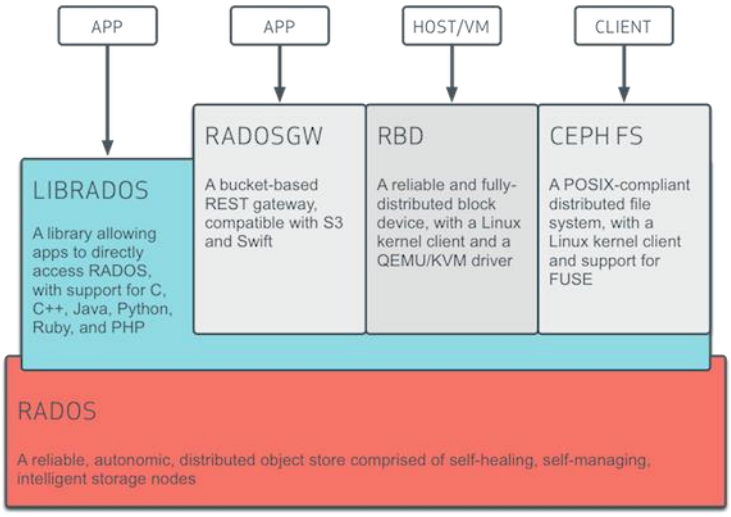
分布式文件系统代表：CEPH



Object：有原生的 API，而且也兼容 Swift 和 S3 的 API。
Block：支持精简配置、快照、克隆
File：Posix 接口，支持快照。
Ceph 底层是 PRDOS，是一个标准的对象存储，在此基础上 Ceph 提供文件，块和对象三种存储服务，其中 RBD 提供了块存储，CephFS 模块和相应的 Clinet 提供了文件服务。当然，Ceph 也通过 RadosGW 存储网关模块向外提供对象存储服务，并且和对象存储的事实标准 Amazon S3 以及 Swift 兼容。
Ceph 特点

-

高性能
a. 摒弃了传统的集中式存储元数据寻址的方案，采用 CRUSH 算法，数据分布均衡，并行度高。
b.考虑了容灾域的隔离，能够实现各类负载的副本放置规则，例如跨机房、机架感知等。
c. 能够支持上千个存储节点的规模，支持 TB 到 PB 级的数据。

高可用性
a. 副本数可以灵活控制。
b. 支持故障域分隔，数据强一致性。
c. 多种故障场景自动进行修复自愈。
d. 没有单点故障，自动管理。
高可扩展性
a. 去中心化。
b. 扩展灵活。
c. 随着节点增加而线性增长。
特性丰富
a. 支持三种存储接口：块存储、文件存储、对象存储。
b. 支持自定义接口，支持多种语言驱动。

1.5 三种存储类型-块存储

rbd

典型设备： 磁盘阵列，硬盘

主要是将裸磁盘空间映射给主机使用的。

优点：

通过 Raid 与 LVM 等手段，对数据提供了保护。

多块廉价的硬盘组合起来，提高容量。

多块磁盘组合出来的逻辑盘，提升读写效率。

缺点：

采用 SAN 架构组网时，光纤交换机，造价成本高。

主机之间无法共享数据。

使用场景：

docker 容器、虚拟机磁盘存储分配。

日志存储。

文件存储。

…

1.6 三种存储类型-文件存储

fs

典型设备： FTP、NFS 服务器

为了克服块存储文件无法共享的问题，所以有了文件存储。

在服务器上架设 FTP 与 NFS 服务，就是文件存储。

优点：

造价低，随便一台机器就可以了。

方便文件共享。

缺点：

读写速率低。

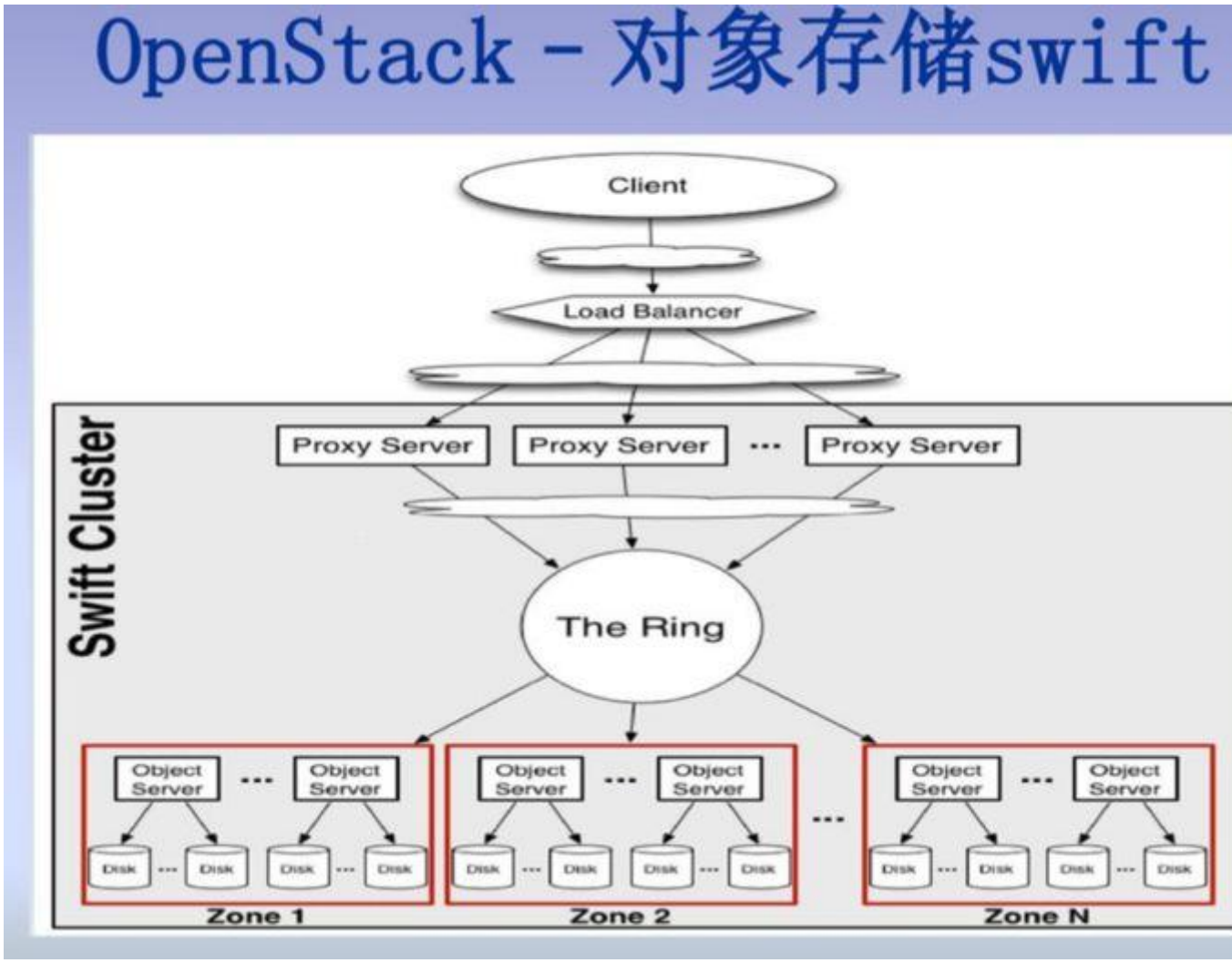
传输速率慢。

使用场景：

日志存储。

有目录结构的文件存储。

1.7 三种存储类型-对象存储



典型设备： 内置大容量硬盘的分布式服务器(swift, s3)

多台服务器内置大容量硬盘,安装上对象存储管理软件,对外提供读写访问功能。

优点:

具备块存储的读写高速。

具备文件存储的共享等特性。

使用场景： (适合更新变动较少的数据)

图片存储。

视频存储。

…

Ceph 核心组件及概念介绍

Monitor
一个 Ceph 集群需要多个 Monitor 组成的小集群，它们通过 Paxos 同步数据，用来保存 OSD 的元数据。

OSD
OSD 全称 Object Storage Device，也就是负责响应客户端请求返回具体数据的进程。一个 Ceph 集群一般都有很多个 OSD。

MDS
MDS 全称 Ceph Metadata Server，是 CephFS 服务依赖的元数据服务。

Object
Ceph 最底层的存储单元是 Object 对象，每个 Object 包含元数据和原始数据。

PG
PG 全称 Placement Grouops，是一个逻辑的概念，一个 PG 包含多个 OSD。引入 PG 这一层其实是为了更好的分配数据和定位数据。

RADOS
RADOS 全称 Reliable Autonomic Distributed Object Store，是 Ceph 集群的精华，用户实现数据分配、Failover 等集群操作。

Libradio
Librados 是 Rados 提供库，因为 RADOS 是协议很难直接访问，因此上层的 RBD、RGW 和 CephFS 都是通过 librados 访问的，目前提供 PHP、Ruby、Java、Python、C 和 C++支持。

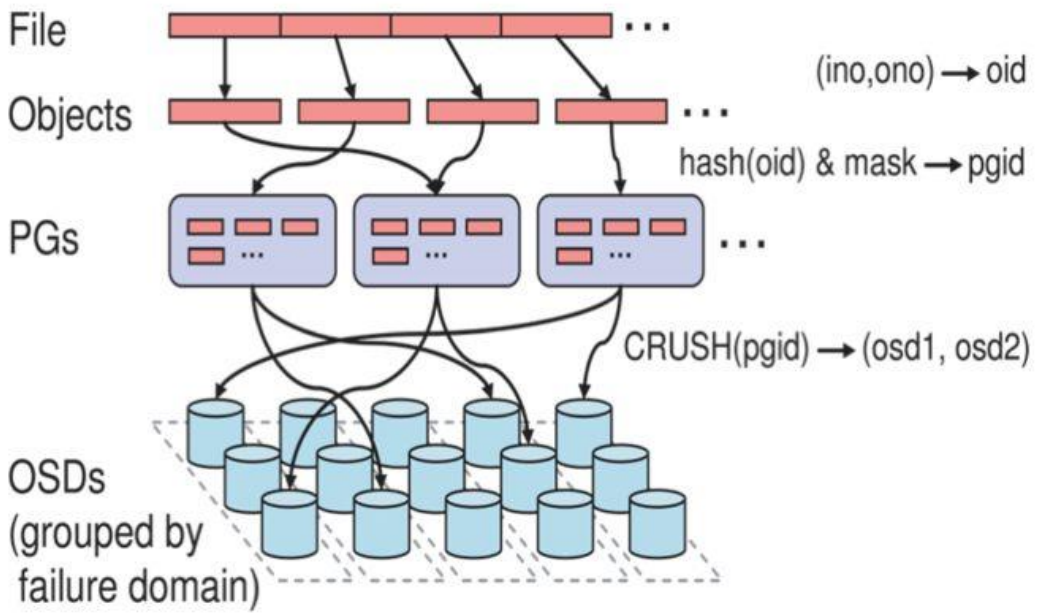
CRUSH
CRUSH 是 Ceph 使用的数据分布算法，类似一致性哈希，让数据分配到预期的地方。

RBD
RBD 全称 RADOS block device，是 Ceph 对外提供的块设备服务。

RGW
RGW 全称 RADOS gateway，是 Ceph 对外提供的对象存储服务，接口与 S3 和 Swift 兼容。

CephFS
CephFS 全称 Ceph File System，是 Ceph 对外提供的文件系统服务。

Ceph IO 算法流程



1. File 用户需要读写的文件。File->Object 映射：

a. ino (File 的元数据，File 的唯一 id)。
b. ono(File 切分产生的某个 object 的序号，默认以 4M 切分一个块大小)。
c. oid(object id: ino + ono)。

2. Object 是 RADOS 需要的对象。Ceph 指定一个静态 hash 函数计算 oid 的值，将 oid 映射成一个近似均匀分布的伪随机值，然后和 mask 按位相与，得到 pgid。Object->PG 映射：

a. hash(oid) & mask-> pgid 。
b. mask = PG 总数 m(m 为 2 的整数幂)-1 。

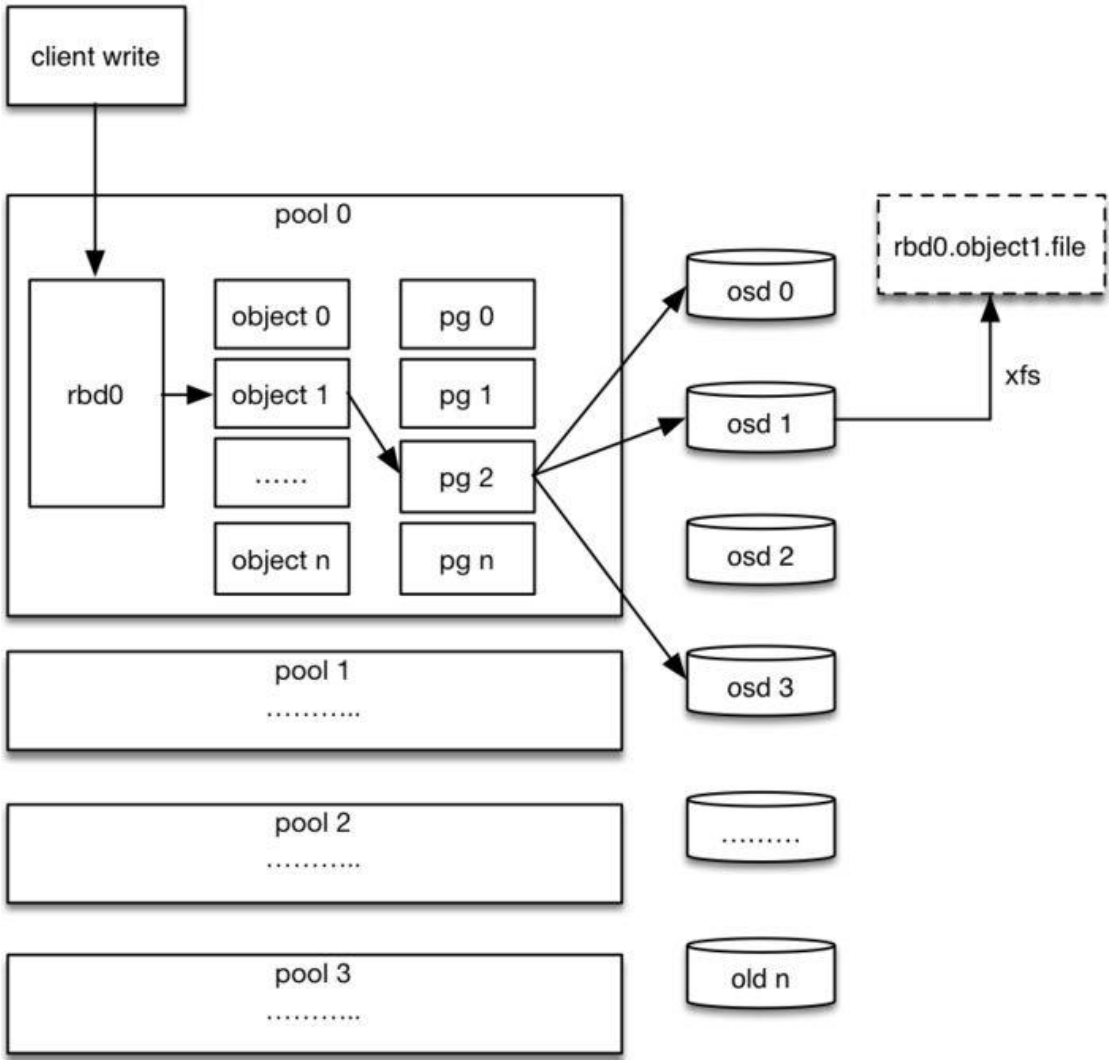
3. PG(Placement Group),用途是对 object 的存储进行组织和位置映射，(类似于 redis cluster 里面的 slot 的概念) 一个 PG 里面会有很多 object。采用 CRUSH 算法，将 pgid 代入其中，然后得到一组 OSD。PG->OSD 映射：

a. CRUSH(pgid)->(osd1,osd2,osd3) 。

伪代码流程

```
locator = object_name
obj_hash =  hash(locator)
pg = obj_hash % num_pg
osds_for_pg = crush(pg)    # returns a list of osdsprimary = osds_for_pg[0]
replicas = osds_for_pg[1:]
```

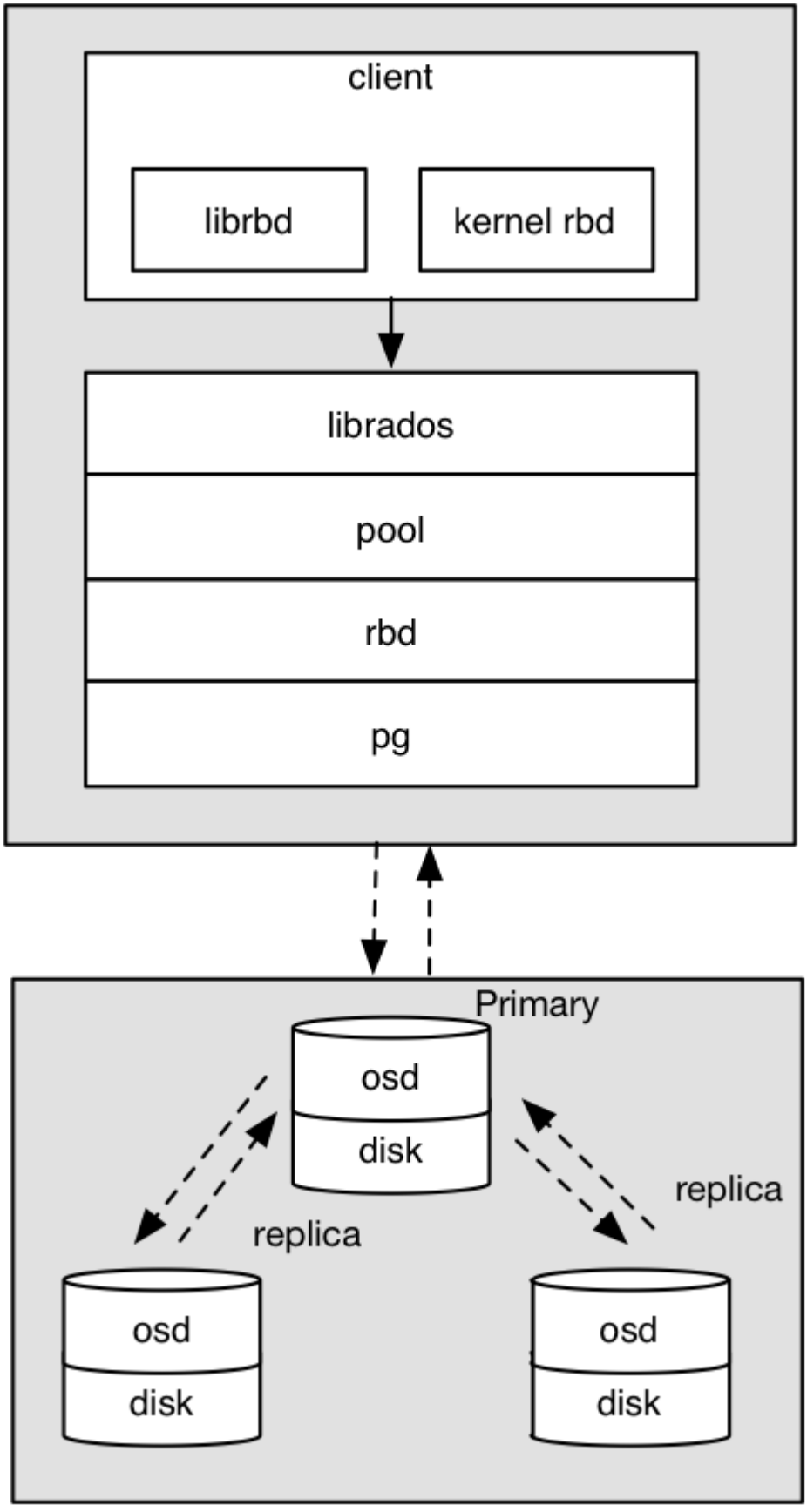
Ceph RBD IO 流程



步骤：

1. 客户端创建一个 pool，需要为这个 pool 指定 pg 的数量。
2. 创建 pool/image rbd 设备进行挂载。
3. 用户写入的数据进行切块，每个块的大小默认为 4M，并且每个块都有一个名字，名字就是 object+序号。
4. 将每个 object 通过 pg 进行副本位置的分配。
5. pg 根据 cursh 算法会寻找 3 个 osd，把这个 object 分别保存在这三个 osd 上。
6. osd 上实际是把底层的 disk 进行了格式化操作，一般部署工具会将它格式化为 xfs 文件系统。
7. object 的存储就变成了存储一个文 rbd0.object1.file。

Ceph RBD IO 框架图



客户端写数据 osd 过程：

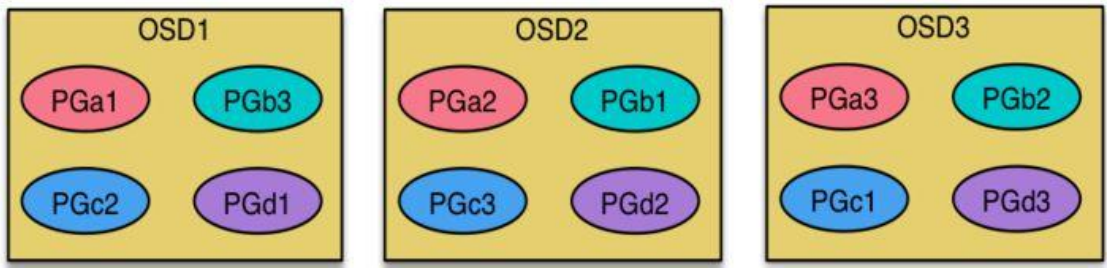
1. 采用的是 librbd 的形式，使用 librbd 创建一个块设备，向这个块设备中写入数据。

2. 在客户端本地同过调用 librados 接口，然后经过 pool，rbd，object、pg 进行层层映射,在 PG 这一层中，可以知道数据保存在哪 3 个 OSD 上，这 3 个 OSD 分为主从的关系。

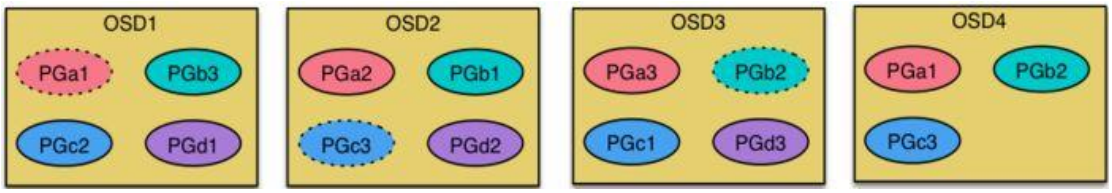
3. 客户端与 primay OSD 建立 SOCKET 通信，将要写入的数据传给 primary OSD，由 primary OSD 再将数据发送给其他 replica OSD 数据节点。

Ceph Pool 和 PG 分布情况

说明：
pool 是 ceph 存储数据时的逻辑分区，它起到 namespace 的作用。
每个 pool 包含一定数量(可配置)的 PG。
PG 里的对象被映射到不同的 Object 上。
pool 是分布到整个集群的。
pool 可以做故障隔离域，根据不同的用户场景不一进行隔离。
2.8 Ceph 数据扩容 PG 分布
场景数据迁移流程：
现状 3 个 OSD, 4 个 PG
扩容到 4 个 OSD, 4 个 PG
现状：



扩容后：



说明

每个 OSD 上分布很多 PG，并且每个 PG 会自动散落在不同的 OSD 上。如果扩容那么相应的 PG 会进行迁移到新的 OSD 上，保证 PG 数量的均衡。

CEPH 的使用：、

预备条件：

Debian 或 RPM 软件包依赖项列表可以安装为：

```
./install-deps.sh
```

搭建 CEPH

构建指令：

```
./do_cmake.sh
cd build
make
```

注意，这些指令是为开发和测试编译代码的开发人员编写的。要构建适合安装的二进制文件，我们建议您构建 DEB 或 RPM 包，或者参考 CEPH.spec.in 或 Debian/规则，查看为生产版本指定了哪些配置选项。 前提条件：CMAKE 3.5.1 构建说明：（注意：doucmake.sh 现在默认为创建 CEPH 的调试构建，在某些工作负载下，它的速度可以降低 5 倍。请通过“-dcmake_build_type=relwithdebinfo”执行“u cmake.sh”以创建非调试版本。）这假定您将 build dir 设置为 ceph.git 签出的子目录。如果你把它放在别的地方，只要换掉……在 do-cmake.sh 中，具有正确的签出路径。在调用 do-cmake 之前，可以指定任何其他 cmake 参数来设置参数。有关详细信息，请参阅 cmake 选项。

如 `ARGS="-DCMAKE_C_COMPILER=gcc-7" ./do_cmake.sh`

要仅生成特定目标，请使用：

```
make [target name]
```

下载：

```
make install
```

CMake 选项

如果手动运行 cmake 命令，可以使用“-d”设置许多选项。例如，构建 Rados 网关的选项默认为打开。在没有雷达罩网关的情况下建造：

```
cmake -DWITH_RADOSGW=OFF [path to top level ceph directory]
```

下面的另一个示例是使用调试和几个外部依赖项的备用位置进行构建：

```
cmake -DLEVELDB_PREFIX="/opt/hyperleveldb" -DOFED_PREFIX="/opt/ofed" \
-DCMAKE_INSTALL_PREFIX=/opt/accelio -DCMAKE_C_FLAGS="-O0 -g3 -gdwarf-4" \
..
```

要查看-d 选项的详细列表，可以使用以下命令调用 cmake：

```
cmake -LH
```

如果您经常将 make 转换为 less，并且希望维护错误和警告的诊断颜色（如果编译器支持该颜色），则可以使用以下命令调用 cmake：

```
cmake -DDIAGNOSTICS_COLOR=always ..
```

然后在执行以下操作时，您将获得诊断颜色：

```
make | less -R
```

“DIAGNOSTICS_COLOR”的其他可用值为“auto”（默认）和“never”。

构建一个源 tarball

要构建一个完整的源 tarball 以及从源代码构建和/或构建一个（deb 或 rpm）包所需的一切，请运行

```
cd build
make vstart        # builds just enough to run vstart
../src/vstart.sh --debug --new -x --localhost --bluestore
./bin/ceph -s
```

几乎所有常用命令都在 bin/目录中可用。例如，

```
./bin/rados -p rbd bench 30 write
./bin/rbd create foo --size 1000
```

运行测试单元

要构建和运行所有测试（并行使用所有处理器），请使用 ctest:

```
cd build
make
ctest -j$(nproc)
```

在 CEPH 中构建和运行所有测试及其依赖项，而不需要其他不必要的目标：

```
cd build
make check -j$(nproc)
```

要手动运行单个测试，请使用-r 运行 ctest（regex 匹配）：

```
ctest -R [regex matching test name(s)]
```

建立文档

先决条件

构建文档的包依赖项列表可以在 doc_deps.deb.txt 中找到：

```
sudo apt-get install `cat doc_deps.deb.txt`
```

建立文档

要构建文档，请确保您位于顶层/ceph 目录中，然后执行构建脚本。例如：

```
admin/build-doc
```