com/ ，与MinIO的开发人员直接交流，或者自己阅读代码。Ceph的各种实现文档、算法说明文档非常丰富。这方面Ceph要比MinIO成熟很多。

04 结论

由以上讨论，可见作为对象存储软件来说，MinIO， Ceph都非常优秀，各自有各自的优势。准备使用对象存储软件的用户，应该根据自己单位的需求、技术储备等实际情况，选择适当的软件。

六 参考硬件



MinIO是符合软件定义存储SDS理念的，兼容主流X86服务器以及ARM/飞腾平台，同时也可以移植到诸如申威（Alpha架构）和龙芯（Mips架构）等硬件平台。

下面这些符合工业标准的、广泛采用的服务器是经过MinIO inc.优化测试过的、MinIO对象存储软件表现优异的服务器：