smartx
MAKE IT SIMPLE

# SMTX ZBS 5.0.0

# 块存储服务技术白皮书

发布日期: 2021-11-22
文档更新: 2022-04-02

MAKE IT SIMPLE

# 目录

# 扉页

## 版本信息

| 软件版本号： | 5.0.0 |
| --- | --- |
| 发布日期： | 2021-11-22 |
| 类型： | 公开 |

## 版权信息



## 关于本文档

SmartX 会定期发布产品的新版本。因此，对于当前使用的某些版本，本文档中介绍的一些功能可能不受支持。有关产品功能的最新信息，请参阅相关产品的发行说明。如果您的 SmartX 产品未提供本文档所述的功能，请联系 SmartX 以获取硬件升级或软件更新。

## 关于文档的意见和建议

您的建议有助于我们提高本文档的准确性、组织结构等，欢迎通过邮件将您对本文档的意见发送到 info@smartx.com。

# 第 1 章

# SMTX 块存储服务技术白皮书

本白皮书介绍了 SmartX 块存储服务的架构设计、核心组件、存储数据结构、数据管理和数据保护原理等。

SmartX 设计研发了一套高性能、高可靠的分布式块存储服务系统（内部项目代号为 ZBS）。为方便叙述和理解，下文中可能根据语境交替使用“ZBS” 或“块存储服务”来指代这套系统。

## 基本概念

为了更好地理解 ZBS 的技术特点，首先介绍几个相关的基本概念：传统数据中心架构、软件定义的存储和超融合架构。

### 传统数据中心架构

数据中心在过去的几十年间发生了巨大变化，先后经历了大型机、独立服务器、集中存储、虚拟化技术的出现和成熟等主要阶段。这些不断演进的技术，相当好地解决了各个时代数据中心所面临的问题，然而当随着时代的进步和科技的发展，这些传统的技术，尤其是存储方面的技术，在应对爆炸式增长的海量数据时，越来越显示出不足之处。如今大量使用的仍然是传统数据中心架构：一般由负责计算的服务器加外部的存储阵列。存储设备之间通过存储区域网络（Storage Area Network - SAN）进行数据传输和通信。这样的架构在应对海量数据时，有如下几个主要缺点：

1. 计算和存储分离。计算资源位于服务器，当计算资源需要访问数据时，必须通过存储网络进行，大大增加了延迟。
2. 传统的存储设备上都有存储控制器，所有的数据读写都要经它处理，而它的读写吞吐能力远远低于计算服务器处理器的读写吞吐能力。当计算服务器的处理器对存储设备提出读写请求时，尤其是大量的读写请求时，由于存储控制器的瓶颈，使得整个系统响应读写的速度大大降低。
3. 同时也因为上述原因 2，计算服务器的处理器需要长时间等待读写操作完成后才能进行下一步操作，导致其利用率大大降低。
4. 大量使用专属存储硬件，且控制这些专属硬件的软件与硬件高度耦合。因为是专属硬件，没有统一的业界标准，不同厂商的设备很有可能互不兼容。遇到不得不更新设备以适应新的业务需求时，投入相当巨大；
5. 各种专属硬件存储设备都有自己的管理机制和界面，使得管理变得复杂；
6. 不适合进行横向扩展。横向扩展（亦称线性扩展），简言之就是用更多的节点支撑更大量的请求。由于传统存储设备上的存储控制器能控制的 CPU、内存和存储插槽有限，所以存储空间的扩展也大大受限。

由此可见，传统的存储方式成为数据中心性能得以提高的瓶颈。为了解决上述的缺点，出现了软件定义的存储和超融合架构的概念。

### 软件定义的存储

软件定义的存储的核心是存储系统的管理和智能层面与底层硬件分离，将存储硬件中关键的控制、处理功能（如存储控制器、去重、压缩等）抽出来由软件实现。这种方式使得存储服务可以运行在通用的存储硬件上，不再需要专属硬件。这就意味着，存储控制软件不仅可以控制甲厂商的存储设备，也可以控制乙丙丁等其他厂商的存储设备。由于数据的可用性由软件提供，而不是冗余的硬件设备提供，这大大降低了投入成本，原因在于：（1）昂贵的专属硬件被便宜得多的通用硬件代替；（2）许多原来由硬件实现的功能现在由软件实现，省去了购买这部分硬件的成本。

存储服务通常包括快照、去重、多副本、克隆、压缩、精简配置等，这些功能在传统存储中由硬件实现；而软件定义的存储，则由软件提供这些服务，从而不再跟专属硬件高度耦合，而且可以通过简单的软件升级轻松提升已有功能和添加新的功能。因此，存储的灵活性和适应性大大提高。

软件定义存储以虚拟技术和池化存储资源为基础。分布在同一集群的不同服务器上的存储资源能够方便地组成一个逻辑上的存储池供用户使用和管理，仿佛使用一块大的本地存储盘。

## 超融合架构

简单地说，超融合架构是节点间没有明确的计算和存储的分工，在同一个服务器硬件资源（目前主要是 x86 服务器）上实现核心的存储和计算功能，封装为单一的、高度虚拟化的解决方案。之所以能把计算和存储资源集成到同一台服务器中，其基础是近些年处理器技术和存储设备硬件本身性能的大幅度提高。

超融合架构的核心特征和优势：

1. 计算、存储和网络等资源融合在同一台服务器上，所以数据的读取可以优先在本地存储进行，只有当本地存储没有相应数据时，才会从远程节点读取，有效地减少跨网络读取数据的开销。
2. 软件定义的存储。
3. 读写性能的大幅度提高，使得计算服务器的处理器的使用率也得到大幅度提升。
4. 不再使用或很少使用专属硬件，大大降低投入成本。
5. 一个统一的管理入口，管理各种资源。
6. 方便进行横向扩展。横向扩展时，是通过增加新的节点来满足对存储的需求。一般来说，一个节点就是一台 x86 服务器。这种扩展的最大优点在于，在扩大存储容量的同时，由于 x86 服务器还含有 CPU、内存以及存储插槽等资源，所以扩展后的集群，性能也同时得到了提升。值得注意的是，这种性能的提升，可以根据实际需要实现精细化而变得更灵活：例如当希望计算能力得到大幅度提升时，可以偏重增加计算资源；当希望存储能力得到大幅度提升时，可以偏重增加存储资源。得益于以通用硬件为基础的软件定义架构，在集群部署完成后，可以通过节点硬件资源的动态热插拔实现节点角色/集群能力的弹性调整以应对不断变化的业务需求。前面讲过，传统存储的扩展受制于其存储控制器能力的上限，而超融合结构由于使用了软件定义的存储，其存储控制器由软件实现，每个节点上都可以方便地运行存储控制器，这种机制使得集群的节点数的扩展不再受制于存储控制器。横向扩展可以根据实际业务需求，从小规模扩展开始，因此投入也较低，用户也不必提前预估数年后对存储的需求。

对比下传统的存储架构，可以发现，超融合架构基本避免或者大大缓解了传统存储架构存在的问题。

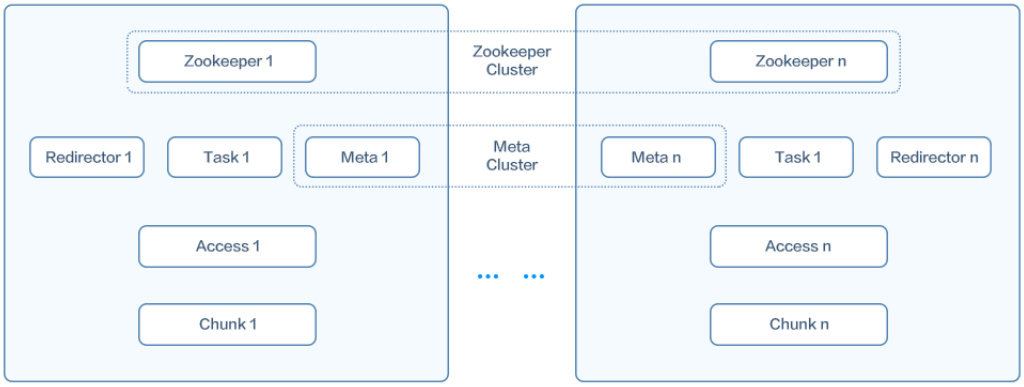
# ZBS 简介

ZBS 充分利用了虚拟化平台特点以及 SSD 设备的特性，进行了深入地优化与整合，使得 ZBS 在面对大规模虚拟化的场景中，和传统集中式存储以及其他分布式存储相比，占有绝对的优势。ZBS 具有以下功能和优点：

1. 高性能
   - 以 SSD 设备作为缓存，充分利用 SSD 低延迟、高 IOPS 的特性；
   - 支持超融合架构：计算和存储运行在同一物理服务器上，存储对计算感知，大部分 IO 可以在本地完成，避免网络延迟。
2. 高可靠
   - 通过多副本，保证在服务器宕机时，数据不会丢失；
   - 多副本采用强一致性写，保证数据一致性。
3. 低消耗
   - 兼容商用 x86 服务器，无特殊硬件需求；
   - 存储仅占用有限的计算资源，避免额外购置存储服务器，完美适配超融合架构。
4. 横向可扩展
   - 性能与容量同步线性扩展；
   - 单集群最大可支持 255 个节点。
5. 易于集成
   - 提供标准的 NFS 及 iSCSI 接口，完整的 RESTful API 支持;
   - 支持 KVM、VMware ESXi、XenServer 等虚拟化/云平台。

# 架构与核心组件

ZBS 的架构图如下：




## ZooKeeper

ZooKeeper 是分布式系统中常用的一个组件，用于在分布式系统中实现一致性。ZooKeeper 可以避免因网络故障而导致的脑裂问题。

脑裂是指本来为一整体、动作协调的集群，由于网络故障而分裂成为两个独立的个体。此时尽管实际上各节点仍然正常工作，但由于节点间相互失去了联系，都以为是对方出了故障，因此多个节点有可能会同时读写数据，进而导致数据被损坏。

ZooKeeper 之所以能够避免脑裂，在于其机制规定：必须在节点中选出一个作为 Leader，且任何时候一个集群中只能有一个 Leader。假设有 n 个节点的集群，如果发生脑裂分为两部分，则含有**多于** n/2 个节点的“子群”会投票选出一个新的 Leader （如果原来的 Leader 不在这个“子群”中的话）继续协调 ZooKeeper 服务。如果原来的 Leader 正好在这个“子群”中，则无需重选 Leader。

运行 ZooKeeper 的节点称为主节点（Master），每个主节点上只能运行一个 ZooKeeper 实例。在实际部署中，根据集群规模的不同，可采用三个、五个主节点的部署方式。集群规模不超过四个节点时，必须使用三个主节点的部署方式，此时可以容忍集群中任意一个主节点宕机；当集群规模大于或等于五个节点时，必须使用五个主节点的部署方式，此时可以容忍集群中任意两个主节点宕机。

## MetaServer

MetaServer 是一个分布式的元数据服务，为集群提供所需要的元数据信息。其中包括：

1. 集群成员信息
2. 所有存储卷的元数据信息
3. 数据块在集群中所存放的位置的元信息

多个 MetaServer 组成一个 MetaCluster 集群，每一个 MetaServer 中都有一份元数据的完整拷贝，以保证元数据的可靠性。

当集群中存在多个 MetaServer，MetaServer 会通过 ZooKeeper 进行选举，任意时刻有且仅有一个 MetaServer 会成为 MetaCluster 的 Leader，其他的 MetaServer 将成为 Follower。当作为 Leader 的 MetaServer 因故障无法工作时，Meta 集群会通过 Zookeeper 选出新的 Leader 继续对外提供服务。

当 Meta Leader 对集群进行管理操作时，所有的元数据更新操作都会在 ZooKeeper 集群中生成一条操作日志，其他 Follower 会从 ZooKeeper 处得到最新的日志，在本地重放日志，这样每个 Follower 与 Leader 之间的数据差异都会被控制在一个较小的范围内。每次 MetaServer 重新选取的 Leader 节点在提供服务之前，都会与 ZooKeeper 及其他 MetaServer 联络，确保自己的数据状态是最新的才会对外提供服务。

作为 Leader 的 MetaServer 同时承担了集群管理者的角色。它会负责与所有的 AccessServer 和 ChunkServer 进行通信，并收集相应的状态，用于执行数据恢复、数据迁移等决策。

MetaServer 只运行在主节点上。

## AccessServer

AccessServer 是 ZBS 的协议网关，提供包括 NFS 和 iSCSI 在内的接入层协议。AccessServer 接受 NFS 客户端或 iSCSI 发起程序（iSCSI Initiator） 的读写请求，并将不同的协议对象，例如 NFS 文件 或 iSCSI LUN，将其转义成 ZBS 内部的卷对象，进行读写。

AccessServer 运行在每台物理主机上，每个 AccessServer 都处于 Active 模式，都可以接受客户端的请求。AccessServer 通过 ZBS 的分布式租约（Lease） 协议机制保证：对任意一个客户端的读写请求，在任一时刻，只有一个 AccessServer 在处理其读写请求。这保证多路径软件在进行路径切换时，不会出现访问冲突而导致数据损坏的情形。

AccessServer 可看做传统存储的“存储控制器”，提供 IO 访问控制，但不同于传统存储，一个 ZBS 集群中可以有任意多的 AccessServer。因此，ZBS 可看做是一个高可用的可横向扩展的 iSCSI/NAS 存储。

## ChunkServer

ChunkServer 管理每台物理机上的 SSD 和 HDD 设备，对外提供基于数据块（extent）的读写接口。ChunkServer 的本地文件系统并没有基于任一 Linux 的已有文件系统，而是直接在裸设备上构建了自身的文件系统，称为 LSM（Local Storage Manager），LSM 不但是更适于高性能块存储访问的文件系统，避免了 Linux 已有文件系统的开销，而且通过智能的 SSD 和 HDD 管理，实现冷热数据交换协议。

## TaskServer

TaskServer 是 ZBS 的异步长任务处理模块。提供诸如存储池间移动/复制数据、异步复制等耗时较长的长任务处理服务。与 MetaServer 类似，多台 TaskServer 组成 Task 集群。经由 ZooKeeper 选出唯一的 Task Leader 负责任务的调度分发，其余 TaskServer 只作为 Runner 实际运行长任务。

当 Task Leader 故障时，集群会重新选举出新的 Leader，而已在运行中的任务不会受到影响。新的长任务请求将由新的 Leader 处理。 当 Task Runner 异常时，Leader 将会感知到任务异常，重新将任务调度至新的可用 Task Runner 处理。新的 Task Runner 将会从获取任务的当前进度状态，继续任务。此外，Task Runner 利用 ZooKeeper 来确保每个任务在任意时刻只有至多一个 Runner 在处理。

VIP Service 是 ZBS TaskServer 提供的虚拟 IP 服务。当为某一个服务配置了 VIP 后，集群内有且仅有一个物理节点（即 Task Leader）对应这个虚拟的 IP 地址。VIP 服务使得可以通过一个不变的 IP 地址使用接入服务。例如为 iSCSI 指定 VIP 地址之后，即可通过不变的 VIP 地址访问 ZBS 提供的 iSCSI 接入服务，而无需为每个 iSCSI 目标根据连接配置不同的服务地址。TaskServer 保证仅有 Task Leader 节点持有 VIP。在 Leader 异常切换时，VIP 也随之迁移至新的 Task Leader 所在的物理节点。

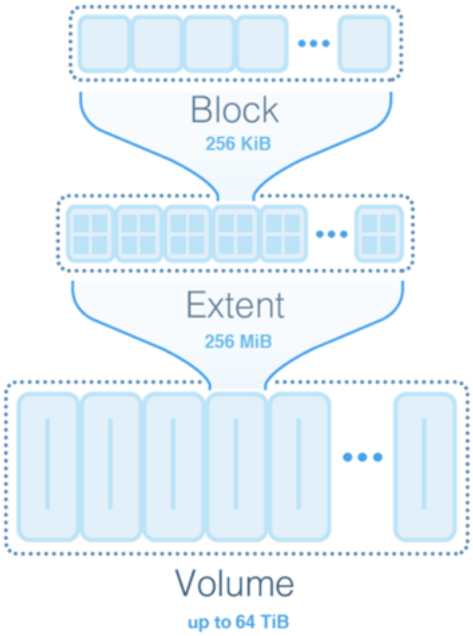
## iSCSI 重定向程序/ IO Rerouter

iSCSI 重定向程序（iSCSI redirector）提供 iSCSI 接入重定向服务。iSCSI 重定向程序通过定期与 MetaServer 通信更新的方式持有活跃的 AccessServer 列表。当 iSCSI 发起程序向 iSCSI 重定向程序发起登录请求时，iSCSI 重定向程序将返回恰当的 AccessServer，iSCSI 发起程序将在对应的 AccessServer 完成登录过程与后续的 IO 处理流程。

iSCSI 重定向程序分布在所有的物理节点上，所有的 iSCSI 重定向程序是逻辑等价的。与 VIP Service 组合使用时，即可保证通过 ZBS 提供的统一地址，可靠地使用 iSCSI 接入服务。

NFS 协议中并不包含重定向或类似功能，为了保证接入层的高可用性，需要通过在 NFS 客户端部署 IO Rerouter，以保证当某个 AccessServer 因故障无法访问时，NFS 客户端可以切换到其他可用的 AccessServer。

# 存储数据结构




## 块

块（block） 是 ChunkServer 内部进行数据存储的基本单元。一个数据块（extent）由多个块组成。块同时也是缓存盘与数据盘之间进行数据交换的基本单位。块默认大小为 256 KiB。

## 数据块

数据块（extent）是 MetaCluster 管理的基本单元。一个存储卷被切分成多个固定大小的数据块，存储在 ChunkServer 中。在 ZBS 的集群中，数据恢复、迁移、写时复制（CopyOnWrite - COW）等，都是以数据块为单位进行的。数据块默认大小为 256 MiB。

## 条带化

条带化（striping）是把连续的数据分割成相同大小的条带（stripe），尽量把每条数据分别写入到不同磁盘上的方法，以便利用多个磁盘的 IO 能力。ZBS 中条带数可以设置为 1、2、 4（默认值），设置的条带数越大，并行越高，对于顺序 IO 的性能更好。每个条带大小可以设置为 4、8、16、32、64、128、256 KiB （默认值）。设置太大容易造成并发度不够，对性能没有帮助，太小容易造成顺序 IO 变成随机 IO，破坏集群整体的性能。

## 存储卷

存储卷（volume），是 ZBS 对外提供的最基本的数据结构。它可以对应到一个虚拟机的存储卷，也可以对应到 iSCSI 中的一个 LUN。

存储卷由多个固定大小数据块组成，数据块表（extent table） 用于记录某个存储卷所包含的数据块 ID。当客户端需要在存储卷上执行 IO 操作时，会通过数据块表，查找到对应的数据块 ID，以及数据块所在的 ChunkServer 副本位置信息。

由于存储卷由多个数据块组成，且这些数据块分布在不同的节点上，因此存储卷的容量可以超过单个物理服务器的总存储空间。存储卷也支持精简配置（thin provisioning），最大可以支持创建 64 TiB 的存储卷。

除此之外，存储卷还具有很多属性，比如可以为其配置不同的存储策略，包括副本数、条带化参数等等。

存储卷也是用户执行快照、克隆、回滚等操作的基本单位。

### 数据存储

数据存储（datastore）是一组存储卷的集合。数据存储中包含的存储卷都具有相同的存储策略，例如副本数、精简配置等。每一个数据存储，都属于某一个特定的存储池。这个数据存储中创建的所有存储卷中包含的数据，也都存储在这个存储池中。

## 数据管理

### 存储池

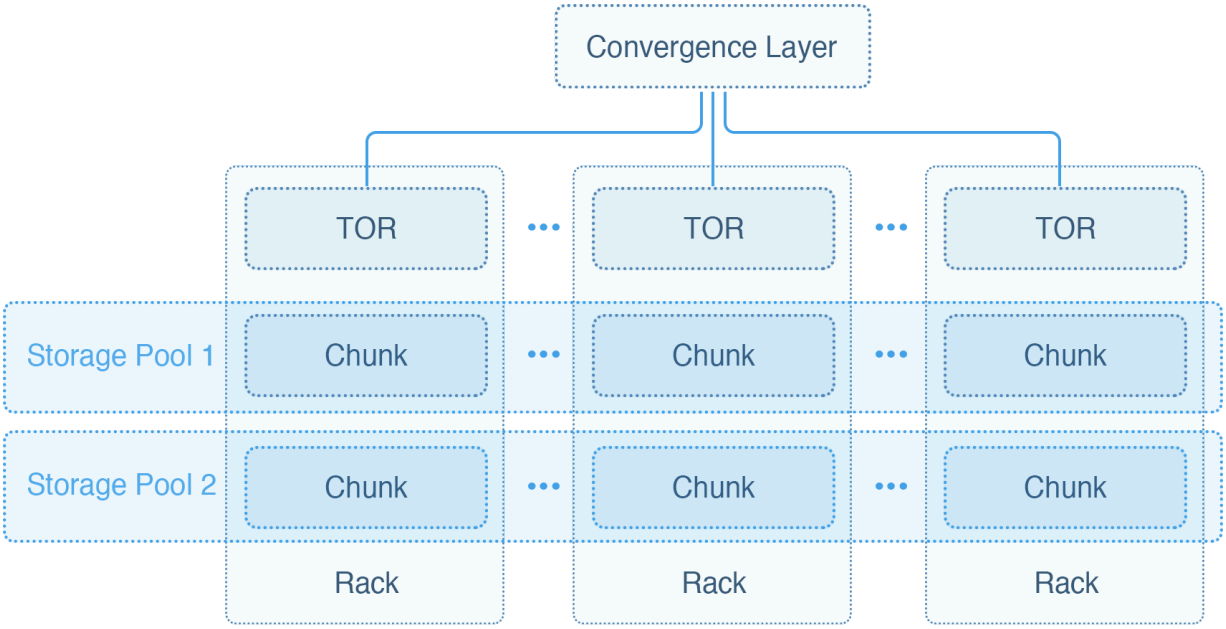
存储池是 ZBS 对存储介质进行组织的单元。在 ZBS 存储服务集群中，不同的存储介质可以加入不同的存储池，使得存储池具备不同的存储特性，例如：SSD/HDD 混合存储池，全 SSD 存储池等（在目前 ZBS 的实现中，服务器是可以加入存储池的最小单元）。

存储池提供了数据物理隔离的能力，一份数据的所有副本，都会保存在同一个存储池中。数据在存储池之间并不共享。如果数据需要在存储池之间进行移动，需要触发数据拷贝。

### 拓扑感知

在实际的部署过程中，并非只有存储节点可能发生物理故障。柜顶式交换机（Top of Rack, TOR）、机柜、UPS 电源等元素均有异常的可能。为了提供更高水平的系统可靠性与可用性, ZBS 的数据分配是拓扑敏感的。在存储节点加入集群时，允许指定存储节点的物理拓扑信息，例如机架组（pod）、机架（rack）、机箱（brick）、节点（node）等信息。ZBS 在进行数据副本分配时，会尽可能地将数据分散至物理拓扑的不同分支上，降低任一设备（存储节点/网络设备/电源模块）故障对系统的影响。

为了利用拓扑感知带来的可靠性提升，配置存储池时，应该尽量选取位于不同拓扑分支的节点组成存储池。例如：




### 精简配置

通常的使用场景里，虚拟机申请的磁盘空间并不是立即用满的。例如一个虚拟机申请了一个 1 T 的磁盘作为数据存储，并不是在申请的当时就立刻填充满 1 T 的磁盘空间，而是在虚拟机上承载的业务运行过程中逐渐的填充数据：即对数据空间的需求通常是随着时间逐渐提出的。为了提高存储空间的有效利用率，节省成本，ZBS 采用了精简配置策略。以数据块为粒度，仅在触发真实使用数据空间时（数据写入）才在 ChunkServer 中分配对应的空间。

精简配置默认开启，在开启了精简配置之后， ZBS 提供的名义逻辑存储空间大小可以超过实际物理空间大小。可以参考日常的业务空间需求来调节空间超分的比例，在实际空间需求达到真实可用空间限制之前补充设备扩展空间即可。

## 访问权限

为了提供数据访问安全等级，ZBS 为 iSCSI 目标/NFS Export 提供白名单机制。可以为每个 NFS Export/iSCSI 目标指定允许访问该对象的 IP 地址列表。接入层在处理接入请求时将校验客户端的 IP 地址。对于不符合白名单中匹配规则的来源 IP，将拒绝该请求。

对 iSCSI 数据存储以及 target 下的每一个 LUN，都可以单独设定白名单，用以隔离访问客户端，防止被非认证客户端访问篡改数据。

ZBS iSCSI 服务支持使用 CHAP 进行访问权限控制，支持单向认证和双向认证。单向认证指 iSCSI Target 可对 Initiator 端提供的用户名和密码进行身份认证。双向认证在单向认证的基础上，Initiator 端对 iSCSI Target 端也要进行身份认证。

## 接入点管理

所有挂载给外部使用的 iSCSI Target 上的数据链路都由 Meta 中的 iSCSI Service 统一管理。按照如下策略为每条数据链路分配接入点：

- 同一个 Target 被不同的 Initiator 连接时尽量使用不同的接入点（在 Target 作为共享数据被多个 Initiator 的情况下这个策略可以保证集群接入点性能被最大化的利用）；
- 同一个 Initiator 连接不同 Target 时尽量使用不同的接入点（在单个 Initiator 连接多个对象时候，可以保证 Initiator 可以使用整个集群的资源）；
- 每个 Access 上维持的活跃链路数量基本一致。

首次分配接入点后，会结合 Access Manager 从 Access 上获取的活跃链路信息，定期清理空闲的链路。周期性的检查所有当前 Access 记录的链路信息，将承担较多数量链路的 Access 的部分链路转移至其他节点。寻找新节点的规则为在所有较低负载（处理的连接数量低于平均值）的 Access 中按照选取新接入点的规则选取。在记录变化之后的周期性检查中，如果正在连接的链路与记录不匹配，则触发 Access 主动断开连接、Initiator 重连至记录的接入点的事件。

在链路分配上不会追求每个接入点处理的链路数量完全一致（实际上在链路数量不为接入点数量整数倍时，这是完全不可能做到的），仅要求负载最高的接入点比最低的接入点链路数量小于等于指定阈值即可（默认为 2 ）。每个平衡检查周期当中最多仅移动一条链路，避免集群剧烈的链路震荡。在新接入点加入集群提供服务时，其上已经分配的链路数量为空。部分链路会自然的经过均衡过程指向新的接入点。

# 数据 IO 路径

## 元数据设计

在分布式存储系统中，元数据服务通常都是设计的难点。如果设计不当，非常容易成为系统的瓶颈。

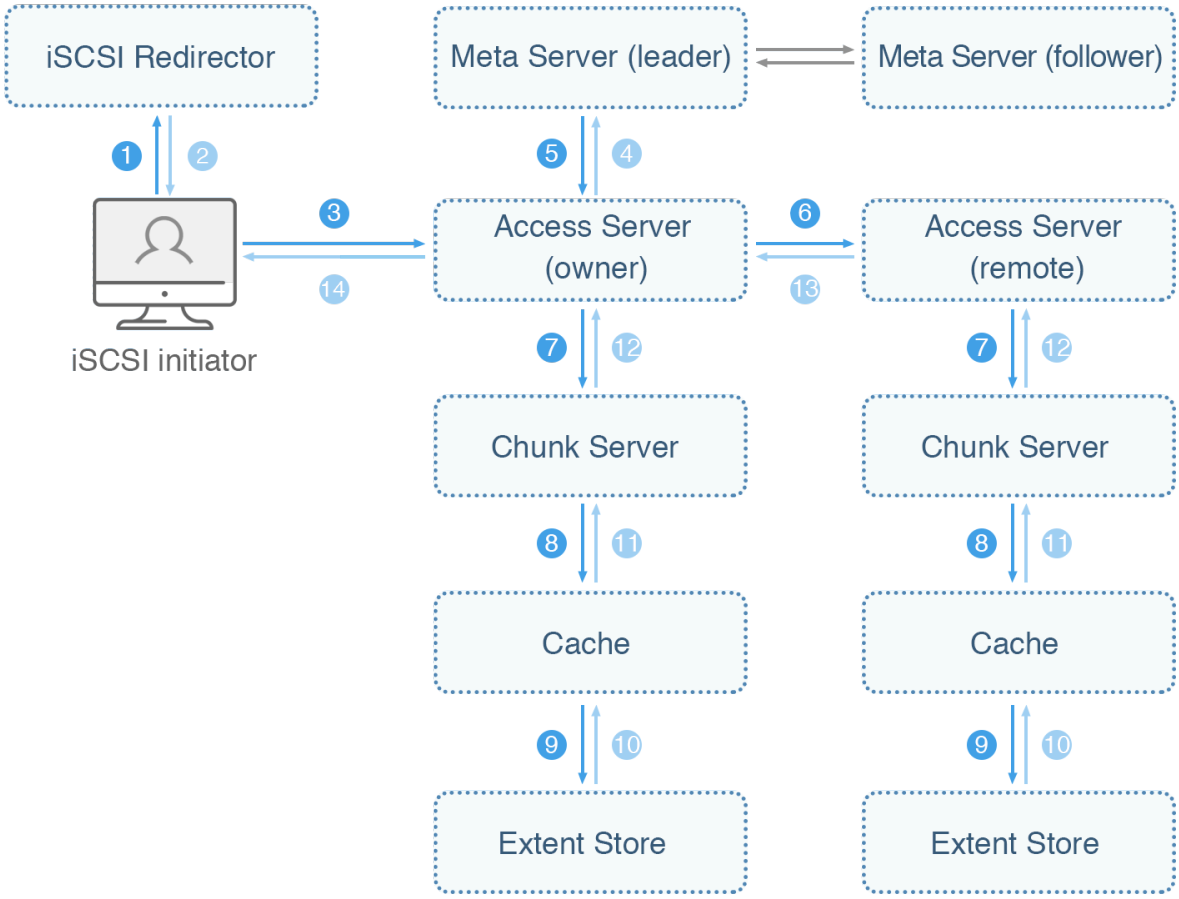
ZBS 元数据管理具有以下几个特点：

1. 采用集中式元数据管理，通过收集集群中数据的状态，做到对集群中数据的精确控制，包括数据的分配、数据的恢复和迁移等等。
2. 采用较大粒度的基本数据单元（数据块），减少元数据消耗的内存资源，保证全部元数据都可以保存在内存中，提高访问效率。
3. 元数据在所有节点中都保存一份副本，保证元数据的高可靠。当有部分服务器发生宕机，元数据不会丢失。
4. 通过对元数据进行缓存，减少各个组件与 MetaServer 的交互，从而减小 MetaServer 的负载，提高 IO 的效率。

## 全局 IO 路径

ZBS 采用了 iSCSI 和 NFS 两种存储协议，下面分别介绍其 IO 路径。

## iSCSI IO 路径




1. iSCSI 发起程序向 iSCSI 重定向程序发起数据接入请求

   iSCSI 发起程序在初次访问 iSCSI 目标（AccessServer） 之前，需要先向任意一个 iSCSI 重定向程序查询服务接入地址。

2. iSCSI 重定向程序返回 iSCSI 目标（AccessServer）的地址

   iSCSI 重定向程序通过和 MetaServer 之间的通信，维护了一个可用的 AccessServer 地址列表。收到请求后，iSCSI 重定向程序将向客户端返回恰当的 AccessServer 服务地址。 AccessServer 是数据访问的入口，每一个可访问的对象（extent）在被访问时，都会指定一个 AccessServer 作为这个对象的所有者。当对象长时间不被访问，则所有者被释放。

   如果 iSCSI 发起程序在访问一个 LUN 时，这个 LUN 已经存在所有者，则 iSCSI 重定向程序返回所有者的地址；否则，iSCSI 重定向程序会返回一个和 iSCSI 发起程序在同一台服务器的 AccessServer，作为新的所有者，以保证数据访问的本地性（locality）。

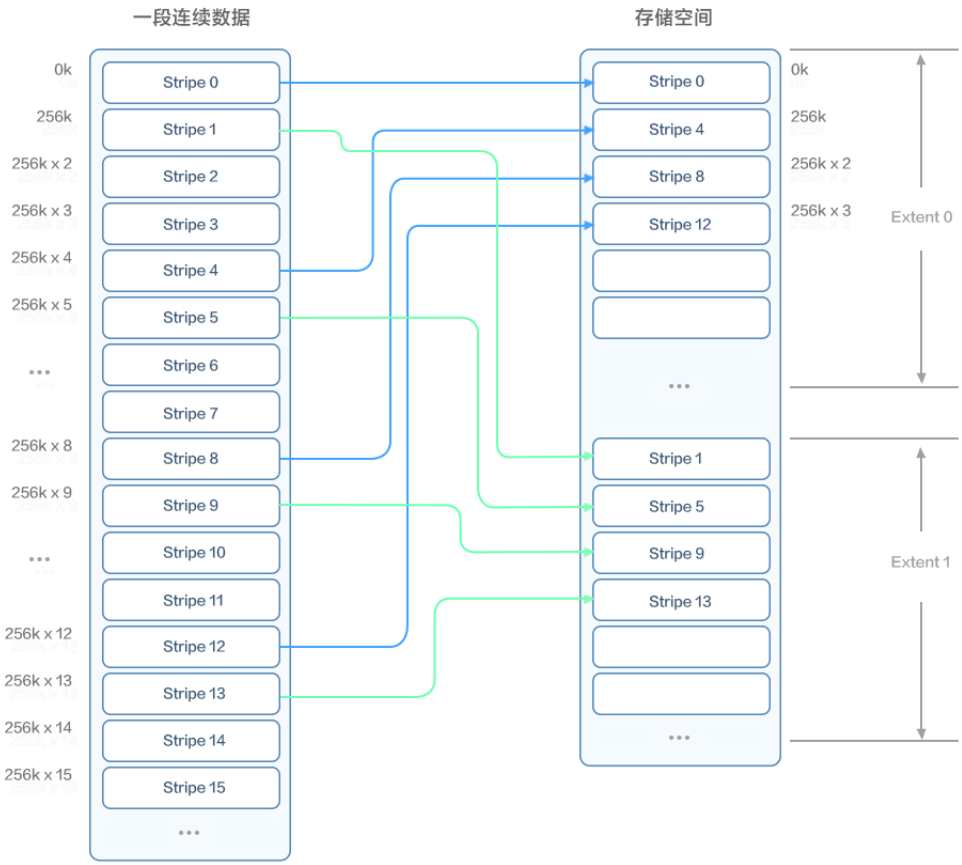
3. iSCSI 发起程序向 AccessServer 发起 IO 请求

   每一个 LUN 都有一个 AccessServer 作为它的所有者，负责这个 LUN 的所有 IO 操作请求。iSCSI 发起程序会首先将请求发送给所访问的 LUN 的所有者，并等待返回结果。

4. AccessServer 向 MetaServer 请求元数据信息

   向 MetaServer 请求元数据的时候，只有当前的 Meta Leader 可以响应请求，其他 MetaServer 不提供元数据服务。AccessServer 发出的请求携带的参数包括 LUN 对应的 Volume ID 与 VExtent No.。VExtent No. 表示数据块对应的 Extent ID 及一些控制属性，例如是否需要 COW 等, VExtent No. 从当前存储卷 IO 请求的偏移量转化而来。每个数据块（Extent） 大小为 256 MiB， 所以 0～256 MiB 的偏移量即对应卷数据块表（Volume Extent Table） 中的第一个数据块，即数据块号（Extent No.）为 0， 256 MiB～512 MiB 对应的数据块号为 1，依此类推。在开启了条带化（striping）功能时，映射方式略有不同：假设设置的条带大小为 256 KiB，条带数为 4，那么数据会按照条带大小（256 KiB）顺序分布在数据块（这些数据块尽可能分布在不同磁盘上）上：即 0～256 KiB， 256 KiB × 4

～256 KiB × 4 + 256 KiB，……，255 MiB～255 MiB + 256 KiB 对应的数据块号为 0，256 KiB～512 KiB 对应的数据块号为 1，依此类推。如下图所示：




5. MetaServer 返回元数据查询结果

   MetaServer 根据 Volume ID 与 VExtent No. 查询卷数据块表，获取表中对应的数据块信息。查询结果包含了数据的副本信息、副本所在的 ChunkServer 的地址等信息。如果是精简模式第一次写入，MetaServer 会为数据块分配副本空间。AccessServer 在收到元数据信息后，会将元数据信息缓存在内存中，减少对 MetaServer 的查询请求次数。

6. 向其他 AccessServer 发送 IO 请求

   如果 iSCSI 发起程序发送的请求是写/读操作，但 AccessServer 本地并没有副本，则会把 IO 请求将发送到其他 AccessServer。对于双活集群，IO 请求优先发至同一可用域内的 AccessServer。

7. IO 请求发送给 ChunkServer

   AccessServer 将 IO 请求转义成 ChunkServer 可处理的数据块 IO 请求。

8. IO 请求发送给缓存系统

   如果数据在缓存中存在，则缓存系统可以直接响应请求。否则，请求将被转发给持久化层。

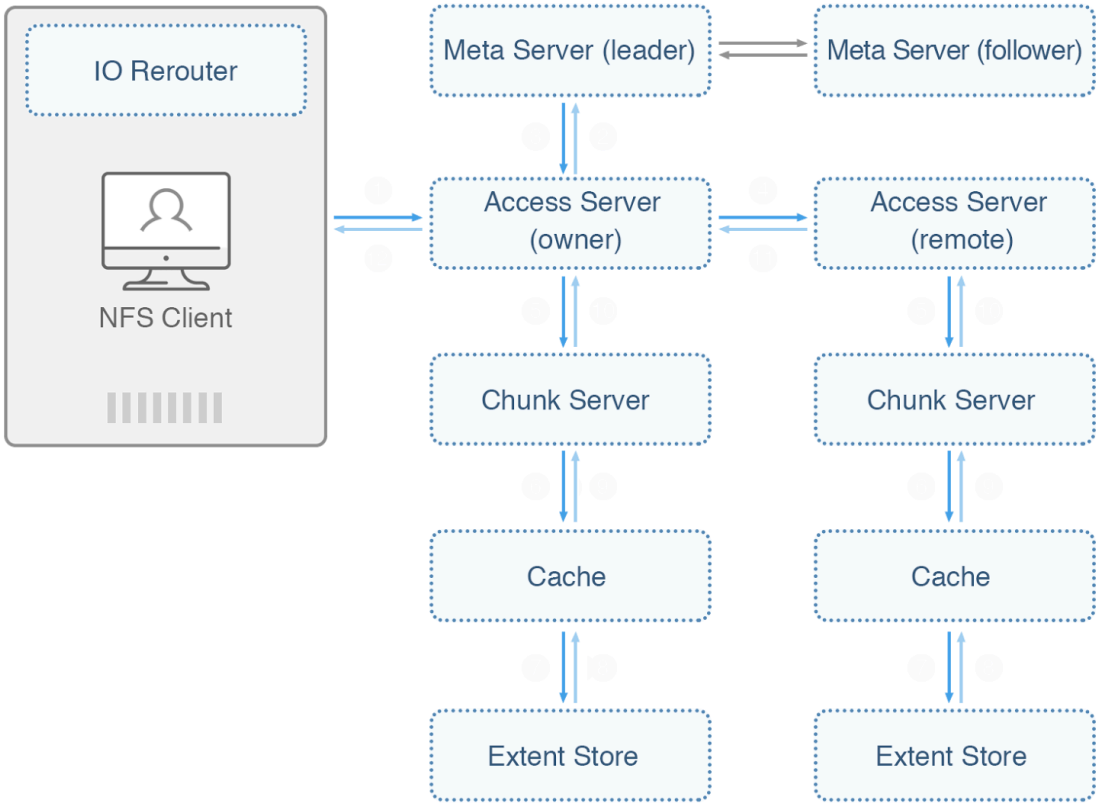
9. IO 请求发送给持久化层

   持久化层（通常管理 HDD，全闪模式下为 SSD）。如果数据不在缓存中，IO 请求将发送给持久化层。持久化层负责在 HDD 磁盘中查找数据，并返回给 ChunkServer。

10. 第10~14，IO 请求执行完成，返回给 AccessServer

    对于写请求，只有所有副本都写成功才会返回成功。当因为网络异常或存储设备异常而导致部分副本写入失败时，AccessServer 会立即把问题副本从集群中剔除，并触发数据恢复流程；如果网络正常，且写操作正在进行，但 30 秒内仍未完成写入，则会终止写操作，由 AccessServer 把问题副本从集群中剔除，并触发数据恢复流程。如果所有副本都写失败，则会立即触发重试操作。对于读请求，如果有一个副本读取失败，则会立即尝试从其他副本读取。

## NFS IO 路径

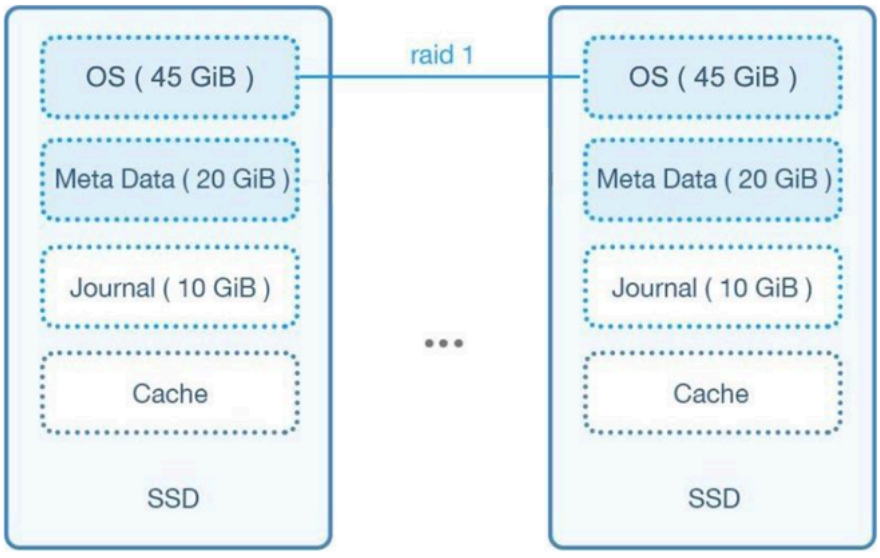



NFS 协议并不直接支持 IO 重定向，所以 NFS 的 IO 路径在寻址上与 iSCSI 稍有不同。为保证 NFS 服务的高可用性，需要在 NFS 客户端所在的物理节点上部署 IO Rerouter 程序。NFS 客户端通过固定的服务地址访问 NFS 服务（例如：192.168.33.2）。IO Rerouter 程序定期检查当前连接的 AccessServer 的存活状态，在 AccessServer 异常时， IO Rerouter 将负责修改 NFS 客户端所在的服务器的路由表，把 192.168.33.2 指向一个当前可用的 AccessServer，NFS 客户端即可切换至新的可用 AccessServer 继续 IO。

AccessServer 与后端的 IO 交流过程与 iSCSI 一致。

## 本地 IO 路径

### 本地磁盘分区



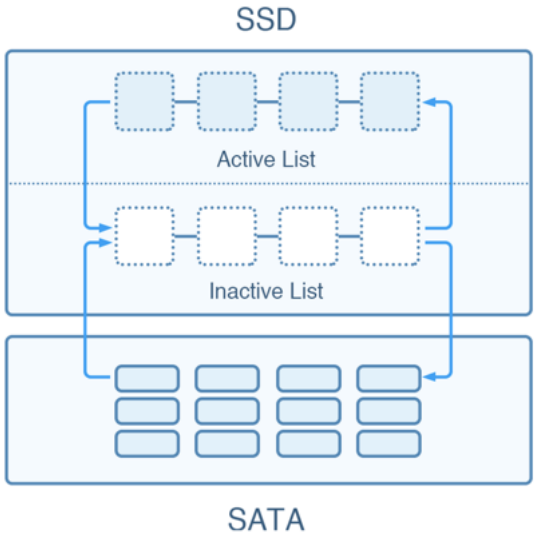

### Journal

Journal 用于记录 ChunkServer 本地的数据操作，包括数据块的创建、删除、修改等等。每个写入请求都会写入一条 Journal，当写入请求大小小于 block 大小时，Journal 中包含完整的请求数据,。而当缓存命中（即缓存有足够的空间供相关数据写入或相关数据已经在缓存中存在）且请求大小为 block 大小时（常见于顺序写场景，接入层会将顺序 IO 合并至 block 大小下发至 Chunk），数据会直接写入缓存，并在稍后异步同步到 ExtentStore。此时 Journal 不携带数据，仅记录操作元数据，避免了写放大。

每一个 ChunkServer 中会包含多个 Journal 。当其中一个 Journal 写满时，会切换到另一个 Journal 。当写满的 Journal 中的数据全部同步到 ExtentStore 中以后，写满的 Journal 会被清空，可以被重新使用。Journal 位于 SSD 上，以提供足够高的 IO 能力。

**CacheManager**

CacheManager 通过近期最少使用算法（Least Recently Used, LRU），缓存用户频繁访问的数据。当数据第一次被写时，数据会直接写到 ExtentStore 上（如果是随机 IO，则会先写到 Journal 中）。当数据再次被访问时，会被缓存到 CacheManager 中进行管理。




CacheManager 通过两级 LRU 链表来管理数据冷热程度。Inactive List 中记录了至少被访问过一次的块（block），Inactive List 按照被访问的先后顺序进行排序，最不经常被访问的数据会被从缓存中移除。Active List 中记录了被访问过多次的块，Active List 同样按照被访问的先后顺序进行排序，最不经常被访问的数据，会转移到 Inactive List 中。

**ExtentStore**

ExtentStore 支持对数据进行校验。如果开启数据校验，则每一个保存的数据块都会同时保存一个 Checksum 值。当数据被访问时，会通过 Checksum 来校验存储数据的正确性。

## 副本分配策略

为了保证 IO 的高效，MetaServer 会通过收集集群信息，对副本分配做出近似最优的选择。相关影响因素包括:

1. 本地优先。由于 ZBS 面向超融合虚拟化场景设计，虚拟化的 Hypervisor 与 ChunkServer 会运行在同一台物理主机上。所以 MetaServer 会尽量将副本中的其中一个，分配在与 Hypervisor 运行在相同的物理主机上的 ChunkServer 中，以减少存储服务对网络的压力；
2. 拓扑安全。数据副本尽可能分布在物理拓扑的不同分支上，减少物理故障带来的损害；
3. 局部化。单个卷的数据分布的范围应当较为集中，而不是分散在整个集群中。不同的卷尽可能使用不同的数据空间，降低卷对物理资源占用的冲突可能，实现集群整体性能的线性扩展能力并降低节点故障的影响范围；
4. 负载均衡。尽可能地让所有存储节点的实际承担容量近似，以便平均访问负载并降低节点异常时的损失；
5. 访问模式感知。通过多种方式感知到卷的访问模式。例如是否多点访问（多个虚拟机使用同一个虚拟磁盘）、是否作为模板等。针对不同的访问模式进行副本位置调整；
6. 数据动态移动。当数据访问需求方在集群中移动时（例如超融合模式下虚拟机迁移），对应数据副本需要随之移动，以达到数据本地化；

7. 副本分配策略动态调整。副本分配的多个目标之间并不是协调一致的，例如本地优先会导致数据集中，而负载均衡希望不同节点数据分散。所以需要根据当前集群的节点空间使用比例，调整不同目标的优先级，使用不同的策略；

其中 1、2、3、4 为数据副本分配的基本目标。其中拓扑安全目前设计为高优先级，但非强制性目标。即在拓扑安全可满足的条件下拓扑安全将作为第一优先级。若无法满足，则允许暂时按照不安全的方式进行数据分配，稍后在可能的情况下再自动修复。而本地优先与局部化分配没有矛盾，可以同时处理，但它们在某些场景下（虚拟机需要的数据空间集中在少数节点上）与容量均衡的目标是互相冲突的。ZBS 将根据不同的负载情况，动态调整本地优先，局部化分配，容量均衡三个策略之间的优先级。

按照集群中数据空间已分配比例最高节点的磁盘空间已分配比例 p 作为标准，策略目标优先级调整如下：

1. 低负载状态（p < 75%）：此时节点间的容量均衡被忽略，以本地优先与数据局部化为目标分配与调整数据，优先级次序为： ② 本地优先 = ① 拓扑安全 > ③ 局部化分配；
2. 中负载状态（75% <= p < 85%）：此时新副本分配还是按照本地优先与数据局部化为目标，优先级次序与低负载时相同。但是迁移扫描将根据当前 IO 状态，尝试迁移部分非本地且非活跃数据以达到容量均衡。再平衡时对非本地活跃数据的优先级次序为： ②本地优先 = ① 拓扑安全 > ④ 容量均衡 > ③ 局部化分配；
3. 高负载状态（85% <= p）：分配时的数据局部化目标将暂时被忽略，以本地优先 + 容量均衡为目标进行数据分配。数据调整时，除中负载时的非本地且非活跃数据之外，还会尝试迁移包含本地的非活跃数据。此时根据 Extent 的 prefer local 节点本身的负载，在初次分配副本时有所不同：

   a. Prefer local 节点自身的容量没有超过 85%，优先级次序为 ②本地优先 = ① 拓扑安全 > ④ 容量均衡；

   b. Prefer Local 节点自身的容量已经超过 85%，放弃本地优先，仅采用 ① 拓扑安全 > ④ 容量均衡策略；
4. 不同负载边界时的处理：为避免节点容量始终处于策略判定的容量边界导致频繁反复变更优先顺序而导致的集群内数据不断被来回迁移，影响正常 IO，会增加容忍阈值，默认 5%。例如当节点容量超过 75% 后不会立即触发中等负载的平衡策略，而是需要等到节点容量>= 80% 时才触发迁移，尝试将节点容量降低至 75% 以下。

访问模式感知，目前 ZBS 感知以下两种访问模式：

1. 快照模板化：当一个快照被多次克隆（默认超过 10次），则它会被处理为模板，关联的数据将在集群中整体均匀分布（忽略局部化与本地化目标）。在 Elf 平台下，还可以通过将存储卷/虚拟机显式地指定为模板触发存储卷的均匀分布。
2. 池化访问，当一个 Volume 同时被多个接入点访问时，Volume 会被认为是一个虚拟的存储池，此时不会根据接入点迁移数据；

数据动态移动则是通过接入点感知完成，当一个活跃的存储卷的访问点发生变化（例如虚拟机迁移），ZBS 接入服务将上报给 MetaServer，MetaServer 在后续的再平衡策略中将数据逐步朝访问点迁移以达到数据本地化目标。

对于双活集群，其副本分配策略略有不同，具体请参见双活集群副本机制（见第 21 页）中对副本分配机制的描述。

## 副本迁移策略

生产集群中的环境会经常发生变化，例如有服务器磁盘故障、服务器故障、集群扩容、虚拟机迁移等等。

这些变化将导致最初的副本分布在新的场景下并不一定是最佳的分布。为了保证集群的最佳分布，MetaServer 会定期对集群状态进行检查，并触发副本迁移任务。副本迁移主要为了达到以下三个目标：

**数据访问局部化**

在虚拟化环境中，虚拟机会经常根据需求进行迁移。当虚拟机发生迁移时，部分原本是本地化的访问，将变成是远程访问。为降低因远程访问带来的额外开销，MetaServer 会触发数据向本地迁移。

通常，为避免不必要的迁移，MetaServer 会在虚拟机迁移以后等待一段时间，再触发数据的迁移。在迁移的时候，会优先迁移被频繁访问的数据。如果迁移的目标端的空间已经用满，则不会触发迁移。

**拓扑再均衡**

当集群因扩容或其他原因，导致集群中拓扑发生变化，为保证集群健康，MetaSever 将会根据最新的拓扑尝试调整数据的期望分配，以达到最佳效果。

**容量均衡**

当集群处在中高负载状态时（有节点的存储空间利用率 > 75%）为了避免数据过度集中，MetaServer 会定期的将高负载节点上的非本地热数据迁移至其他节点，以降低数据集中访问的风险。

除此之外，ZBS 也并不会追求容量绝对的均衡。在集群中，允许一定程度的不均衡情况出现。如集群处于低负载状态（所有节点存储利用率均低于 75%）则不会触发容量均衡，以避免集群中的数据出现抖动。集群处于高负载状态时，会自动加快迁移检查频率，尽快达到均衡状态。

# 数据保护

ZBS 的数据保护机制包括多副本、数据校验、快照、双活和异步复制。

## 副本

不同于硬件 RAID 技术，ZBS 通过软件的方式实现多副本的方式来提供安全的保证。当集群发生故障时，可以对数据进行自我修复。

每一个存储卷都会被划分成以 256 MiB 为长度的数据块。采用副本技术以后，同一个数据块在集群中会有多个副本（拷贝）。同一个数据块的多个副本会分布到不同的 ChunkServer 存储，也就是不同的物理节点上。这样可以保证即使发生服务器节点宕机，只要还有可以访问到的副本，就不会影响数据的可用性和可靠性。

当一个数据块存在多个副本时，为保证数据的一致性，对数据的每一次写都是同时写多份副本，并且当所有副本都写成功后再返回给客户端，也就是保证了数据的强一致性。

数据块的副本数，取决于它所从属的存储卷的存储策略中的设置。为保证数据可靠性，ZBS 元数据服务会周期性地和 ChunkServer 通信，以确定当前系统中可访问的数据块。当发现某个数据块的可用副本数低于设定值，就会触发自动修复功能。例如用户设定某个存储卷副本数为三份，同一份数据会复制到三个不同的 ChunkServer 上。当集群中出现一台 ChunkServer 所在的物理服务器发生宕机，导致可访问的数据块副本数减少，ZBS 的 Meta 服务会及时发现这一事件，并以数据块为单位进行数据恢复。从健康的副本所在的 ChunkServer 中读取正确的数据块，并恢复到另一个健康的 ChunkServer 中（如果宕机的服务器恢复健康，也有可能恢复到这个服务器），以确保数据块始终保持在三副本的状态。

传统 RAID 重建时，经常导致存储系统不可用，而 ZBS 在恢复副本的时候，正常 IO 访问仍然可以被正常执行，并不会出现中断。ZBS 通过控制恢复流量所占的 IO 带宽，尽量减少对正常业务性能的影响。

ZBS 可以通过存储策略，为每一个存储卷设置不同的副本数。目前支持 1、2、3 个副本。不同的副本数定义了不同的数据安全级别。当存储卷的副本数为 3 时，允许同时有 2 台服务器发生宕机事件。当存储卷的副本数为 2 时，允许同时有 1 台服务器发生宕机事件。当副本数为 1 时，意味着数据没有任何保护措施。

用户可以针对业务的重要性，为不同虚拟机，以及对应的不同的存储卷的副本数进行设置。

使用 zbs-meta volume update/zbs-iscsi lun update/zbs-nfs file update 命令可以调整已经分配的 Volume/iSCSI LUN/NFS File 的期望副本数，仅支持提高副本数不支持减少。在提高副本数之后，将会触发恢复事件将数据补齐至指定副本数。

**弹性恢复**

ZBS 使用以下两种策略来控制副本恢复流量以满足不同场景需求：

1. 智能调节（AUTO）：此为默认模式。以保护业务 IO 为前提，系统根据当前业务负载自适应地调整恢复或迁移速度。此外，ZBS 具备硬件感知功能，当主机配置了 25GbE 网卡或 NVMe 磁盘等高速设备时，系统将自动提高允许恢复和迁移速率的上限。
2. 静态调节（STATIC）：用户人工设置速度限额。当用户希望保护正常 IO 时，可以将恢复速度设置为较小的值（例如，可以采用默认的恢复和限速的限速值）。当用户希望加速恢复时，可以设置为较大值（例如，500 MB/s）。静态设置只允许对 Meta 下的所有 Chunk 设置同一限额，不能够对每个 Chunk 单独设置。

## 数据校验

除了副本技术之外，为了提供更高级别数据安全保证，ZBS 同时对所存储的数据进行了数据块校验。每一份存储在 ZBS 中的数据，都会同时保存一份该数据的校验码。ZBS 在数据被访问时，对数据进行校验，以避免磁盘硬件无记录数据损坏（ Silent Data Corruption - SDC） 的发生。当发生数据与数据校验码不一致时，则认为该数据块存在问题，同时会触发数据恢复机制。

## 快照

ZBS 的快照以存储卷为单位。存储卷的快照，就是将存储卷在执行快照那一刻的数据内容保存下来，以便于从这个快照克隆出新的存储卷，或在未来的时间，对快照进行回滚。

副本与数据校验可以有效地避免由于硬件故障导致数据安全问题。而快照，则可以用于避免因人为误操作或软件故障所导致的数据损坏或丢失。

### 快照的实现

ZBS 的快照采用元数据快照和写时复制（CopyOnWrite - COW） 技术，可以实现秒级快照。

在 ZBS 中，快照的元数据和存储卷的元数据的数据结构是相似的，都包含数据块表。快照和存储卷主要的区别是，快照的元数据以及数据都不能被修改。

数据块表示例：卷 A 的数据块表

| VExtent | PExtent | COW |
| --- | --- | --- |
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 2 | 2 | 0 |
| 3 | 3 | 0 |

其中，VExtent 代表存储卷 A 的逻辑地址，PExtent 代表 VExtent 的物理地址，COW 标记是否需要进行 COW。

当对存储卷进行快照操作时，MetaServer 对这个存储卷的数据块表进行复制，组成新的快照的数据块表。通过元数据的复制，这个快照的内容就被保存了下来。而对于原有的存储卷，需要为数据块表中的每一个数据块增加一个 COW 标记位，以表示当这个数据块的内容需要被修改时，需要执行写时复制 操作。

以下是卷 A 的数据块表

| VExtent | PExtent | COW |
| --- | --- | --- |
| 0 | 0 | 1 |
| 1 | 1 | 1 |
| 2 | 2 | 1 |
| 3 | 3 | 1 |

以下是快照 A 的数据块表

| VExtent | PExtent | COW |
| --- | --- | --- |
| 0 | 0 | 1 |
| 1 | 1 | 1 |
| 2 | 2 | 1 |
| 3 | 3 | 1 |

写时复制操作分为 MetaServer 端与 ChunkServer 端，分别对应了元数据的 CopyOnWrite 和数据的 CopyOnWrite。假设 VExtent1 拥有 COW 标记位，修改它的内容前，需要先在 MetaServer 端分配一个新的 PExtent4。

VExtent1 被写入前卷 A 的数据块表 VExtent1 被写入后卷 A 的数据块表

| VExtent | PExtent | COW | | VExtent | PExtent | COW |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | | 0 | 0 | 1 |
| **1** | **1** | **1** | # | **1** | **4** | **0** |
| 2 | 2 | 1 | | 2 | 2 | 1 |
| 3 | 3 | 1 | | 3 | 3 | 1 |

在 ChunkServer 端，也会创建对应的 PExtent4。PExtent4 的 Parent 会指向 PExtent1。每一个 PExtent 在 ChunkServer 端会保存一个块位图（Block Bitmap），用于标记一个该块是否被写过数据。

当 ChunkServer 对 PExtent1 进行写时复制时，如果写请求没有按照块对齐，会先按照块对齐的边界，对写请求切分成多个请求，分别处理。切分后的请求中，对于请求数据不足一个块的情况，对应的块的数据会从 PExtent1 中读取出来，与新写入的数据进行合并，然后写入 PExtent4 的对应的块中。如果切分后的请求刚好为一个块的大小，则不需要从 PExtent1 中读取任何数据。由于一个块的大小是 256 KiB，以及使用 SSD 进行加速，一次写时复制的操作是非常快速的。当一个块执行过写时复制操作以后，在这个块上后续的读写操作都和正常的读写操作一致。所以快照并不会影响存储卷的写性能。

当从 PExtent4 读取数据时，如果对应的块的位图的比特位为空，则从 PExtent1 中读取数据。由于所有元数据信息都缓存在内存中，查找数据的物理位置，以及位图的查找都在内存中，所以查找的速度也非常快。这也意味着每一次读操作只需要读取一次磁盘，而并不需要像其他快照技术那样读取多次。这使得在对做过快照的存储卷进行读操作时，速度可以和对普通存储卷进行读操作是一致的，快照并不会影响存储卷的读性能。

| PExtent1 block | 0 | 1 | 2 | 3 | ... |
|---|---|---|---|---|---|
| | x | x | x | x | ... |

| PExtent4 block | 0 | 1 | 2 | 3 | ... |
|---|---|---|---|---|---|
| | x | o | x | o | ... |

ZBS 的快照之间不存在任何依赖关系。删除快照只需要在 MetaServer 端删除这个快照对应的数据块表即可。因快照删除所触发的空间释放，以及数据合并操作，会在空闲时完成，并不会影响正常的数据访问的性能。

## 克隆的实现

ZBS 支持由存储卷到存储卷的克隆，也支持直接由快照到存储卷的克隆。

对于从存储卷到存储卷的克隆，和对存储卷执行快照的操作非常相似。区别在于：

1. 存储卷到存储卷的克隆操作创建的是基于源存储卷快照创建的存储卷，而对存储卷进行快照操作创建的是不可变的数据块表。
2. 对克隆出来的存储卷也可以执行 IO 操作，数据块表中的每一个数据块具有 COW 标记位。

对于从快照到存储卷的克隆，和对存储卷执行快照操作也非常相似。区别在于：

1. 从快照克隆存储卷的操作创建的是基于快照的不可变的数据块表创建的存储卷，而对存储卷执行快照操作创建的是不可变的数据块表。
2. 对克隆出来的存储卷也可以执行 IO 操作，数据块表中的每一个数据块具有 COW 标记位。
3. 原有快照的数据块表不做任何改动。

## 回滚的实现

ZBS 中的存储卷可以在任一时间回滚到任一快照。为保证虚拟机数据的一致性，用户不应该在开机的状态下对存储卷进行回滚。

快照回滚的操作同样是基于数据块表的操作，可以实现秒级回滚。用将要回滚的快照的数据块表覆盖当前虚拟磁盘的数据块表，并为数据块表中的每一个数据块添加 COW 标记位。

### 快照的删除与数据回收

ZBS 中的快照之间不存在依赖关系，快照可以在任一时间被删除。

因为快照和克隆的存在，一个数据块可能同时被多个存储卷或快照引用。MetaServer 维护了数据块被引用的次数。当快照被删除以后，这个快照对它所属的数据块的引用将被释放。当数据块被引用的次数为 0 时，意味着这个数据块对应的数据空间和 PExtent ID 可以被重新分配（亦称被回收)。MetaServer 将回收命令发送给数据块的副本所在的 ChunkServer。

ChunkServer 对于数据块的 CopyOnWrite 操作是以块为单位的。一个数据块在被回收之前，可能包含了其他数据块的数据。这个时候 ChunkServer 需要触发合并操作，以保证这个数据块可以被安全的回收。数据块合并操作会在空闲时执行，并不会影响正常 IO 操作。

在节点上因为大量卷或快照删除而导致待回收数据较多时（已经使用的数据空间占比超过 95%），如果集群没有进入极高负载状态（整体空间分配比例达到 90%），不向该节点迁移数据以保证回收顺利进行。

## 双活集群

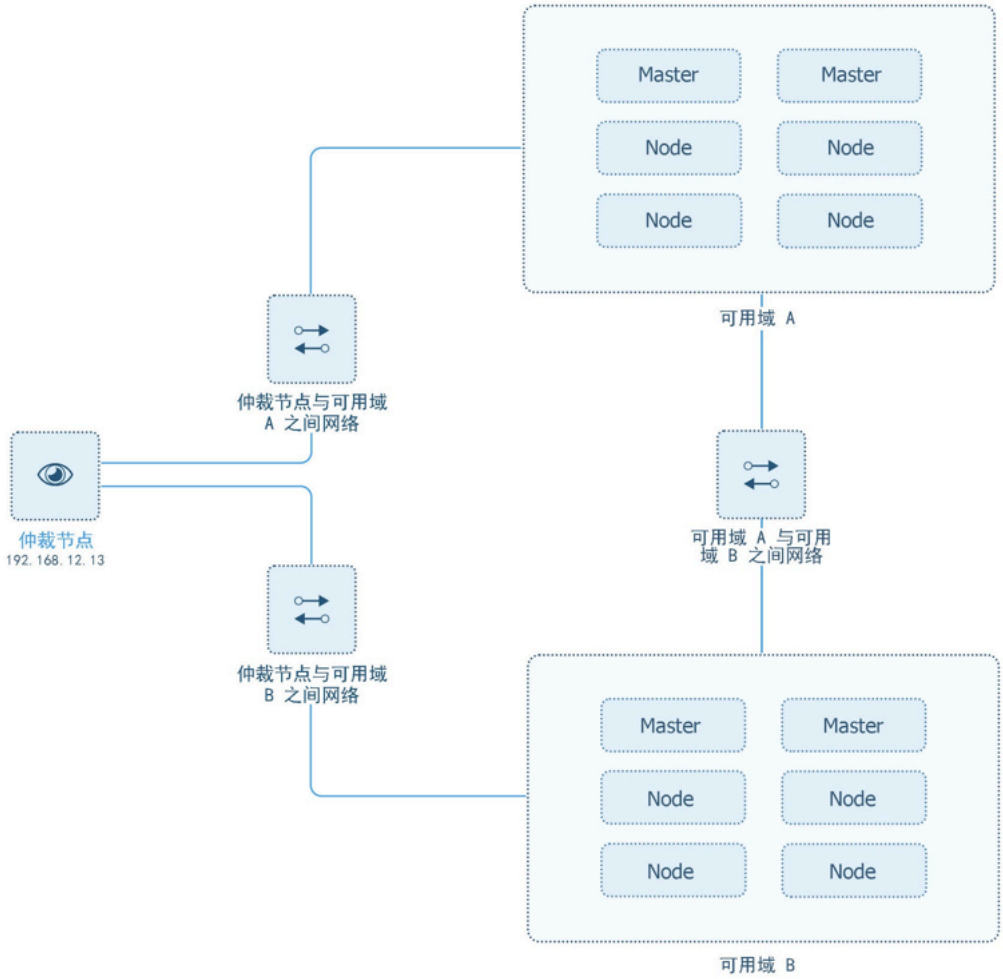
双活集群是指一个含有两个可用域的集群。一个可用域通常指一个数据中心，这个数据中心可以完全独立地提供计算和存储服务。属于同一个双活集群的两个可用域，除非特殊说明，一般是位于同一个城市。当两个域都工作正常时，集群会确保每个数据块在两个可用域内都保存有副本，当一个可用域失效时，可以从存活的可用域中获得完整的数据恢复业务。

### 双活集群结构

一个双活集群由两个可用域和一个仲裁节点组成。

仲裁节点不属于任何一个可用域，它可以是物理机，也可以是虚拟机，只参与选主和运行关键服务（具体指 ZooKeeper、MongoDB、NTPD 和 Master Monitor），但不用来存储数据。

每个可用域至少含有 3 个节点，其中至少 2 个是主节点（即运行 ZooKeeper 的节点）。两个可用域之间、仲裁节点和两个可用域之间通过网络连接。如下图：




根据业务要求，需要指定一个可用域为优先可用域，另一个则自动成为次级可用域。

## 双活集群副本机制

双活集群的保护能力，源于其多副本机制。双活集群中副本数量的分配基本原则是：优先可用域保留 2 个副本，次级可用域保留 1 个副本；副本迁移和分配的原则基本同非双活集群，但又有些差别。

### 新副本分配

新副本是指首次生成副本的情况。双活集群新副本分配基本遵循前述的本地优先与局部化原则。但需注意的是，跟非双活集群略有不同，此处的“本地”是指“本地可用域”（即虚拟机或者物理机等连接至存储的接入点所在的可用域），而不是指“本地磁盘“。

副本具体分配顺序如下：

1. 第一个副本的磁盘分配：
   a. 当启用本地优先策略时，本地可用域就是数据块（extent）的优先可用域：
      1. 若本地磁盘空间满足需求，则本地磁盘的数据作为第一个副本；
      2. 若本地磁盘空间不满足需求，则在本地可用域内的所有节点中，在当前磁盘利用率最低的节点上创建第一个副本；若本地可用域内无可用节点，则在另一个可用域内磁盘利用率最低的节点上创建第一个副本；
   b. 当不开启本地优先策略时，双活集群的优先可用域同时也是数据块（extent）的优先可用域：
      1. 集群优先可用域中有节点空间可满足需求，则在此域中当前磁盘利用率最低的节点上创建第一个副本；
      2. 集群优先可用域中无节点空间可满足需求，则在集群次级可用域中当前磁盘利用率最低的节点上创建第一个副本；

2. 第二个副本的磁盘分配，检查数据块优先可用域内的节点状态：
   - 若有节点可用，则按照局部化规则补齐数据块优先可用域内的副本数量至 2 副本；剩余的第 3 个副本的分配原则，见以下第 3 步；
   - 若无节点可用，按照局部化规则在数据块次级可用域补齐至 3 副本，分配结束；
3. 第三个副本的磁盘分配：如果副本数未达到 3 ，检查数据块次级可用域节点状态:
   - 若有节点可用，按照局部化规则补齐数据块次级可用域内的副本数量至 3 副本；
   - 若无节点可用，按照局部化规则在数据块优先可用域补齐至 3 副本；

**副本恢复**

当有副本失效时，需要恢复副本至 3，规则如下：

1. 活跃副本数为 1 时，首先检查与源副本同一可用域内是否有节点空间满足需求。若有，则按照局部化规则在源副本可用域内的节点上创建 1 个副本，否则在另一个可用域内创建。此时，活跃副本数变为 2，后续按下述规则最终把副本数恢复为 3。
2. 活跃副本数为 2 时：
   a. 当 2 个活跃副本都在同一个可用域，可任意选取其中一个副本作为源副本。此时若另外一个可用域有可用节点，则在另一个可用域内按局部化规则，创建 1 个副本。如果另外一个可用域内没有可用节点，则不会尝试在活跃副本所在可用域内恢复至 3 副本；
   b. 当活跃副本位于不同的可用域:
      1. 若本地可用域内有节点可用，则选取本地可用域副本作为源，在本地可用域内按照局部化规则创建 1 个副本;
      2. 若本地可用域内无节点可用，则选取另一可用域副本作为源，在另一可用域内按照局部化规则创建 1 个副本；

策略 1 将保证在活跃副本数为 1 时，优先在当前可用域将数据块恢复至 2 副本状态。对优先可用域不稳定，反复异常的场景，该策略将可以保证数据块可靠性快速提升至 2 副本水平。

**副本再均衡**

所有基于节点容量平衡的再均衡策略将只在同一个可用域内进行，不会尝试跨越可用域传输数据。即便一个可用域内数据空间消耗殆尽也不会尝试迁移副本至其他可用域。但是基于拓扑安全的再均衡策略将会允许跨域传输，例如某些场景下 3 副本位于同一个可用域，则允许迁移一个副本置另外一个可用域。

双活集群中的模板卷，副本分配规则将调整为，各个可用域内各自保持均匀。具体来说，即单个可用域内所有节点均保存近似数量的卷的副本（相差小于 4 个数据块，即 1 GB）。不同可用域的节点所保存的副本数量没有要求。

# 异步复制

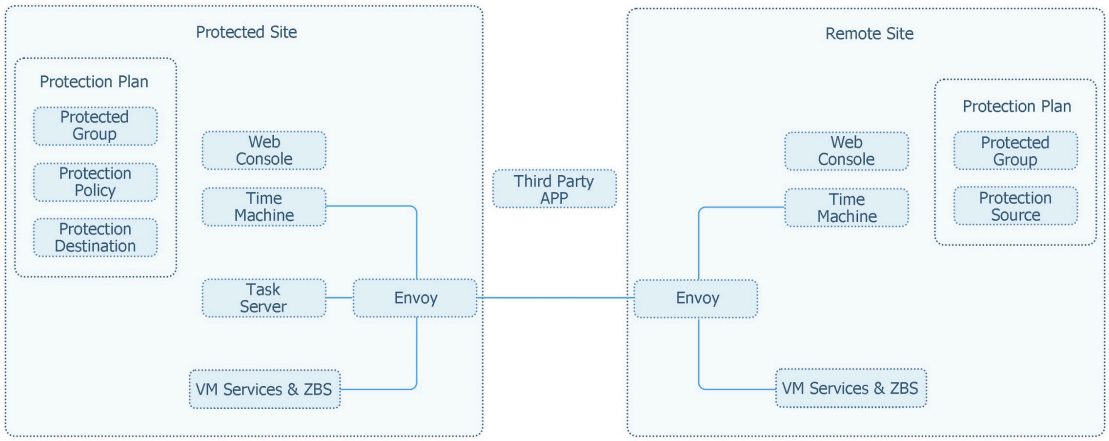
ZBS 提供本地快照计划和异地异步复制功能，其中异步复制目前仅支持异地目标站点也是 ZBS 集群的情形。

异步复制实际上就是将快照数据搬运到一个或多个异地存放的过程。但传输数据的网络通常需要经过广域网，带宽非常有限。为此，ZBS 对异步复制功能内建了许多数据传输量的优化，大大加快了异步复制数据传输的速度。

与双活集群的实时复制不同，异步复制是以复制非实时的快照到异地为基础，所以对延时要求低，从而使得受保护站点和异步复制目标站点的距离，在带宽足够的情况下，可以达数千公里。

## 架构

异步复制包括本地的受保护站点，以及一个或多个异地的站点，其基本架构如下：




受保护站点：

- 保护计划：快照计划和异步复制计划的统称，指定受保护对象、快照/异步复制策略以及远程异步复制站点（亦称为保护目的地站点、保护目的地或目的地）。不为保护计划指定远程异步复制站点时，保护计划被称为快照计划；保护计划包含远程异步复制站点时被称为异步复制计划。
- Web 控制台: SMTX OS 的 Web 控制台。它一方面从后端接收各种数据后，显示在用户界面；另一方面，它也接收用户的各种设置，传递给后端，并由后端处理。
- Time Machine（时光机）：负责保护计划任务的管理与触发。在异步复制站点中创建受保护对象的镜像对象。按照预设的保护周期在受保护站点中创建快照组，再将快照组内的数据写入至异步复制站点的镜像对象中。在数据传输完成之后，它再对异步复制站点中的镜像对象执行一次快照，得到镜像快照组。用户需要时可以从保护计划的执行记录中找到对应的快照组，选择回滚或者远程重建恢复当时的数据状态。在每次成功执行保护计划之后，Time Machine 还将按照指定的保留策略，清理过期快照组。
- Task Server：负责保护计划任务具体执行。受保护站点的 Task Server 将通过 Envoy 提供的远端接入服务来访问异步复制站点的块（block）服务， 直接对数据进行操作。此模块只处理存储卷和快照的异步复制、克隆、回滚等数据层的任务，不处理关联的元数据信息。
- Envoy：开源的专为大型现代服务导向架构设计的 L7 代理和通讯总线。Envoy是一个独立的进程，旨在与每个应用程序服务器并行运行。所有的 Envoy 形成一个透明的通信网格，每个应用程序发送和接收来自本地主机的消息，并且不知道网络的拓扑结构。Time Machine 利用 Envoy 管理与 ZBS 集群及远端站点的链路维护。在两个站点之间的 Envoy 将使用 SSL 加密通道进行传输。
- VM Services & ZBS：SMTX OS 中的虚拟机与块存储服务。

远程（异步复制）站点：

- 外来异步复制计划：与受保护站点的“异步复制计划”对应，简称为镜像计划。当受保护站点的受保护数据被异步复制到远程异步复制站点时，会在远程异步复制站点生成镜像计划，用来标示远程异步复制站点上这块数据的类型和用途，防止目的地站点修改镜像计划管理的数据。另外，也可方便目的地数据在受保护站点崩溃时，用目的地站点的异步复制数据进行重建。
- VM Services & ZBS、Time Machine、Envoy：与保护站点的同名模块相同。

## 增量复制

对一个受保护的对象（虚拟机、存储卷、iSCSI LUN 和 NFS 文件）的首次复制是一次全量复制，而之后则每次进行增量复制，大大降低了数据传输量。

在做增量复制时，仅计算本次快照和上次快照的数据块表的差异，得到那些被修改过的脏数据块（Extent，大小为 256 MiB）。由于这个计算过程是通过比较元数据进行的，所以不产生访问数据。ZBS 仅同步这些差异的数据块，并在异地目标端结合上次的异步复制快照来构建出本次快照的完整异步复制数据。

### 从异步复制恢复

从异步复制恢复是指将数据恢复（又称为回滚）至本地保留的或目标站点储存的快照数据。在每次回滚动作之前，Time Mahcine 会将回滚前的数据状态保存为临时快照。如果回滚动作失败，则会把数据恢复至临时快照，从而保证失败的回滚动作不会影响到原有数据的状态。所有的回滚动作针对的均只能是保护计划产生的快照组对象整体，不能仅选择部分对象进行回滚。

### 从异步复制中克隆

Time Machine 支持从异步复制中重新克隆出新的数据对象。克隆时可以选择克隆至其他远端站点。克隆时允许选择仅克隆快照组中的部分对象。

### 数据传输时优化

多集群间数据异步复制场景下，在源集群将需要异步复制的数据进行压缩后再进行网络传输，到异步复制集群进行解压后储存数据。这样能够减少实际的数据传输量，减小网络流量和带宽的消耗，从而减小对集群间网络带宽的依赖，并减少数据传输时间。

压缩算法使用 Zstandard，该功能默认开启。

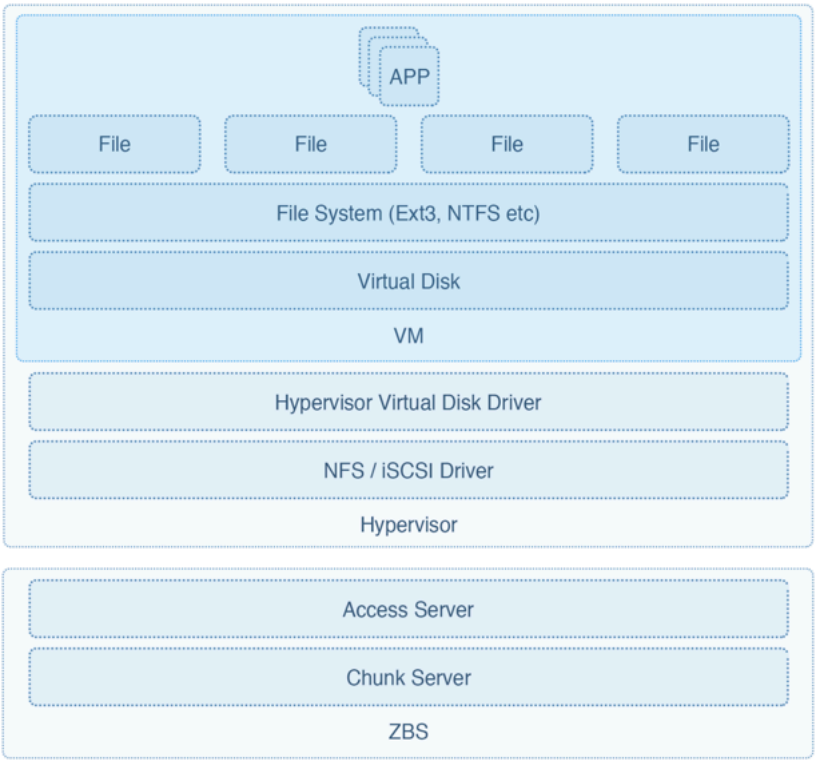
## 数据接入模式

ZBS 数据接入模式包括：

1. ZBS 卷：直接访问系统内置的 ZBS 卷对象。
2. NFS：通过 NFS 协议访问转化为 NFS 目录/文件 的 ZBS 存储对象。
3. iSCSI：通过 iSCSI 协议访问转化为 iSCSI 目标/LUN 的 ZBS 存储对象。

访问源本地的客户端（标准 NFS/iSCSI Client）通过 NFS/iSCSI 协议访问 AccessServer 提供的NFS/iSCSI 协议服务端。AccessServer 将 NFS/iSCSI 服务请求转化为标准的 ZBS volume 数据请求发往 ChunkServer。

一个典型的为虚拟机提供虚拟磁盘服务的 ZBS 接入层级简化示意图如下：




ZBS 以虚拟磁盘的形式为虚拟机提供存储服务。虚拟磁盘与 NFS File/iSCSI LUN 一一对应。虚拟机上的应用程序可见的文件对象，是虚拟机内部文件系统在虚拟磁盘之上包装的文件服务提供的。虚拟机内的应用程序对文件的操作首先会交由虚拟机内部的文件系统处理，文件系统将文件操作转化为虚拟磁盘的 IO 操作发往驱动层。Hypervisor 的虚拟磁盘驱动将对应的虚拟磁盘操作转化为对 NFS 文件/

iSCSI LUN 的 IO 操作放往 ZBS AccessServer。AccessServer 再将操作转化为 ZBS 数据块 IO 操作发往 ChunkServer。

在分离模式部署下，默认启动 DataChannel 线程分离。使 ZBS 的节点能够独占 CPU 资源，由此提高节点间数据传输的效率，以提高分离模式的写操作的性能。在超融合模式下，默认不启动此功能。

### 对 iSCSI PR 的支持

PR 是 SCSI 3 Persistent Group Reservation 的简称，是一种支持集群文件系统的底层技术，用于协调解决多个 Initiator 来访问同一个 SCSI 设备时的访问冲突问题。基本原理是通过一系列 SCSI 指令在一定的时间区间内获得对 SCSI 设备的排他性访问权限，避免并发更新引发的数据异常。通过对 PR 的支持，可以防止用户在多 Initiator 挂载的同一 iSCSI LUN 时进行多点写入，造成数据损失。

部署需要 PR 支持的文件系统，如 Windows Failover Cluster Filesystem，Initiator 端需要通过 iSCSI 协议直接连接并挂载 LUN 为本地磁盘，再启动相关文件系统功能。

Elf 或 Vmware 集群上的虚拟机无法直接在虚拟机虚拟磁盘上部署需要 PR 支持的文件系统。

### 支持 VMware ATS 挂载 ZBS 存储服务

用户可以通过 VMware 添加 VMFS 的方式，将 ZBS 集群提供的 iSCSI 数据存储挂载给 VMware 集群。从而实现分离模式部署，使 VMware 集群提供虚拟化服务，ZBS 负责提供数据存储服务。

## 异常处理

### 磁盘异常处理

ChunkServer 负责检查本地磁盘的健康状态。通常来说，磁盘故障分来两种：

1. 可直接检测到的故障。这种故障的现象是磁盘 IO 发生 IO 错误，磁盘无法被系统识别，磁盘检测工具报告磁盘状态异常等等。这种故障很容易被检测到。
2. 不可直接检测到的故障。这种故障并不会引发磁盘 IO 错误，也无法通过磁盘检测工具检测到，但磁盘读取返回的数据可能存在异常。这种故障通常由磁盘的固件缺陷、磁盘的比特翻转（bit flipping）引起。这种故障很难被检测到，只有开启数据校验以后才可能被发现。

以上的两种磁盘故障都会触发磁盘故障处理的流程。ChunkServer 在检测到这两种错误后，会将这个故障磁盘上错误包含的所有数据块汇报给 MetaServer，并立即触发数据恢复的过程。而有故障的磁盘，将不再被使用。

数据恢复任务以数据块（Extent）为粒度，将被均匀地分配给集群中所有健康的 ChunkServer，以达到并行数据恢复的效果。同时，为了最大化磁盘的使用效率，所有的数据恢复 IO 都是顺序 IO。

ZBS 允许用户对数据恢复的任务进行限速。如果用户想减少数据恢复任务对业务的影响，可以调低数据恢复的限速。

### 节点异常处理

当节点因为硬件故障、网络故障等导致不可访问，将触发节点异常处理。MetaServer 会负责为这个节点上的数据块创建一个恢复任务，并加入数据恢复调度队列。

节点异常有时候是间歇性的，例如网络抖动、服务器意外重启等。在全部数据恢复完以前，这个节点很可能又恢复了正常，重新加入了集群。

为了避免因这种场景带来的不必要的数据恢复，我们会对重新取得连接的节点上的数据进行检查。这种检查通过与其它节点比对数据块的版本（只需要比对版本，不需要比对数据）进行。如果版本是一致的，说明这个数据块依然是一个可用的副本，则对应的恢复任务可以被取消；如果版本不一致，则这个数据块的副本需要在这个节点上被删除掉，等待恢复任务恢复出正确版本的数据块副本。

### 服务异常时的可用性保证

ZBS 使用了 VIP 外加 iSCSI 重定向程序/ IO Rerouter 的多重可靠性行机制来保证节点/网络/服务异常时 IO 尽可能不受到影响。

根据数据 IO 路径中提到的，客户端在访问 ZBS 数据时，VIP 和 iSCSI 重定向程序/ IO Rerouter 能够保证客户端能够访问到当前可用的 AccessServer。所有的 AccessServer 服务均是等价的，可以提供数据访问服务。客户端接入任一 AccessServer 访问数据均是可靠与一致的。

在 IO 路径上如下角色节点发生异常时，数据 IO 服务均不会受到显著影响：

1. VIP Server （iSCSI Redirector）节点异常：ZBS 通过集群状态探测机制会即时将 VIP Service 切换至其它可用节点。客户端无需调整配置，异常之后新的探查路径请求将被导向新的 VIP Server。已建立连接的 IO 路径则不再需要 VIP Server 提供服务，不会受到影响。
2. AccessServer 节点异常：iSCSI 发起程序将发现连接短暂中断，重新向 VIP Server（iSCSI Redirector）请求新的 AccessServer 服务地址；IO 重定向程序会主动将 NFS 客户端的路由切换到可用的 AccessServer 所在的地址。客户端与新的 AccessServer 建立连接，继续服务。
3. 数据节点 ChunkServer 异常：MetaServer 会迅速的进行副本切换，剔除异常 ChunkServer, 保证 IO 顺利进行，同时触发恢复任务进行数据恢复。
4. Meta 节点异常：ZooKeeper 集群会重新选举，确定新的 Meta Leader 节点恢复服务。

# 与计算平台的集成

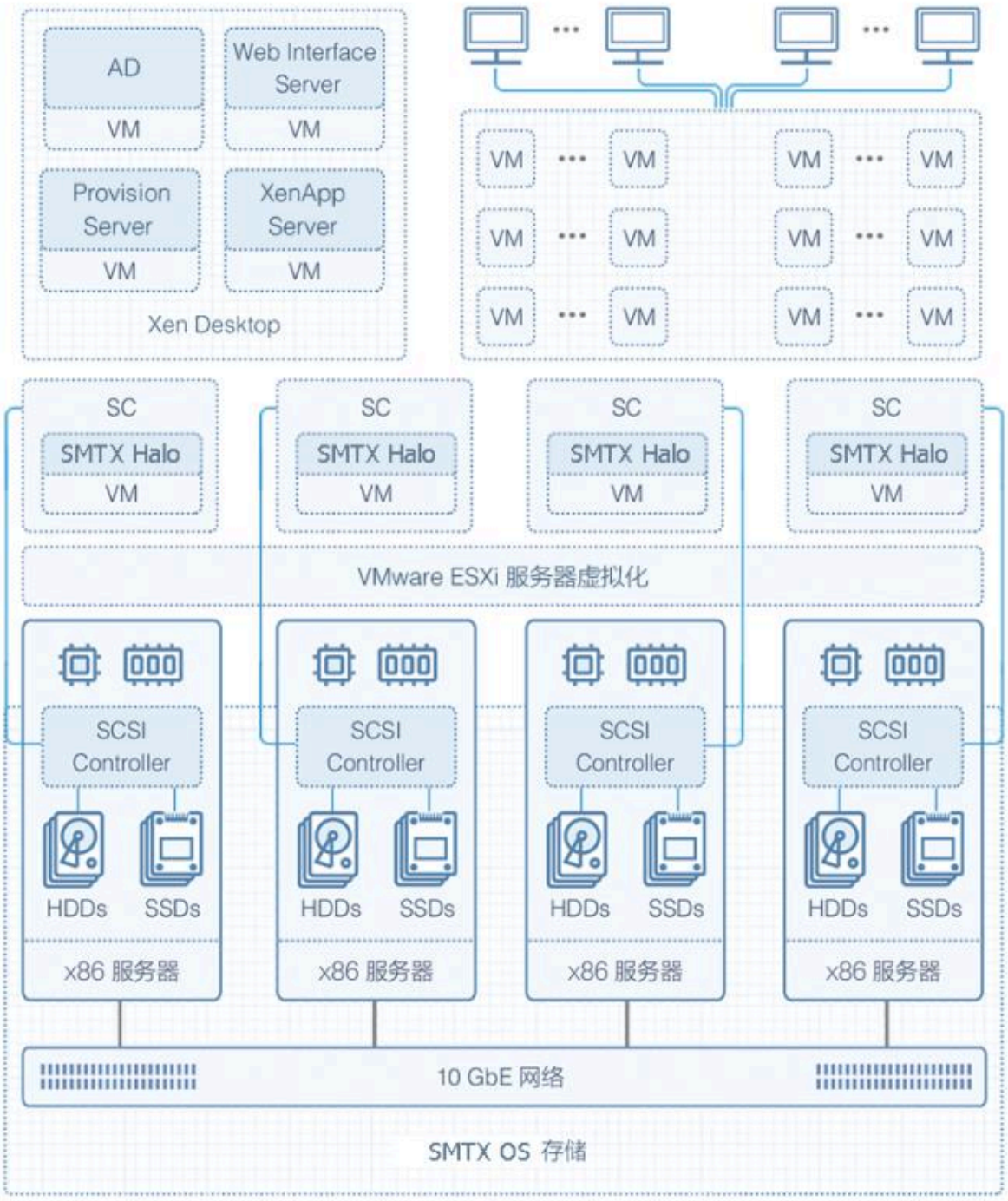
ZBS 可以与 VMware 虚拟化平台和 Kubernetes 进行集成。

## 与 VMware 的集成

SMTX 控制虚拟机（SMTX Controller Virtual Machine - SCVM）和 VMware vSphere 存储 API (VMware vSphere Storage APIs - Array Integration - VAAI）是与 VMware 集成相关的重要概念，在本节进行介绍。

### 部署架构

ZBS 与 VMware ESXi 的集成方式为：ZBS 作为存储服务运行在 SCVM 中，并对外提供标准的 NFS 访问接口。




## SCVM

SCVM 是运行在 VMware ESXi 上的一个虚拟机。和普通的虚拟机不同，这个虚拟机运行的是装有 ZBS 的操作系统（SMTX OS）。同时，需要将 VMware ESXi 的服务器上的存储磁盘直通（passthrough）给 SCVM，使得这些磁盘统一由 ZBS 负责管理。集群中多个 SCVM 通过万兆网络进行通信，组成了 ZBS 集群，并提供存储服务。

由于 ESXi 对直通操作的限制，只允许直通整个 IO 控制器或 RAID 卡（PCI 设备），而不是某个磁盘，且 ESXi 的系统盘所在的 IO 控制器或 RAID 卡无法被直通给 SCVM，所以 VMware ESXi 和用于存储的磁盘，需要接入不同的 IO 控制器或 RAID 卡。如果服务器不支持多个 IO 控制器或多个 RAID 卡，也可以选择将 VMware ESXi 安装在 SATADOM 上。 SATADOM 可以通过板载的 SATA 控制器，直接为 ESXi 提供存储服务，而不占用额外的 IO 控制器，或 RAID 卡。

直通使得 SCVM 对物理磁盘的访问可以不经过 ESXi，保证了物理磁盘的访问速度。

与此同时，每一个 SCVM 又对外暴露一个 NFS 的存储服务，VMware ESXi 可以通过私有网络访问到 NFS 服务，并创建数据存储，供给其它虚拟机使用。

每一个 SCVM 暴露的 NFS 服务监听地址相同，例如：192.168.33.2。因为 VMware ESXi 和 NFS 通信是通过私有网络，所以不同 SCVM 之间并不会存在地址冲突。由于挂载地址相同，不同的 VMware ESXi 会认为这是一个共享存储，可以提供对热迁移、虚拟机 HA 等高级功能的支持。

### VAAI

VAAI 是 VMware 开放的一组 API，用于在 ESXi 和存储后端之间进行通信。通过 VAAI，ESXi 可以将克隆、置零等操作，交给后端存储来完成，这样极大的降低了 ESXi 的负载，提高了 IO 操作的速度。

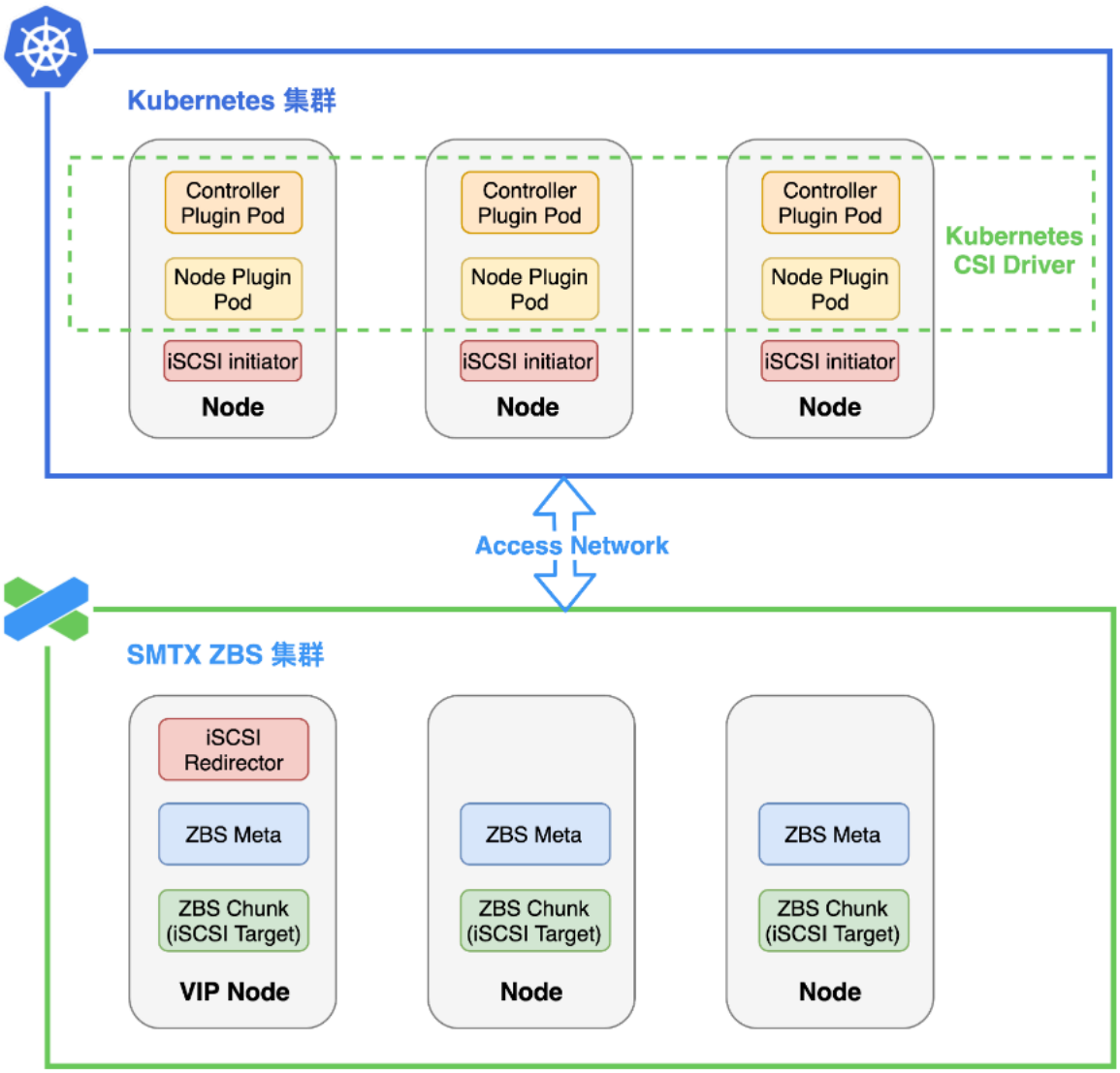
在目前 ZBS + VMware ESXi 的最佳实践中，推荐使用 NFS 接口访问 ZBS 存储服务，所以用户需要安装 ZBS VAAI-NAS 插件。通过这个插件，可以对 ESXi 的虚拟机克隆操作进行加速，实现秒级克隆，且克隆操作对性能没有任何影响。

## 与 Kubernetes 的集成

Kubernetes CSI Driver 是 SmartX 开发的符合 Kubernetes CSI 规范的 CSI Driver。通过它，Kubernetes 可以以 iSCSI 的方式从 SMTX ZBS 集群中获得持久存储服务。当前支持版本为 Kubernetes CSI 规范 1.4。

Kubernetes CSI Driver 存储插件运行在 CSI Pod 中，CSI Pod 通过此插件以 Remote Procedure Call 的方式管理持久卷的生命周期，包括创建、删除、挂载和卸载持久卷等操作。每个持久卷在 SMTX ZBS 存储集群中对应一个 iSCSI LUN。当 Kubernetes 中的 Pod 需要使用该卷时，将通过 iSCSI 协议在 Pod 所在的物理节点上挂载该 LUN，并在 LUN 上构建文件系统，将得到的文件系统挂载到 Pod 作为数据目录使用。

Kubernetes 集群通过 Kubernetes CSI Driver 接入 SMTX ZBS 集群，部署形态如下图所示。




- **CSI Plugin Pod**

  Kubernetes CSI Driver 以 Pod 的形式部署在 Kubernetes 集群中，根据其部署的 Pod 不同，所提供的功能也不同。

  - Controller Plugin Pod

    Controller Plugin Pod 以多副本的形式运行在 Kubernetes 集群中的多个节点上（默认为 3 个节点），负责存储对象（Volume）的生命周期管理。运行在 Controller Plugin Pod 中的 Kubernetes CSI Driver 容器主要提供控制服务，包括创建和删除 CSI Volume，对 CSI Volume 执行 Attach/Detach，以及快照等操作。
  - Node Plugin Pod

    Node Plugin Pod 运行在 Kubernetes 集群中的每个节点上，负责与其所在节点上使用 Volume 的容器进行交互。运行在 Node Plugin Pod 中的 Kubernetes CSI Driver 容器主要提供节点服务，执行当前节点上的 Volume 挂载、卸载或扩容等操作。
- **Access Network**

  Kubernetes 集群和 SMTX ZBS 集群通过 Access Network （接入网络） 进行通信，该网络需要独立于 Kubernetes 集群中的业务网络和 SMTX ZBS 集群使用的存储网络。

- **ZBS iSCSI Redirector**

  SMTX ZBS 集群使用 iSCSI Redirector 模式提供 iSCSI 接入服务。 iSCSI Redirector 运行在服务虚拟 IP （VIP） 节点上。SMTX ZBS 集群部署完后会默认配置一个 VIP。当 VIP 节点异常时，集群会自动切换到新的 VIP 节点提供 iSCSI Redirector 服务，从而保证 iSCSI Redirector 的高可用性。

  Kubernetes CSI Driver 使用的 iSCSI 服务端地址为 iSCSI Redirector 地址，iSCSI initiator 尝试连接 iSCSI Server 时，Redirector 将会引导 initiator 重新连接向一个可用的 iSCSI Server。在重定向之后，所有的数据请求仅在 iSCSI initiator 与 Server 之间进行，无需经过 Redirector。

# 第 2 章

# 附录

## 术语与缩略语

**SSD**

Solid State Disk/Drive

固态硬盘。一种以闪存作为永久性存储器的电脑存储设备。

**HDD**

Hard Disk Drive

**SATA**

Serial Advanced Technology Attachment

串行 ATA。一种电脑总线，负责主板和大量存储设备（如硬盘及光盘驱动器）之间的数据传输。

**SATA DOM**

SATA disk-on-a-module

**LUN**

Logical Unit Number

逻辑单元号

**LSM**

Local Storage Manager

SmartX 在裸设备上构建的文件系统，在 ChunkServer 中使用。

**缓存盘**

Cache Disk

缓存盘用于缓存访问次数较多的数据，通常会使用读写速度更快的固态硬盘。

**数据盘**

Data Disk

数据盘用于存放访问相对较少的数据，通常使用机械硬盘作为数据盘。

**iSCSI**

Internet Small Computer System Interface

互联网小型计算机系统接口。

一种基于 TCP/IP 及 SCSI-3 协议下的存储技术，用来建立和管理 IP 存储设备、主机和客户机等之间的相互连接，并创建存储区域网络（SAN）。

**NAS**

Network Attached Storage

**HA**

High Availability

高可用性。指在某主机故障时，自动将业务迁移至健康主机的行为，从而提高整个系统的可用性。

**VNC**

Virtual Network Console

虚拟网络控制台。一款远程控制的工具软件。

| | |
|---|---|
| **DataStore** | VMware vSphere DataStore |
| **OVS** | Open vSwitch |
| **UUID** | Universally Unique Identifier<br>通用唯一识别码。在一台机器上生成的数字，它保证对在同一时空中的所有机器都是唯一的。它通常包括当前的日期和时间、时钟序列以及全局唯一的 IEEE 机器识别号。 |
| **VLAN** | Virtual Local Area Network<br>虚拟局域网 |
| **DHCP** | Dynamic Host Configuration Protocol<br>动态主机配置协议。通常被应用在大型的局域网络环境中，主要作用是集中管理、分配 IP 地址，使网络环境中的主机动态获得 IP 地址、网关地址、DNS 服务器地址等信息，并能够提升地址的使用率。 |
| **TOR** | Top of Rack<br>柜顶式交换机。 |
| **IPMI** | Intelligent Platform Management Interface<br>智能平台管理接口。一种开放标准的硬件管理接口规格，定义了嵌入式管理子系统进行通信的特定方法。 |
| **副本集** | Replica set<br>一组保留有相同数据的 MongoDB 进程。通过将数据部署在多个不同的服务器上，防止因单机故障而造成数据的丢失，借助数据冗余来提高数据的可靠性和安全性。 |
| **可重入性** | 若一个程序或子程序可以“在任意时刻被中断，然后操作系统调度执行另外一段代码，这段代码又调用了该子程序不会出错”，则称其为可重入的（reentrant）或具有可重入性（reentrancy）。 |
| **DAG** | Directed Acyclic Graph<br>有向无环图。 |
| **DRF** | Dominant Resource Fairness |
| **COW** | CopyOnWrite<br>写时复制。 |
| **LRU** | Least Recently Used<br>近期最少使用算法。 |
| **MongoDB** | 一个基于分布式 NoSQL 数据库。由 C++ 语言编写。旨在为 Web 应用提供可扩展的高性能数据存储解决方案。 |

| | |
|---|---|
| **CSI** | Container Storage Interface<br>容器存储接口。 |
| **Kubernetes（k8s）** | 开源的容器编排系统 |
| **PV** | Persistent Volume<br>持久卷。Kubernetes 中一种存储原语。持久卷是外部存储系统中的一块存储空间，由管理员创建和维护。保存在持久卷中的数据不会因为 Pod 生命周期的结束而消失。 |
| **PVC** | PersistentVolumeClaim<br>Kubernetes 中一种存储声明原语。PersistentVolumeClaim (PVC) 是对 PV 的申请 (Claim)。PVC 通常由普通用户创建和维护。需要为 Pod 分配存储资源时，用户可以创建一个 PVC，指明存储资源的容量大小和访问模式（比如只读）等信息，Kubernetes 会查找并提供满足条件的 PV。 |
| **Storage class** | Kubernetes 中一种资源动态提供方式。 |
| **Sidecar** | 一种辅助或增强容器的容器总称。 |
| **精简置备** | 精简置备的磁盘，其存储空间使用量根据实际使用量动态变化，为其置备的存储空间为可以使用的最大值。精简置备的磁盘一开始仅使用磁盘初始操作所需的存储空间，如果磁盘未来需要更多的空间，则可以扩展到预设的最大值。 |
| **厚置备** | 厚置备的磁盘，在创建时即为该磁盘置备固定量的存储空间。不论该磁盘实际用量多少，均占据整个置备的存储空间。 |
| **socket** | 物理机主板上用来安装 CPU 的插槽。 |
| **core** | CPU 的处理器。一个 CPU 可以有多个处理器，每个处理器都可以独立执行计算任务。 |
| **RPO** | Recovery Point Objective<br>数据恢复点目标。主要指业务系统所能容忍的数据丢失量。 |
| **RTO** | Recovery Time Objective<br>恢复时间目标。主要指能容忍的业务停止服务的最长时间，即从灾难发生到业务系统恢复服务功能所需要的最短时间周期。 |