基于 ceph RBD 的 Iscsi target 实现分析

runsisi AT hust.edu.cn
2015/06/12

本文总结基于 ceph 存储后端的 iSCSI target 实现，针对几种可能的实现思路进行分析，并输出本文档，本文内容组织结构如下：

1. 描述本文的目的及内容；

2. iSCSI 使用 ceph 作为存储后端的基本实现思路；

3. 分别阐述几种可能的实现方式，分析其各自的特点和可能存在的一些技术障碍；

4. 最后简单总结全文。

1. 问题提出

ceph 作为一个真正意义上的统一存储系统，有着很好的应用前景，但到目前为止有多种原因限制了其在传统存储应用领域真正大范围的应用，如客户端仅支持 GNU/Linux 系统，内核态客户端实现也仅会合入高版本的内核中等。而对于 iSCSI 这种传统的存储应用而言，由于客户端配置简单且足够通用，常见的各种系统（包括操作系统和应用系统）一般都对 iSCSI 有很好的支持，因此为了扩大 ceph 的应用范围，特别是应对只支持 iSCSI 的系统，ceph 必须通过某种途径实现对 iSCSI 的支持。

2. 基本思考方向

ceph 集群目前支持三种形式的存储接口：文件、对象、块，其中块接口(即 RBD)与 SCSI 块设备读写所要求的接口一致，因此可以基于 ceph 的 RBD 提供 SCSI 存储系统后端，当然如果有足够信心的话也可以完全抛弃 ceph 提供的这三种基础接口，而在原始的 RADOS 接口上开发新的块接口，当然除非原始的 RBD 接口有重要缺陷，否则暂时还看不到重新发明轮子的必要，注意后文的讨论都将基于这一基本假设。

基于 RBD 的 iSCSI 支持有多种实现思路，目前考虑到的有如下三类，概述如下：

1) 为 iSCSI 客户端增加 RBD 客户端的功能，即在客户端实现 iSCSI initiator 与 RBD client 的融合；
2) 类似于 MDS 的实现，为 ceph 集群增加 LUS（Logical Unit Server）服务器，由 LUS 负责 iSCSI 业务接入；
3) 选择合适的 iSCSI target 为其增加 RBD 后端支持，并在终端用户与 ceph 集群之间架设 iSCSI 网关（可以扩展成其它类型的如 FC 网关）；

3. 具体实现分析

上一节针对 ceph 支持 iSCSI 的需求提出了三种可能的实现方向，下面将对这几种思路进行展开分析。

需要注意的是，由于 ceph 集群的设计，RBD 设备的读写需要专门的 RBD 客户端，因此支持直接读写 RBD 的机器都需要安装 RBD 客户端软件。目前已有的 RBD 客户端有两种实现，一种是直接集成到内核中的内核驱动形式的实现，另一种是动态库形式的用户态实现（包括 python、java 等语言的绑定），这两种实现暂时都只支持 GNU/Linux 系统，同时相比后者，前者还存在一些问题，如：1）只有较新的内核才支持 RBD，2）某些发行版不一定会将 RBD 支持编译进内核，3）ceph 的开发迭代周期很短，新特性、BUG 修复等不可能很及时的合入内核，4）稳定性还有待验证。基于这些问题的考虑，下文如无特殊说明将不考虑使用内核态的 RBD 模块作为 RBD 客户端。

3.1. 增强型 iSCSI initiator

类似于已使能 RBD 支持的 qemu，这种方式在原有的 iSCSI 客户端基础上集成 RBD 管理、读写等功能，当然这种实现远远比 qemu 支持 RBD 要复杂，qemu 只需要在用户态直接调用 RBD 接口，而 iSCSI initiator 需要在内核中注册块设备，然后将读写请求发往用户态的 RBD 客户端，图 1 的简单对比可以看到两者实现的主要差异。

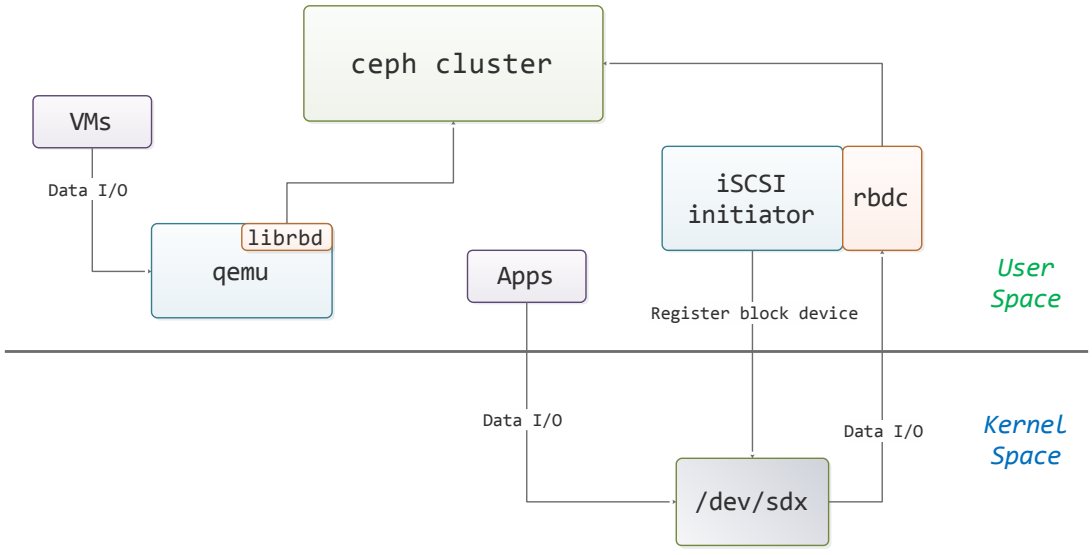

图 1 iSCSI initiator enhanced

使用增强型 iSCSI initiator 的实现存在如下一些问题：

1） 实现稍嫌复杂，需要开发内核模块，转发 IO 请求到用户态 RBD 客户端的处理流程可能存在一定的技术障碍（可以参考 UIO 以及 TCMU）；

2） 普通的 iSCSI initiator 能够发现的 LUN 都是管理员在 iSCSI target 端针对各个客户端进行配置的，因此在这里需要将相关配置前移至终端用户侧；

3） 由于对 iSCSI initiator 进行了修改，用户需要安装新的软件，需要熟悉新的配置，而已有的自动化软件也可能存在兼容性问题；

4） 最后一个非常致命的问题，GNU/Linux 下修改 iSCSI initiator 容易，但是 Windows 等系统下的闭源客户端我们无法进行修改，而且 RBD 客户端暂时也不支持相关的操作系统。

ceph 支持 iSCSI 的主要目的就是为了解决通用性问题，但显然增强型 iSCSI initiator 并不能解决该问题，如果只需要支持 GNU/Linux 下的 iSCSI 应用，则这种思路应该是值得尝试的。

3.2. LUS 服务器

与增强型 iSCSI initiator 的出发点相反，终端用户不用做任何改变，仍然继续使用原有的 iSCSI 系统或设备，所有的修改都在服务端即 ceph 集群侧进行。

ceph 集群由三种类型的服务角色组成，包括：MON、OSD 以及 MDS，其组织结构如图 2 所示。

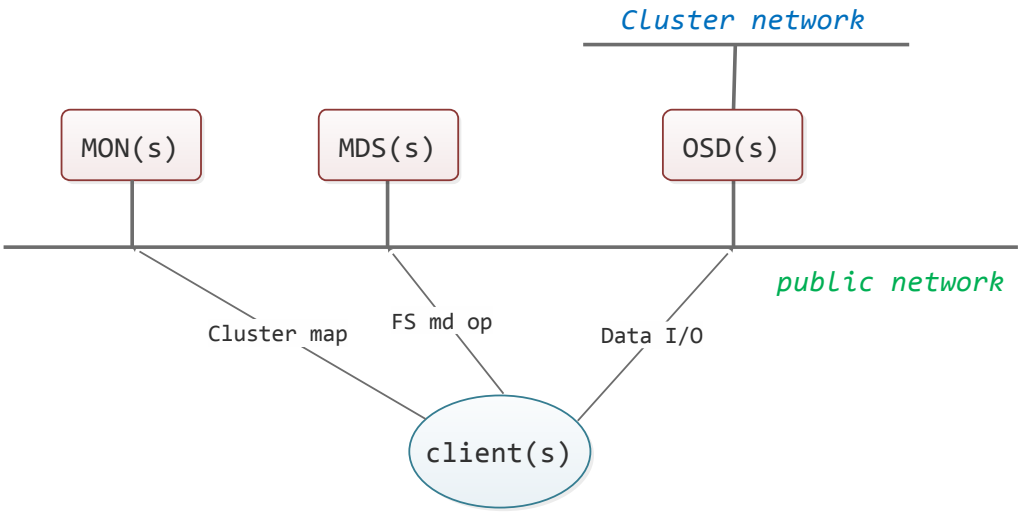

图 2 ceph 集群结构

其中 MON(Monitor)是集群的控制中心，主要通过管理 MON map、OSD map 等几张集群状态表维护集群状态的一致性；OSD(Object Storage Device)是集群数据存储仓库，其内部以对象的形式组织数据，OSD 可以相对自由的加入或退出集群从而实现集群存储能力动态调整的功能；MDS(MetaData Server)负责 posix 文件系统访问接入，实际上就是文件元数据的管理，但需要注意的是 MDS 本身并不存储数据，所有的文件元数据以及真实数据都存储在 OSD 上，MDS 可以看作一个大的文件元数据缓存服务器。

对一个存储系统最终的要求就是存取数据，实现这样的系统从根本上来说应该只需要两类角色：1)控制器、2)存储设备。通过 ceph 提供的 rados、rbd 接口可实现对 ceph 存储集群的读写，因此实际上 ceph MON+OSD 的架构是完全满足存储系统的要求的。MDS 的存在是为了兼容传统的文件存储接口，照此类推，如果要兼容 iSCSI 接口也只需要增加 LUS(Logical Unit Server)服务器即可，当然为了 HA 考虑，LUS 集群的也需要在 MON 的 LUS map 表中进行注册，此时 ceph 集群结构如图 3 所示。

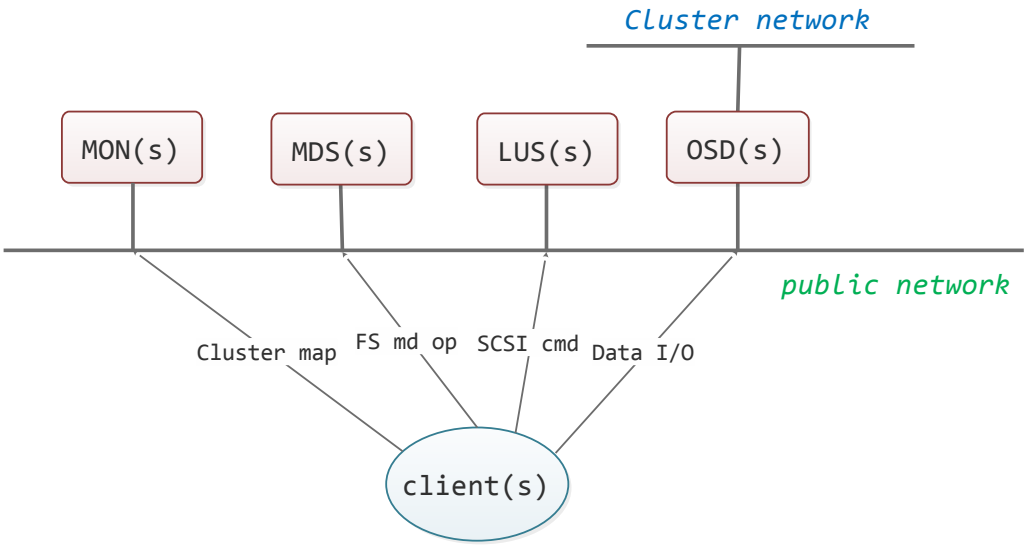

图 3 支持 SCSI 的 ceph 集群结构

LUS 看上去很好，但是实际上仍然存在很多问题，考虑 ceph 支持 posix 文件访问接入的实现即可看出问题所在。

1) 本地 GNU/Linux 系统访问 MDS 需要通过 ceph 网络文件系统内核模块的支持，即由该内核模块实现类似 ceph 文件访问客户端的功能，因此即使在集群中增加 LUS 服务器，终端用户机器上仍然需要一个类似的模块实现与 ceph 集群的交互，这又引入了增强型 iSCSI initiator 所遇到的问题，闭源的 iSCSI initiator 实现我们很难将它与 ceph 客户端模块融合，考虑到 iSCSI 的通用性，这是无法回避的问题；

2) 由于 iSCSI initiator 与 target 的交互实际上是在内核态，因此该模块也必须是内核模块，整个 ceph 集群的交互又得全部实现一遍，和从头实现 ceph 文件系统内核模块没什么差异，工作量大、维护麻烦且稳定性是否有保证还不得而知；

3) 如果 SCSI 命令不在 iSCSI initiator 所在机器上解析，则 SCSI 命令必须先由 LUS 处理然后转成 RBD 读写，或者在 iSCSI initiator 所在机器上只解析 iSCSI Data Out/Data In 命令，其它的管理命令仍然发给 LUS 去处理，这样问题都是模块实现时需要考虑的，

因此该模块的实现不一定会比已有的文件系统模块简单；

4） MDS 是 ceph 最早支持的功能，但直到现在其功能和稳定性仍然达不到生产环境的要求，至于新增 LUS 的工作量也不好说；

5） HA，LUN 配置同步等问题也需要 LUS 集群解决，当然这是所有 iSCSI 集群都会遇到的难题。

3.3. iSCSI 网关

对上一节的 LUS 进行展开，可以自然而然的想到 iSCSI 网关，将 LUS 从 ceph 集群中移出来作为 ceph 集群的客户端，同时又作为终端用户的 iSCSI 服务端，此时整个系统的组织架构如图 4 所示，这里的 iSCSI 网关实际上就是一个以 ceph 集群为存储后端的 iSCSI target。当然图 4 中的 iSCSI 网关还可以进一步扩展成 FC 网关等其它任何类型的网关，实际上 ceph 对外所提供的与 swift、S3 兼容的对象存储（注意这里的对象与 ceph 集群内部的数据对象不是同一个概念）同样也是以网关的形式存在的。

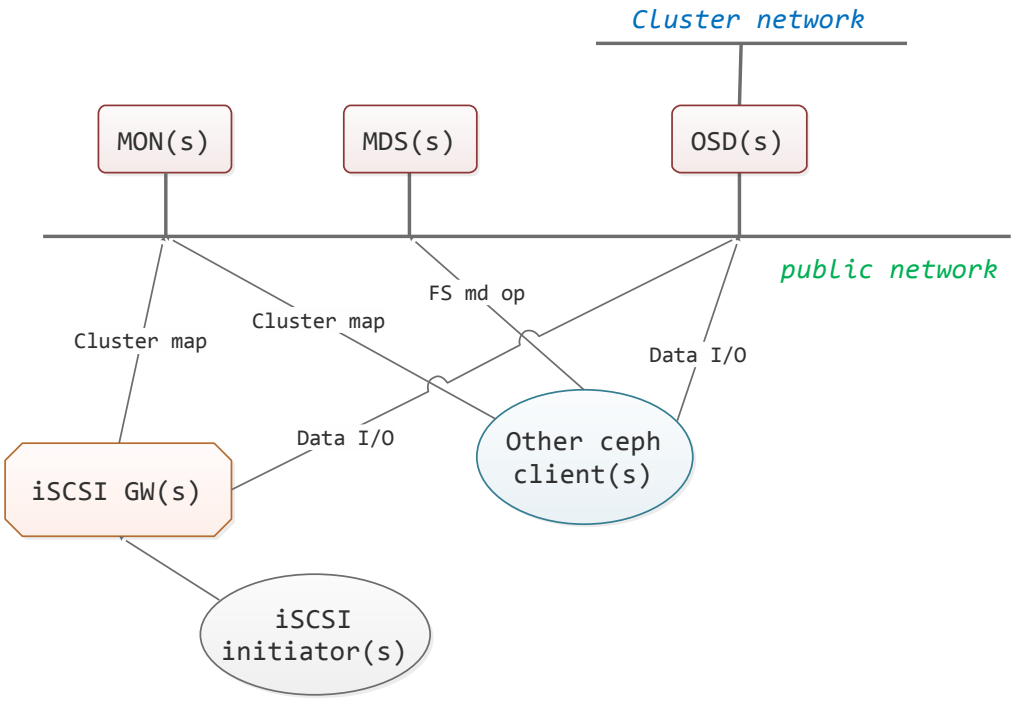

图 4 iSCSI 网关系统组织架构

先简单介绍一下 iSCSI target 的基本原理。对于 GNU/Linux 或 FreeBSD 上较新的 iSCSI target 实现而言，iSCSI target 并不是一个孤立的功能模块，系统对 iSCSI、FC 等 SCSI 传输层以后多种后端存储的支持构成了整个 SCSI target 框架，而 iSCSI 功能只是 target 框架中的的一个前端模块，典型的 target 框架如图 5 所示，分为前端接入模块、SCSI 框架、后端存储引擎三个部分。

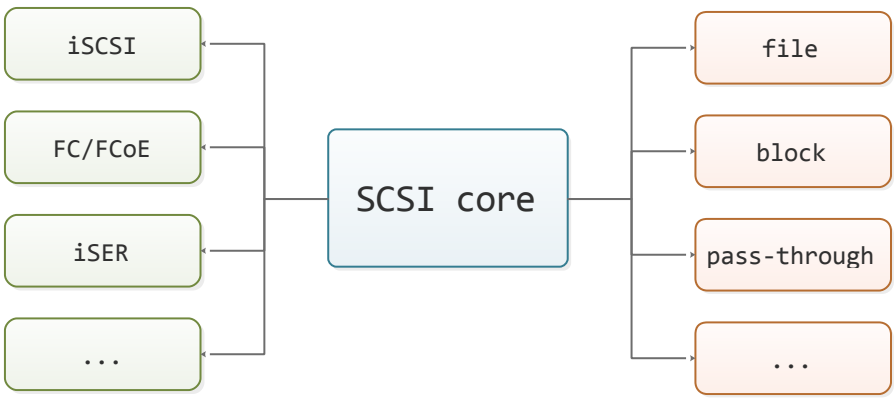

图 5 target 框架

通常而言，iSCSI 前端是 target 框架最早实现且支持的最好的前端模块，因此如果是在已有的框架上进行 iSCSI 支持则所需要的开发工作并不多。而同时，由于存储后端也是模块化的，即使是类似 IET(iSCSI Enterprise Target)这样的纯 iSCSI target 实现，其存储后端的读写也是使用回调形式实现的，因此如果需要新增存储后端，需要做的工作就是按照预定义的回调接口处理 SCSI 读写命令即可。

回到 iSCSI 与 ceph 的 RBD 结合的问题上来，由于网关是独立于终端用户以及 ceph 集群之外独立的机器，因此问题得到了极大的简化，为了使得 ceph 集群提供的 RBD 块设备能够被 iSCSI 网关作为 LUN 导出给终端用户使用，这里有两种实现思路：1）使用内核态 RBD 客户端将 RBD 块设备映射到 iSCSI 网关本地，然后再由 iSCSI target 将映射到本地的块设备作为 LUN 导出给终端用户使用，2）选择合适的 iSCSI target，并为其增加用户态的 RBD 存储后端。下面将对这两种实现思路进行具体分析，注意这里我们只讨论 GNU/Linux 作为网关机器操作系统的情况。

3.3.1. 映射 RBD 到本地再导出

由于 GNU/Linux 上常见的 iSCSI target 都支持将块设备作为 target 的后端存储，而 ceph 的内核态 RBD 客户端支持将远端的 RBD 块设备映射到本地，因此可以选择合适的 iSCSI target 将映射到本地的块设备作为 LUN 导出，这是现在流行的或者说暂时相对可行的 ceph iSCSI 解决方案，具体实现流程如下：

1） ceph 的内核态文件系统客户端模块在 2.6.34 版本合入内核主线分支，内核态 RBD 客户端的合入更晚，因此需要升级网关机器的内核到较新的版本（3.xx ?）；

2） 选择合适的 iSCSI target 软件（后文对常见的 target 实现有简单的对比），并在 iSCSI 网关机器上安装；

3） 在网关机器上安装 ceph RBD 用户态软件包；

4） 使用 RBD 命令映射远端 RBD 设备到网关本地；

5） 配置 iSCSI target，将映射到本地的 RBD 设备作为后端 LUN 导出。

从这些流程来看，虽然忽略了一些机器重启等异常情况的考虑，但总的来说这种实现思路实际上是非常简单易行的，并且由于所有的数据路径都在内核态，除非内核态 RBD 实现的不好，否则性能应该不存在太大问题。当然除了优点之外也存在如下一些问题：

1） 如前所述，对于 GNU/Linux 系统来说由于内核更新速度太快，接口变动频繁，内核模块的升级始终是个大问题，因此新特性的支持不可能做到及时提供；

2） 同时内核模块的稳定性也仍然存在一定的顾虑；

3） 前面在 LUS 问题总结中提到的 HA，LUN 配置同步等问题也没有好的解决方法。

总的来说，如果是要项目快速上马，要求迅速推出 iSCSI 功能，这可能是一个不错的解决方案，毕竟外面都是这样用的。

3.3.2. 将 RBD 作为存储后端直接导出

出于对内核态 RBD 驱动的顾虑，而且如果使用内核态 RBD 驱动的话可以直接参考前一种解决方案，此处的新思路是增加新的 iSCSI target 存储后端，新的存储后端通过调用用户态的 RBD 接口实现数据的读写。

GNU/Linux 系统中流行的 SCSI target 框架或 iSCSI target 实现有如下几种：tgt、IET、SCST、Linux-IO，再加上 ZXUSP 上的实现，下面对它们与用户态 RBD 接口融合的可行性及一些特点进行简要总结。

1） tgt

tgt 是一个用户态的 SCSI target 框架，在 GNU/Linux 内核直接集成 SCSI target 框架之前，这是一个绝对主流的框架。由于是基于用户态的框架，因此对集成同在用户态的 RBD 接口是非常简单的，实际上早在 2013 年就已经有 ceph 的开发人员就开发了适配 tgt 的 RBD 后端，并且合入了 tgt 的主干分支，如图 6 就是我测试 tgt + librbd 后端的情况（注意数据不重要，关键是验证功能）。

作为用户态框架，不可避免的就是性能问题，根据网上的相关数据，tgt 在使用本地存储的情况下，性能相比后面会提到的 IET、LIO 等是有一定差距的，如果从性能的角度考虑，tgt 应该是不推荐使用的，但在新特性验证等方面是很合适的。

```
root@runsisi-hust:/home/runsisi# parted /dev/sdc print
Model: IET VIRTUAL-DISK (scsi)
磁盘 /dev/sdc: 10.5GB
Sector size (logical/physical): 512B/4194304B
分区表：gpt
Disk Flags:

数字  开始：  End     大小    文件系统  Name  标志
 1    1049kB  9663MB  9662MB  xfs       hust

root@runsisi-hust:/home/runsisi# dd if=/dev/zero of=/dev/sdc1 bs=1M count=300
记录了300+0 的读入
记录了300+0 的写出
314572800字节(315 MB)已复制，40.9631 秒，7.7 MB/秒
root@runsisi-hust:/home/runsisi#
```

图 6 tgt + librbd

2) IET

IET 是 iSCSI Enterprise Target 的缩写，是一个纯粹的 iSCSI target 实现，实际上这也是 SCST 等框架的参考实现，不过其早在 2010 年就基本上停止开发了。代码实现分两部分，iSCSI 登录阶段的处理在用户态处理，然后 iSCSI 全特征阶段由于主要是数据的读写操作，对性能要求比较高，所以都放在内核态以内核驱动的形式实现。

由于数据读写部分在内核态，但 RBD 接口在用户态，因此需要有一种机制将内核态的读写请求转发到用户态然后再由用户态 RBD 接口处理，目前比较流行的方法是使用 UIO，如 DPDK 提交报文给用户态处理就是采用的这种方式，因此这里可以将读写命令进行封装然后提交给用户态程序调用 RBD 接口进行处理。通过这种途径，理论上是能够解决内核态调用用户态接口的问题的，但需要开发用户态驱动仍然不是一件非常容易的事情。

虽然 2010 年以后 IET 的代码继续有内核兼容性、LUN 预留方面的更新，现有的 Ubuntu 等发行版的软件源里也有相关的软件包下载，但总的来说这个项目基本濒临废弃，加上如果要集成用户态 RBD 接口也有一定的技术风险，因此不推荐。

3) SCST

SCST 是 SCSI target subsystem 的简称，代码继承自 IET，也分用户态和内核态两部分，不过由于已经扩展成一个 SCSI target 框架，因此功能远远比 IET 丰富，有多种前后端支持。实际上当时 SCST 是与 LIO 一起竞争进入 GNU/Linux 内核的，只是后来 SCST 失败了。SCST 支持用户态后端，而且根据其官方说明，相比纯内核态实现只有万分之一数量级的性能损失，因此实际上也就是说前面提到的 IET 需要自己实现的用户态驱动 SCST 框架已经实现了。

4) LIO-TCMU

LIO 也即 Linux-IO，是目前 GNU/Linux 内核自带的 SCSI target 框架（自 2.6.38

版本开始引入，真正支持 iSCSI 需要到 3.1 版本），对 iSCSI RFC 规范的支持非常好，包括完整的错误恢复都有支持。整个 LIO 是纯内核态实现的，包括前端接入和后端存储模块，为了支持用户态后端，从内核 3.17 开始引入用户态后端支持，即 TCMU(Target Core Module in Userspace)，TCMU 基于前面所提到的 UIO，其最初的设计目标就是为了支持用户态的 RBD、gluster 等，因此如果需要 LIO-TCMU 支持用户态 RBD，只需要设计用户态应用程序即可。

目前在 github 上已经有红帽的开发者写了一个 TCMU 的用户态程序框架，其设计目的是让用户态存储后端开发人员只需要开发相关的插件（即动态库）即可，由框架自动调用相关的插件处理 SCSI 命令，我已经实现了一个简单的 RBD 插件，效果如图 7 所示（同样不要纠结于性能，我用的是单线程、同步 IO）。

```
root@runsisi-hust:/home/runsisi# parted /dev/sdc print
Model: LIO-ORG rbd (scsi)
磁盘 /dev/sdc: 9664MB
Sector size (logical/physical): 512B/512B
分区表：gpt
Disk Flags:

数字  开始：  End     大小    文件系统  Name  标志
 1    1049kB  9663MB  9662MB            hust

root@runsisi-hust:/home/runsisi# dd if=/dev/zero of=/dev/sdc1 bs=1M count=300
记录了300+0 的读入
记录了300+0 的写出
314572800字节(315 MB)已复制，920.335 秒，342 kB/秒
root@runsisi-hust:/home/runsisi#
```

图 7 LIO-TCMU + librbd

实际上我手上也没有 SCST 与 LIO 实际的性能数据对比，但从社区的支持度，功能特性的丰富度来说，SCST 并不如 LIO，如果要两者二选一的话，LIO 似乎更合适。

通过前面的总结，我应该是更倾向于基于 LIO-TCMU 来实现 iSCSI 网关，不过前面提到过的 HA、LUN 配置同步等是 iSCSI 网关无法回避的一个问题，ceph 是一个 HA 集群，但现有的 SCSI target 框架都没有好的 HA 解决方案，配置数据可能采用怎样的方式保持一致性等问题都需要进一步的学习研究。

4. 总结

本文就几种 ceph 支持 iSCSI 服务的实现思路进行了简要分析总结。从个人纯技术的角度来看，使用 LIO-TCMU 来实现 iSCSI 网关应该是一个可行的方向，除了要解决 HA 等集群 iSCSI 都需要面对的问题，其它方面应该没有太大技术难题。

5. 附录

1） 实际上关于 iSCSI 网关的实现还有一种思路，将 iSCSI 网关假设在 ceph 集群所在的节点上，不过这种方案感觉不太合适，如有必要再做分析。

2） 文内有许多地方需要标明参考文献，由于时间比较匆忙，未来得及补上，不过通过搜索引擎应该能够很容易搜索到相关信息。