分布式存储系统的思考与设计

一. 分布式存储系统介绍

1.什么是分布式存储系统？

首先，它是存储系统，即对数据进行可持久化存储（断电后依然存在)。分布式的意思是大量普通的节点（包括 cpu，内存，磁盘等）通过网络互连，对外作为一个整体来提供服务。

2.为什么需要分布式存储系统？

机构数据量逐年增加，这么庞大的数据量使用单机系统无法有效存储及处理。

3.分布式存储系统是如何做到的？

后面会分不同的系统来介绍其对应的方法。

4.怎样的一个分布式存储系统才是一个好的分布式存储系统（或者说分布式存储系统的目标是什么）？

我对应奥运会的口号总结为三点：更快，更高，更强。

(1) 更快，即性能高，读写数据的速度很快。涉及的技术有条带化（高并发)，负载均衡。

(2) 更高，高，即存储容量高。涉及的技术有可线性扩展，地址管理。

(3) 更强，强，即身强体健，稳若磐石。涉及的技术有副本策略，一致性，故障恢复，并发控制等。

(4) 走群众路线。系统应低成本，易用。

5.分布式存储系统分类？

说到分类，往往会考虑以什么来分？

首先，存储的数据类型有:

(1) 非结构化数据(图片，视频，歌曲，文档等)

(2) 结构化数据（关系数据库中的二维关系表，数据的模式结构和内容分开存储)

(3) 半结构化数据(自描述的,数据的模式结构和内容混在一起，如Html文档)

针对不同的数据存储和处理需求，不同的存储系统会提供不同的服务接口，按接口来分:

(1) 对象存储:即键值存储,其接口就是简单的GET、PUT、DEL等。

(2) 块存储:磁盘块通过底层存储协议访问,开销很小而且没有其它额外的抽象层。

(3) 文件存储:POSIX接口，它跟传统的文件系统如Ext4是一个类型的,对用户友好。

二．分布式存储系统设计

了解完基本概念后,我们来看看如何来设计一个分布式存储系统呢？

想想一个分布式存储系统的首要目标是什么呢？即解决单机存储系统无法解决的问题，大，系统的容量随着节点的不断加入而线性增加。这个的设计与实现不算难，我们简单思考:

(1) 可以直接往系统里增加存储节点，系统容量即这些节点的容量总和。

(2) 还需要一套映射机制，使得我们的读写请求所操作的数据可以被正确映射到相应节点的相应数据块中。

具体怎么实现呢（实现的过程我首先会谈谈自己的想法，下来会分别介绍 lustre 和 ceph 在相应处是如何来设计的）？

首先，增加节点时，我们给每个节点一个唯一的节点标识，这样使得映射机制可以找到他，同样地，节点内基本数据块的寻找可以有另一套机制来完成(一般都是本地文件系统)；节点和节点间通过集群内部的网络进行连接。

其次，映射机制可由单独设立映射节点来管理，这个节点负责为新创建的文件或对象分配相应的地址，并通过一个哈希表把对象标识和地址记录下来。下次要读的时候先通过查表找到地址，再直接过去进行读写。

上面即是我对可扩展性的简单实现想法，可以看到实现不难，其性能瓶颈在映射节点的处理能力以及得到地址

后系统数据的传输能力。下面我们分别来看看 lustre 和 ceph 是如何设计实现的？

1. 可扩展性

1.1 Lustre

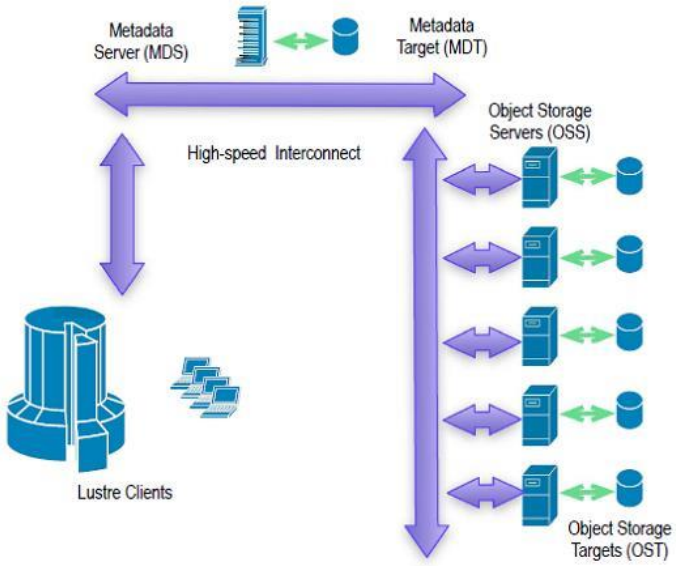


Figure 1: Lustre components.

上图看到的是 lustre 基本的架构，她由

(1) MDS（元数据服务器）。为客户端和 MDT 之间提供桥梁。

(2) MDT（元数据存储对象）。存储元数据（例如文件名，目录，权限和文件布局）。

(3) OSS（对象存储服务器）。为客户端和 OST 之间提供桥梁。

(4) OST（对象存储对象）。存储真实数据。

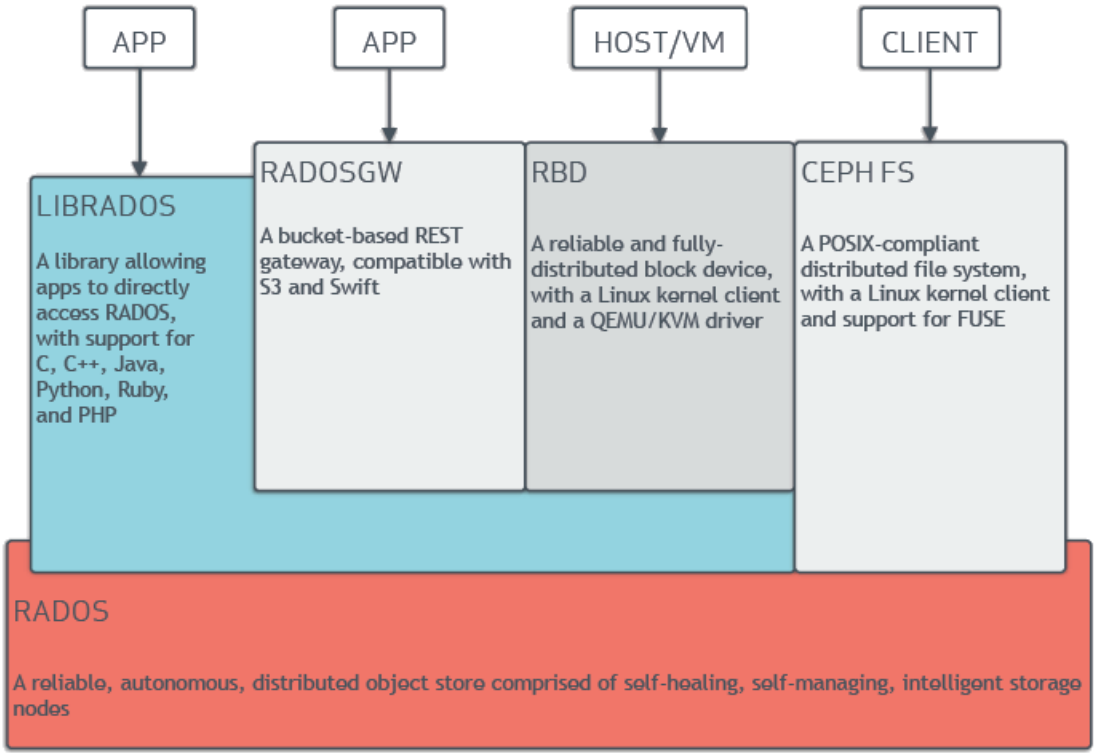
(5) Clients（客户端）。运行 Lustre 客户端软件的计算、可视化、桌面节点。

(6) MGS（管理服务器）。管理 Lustre 文件系统的配置信息。MDS 和 OSS 都可以有多个节点，通过 MGS 进行管理，MGS 只能有一个。

Lustre 客户端软件为 Linux 虚拟文件系统和 Lustre 服务器之间提供了接口。客户端软件包括一个管理客户端（MGC），一个元数据客户端（MDC）和多个对象存储客户端（OSC）。一个客户端软件对应于文件系统中的一个 OST。

所有的文件操作都需要由服务器端的 MDC 询问 MDS 应当与哪些 OST 对话，MDS 返回信息后，服务器端的 OSC 则直接与 OST（由 OSS 负责处理）对话，不需要 MDS 再参与。

1.2 Ceph

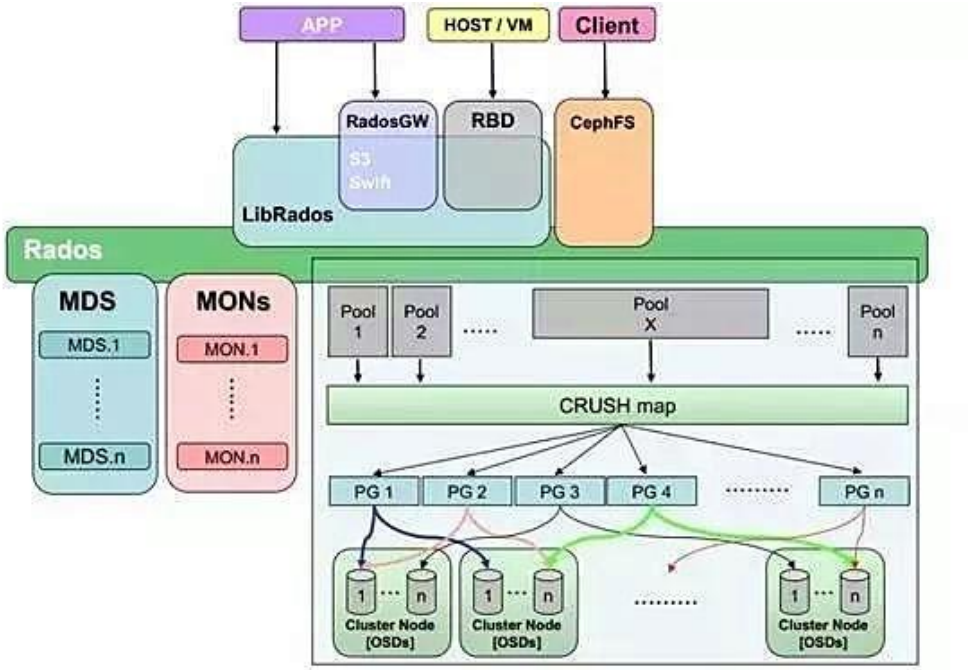


每一个 Ceph 部署的核心就是’Ceph 存储集群’。 集

群主要是由 2 类后台守护进程组成：

Ceph OSD 守护进程：Ceph OSD 为 Ceph 客户端存储数据提供支持。(另外，Ceph OSD 利用 Ceph 节点的 CPU 和内存来执行数据复制、数据再平衡、数据恢复、状态监视以及状态上报等功能。)

Ceph mon 监视器：Ceph 监视器使用存储集群的当前状态维护 Ceph 存储集群映射关系的一份主副本。



Ceph 客户端维护对象 ID 和存储对象的存储池名称，但它们既不需要维护对象到 OSD 的索引，也不需要与一个集中的对象索引进行通信来查找数据对象的位置。为了能够存储并获取数据，Ceph 客户端首先会访问一台 Ceph mon 并得到最新的存储集群映射关系，然后 Ceph 客户端可以通过

提供的对象名称与存储池名称，使用集群映射关系和 CRUSH 算法（可控的、可扩展的、分布式的副本数据放置算法）来计算出提供对象所在的 PG 和主 Ceph OSD，最后，Ceph 客户端连接到可执行读写操作的主 OSD 上进而达到数据的存储与获取。

一个对象的 ID 在整个集群中是唯一的，（全局唯一）而不仅仅是本地文件系统中的唯一。Ceph OSD 将所有数据作为对象存储在扁平结构的命名空间中（例如，没有目录层次结构）。对象在集群范围内具有唯一的标识、二进制数据、以及由一组名称/值的键值对组成的元数据。而这些语义完全取决于 Ceph 的客户端。例如，Ceph 块设备将块设备镜像映射到集群中存储的一系列对象上。由唯一 ID、数据、名称/值构成键值对的元数据组成的对象可以表示结构化和非结构化数据，以及前沿新的数据存储接口或者原始老旧的数据存储接口。

ID	Binary Data	Metadata
1234	01010101010101001101010010 01011000010101001101010010 01011000010101001101010010	name1 value1 name2 value2 nameN valueN

Ceph 存储集群通过‘存储池’这一逻辑划分的概念对数据对象进行存储。可以为特定类型的数据创建存储池，比如块设备、对象网关，亦或仅仅是为了将一组用户与另一组用户分开。从 Ceph 客户端来看，存储集群非常简单。当有 Ceph 客户端想读写数据时（例如，会调用 I/O 上下

文），客户端总是会连接到存储集群中的一个存储池上。客户端指定存储池名称、用户以及密钥，所以存储池会充当逻辑划分的角色，这一角色使得对数据对象访问进行控制。

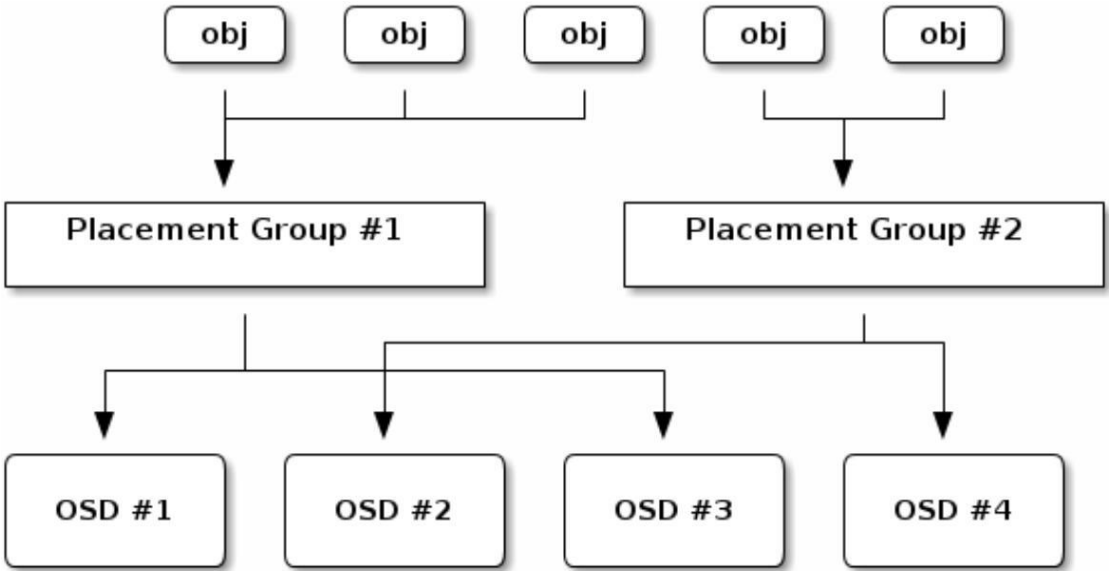
考虑一个问题，在 EB 规模的存储集群中，一个 Ceph 存储池可能会存储数百万或更多的数据对象。Ceph 必须处理数据持久化（副本或纠删码数据块）、清理校验、复制、重新再平衡以及数据恢复，因此在每个对象基础上的管理就会出现扩展性和性能上的瓶颈。Ceph 通过散列存储池到 PG 的方式来解决这个瓶颈问题。CRUSH 则分配每一个对象到指定的 PG 中，每个 PG 再到一组 OSD 中。

在 ceph 的映射部分中，我们着重了解一下 crush 算法和 PG，他们也是 ceph 的核心之处。

1.2.1 PG

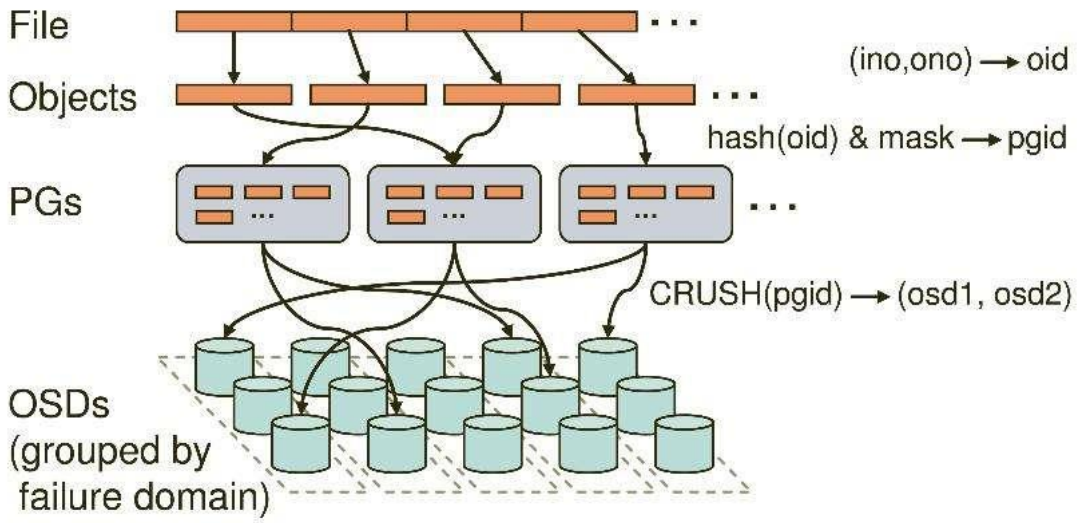
Ceph 将存储池分片处理成在集群中均匀且伪随机分布的 PG。CRUSH 算法将每个对象分配到一个指定的 PG 中，并且将每个 PG 分配到对应的 Acting Set 集合中一也就是在 Ceph 客户端和存储对象副本的 OSD 之间创建一个间接层。如果 Ceph 客户端直接就能知道对象存放到具体的哪个 OSD 中的话，那么 Ceph 客户端和 Ceph OSD 之间耦合性就太强了。相反的，CRUSH 算法会动态的将对象分配到 PG 中，然后再将 PG 分配到一组 Ceph 的 OSD 中。有了这个间接层之

后，当新 Ceph OSD 加入或者 Ceph OSD 出现问题时，Ceph 存储集群就可以动态的进行数据再平衡。通过在数百到数千个放置组的环境中管理数百万个对象，Ceph 存储集群可以高效地增长和收缩以及从故障中恢复。



相对整体集群规模来说，如果存储池设置的 PG 较少，那么在每个 PG 上 Ceph 将会存储大量的数据；如果存储池设置的 PG 过大，那么 Ceph OSD 将会消耗更多的 CPU 与内存，不管哪一种情况都不会有较好的处理性能。 所以，为每个存储池设置适当数量的 PG，以及分配给集群中每个 OSD 的 PG 数量的上限对 Ceph 性能至关重要。PG 是对象的集合， 在同一个集合里的对象放置规则都一样（比如同一集合中的对象统一都存储到 osd.1，osd.5. osd.8 这几台机器中）；同时，一个对象只能属于一个 PG，而一个 PG 又对应于所放置的 OSD 列表；另外就是每个 OSD 上一般会分布很多个 PG。

1.2.2 Crush



Ceph 会将 CRUSH 规则集分配给存储池。当 Ceph 客户端存储或检索存储池中的数据时，Ceph 会自动识别 CRUSH 规则集、以及存储和检索数据这一规则中的顶级 bucket。当 Ceph 处理 CRUSH 规则时，它会识别出包含某个 PG 的主 OSD，这样就可以使客户端直接与主 OSD 进行连接进行数据的读写。

Ceph 客户端经过以下步骤来计算出 PG ID 信息:

(1) 客户端输入存储池 ID 以及对象 ID (例如，存储池 pool=”liverpool”，对象 ID=”john”)。

(2) CRUSH 获取对象 ID 后对其进行 HASH 编码。

(3) CRUSH 根据上一步的 HASH 编码与 PG 总数求模后得到 PG 的 ID。(例如 HASH 编码后为 186，而 PG 总数为 128，则求模得 58，所以这个对象会存储在

PG_58 中；另外这也可以看出 PG 数对存储的影响，因为涉及到对象与 PG 的映射关系，所以轻易不要调整 PG 数)

(4) CRUSH 计算对应 PG ID 的主 OSD。

(5) 客户端根据存储池名称得到存储池 ID (例如”liverpool”=4)。

(6) 客户端将 PG ID 与存储池 ID 拼接 (例如 4.58)

(7) 客户端直接与 Activtin Set 集合中的主 OSD 通信，来执行对象的 IO 操作 (例如，写入、读取、删除等)。

2. 可靠性

了解完扩展后，接下来我们回想还需要什么呢？是可靠性重要还是高性能？可靠性即保证了数据存取的正确性，如果正确性不能保证，那再快又有什么用呢？所以接下来我们来谈谈如何来保证数据的正确性—即可靠性。

2.1 容错性

可靠性即容错性，当某些设备发生错误（故障）时，整个系统的运行不受影响。那错误是怎么来的呢？

根据 google 统计，我们了解到

表 3-1　Google 某数据中心第一年运行故障

发生频率	故障类型	影响范围
0.5	数据中心过热	5 分钟之内大部分机器断电，一到两天恢复
1	配电装置（PDU）故障	大约 500 到 1000 台机器瞬间下线，6 小时恢复
1	机架调整	大量告警，500~1000 台机器断电，6 小时恢复
1	网络重新布线	大约 5% 机器下线超过两天
20	机架故障	40 到 80 台机器瞬间下线，1 到 6 小时恢复
5	机架不稳定	40 到 80 台机器发生 50% 丢包
12	路由器重启	DNS 和对外虚 IP 服务失效约几分钟

（续）

发生频率	故障类型	影响范围
3	路由器故障	需要立即切换流量，持续约 1 小时
几十	DNS 故障	持续约 30 秒
1000	单机故障	机器无法提供服务
几千	硬盘故障	硬盘数据丢失

即当在大规模分布式存储系统中，小概率的设备故障事件会经常发生，而当这些设备故障时其自然无法提供服务了。如上表所示，单机故障和磁盘故障发生的概率最高。

分布式存储系统中，数据都被分割成最基本的数据块（或对象）来存储。经过思考发现我们可以为每个数据块保存几个副本，并将这几个副本存在不同的故障区域中（即不同的机架，节点，磁盘等），并记录每个副本的位置，当一个副本故障，我们可以去请求其他副本来完成读写（如果所有的副本都故障，那只能说明你运气忒不好，这样的情况概率也是极低的）。

有人考虑到这样做太浪费存储空间（比如考虑 3 副本策略，集群中 70%的空间用来存储用户数据，则有效存储容量为 70%/3，约为 23%），于是结合线性代数（范德蒙德矩阵，位矩阵，柯西矩阵等）提出纠删码（RAID）。纠删码首

先将数据分块，并根据分块数据进行编码生成 N 个校验数据块，并将数据块存于不同的故障域中，其最高可忍受 N 个块故障。纠删码通过数学变换，将额外的消耗空间控制在 1 倍以内，而多副本策略则要消耗 m 倍的额外空间。在节省空间的同时，纠删码增加了系统的复杂性，并需要额外的计算力。

事实上，不同的存储需求对应不同的备份策略。下面我们来看看两大存储系统分别是如何解决这个问题的。

2.1.1 Lustre

Lustre 在文件系统级只提供了服务器故障切换，即暂不提供数据冗余，它依赖于备用存储设备的冗余性。备用 OST 存储应为 RAID 5，或最好为 RAID 6。MDT 存储应为 RAID 1 或 RAID 10。

服务器故障切换通过共享存储来实现，如 SAN (Storage Area Network，存储区域网络)：通过光纤交换机或者以太网交换机把服务器和存储设备连接在一起，实现多服务器共享访问或使用一个存储阵列或集群存储。

其故障切换配置方式包括：

- `主动/被动` 对： 主动节点提供资源并提供数据，而被动节点通常闲置。如果主动节点发生故障，则被动节点将接管并激活。
- `主动/主动` 对：两个节点都处于活动状态，每个节

点都提供一个资源子集。在发生故障的情况下，第二个节点从故障节点接管资源。

在 Lustre 2.4 之前的版本中，可将 MDS 配置为`主动/被动` 对，而 OSS 可部署在`主动/主动` 配置中，以提供冗余且避免额外开销。

Lustre 软件仅在文件系统级别提供故障切换功能。在完整的故障切换解决方案中，系统级组件的故障切换功能（如节点故障检测或电源控制）必须由第三方工具提供。

2.1.2 Ceph

Ceph 中对两种副本策略都有应用。在以前的老版本中，一个存储池只是简单的维护对象的多个深拷贝。而现在，Ceph 能够维护一个对象的多个副本，或者能够使用纠删码。正因为保证数据持久化的 2 种方法（副本方式与纠删码方式）存在差异，所以 Ceph 支持存储池类型。存储池类型对于客户端也是透明的。

Crush 算法的计算过程中天然地考虑到集群的层次故障域，通过 crush 规则集合可指定数据块放置到对应的故障域级别中。

(1) 多副本

和 Ceph 客户端一样， Ceph OSD 也是通过与 Ceph mon 交互来获取到最新的集群映射关系。Ceph OSD 也使

用 CRUSH 算法，但是用这个算法是用来计算对象的副本应该存储在什么位置（客户端用 CRUSH 是用来找主 OSD 以及计算出 Acting Set 列表，而 OSD 用 CRUSH 则是主 OSD 定位对应的副本是谁)。

(2) 纠删码

和副本存储池类似，纠删码存储池也是由 Acting set 列表中的主 OSD 来接收所有的写操作。在副本存储池中，Ceph 对 PG 中的每个对象在从属 OSD 上都会有一份一样的数据对象；而对于纠删码存储池来说，可能略有不同。每个纠删码存储池都会以 K+M 个块来存储每一个对象。对象(的数据内容)会被切分成 K 个数据块以及 M 个编码块。纠删码存储池创建时也需要配置成 K+M 的大小\(size)以便每个块都可以存储到 Acting Set 列表中的每个 OSD 上。对象的属性存储这些块的等级。主 OSD 负责数据划分到 K+M 个块的纠删编码以及将这些编码信息发送到其它的 OSD 上。同时主 OSD 也会维护 PG 的权威日志（权威日志实际是一种进度控制机制，尤其当某些节点出现问题时，可以根据权威日志进行数据的恢复)。

纠删码存储池迟迟未能达到商用水准，究其原因:

(1) 相较于多副本而言，实现更复杂。

每个逻辑 PG 映射出来的 PG 实例数目要比副本存储池多（因为纠删码条带中的每个块都需要通过 PG 存储到不

同故障域的OSD中来保证可靠性)，这给PG的管理带来更大挑战。

(2) 性能更差

存储系统的读性能至关重要。而任何客户端的读都会被纠删码转化为集群内多个跨节点的读，且跨节点读的个数和k值（数据块个数）成正比。而跨节点的传输时延和协同消耗使其性能必然较差。其最适合的应用场景为追加写和删除。

当提出副本备份策略时，自然而然的抛出一个问题：如果各个副本之间的数据不一样，那用户读写可能会发生意想不到的错误，那如何保证各个副本之间的数据是一样的呢？此即一致性问题。

2.2 一致性

可以从多个角度理解一致性。

从用户角度（假设有三个用户a， b， c）:

- 强一致性：A写入值到存储系统，系统保证b， c读取操作返回最新值。
- 弱一致性：不保证b， c读取操作返回最新值。
- 最终一致性：A写入值到存储系统，系统保证在后续没有操作更新相同值得情况下， b， c读取操作返回

最新值。

从存储系统角度看：

- 副本一致性
- 更新顺序一致性：多个副本之间按相同的顺序执行操作。

CAP理论：一致性，可用性和分区可容忍性不能同时满足。分区可容忍性必须满足，故需要在一致性和可用性之间做出权衡。

根据不同的存储需求，我们有不同的权衡方法。有的应用需要极致的安全性，不能忍受数据丢失，如银行的操作数据；有的性能更加重要，允许小部分小概率丢失，如视频流，音频流数据。提出三种模式：

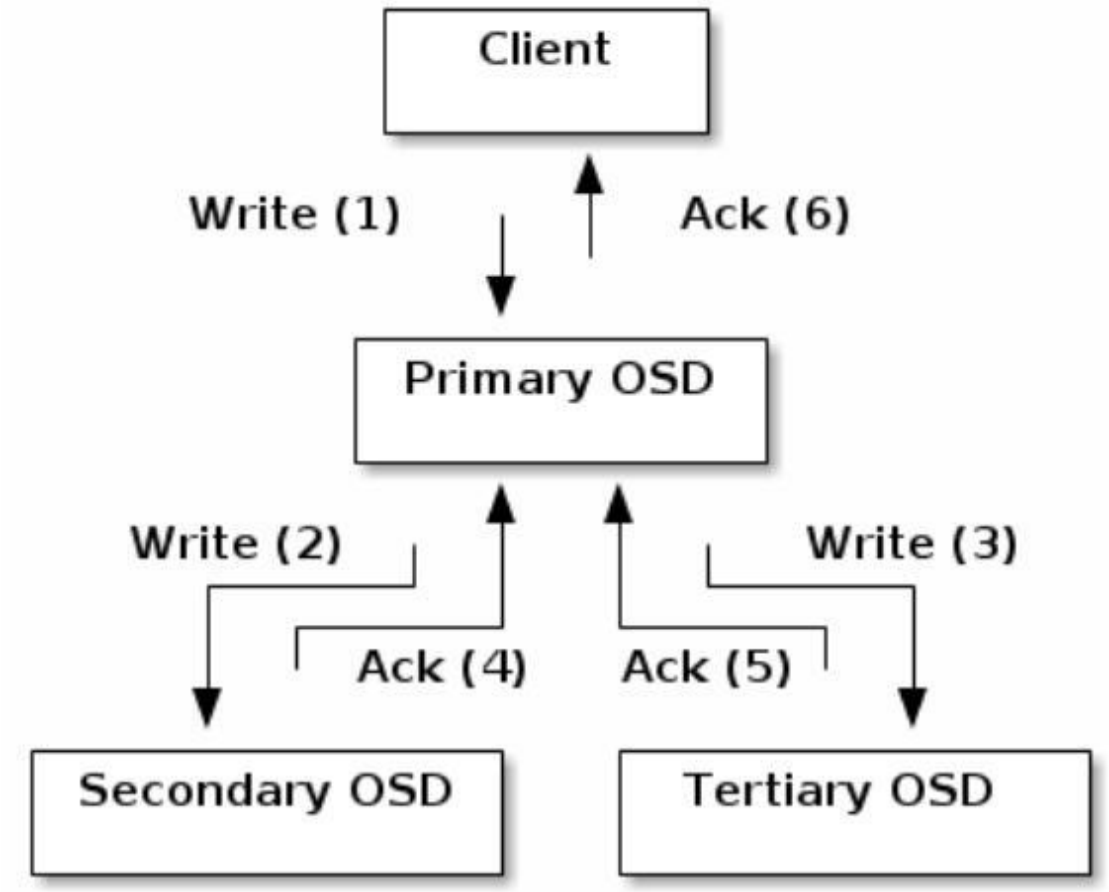
- 保护模式：强同步复制，写操作要求主副本先将操作日志同步到各个从属副本后，收到确认消息并且自身也完成了写操作，才返回写成功。
- 性能模式：异步复制，写操作只要在主副本上执行成功，就可以返回写成功。主副本上的后台线程会通过异步的方式将操作日志复制到从属副本，这个过程可能丢失数据。
- 可用模式：上述两种的折衷。正常情况下为保护模式，当主备副本之间网络出现问题则切换为性能模式。

2.2.1 Lustre

Lustre 中备份策略由磁盘阵列来维护，因此不涉及到一致性的问题。

2.2.2 Ceph

保护模式：在典型的写操作场景下，Ceph 客户端使用 CRUSH 算法计算对象所在的 PG ID 以及 Acting Set 列表中的主 OSD，当客户端将对象写到主 OSD 时，主 OSD 会查看这个对象应该存储的副本个数 (例如，osd_pool_default_size = n)，然后主 OSD 根据对象 ID、存储池名称、集群映射关系这些信息再根据 CRUSH 算法来计算出 Acting Set 列表中的从属 OSD (译者注：除列表中第一个 OSD 外，其它的都是从属 OSD)。主 OSD 将对象写入从属 OSD 中，当主 OSD 收到从属 OSD 回复的 ACK 确认并且主 OSD 自身也完成了写操作后，主 OSD 才会给 Ceph 客户端回复真正写入成功的 ACK 确认。



Ceph 默认也是一个对象保持 3 副本的设置，写操作则要求至少 2 个副本为 clean 状态。如果有 2 个 OSD 出现问题，Ceph 仍然可以保留数据不会丢失，但是就不能进行写操作了。在纠删码存储池中，Ceph 需要多个 OSD 来存储对象分割后的块以便在降级状态仍然可以操作。与副本存储池类似，理想情况下，在降级状态下纠删码存储池也支持 Ceph 客户端进行读写操作。基于此，我们则建议设置 K+M=5 通过 5 个 OSD 来存储块信息，同时设置 M=2 以保证即使 2 个 OSD 出现问题也可以根据剩余的 OSD 进行数据的恢复重建。

2.3 故障恢复

当集群中的设备出现故障时，备份策略可以保证有其他的备份来满足客户端的读写请求。但是这还不够，如果对出现故障的设备撒手不管，不仅是对资源的浪费，也会对集群的管理带来问题。所以我们需要故障恢复机制来对那些出现故障的设备进行处理。

故障恢复首先要求系统能够检测出哪些设备出现故障了。最简单的想法就是控制节点每隔固定时间向每个工作节点发去问候（即所谓的心跳），如果工作节点正常工作则还以问候；否则，控制节点重试一定次数后还没有收到回应，则认为该工作节点发生了故障。但是我们想想，如果是两个节点之间交流的网络发生了故障，或者说工作节点工作太拼命而无暇搭理控制节点（即负载过高），此时岂不是冤枉了工作节点对他下了斩立决令。所以这种方法不能确保工作节点是不是发生了故障。此时，如果工作节点实际上没有故障，而控制节点认为他发生了故障，会导致多台节点服务同一份数据而产生不一致的问题。

此时，我们发现关键之处在于当控制节点认为工作节点出现故障时，系统能保证工作节点一定是不在服务了（不管他是由于什么原因，诸如网络原因，过于繁忙等），就算他可以服务也得强制下线。

所以我们反过来设计，由工作节点定期向控制节点发送消息，那发送的是什么消息呢？续租的消息。控制节点给工作

节点一份租赁合同，合同期限内工作节点可以服务相应的数据，当合同快到期时由工作节点向控制节点发出消息进行续租。正常情况下，工作节点可以一直发消息来续租（又不要钱)；当工作节点故障或其他原因（上述）没有及时续约（即控制节点会等一个固定的时间段)，那么租约过期，工作节点主动停止服务，控制节点也会认为工作节点不再提供服务。

当我们可以检测出哪些节点故障时，怎么来执行故障恢复？

由于内存的存储介质是易失性的，即断电后数据丢失。如果程序执行到一半发生故障，则数据丢失。所以需要可持久化的存储介质来恢复故障，可以考虑将系统每一步的操作记录下来并存放到持久化介质如磁盘中，恢复时通过重新做一遍这些操作（即重放）来恢复。在分布式系统中，这种方式叫做操作日志。同时，比如在数据库中，如果每次操作都持久化到磁盘（随机更新磁盘的某个数据块)，系统性能会很差。通过这种操作日志的方式来顺序记录每个操作并在内存中执行这些操作，内存中的数据定期刷新到磁盘，将随机写转化为顺序写。注意到，我们应先写操作日志，再修改内存中的数据。如果先写内存后写日志，当写完内存后，用户可以立刻督导修改后的结果，一旦在写内存和写日志间发生故障，则最近的修改操作无法恢复，产生不一致的问题。

节点故障又可分为几种情况。

临时/永久故障：即由控制节点等待一个固定时间，如果

其再次上线，则为临时性；如果没有上线，则为永久性。临时性又分该节点掉线期间其他备份节点是否更新（为每个节点分配版本号，更新则版本号加一），若重新上线后发现版本号较低，则执行垃圾回收，增加新的副本。永久性即为备份数据分配新的节点。

主/副节点故障：当副备份节点故障，由主备份节点来完成日志传输工作；而当主节点故障，通过 Paxos 选举新的节点为主节点。

控制节点通过镜像的方式来容错和恢复。

2.3.1 Lustre

Lustre 没有提供故障恢复机制，通过外部软件来提供。

2.3.2 Ceph

Ceph 的故障恢复同上所述。

考虑到多个用户可能对同一个数据对象进行读写，如果不加控制则会出现数据不一致的情况。

2.4 并发控制

（1）锁：读锁以及写锁。允许对同一个元素加多个读锁，但只允许加一个写锁，写事务将阻塞读事务。元素可以是一行，也可以是一个数据块甚至一个表格。

（2） 写时复制：读事务占的比例往往远远超过写事务，很多应用的读写比例达到 6:1，甚至 10:1。写时复制（Copy-On-Write,COW）读操作不用加锁，极大地提高了读取性能。

1）拷贝:将从叶子到根节点路径上的所有节点拷贝出来。

2）修改：对拷贝的节点执行修改。

3） 提交：原子地切换根节点的指针，使之指向新的根节点。

如果读操作发生在第 3 步提交之前，那么，将读取老节点的数据，否则将读取新节点，读操作不需要加锁保护。写时复制技术涉及引用计数，对每个节点维护一个引用计数，表示被多少节点引用，如果引用计数变为 0，说明没有节点引用，可以被垃圾回收。

2.4.1 Lustre-分布式锁管理器模型 (DLM)

DLM在很大程度上借鉴了传统的分布式锁管理器的设计理念， 其基本模型在 Lustre 文件系统中被称为普通锁(Plain Lo ck)。Lustre 锁管理器也有一些它特有的新特征。它对普通锁模型进行了扩展,引入了两种新类型锁:意图锁(Intent Lo ck)和范围锁(Extent Lock)。

Lustre DLM 模型使用锁命名空间(Lock Namespace)来组织和管理共享资源(Re sour ce)以及资源上的锁。当要获得某个资源的锁时，先要将该资源在锁命名空间中命名。

任何资源都属于一个锁资源命名空间， 而且都有一个相应的父资源(根资源除外，其父资源一般设置为 0)每个锁资源都有一个资源名，该资源名对于其父节点是唯一的。所有的锁资源组成一个锁资源树型结构。当要获得某个资源的锁

时，系统必须首先从该资源的所有祖先获得锁。

Lustre 支持六种锁模式：

模式	名称	访问授权	含义
EX	执行	RW	允许对资源的读写访问，且其他任何进程不能获得读或写权限
PW	保护写	W	允许对资源的写访问，且其他任何进程不能获得写权限
PR	保护读	R	允许对资源的读访问，且其他任何进程不能获得写权限但可共享读
CW	并发写	W	对其他进程没有限制，可以对资源并发写访问.无保护方式写
CR	并发读	R	对其他进程没有限制，可以对资源并发读访问.无保护方式读
NL	空	无	仅仅表示对该资源有兴趣，对资源没有访问权限

每种锁模式都有一定的兼容性，如果一个锁可以与已经被授权的锁共享访问资源，则称为锁模式兼容。表 2 是锁兼容性转换表，“1”表示锁模式可以并存，从表中可以看出，该表是对称的。其中，执行锁严格性最高，兼容性最差;而空锁严格性最低，兼容性最好。

锁模式的兼容性

	EX	PW	PR	CW	CR	NL
EX	0	0	0	0	0	1
PW	0	0	0	0	1	1
PR	0	0	1	0	1	1
CW	0	0	0	1	1	1
CR	0	1	1	1	1	1
NL	1	1	1	1	1	1

在 Lustre 锁管理器中，锁命名空间中每个资源都维持着三个锁队列：

(1)授权锁队列:该队列包含所有已被锁管理器授权的锁，但正在转换为与已授权锁模式不兼容的锁除外，该队列的锁的持有者可以对资源进行访问。

(2)转换锁队列:该队列的锁已经授权，但试图转换的锁模式与当前授权队列中的锁模式不兼容，正在等待其他锁的释放或降低访问权限。锁管理器按照 FIFO 的顺序处理转化队列中的锁。

(3)等待锁队列:该队列包含所有正在等待授权的新锁请求。该队列中的锁模式与已授权的锁模式不兼容。锁管理器按 FIFO 的顺序处理等待队列中的锁。新锁请求只有在三个锁队列为空或者请求的锁模式与该资源所有的锁模式兼容时才会被授权，否则就要加入到等待锁队列中。

意图锁：在 Lustre 文件系统中，意图锁主要用于文件

元数据的访问，它通过执行锁的意图来减少元数据访问所需的消息传递的次数，从而减少每次操作的延迟。当客户端向 MDS 发送元数据操作请求时，在请求中指明操作的意图，在交付给锁管理器处理之前先由 MDS 执行锁的意图，然后返回不同的锁资源。例如，在客户端请求创建一个新文件时，首先将该请求标志为文件创建的意图(IT CREATE)，然后必须请求从 MDS 获得它的父目录的锁来执行 lo okup 操作，如果锁请求被授权，那么 MDS 就用锁请求指定的意图来修改目录，创建请求的文件，成功之后并不返回父目录的锁，而是返回新创建文件的锁给客户端，整个过程如图 3 所示。而传统的文件创建操作要先发送 lookup 请求给服务器锁定父目录，然后再发送 create 请求创建文件。与传统的操作相比，Lustre 文件系统的整个操作只需要一次 RPC 调用，大大减少了访问延迟。

范围锁：在 Lustre 文件系统，范围锁主要用来维护细粒度的文件数据并发访问。与普通锁不同的是，范围锁增加了一个表示获得授权的锁定文件范围的域。

2.4.2 Ceph

独占锁提供一种功能特性：任一客户端可以对 RBD 中资源进行’排它的’锁定（如果有多个终端对同一 RBD 资源进行操作时)。这有助于解决当有多个客户端尝试写入同一对象时发生冲突的场景。此功能基于前一节中介绍的对象的监视

与通知。因此，在写入时，如果一个客户端首先在对象上建立独占锁，那么其它的客户端如果想写入数据的话就需要在写入前先检查是否在对象上已经放置了独占锁。设置了这一特性的话，同一时刻只有一个客户端能够对 RBD 资源进行修改，尤其像快照创建与删除这种改变 RBD 内部结构的时候。

3. 高性能

在保证可扩展和可靠性的基础上，我们就可以来谈一谈高性能了。

如果完全只对一块磁盘进行读写操作的话，受限就比较多：磁头的移动（例如每次 6ms 的寻址时间开销）、设备的带宽（例如每秒最大 100MB）等。如果我们把数据分成块（对象），将每个块分配到不同的磁盘中，那么可以通过并发地读写这些块来提升性能，此即条带化。Lustre 和 Ceph 均对数据做了条带化。

考虑到当某个节点负载过高时，访问它上面的数据时速度相应会很慢，故需要将它服务的部分数据迁移到其他较空闲的节点，此即负载均衡。

Lustre 中并没有实现自动的负载均衡，需要用户自己去分析和迁移。Ceph 的 crush 算法会针对每个设备的权重（存储容量）来决定当其的数据应放到哪个设备上，这就是预先考虑到负载均衡而设计的机制。首先是，分配的时候就尽可能

往容量大的节点分，尽量避免以后的迁移；二是当新节点加入进来或要删除旧节点时，都不会导致数据在处被添加或删除之外的两个节点之间进行迁移。