登录 (http://dockone.io/login/) 注册 (http://dockone.io/account/register/)

微信分享 (http://dockone.io/topic/%E5%BE%AE%E4%BF%A1%E5%88%86%E4%BA%AB)

DockOne微信分享（六十一）：虚拟化老兵介绍虚拟化技术

【编者的话】本次分享从以下4方面展开：

1. 虚拟化主流技术介绍
2. 虚拟化前沿技术介绍
3. Docker技术介绍
4. MixSAN技术介绍

大家好，我是徐安，一位虚拟化老兵。2010年开始在世纪互联（云快线）接触云计算和虚拟化技术，应该算是国内较早的一批人吧。目前在汉柏科技有限公司，负责服务器虚拟化以及桌面虚拟化产品的技术工作。

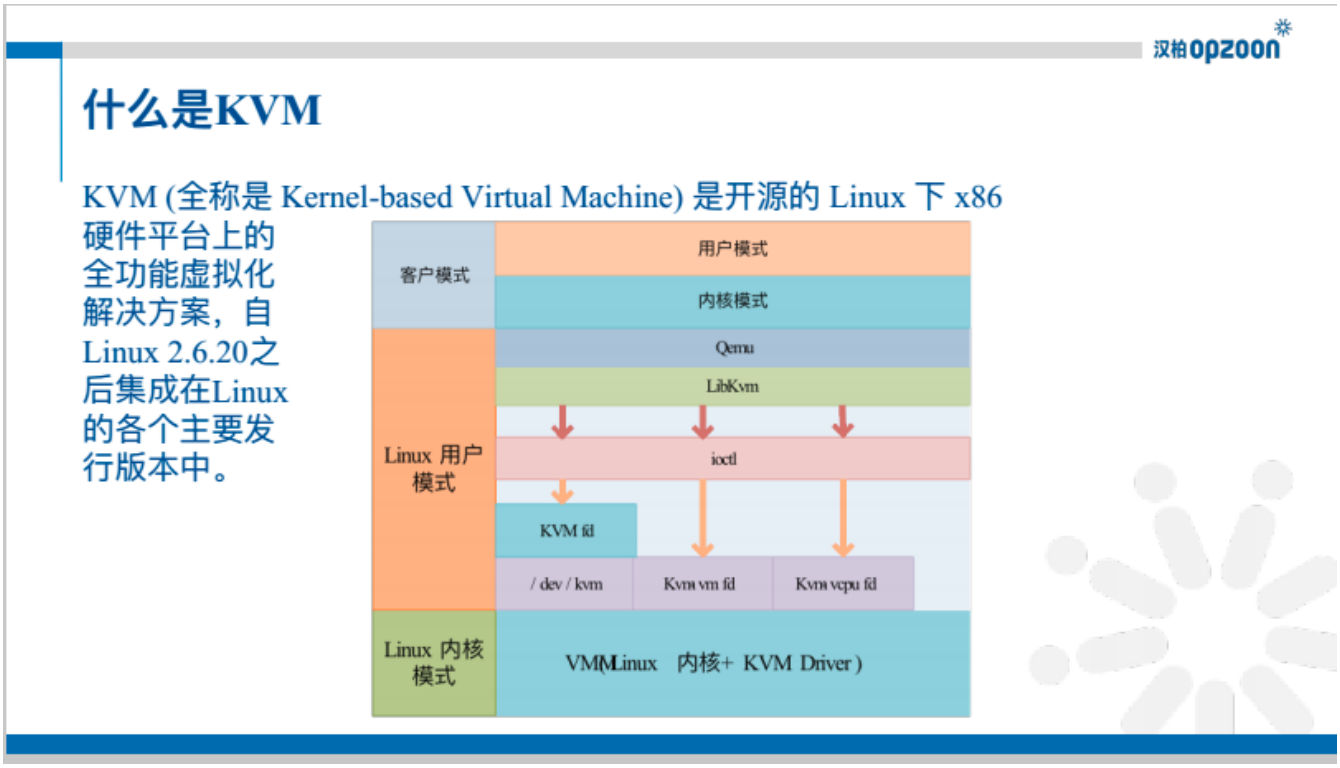
(http://dockerone.com/uploads/article/20160530/d4bae248ea71f27be51db31deb84bc08.PNG)

(http://dockerone.com/uploads/article/20160530/62da15a77f45dd7bb0c59293c26761a3.PNG)

我将从虚拟化的主流技术介绍，前沿技术介绍，Docker技术介绍，MixSAN技术介绍四个方面展开今天的分享。由于笔者水平和知识所限，难免有理解不正确的地方，请各位大牛批评指正。

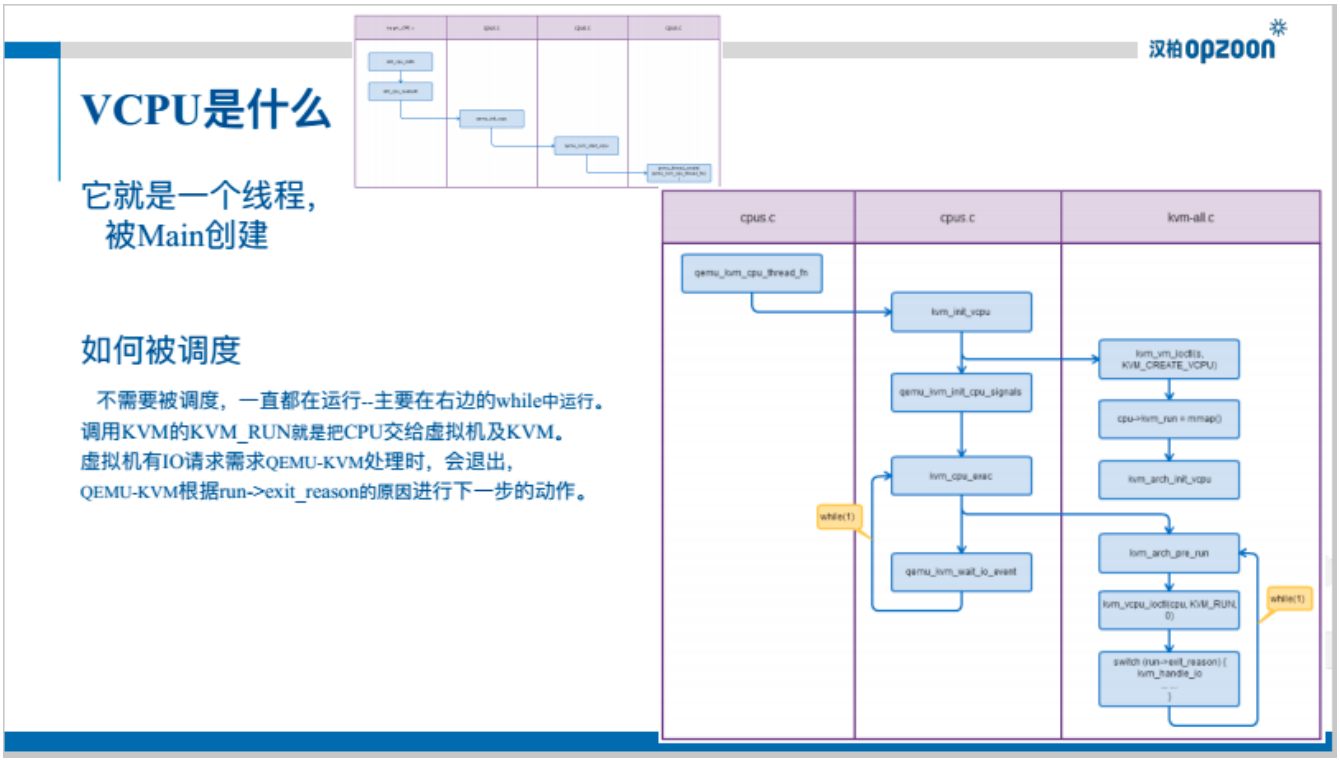
(http://dockerone.com/uploads/article/20160530/f13b0a6fb0d8428ece8889fae92f4df0.PNG)

首先让我们看看主流虚拟化技术有哪些，无非就是CPU虚拟化，内存虚拟化，网卡虚拟化，磁盘虚拟化。



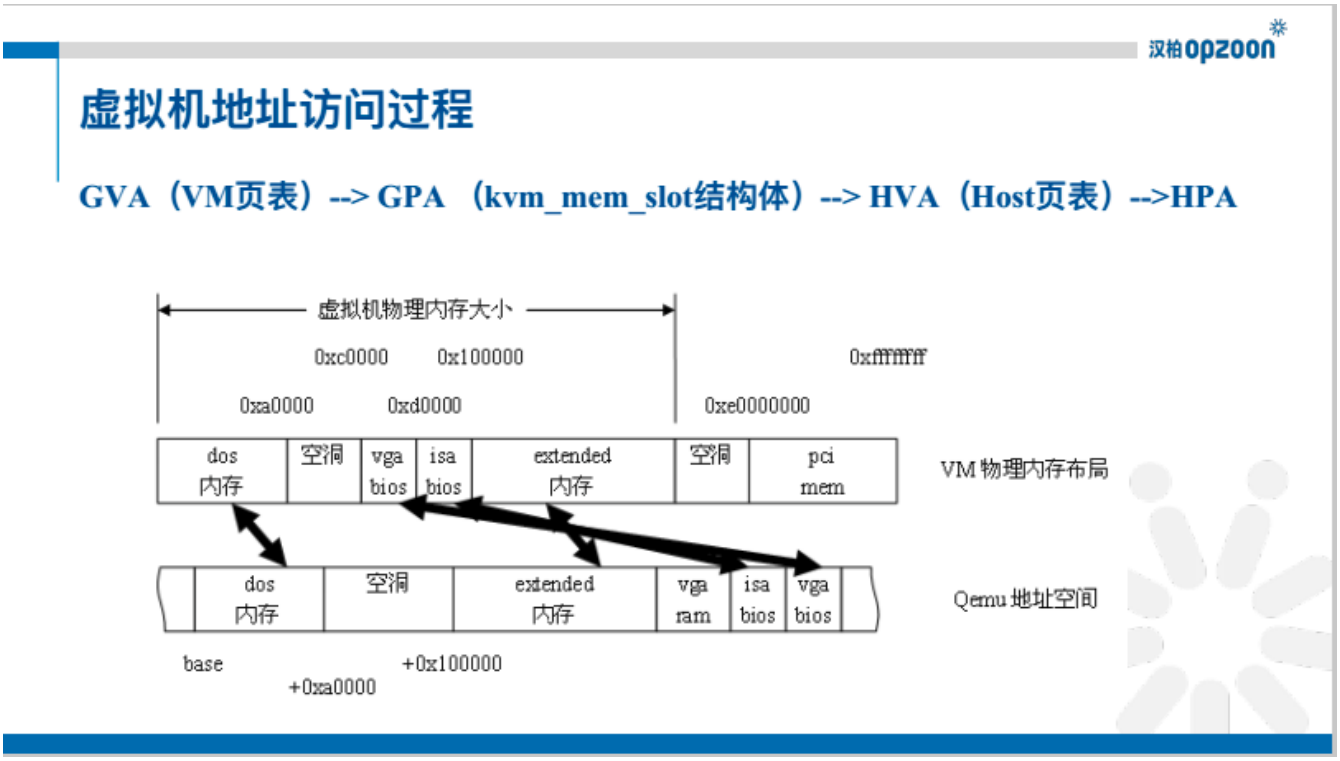
(http://dockerone.com/uploads/article/20160530/2d5929245f27cf0d5aaf2287edb6445d.PNG)

KVM是目前最主流的虚拟化技术，自Linux 2.6.20之后集成在各主要Linux发行版本中。KVM分为四个模式，分别是客户（虚拟机）用户模式，客户（虚拟机）内核模式，Host Linux用户模式，Host Linux内核模式。虚拟机的用户模式和内核模式与虚拟化之前的操作系统对应，没有什么好解释的。Qemu-kvm是一个Host Linux用户态程序，就是一个进程，代表着一个虚拟机。Qemu-kvm主要负责为虚拟机模拟硬件，Qemu-kvm通过ioctl控制KVM内核模块。



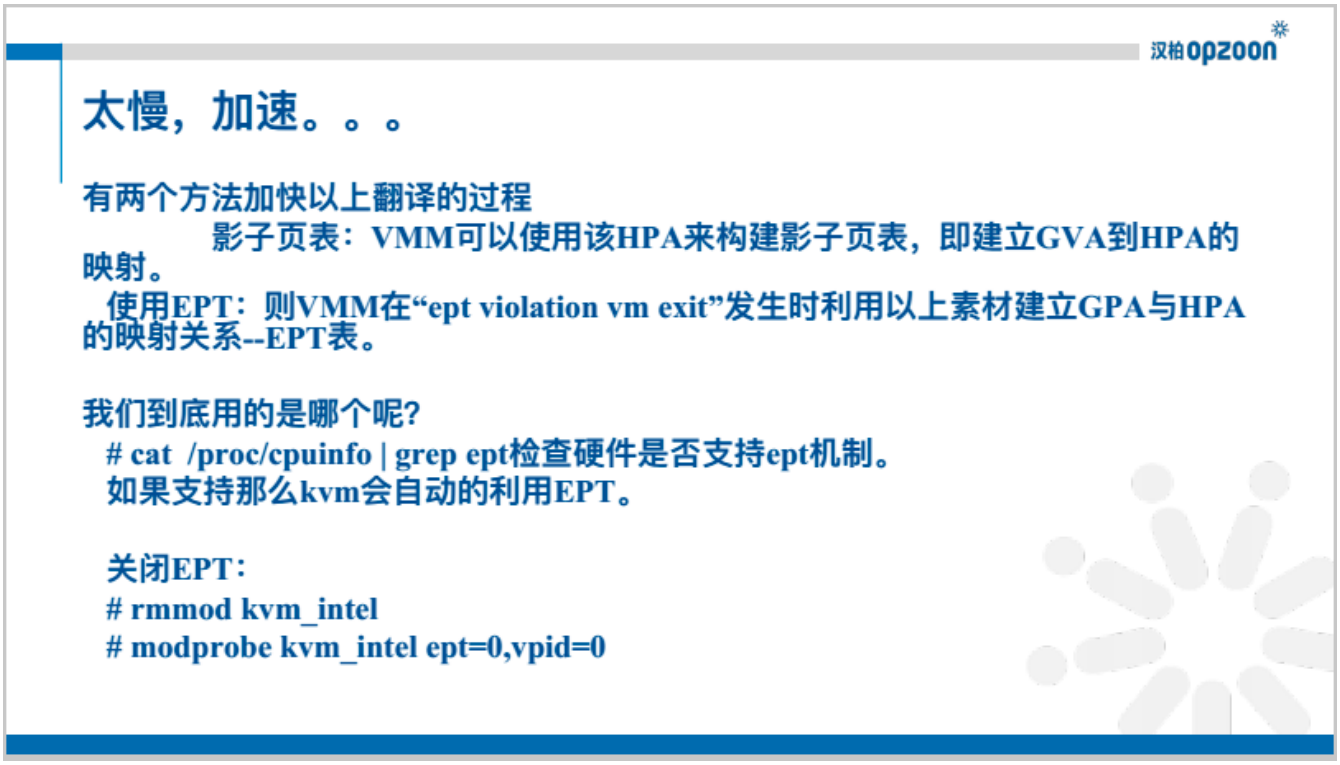
(http://dockerone.com/uploads/article/20160530/b3593c1b1794a9e47202ec4da6db1d1f.PNG)

一个虚拟机就是一个进程，那一个vCPU就是一个线程，它被Main线程创建。这么设计的好处是无需单独设计vCPU的调度算法了，就用Linux的线程调度即可。所以，对于一个vCPU来说，它就是拼了老命runing，看图可以见到两个while循环，第二个while就是让KVM内核运行该vCPU的上下文，一旦KVM内核运行不下去了，就看看是什么原因，qemu-kvm根据原因执行下一步动作，比如需要读写磁盘了，那就用qemu-kvm去打开该磁盘对应的文件，执行read，write操作。



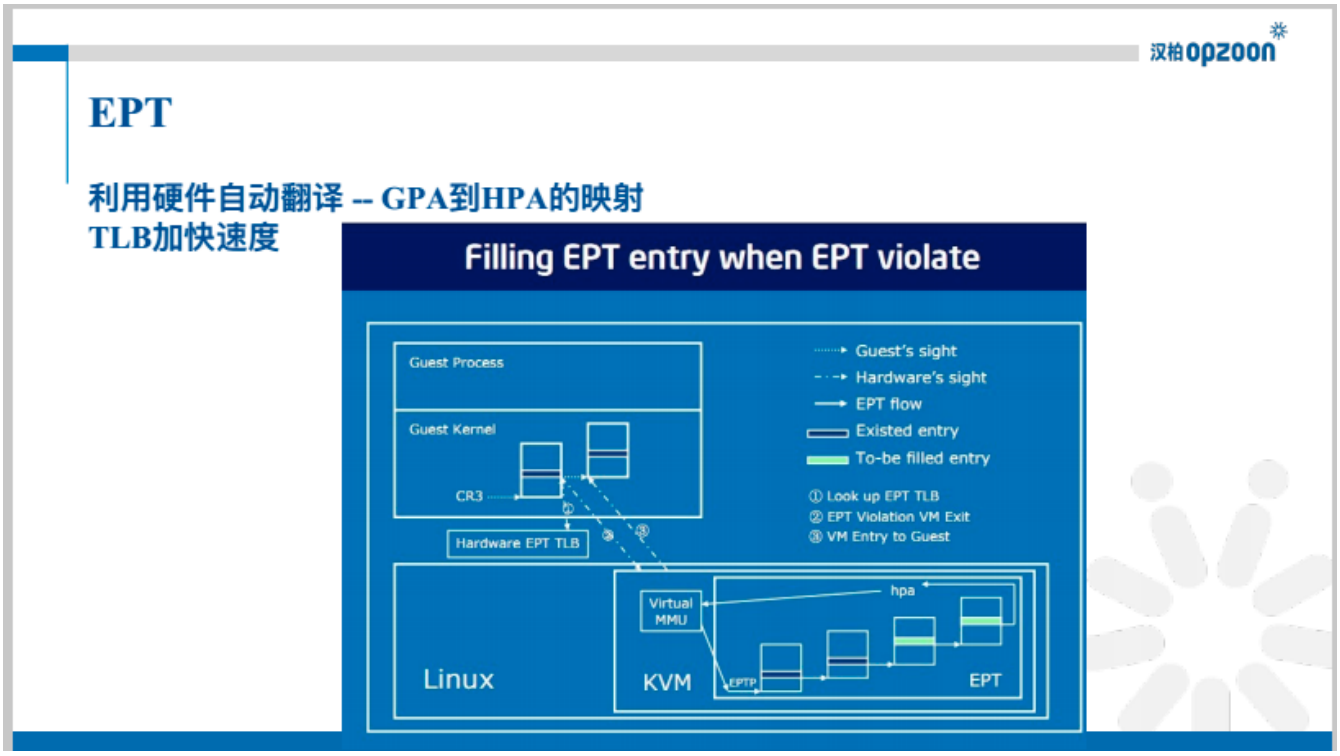
(http://dockerone.com/uploads/article/20160530/6b057691951d81174391b69b14231920.PNG)

整个虚拟机的虚拟地址到实际物理内存地址的访问（转换）过程为：GVA（虚拟机虚拟地址），通过VM页表转换为GPA（虚拟机物理地址），然后通过一个数据结构映射为HVA（物理机虚拟地址），再通过Host页表，转换为HPA（物理机物理地址）。



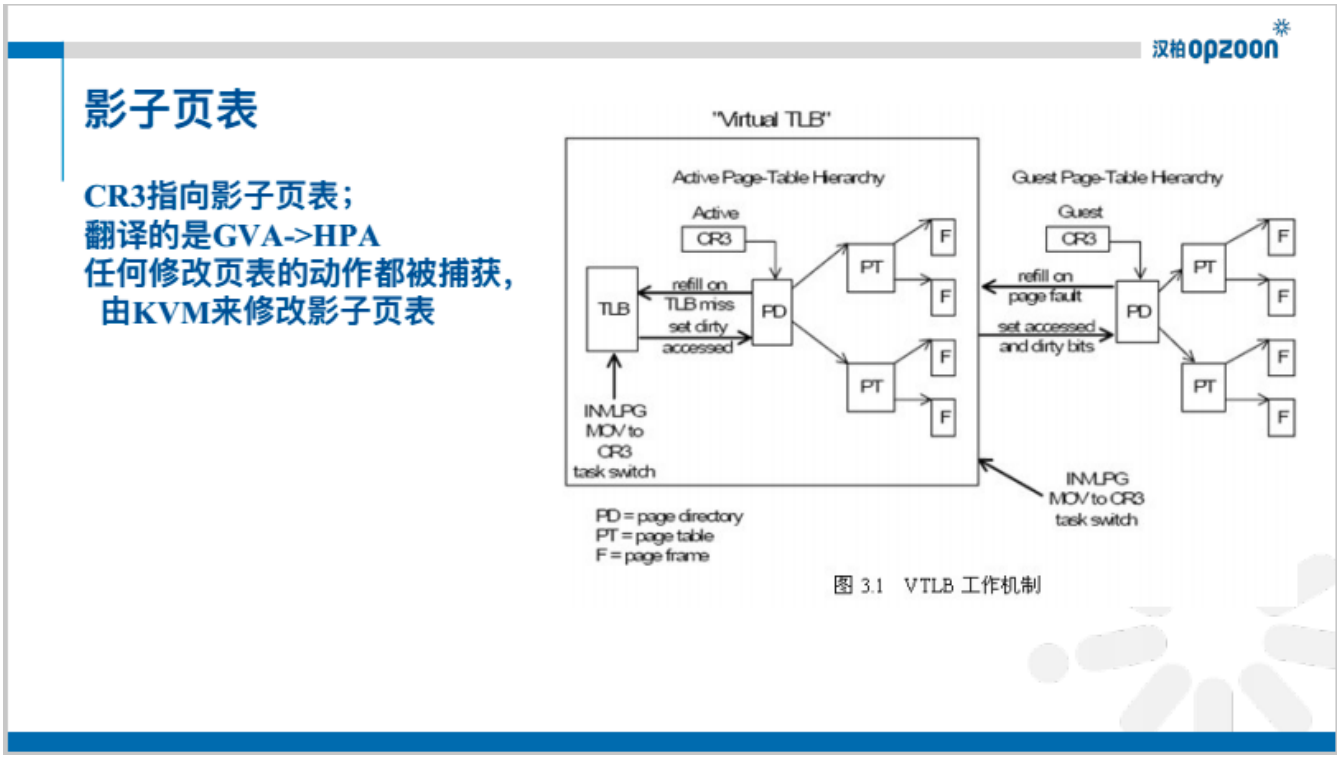
(http://dockerone.com/uploads/article/20160530/0247745bcc30ea4878573fb81021cc53.PNG)

从上页可以看出，这么转换太慢了，需要加速。有两个办法来加速上一页的翻译过程。1）影子页表，2）EPT功能。那KVM使用哪个呢？如CPU支持EPT功能，那就走EPT。



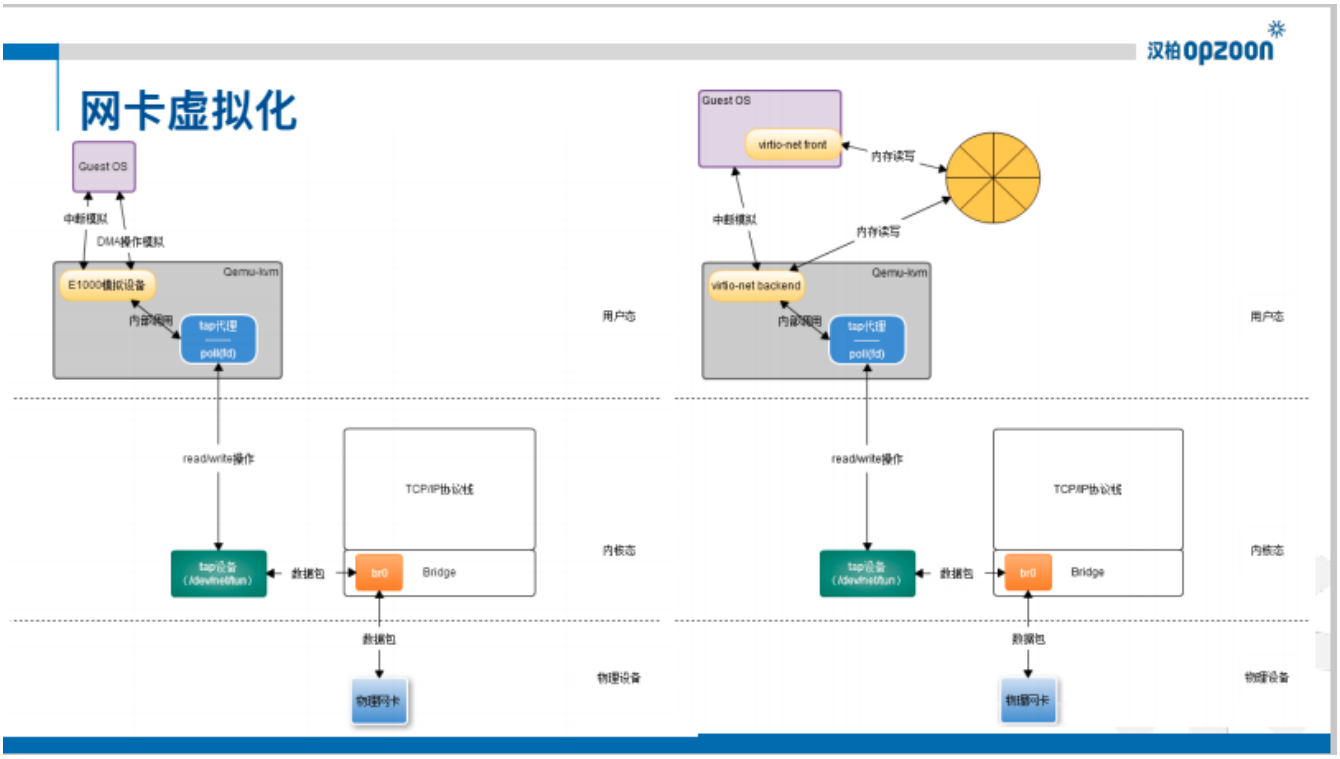
(http://dockerone.com/uploads/article/20160530/f8189b20bb2495661465840c07951d82.PNG)

EPT是利用硬件自动翻译GPA到HPA的地址映射：在构建VM的页表时，EPT硬件把翻译好的HPA地址反馈给VM页表，VM直接使用物理地址。



(http://dockerone.com/uploads/article/20160530/d86e6e881605194e40baa8cb859f540c.PNG)

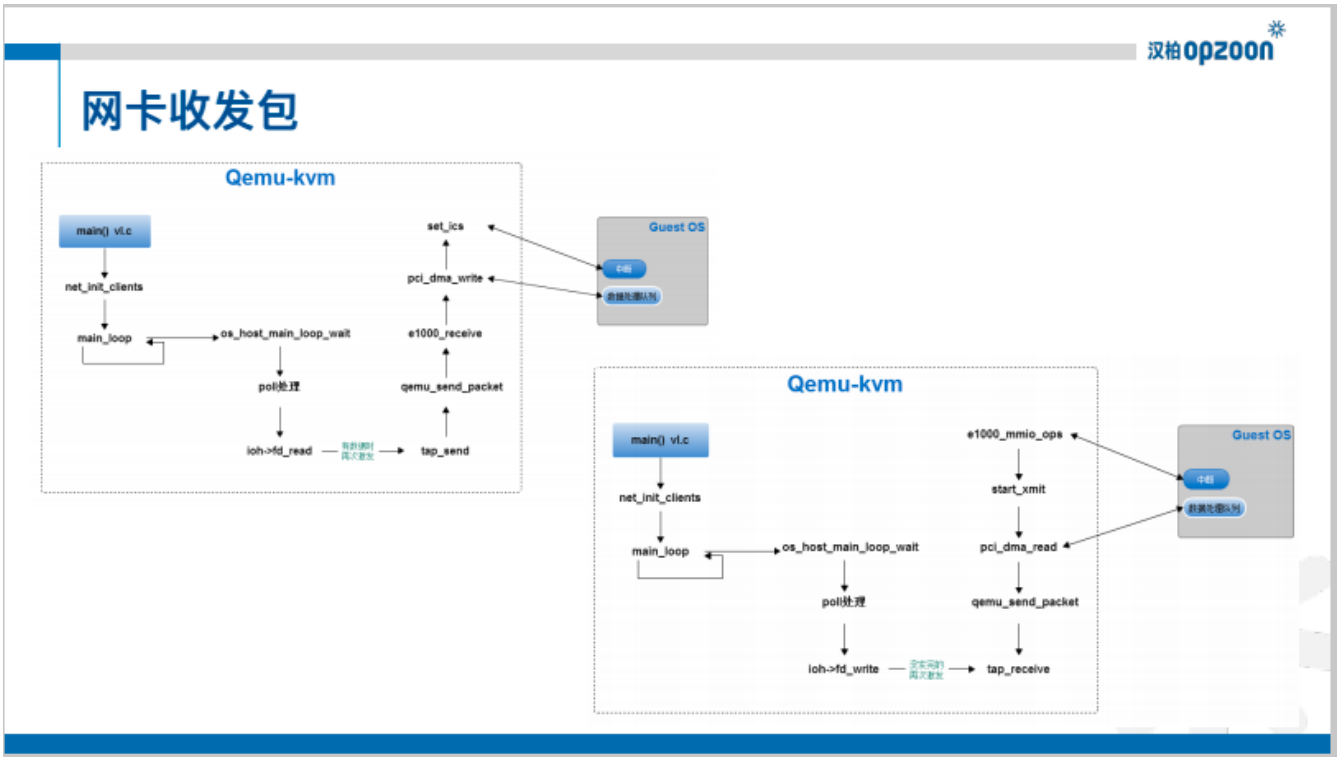
利用软件"欺骗VM"直接构建GVA到HPA的页表，VM还正常走页表翻译的流程，当需要加载页表的物理内存时，KVM接管起来，然后把页表里的内容直接替换成真实的HPA地址即可。



(http://dockerone.com/uploads/article/20160530/d106d7ccc655869610eff3c4aee3d6cc.PNG)

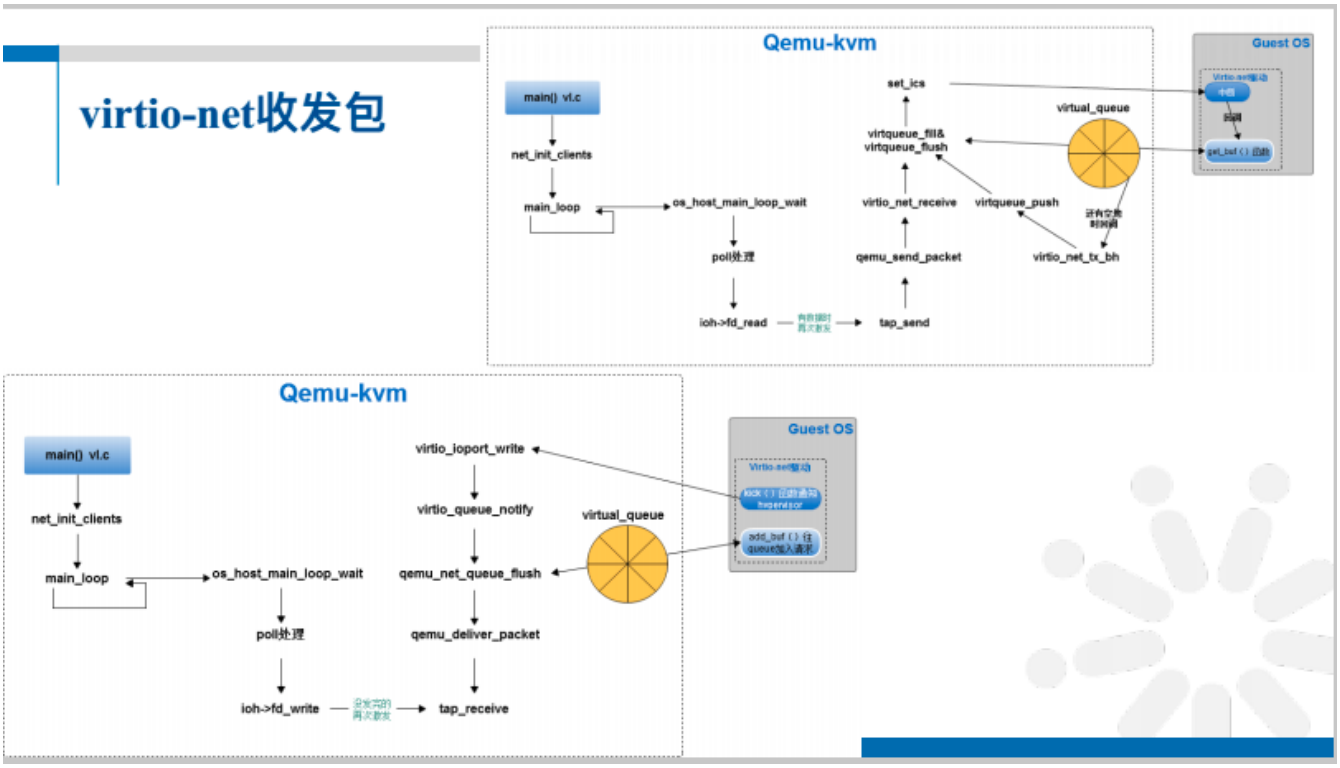
左边是普通网卡的虚拟化，首先qemu-kvm会模拟"E1000"的物理网卡，Guest OS通过模拟的中断和DMA操作与E1000交互，E1000在Qemu-kvm内用内部调用把这些请求转发到"tap代理"上，tap代理实际是open的Host OS的一个tap设备，所以收发包的流程就转换为针对tap设备的read，write操作。tap设备连接在bridge上，再通过TCP/IP协议栈或bridge上的物理网卡把网络包收发起来。

右边是virtio网卡，与普通网卡不同的地方在DMA模拟操作变成了内存共享。



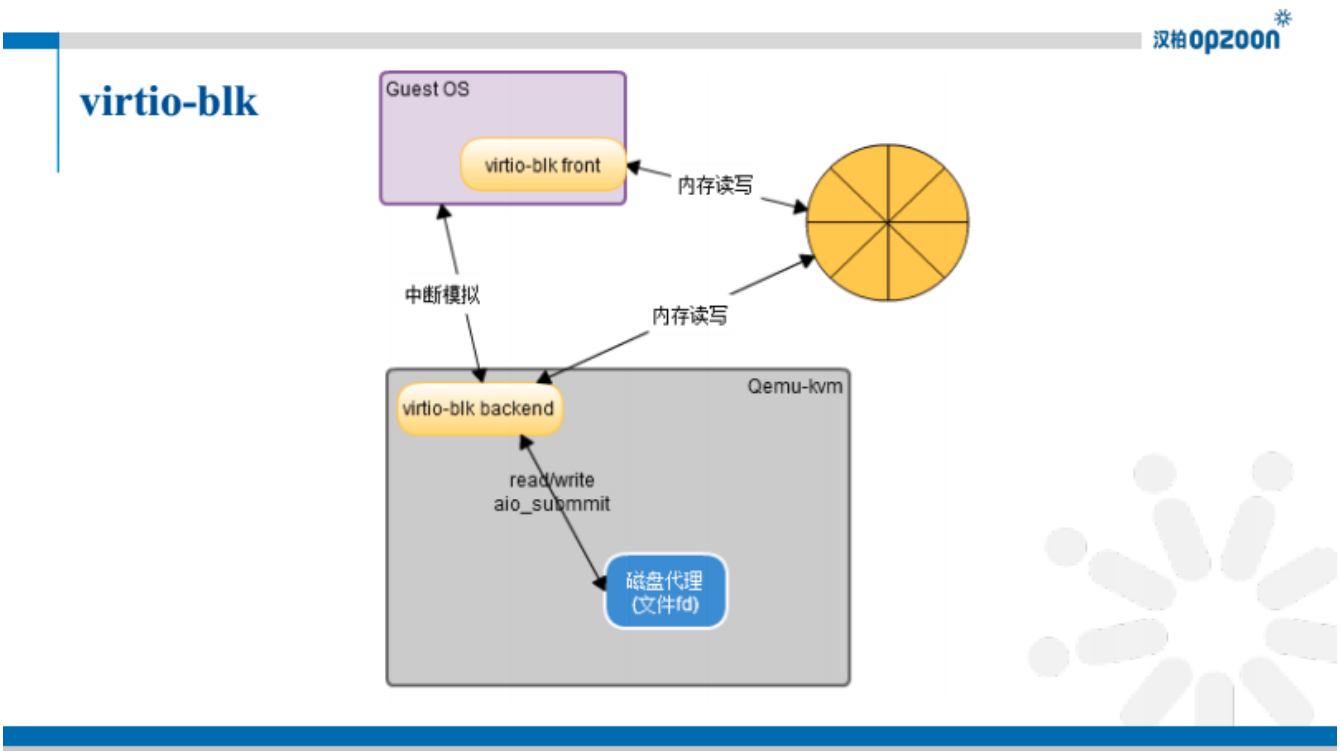
(http://dockerone.com/uploads/article/20160530/093e88e46ae45a75c1d572f1e99aef55.PNG)

这是具体的函数调用过程，强调一下：tap的操作都会经过Qemu-kvm进程的poll处理。左边的是收包过程，右边的是发包过程。



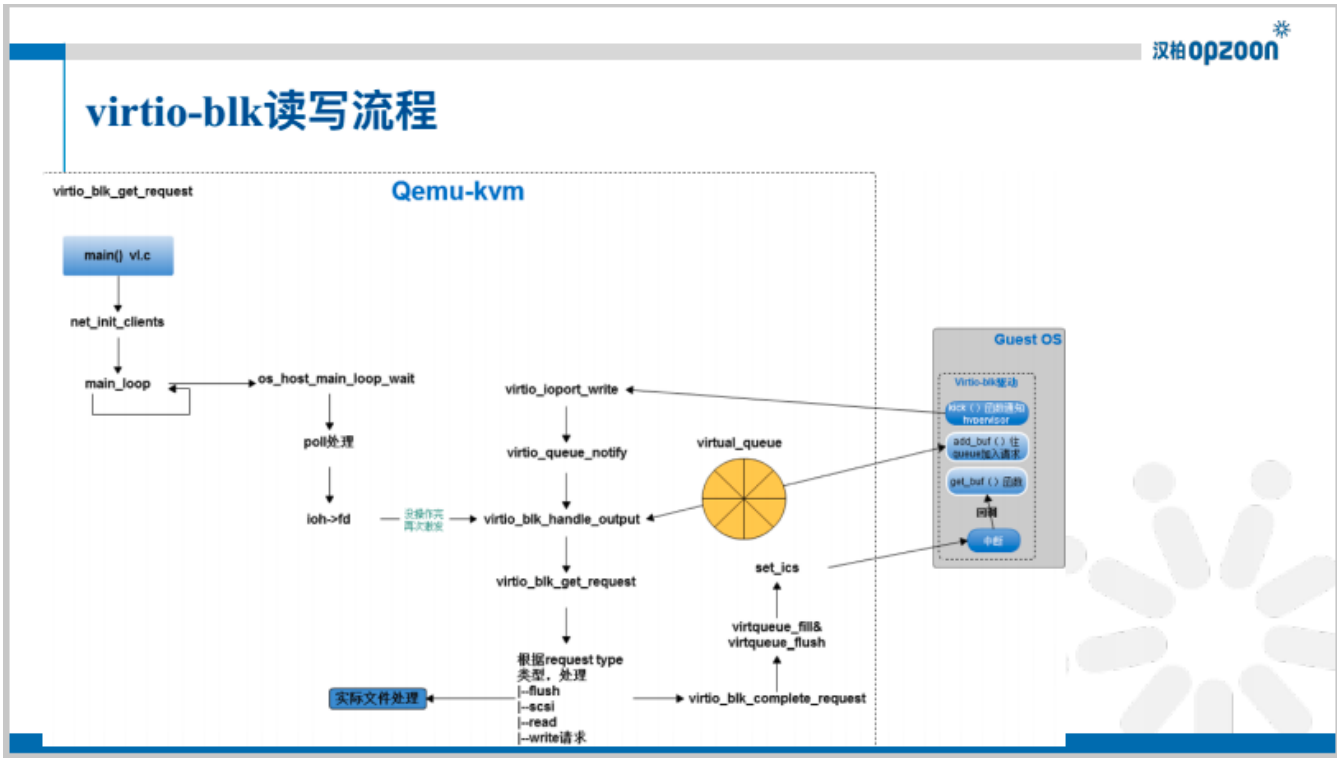
(http://dockerone.com/uploads/article/20160530/d744e636f5d51bfe44ab739cf193a074.PNG)

这是virtio-net收发包的调用流程。



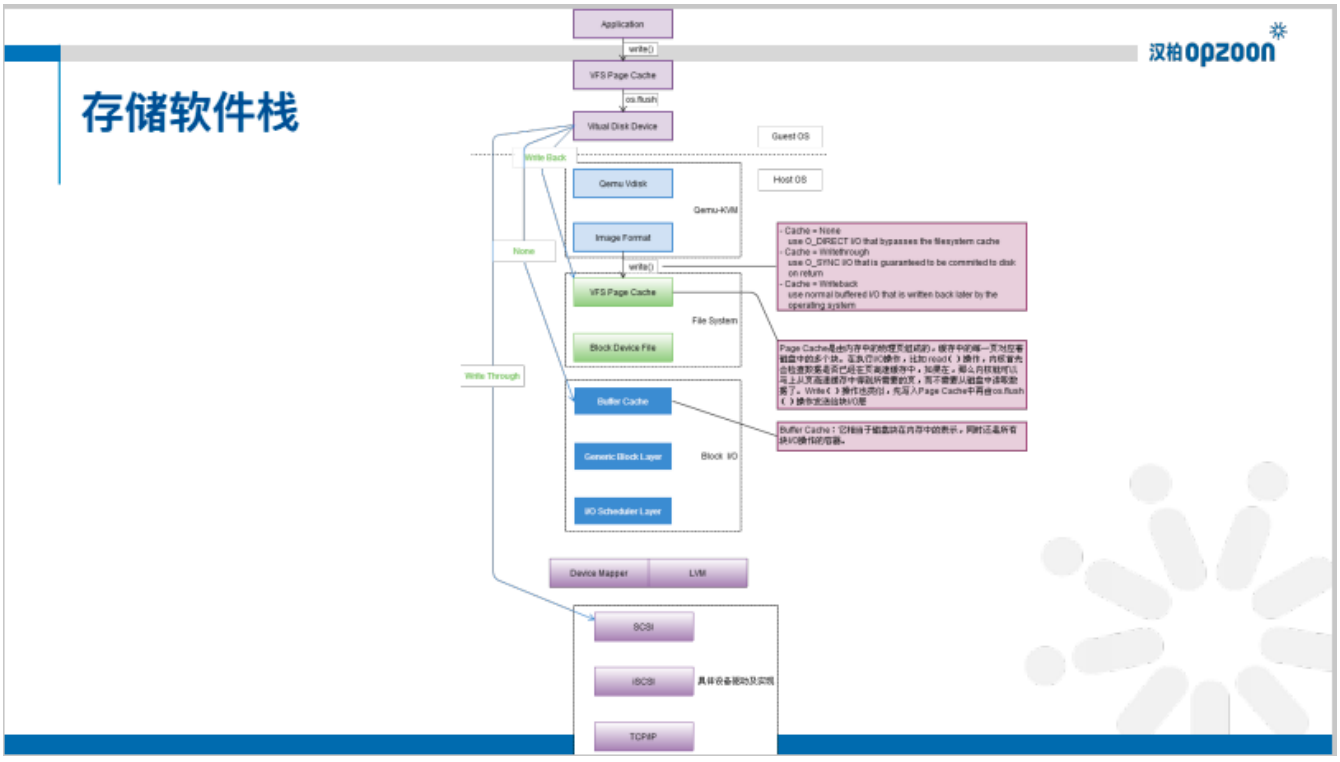
(http://dockerone.com/uploads/article/20160530/14086e05b91a9fe735c25d99850ecbd9.PNG)

vrio-blk与网卡虚拟化类似，首先模拟一个硬件设备，对硬件设备的读写操作都转换为磁盘代理的读写操作。这个磁盘代理可能是一个文件，一个块设备，一个网络设备，或者仅仅是一个模拟。



(http://dockerone.com/uploads/article/20160530/c09434f56de7e15ca952a321950d123a.PNG)

有兴趣的可以私下讨论，时间关系先过。

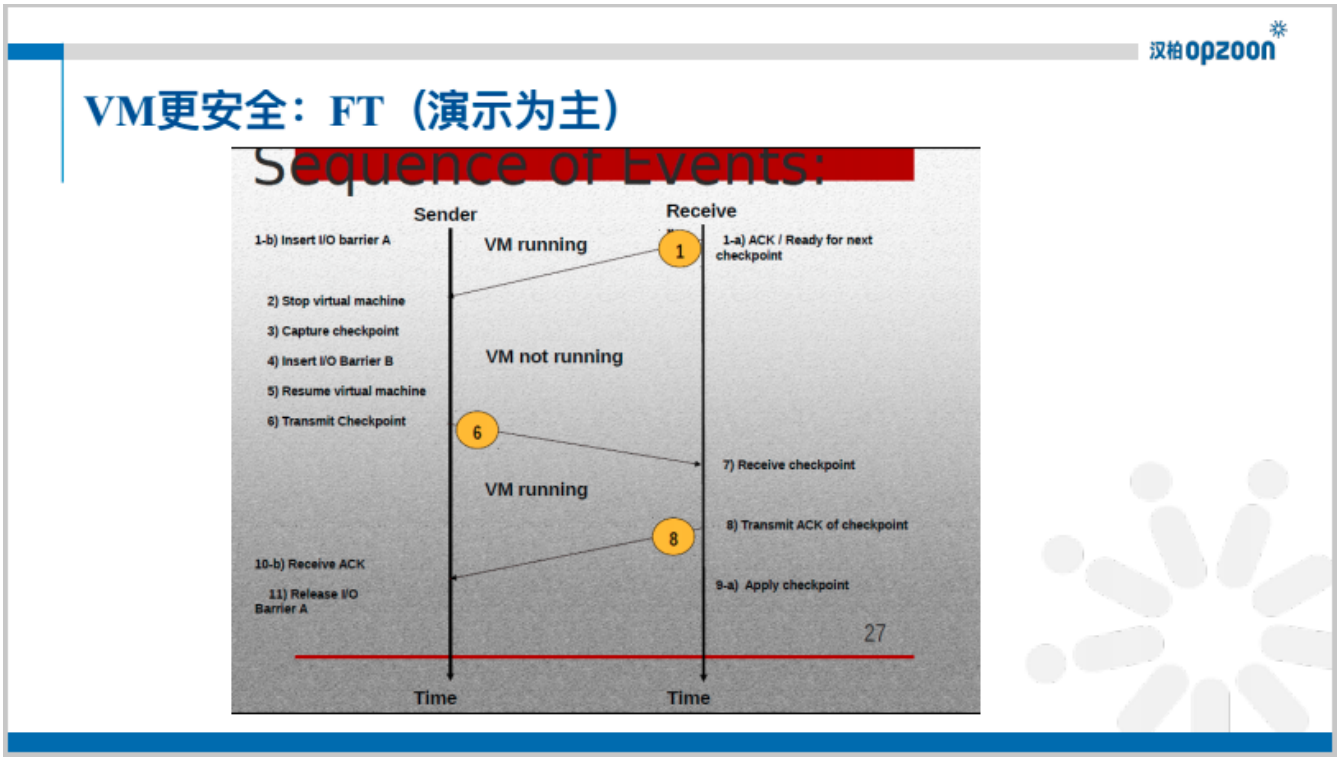


(http://dockerone.com/uploads/article/20160530/0b6777faa2e5385d4f243640a74a963c.PNG)

这个讲起来有点复杂，总之可设定虚拟机某个磁盘的缓冲策略，write-back，none，write-thru。实际上就是qemu-kvm打开文件的类型不同。write-back的意思是数据写入page cache中就不管了，none的话会写入buffer cache（设备缓冲中），write-thru那么设备缓冲也要跨过，直接写入设备中。

(http://dockerone.com/uploads/article/20160530/658b4e3b5db002ee83dc1c4fc5ceb9c7.PNG)

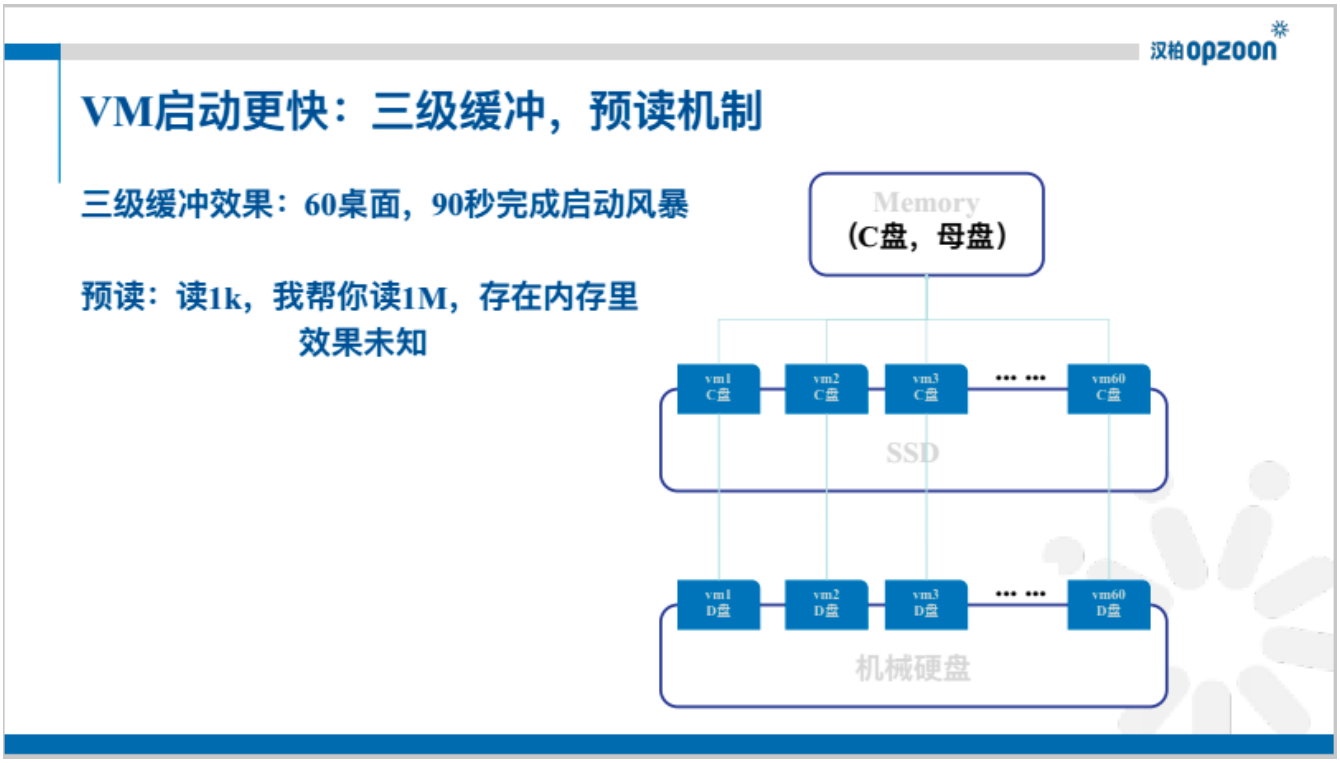
所谓前沿技术，只是个人的认识，不一定正确，请批评指正。



(http://dockerone.com/uploads/article/20160530/1d446cab50ff134a4fb9f750dde1762a.PNG)

所谓的FT技术就是建立两个VM，一主一备。隔一段时间主VM暂停下，做一个checkpoint，然后恢复VM，在下一次checkpoint点之前，把上两次checkpoint之间的差异（CPU，内存）数据传递给备份VM，备份VM同步更新这些差异数据。

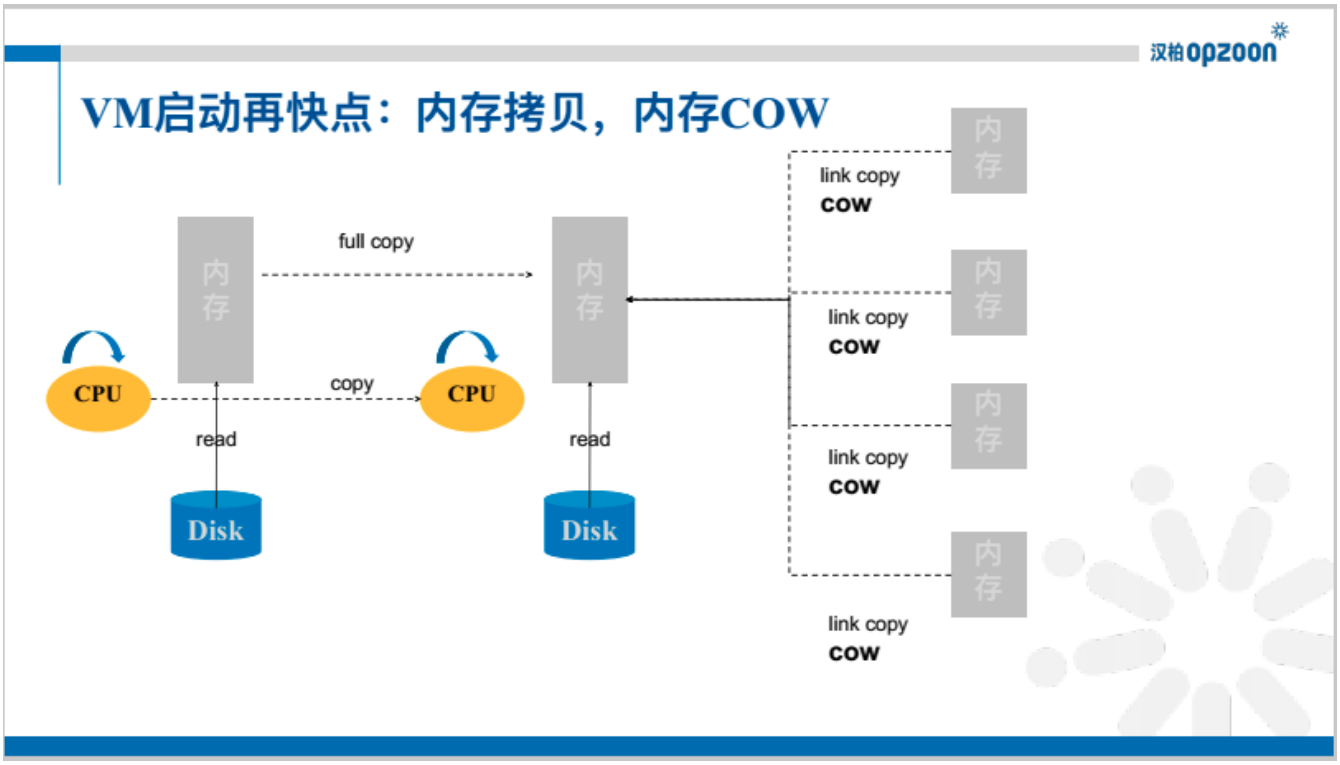
这里的困难点就是checkpoint之间的时间间距尽量短，如果这段时间内存数据变化比较大，那传输的数据流就会巨大，那对主备之间的网络带宽压力就会比较大。



(http://dockerone.com/uploads/article/20160530/9314aae0483ca61b277f19fd8ebb6572.PNG)

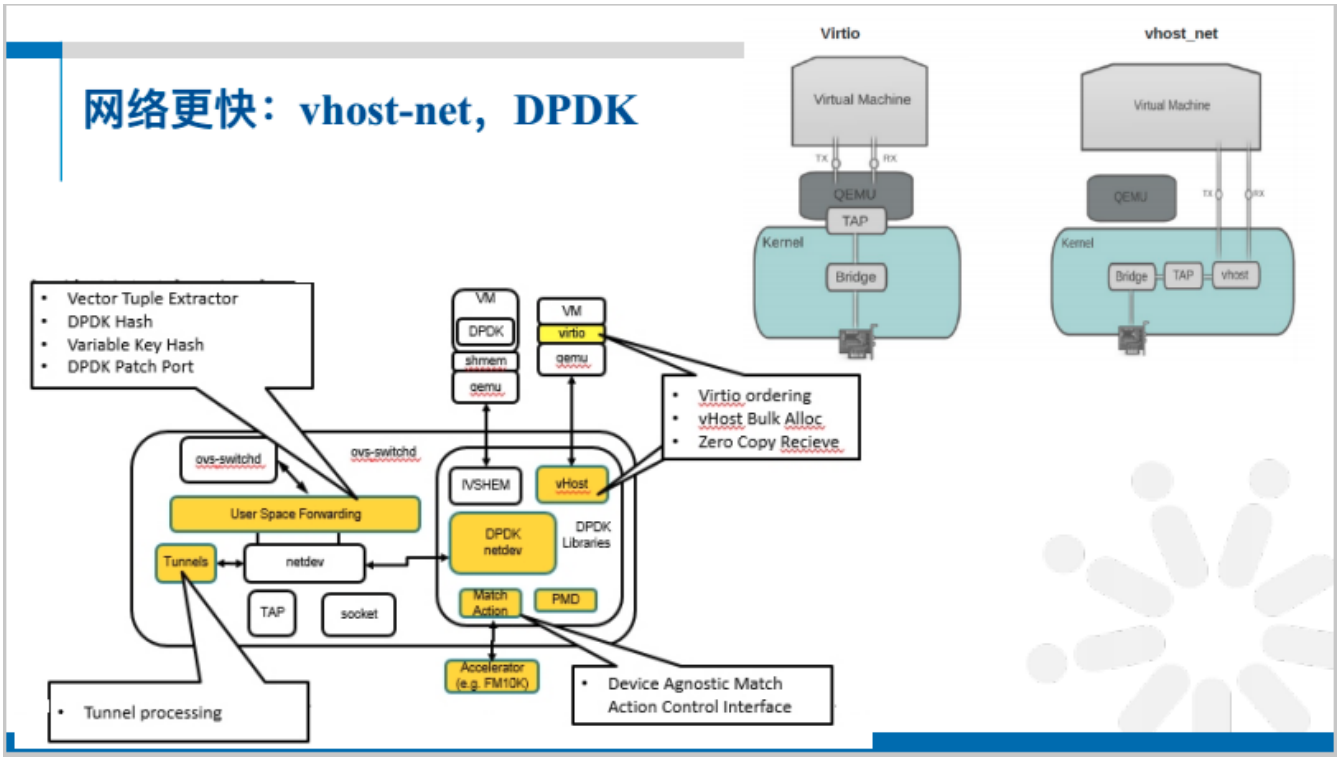
三级缓冲是这样的，以Widows系统举例，把C盘的母盘放在内存中，C盘的link clone出来的子盘放在SSD中，D盘放入机械硬盘中。这样的VM启动速度，我们测试的结果是60个桌面，90秒内可启动完。

预读机制是指，虚拟机让我读取1k的数据，我自动帮你读1M的数据让在内存中。



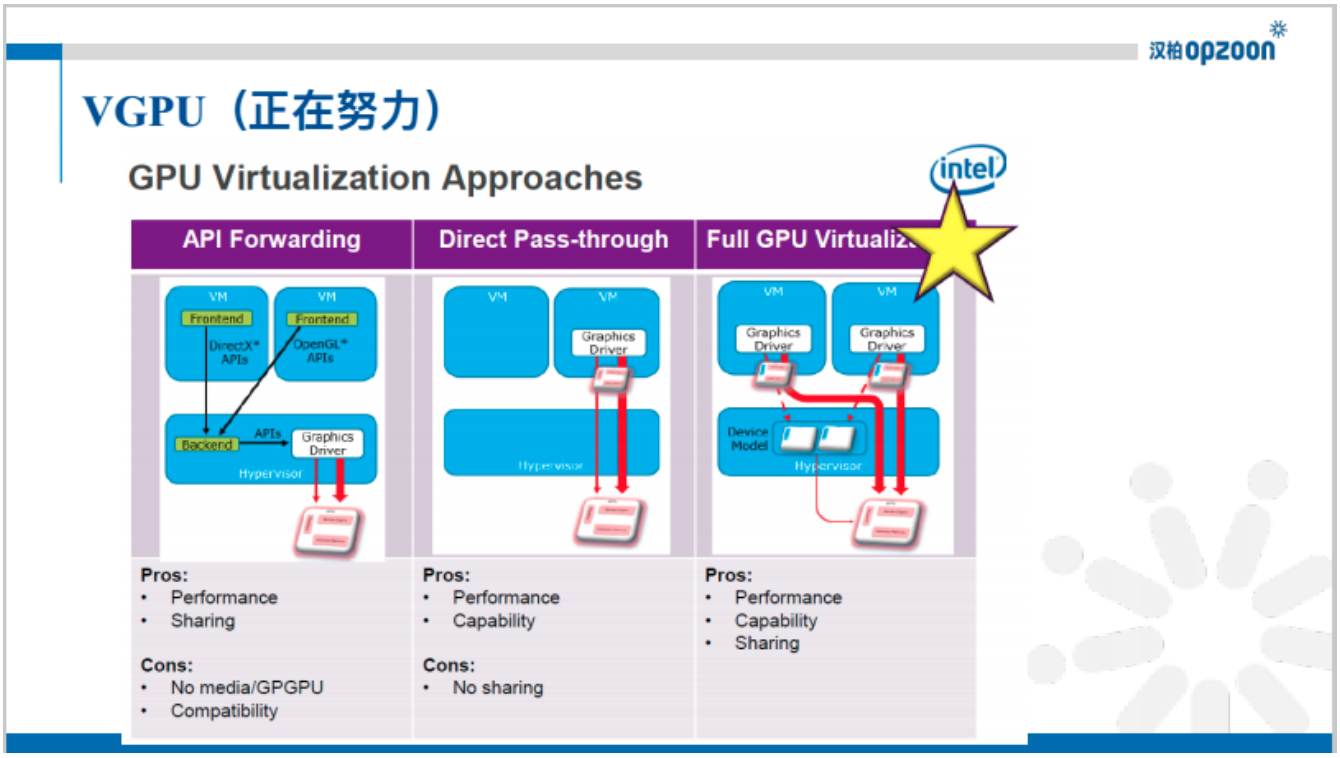
(http://dockerone.com/uploads/article/20160530/07094d65a31ddfe831cf23f946d5747a.PNG)

如果还嫌VM启动太慢了，那还有办法，我们看一个虚拟机启动的过程无非是这样的：CPU转起来，读取Disk里的数据放入内存中。那启动多个一模一样虚拟机的时候能不能不走这个流程呢，直接把内存和CPU的数据从虚拟机A拷贝到虚拟机B，那理论上虚拟机B的启动速度等于内存对拷的时间。还可以再加速，如果虚拟机基本差不多，有些内存数据根本不会变化，那内存拷贝的时候能不能只是做一个link链接，当子内存某一块有变化时，再从母内存中copy on write即可。



(http://dockerone.com/uploads/article/20160530/6418fcd5fbbc6e2fe1ee5bfe505aac6a.PNG)

网卡性能加速第一方向是使用vhost-net，网络数据包转发可抛掉qemu-kvm，直接走到vhost设备，在内核中转发到tap设备，然后走host内核的bridge走出去。说的直白点，在创建VM时，给qemu-kvm的网卡设备文件是vhost的。DPDK可加速交换模块OVS的转发效率，这部分我们还没有深入研究，就一笔带过了。



(http://dockerone.com/uploads/article/20160530/a1956bdf86fc7c38649d643223f667a0.PNG)

针对KVM的vGPU方案还比较少，时间关系，也一笔带过了。

(http://dockerone.com/uploads/article/20160530/c87629763d6c96f9c6184e3ee7559d71.PNG)

关于Docker，我们也是刚刚开始研究，算是学习笔记吧。有说的不对的地方，请各位大牛批评指正。

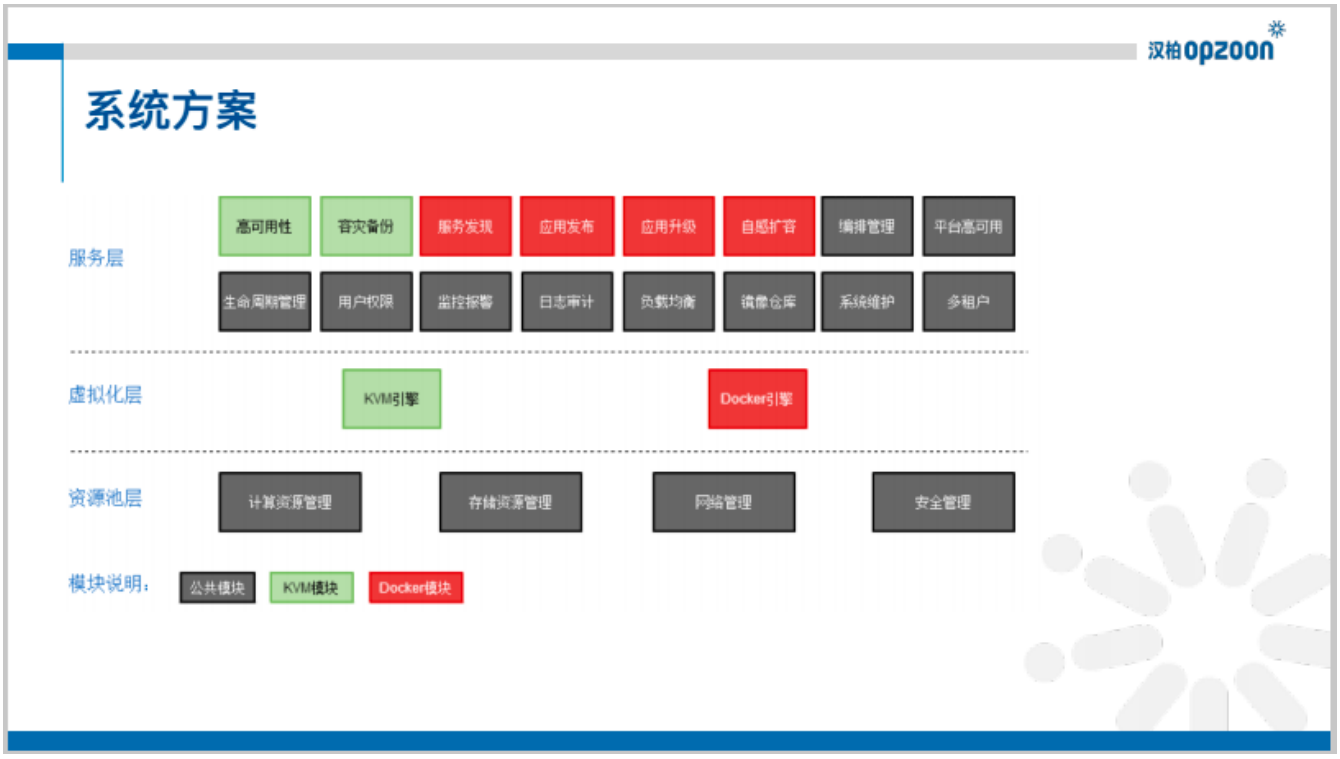
对比项	Docker	对比结果	虚拟化
快速创建、删除	启动应用	>>	启动Guest OS+启动应用
交付、部署	容器镜像	==	虚拟机镜像
密度	单Node 100~1000	>>	但Node 10~100
更新管理	迭代式更新，修改Dockerfile，对增量内容进行分发、存储、传输、节点启动和恢复迅速	>>	向虚拟机推送安装、升级应用软件补丁包
Windows的支持	不支持	<<	支持
稳定性	每月更新一个版本	<<	KVM，Xen，VMware都已很稳定
安全性	Docker具有宿主机root权限	<<	硬件隔离：Guest OS运行在非根模式
监控成熟度	还在发展过程中	<<	Host，Hypervisor，VM的监控工具在生产环境已使用多年
高可用性	通过业务本身的高可用性来保证	<<	武器库很丰富：快照，克隆，HA，动态迁移，异地容灾，异地双活
管理平台成熟度	以k8s为代表，还在快速发展过程中	<<	以OpenStack，vCenter，汉柏OPV-Suite为代表，已经在生产环境使用多年

(http://dockerone.com/uploads/article/20160530/0871fb78e2c5777f4dcb44369298fb0f.PNG)

左边是常用的对比图，可以明显看出Docker少了Guest OS Kernel这一层，所以它的密度更高，启动速度更快。但用户使用Docker和虚拟机，不会直接使用命令，一般是使用一个系统。所以，正确的对比办法是对比它们的系统。右边是从这么几个方面我认识到的对比。可以明显看出，Docker在启动速度，密度，更新管理上占有比较大的优势，其他方面，比如对Windows的支持行，稳定性，安全性，监控成熟度，高可用性，管理平台成熟度上来看，都低于虚拟化技术。当让，再次强调这个是我个人的认识，不一定正确。欢迎批评指正。

(http://dockerone.com/uploads/article/20160530/2720dac9db041bd40d9ab4645e59f996.PNG)

可以看出，这是个群雄逐鹿的时代，主要有OpenStack、Mesos、Kubernetes、Docker公司（社区）四个玩家。它们争夺的当然是云时代，开源平台的份额，也就是云时代的控制权。



(http://dockerone.com/uploads/article/20160530/d447dab740ad2d5365e1dcd67315f2a5.PNG)

笔者一直认为，云消费者并不关心，你这个服务商使用的是虚拟化技术还是Docker，更不关心你自己写的还是基于开源平台改的。他关心的是你的服务是否可靠，是否稳定，是否便宜，是否安全。所以，根据你团队的特点，选择你们自己最擅长的技术，为云消费者提供有竞争力的服务，才是未来我们能否立足的核心。

笔者认为融合了Docker与虚拟化的云平台应该包括三个层次：资源管理层，虚拟化层，服务层。当然docker与虚拟化会共用大部分模块，这也是笔者认为要构筑融合平台的好处。资源管理层至少包括：计算资源管理，存储资源管理，网络管理，安全管理。虚拟化层肯定包括：虚拟化引擎（一般情况下就是KVM），容器引擎（一般情况下就是Docker）。服务层至少包括：高可用性，编排管理，容灾备份，服务发现，应用发布，应用升级，自感扩容，编排管理，平台高可用性，生命周期管理，用户权限认证，监控报警，日志审计，负载均衡，镜像管理，系统维护等模块。

比较项目	kubernetes	Docker Swarm	Mesos	OpenStack Magnum（不包括OpenStack）
计算资源管理	2	2	2	2
存储资源管理	0	0	0	0
网络管理	1	1	1	1
平台高可用	0	0	2	0
编排管理	2	1	2	2
生命周期管理	2	1	1	1
用户管理	0	0	0	2
多租户管理	0	0	0	2
监控报警	0	0	0	0
负载均衡	2	1	1	0
镜像管理	2	2	2	2
日志审计	0	0	0	0
系统维护	0	0	0	0
高可用性	1	0	0	0
容灾备份	0	0	0	0
服务发现	2	1	1	1
应用发布	2	2	2	1
应用升级	0	0	0	0
自愿扩容	0	0	0	0
综合得分	16（42%）	11（29%）	14（37%）	14（37%）

(http://dockerone.com/uploads/article/20160530/49e7725da88d0a5121de89c041e7c22f.PNG)

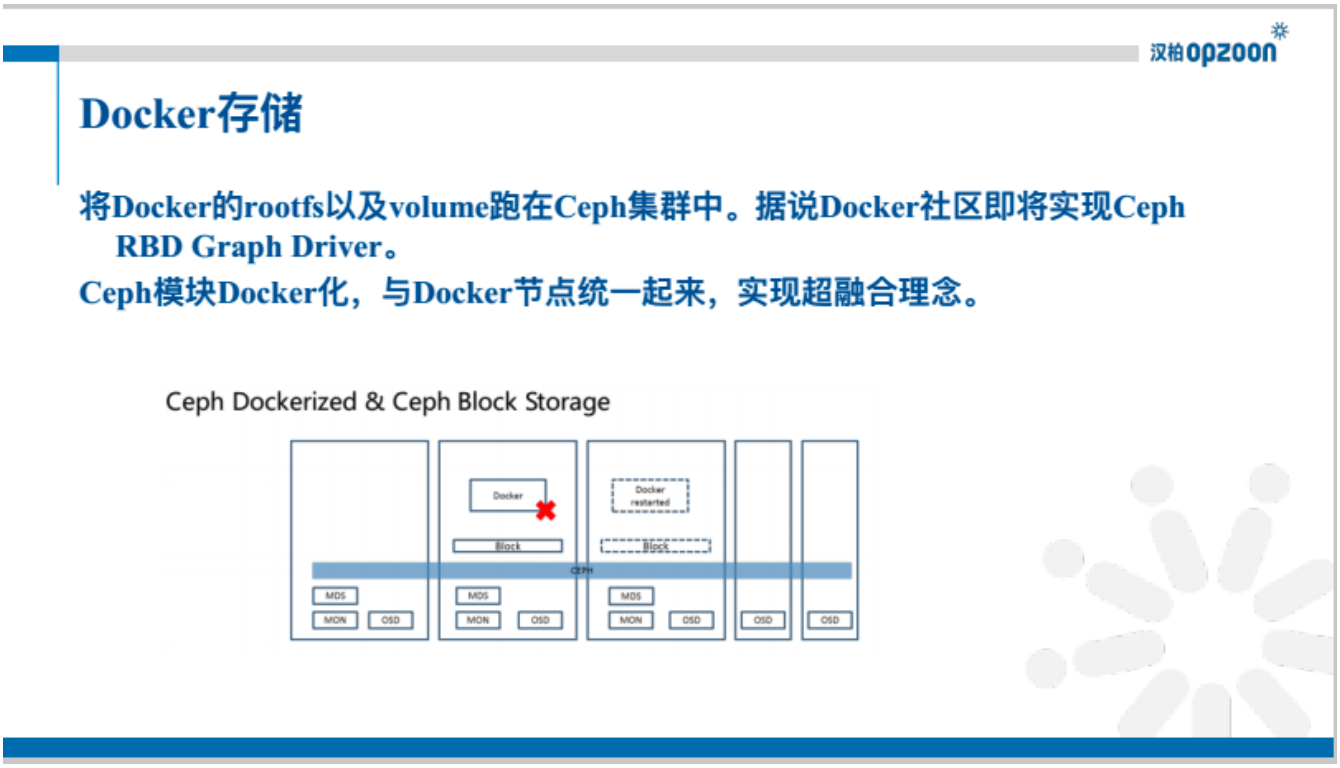
Kubernetes的优势在于它是第一个Docker集群管理平台，第一个提出并实现了Pod，Replication，Services Discovery等概念。关于技术细节，笔者在此不做过多介绍，请自行百度，Google，或者自己搭建平台体验。下面的其他方案也相同。

Swarm是Docker在2014年12月份新发布的Docker集群管理工具。Swarm可管理Docker集群，管理和分配计算资源，也包含服务发现（可以选用etcd、ZooKeeper、Consul），容器编排等功能。Swarm的优势是与Docker接口API统一。

Mesos的目标是下一代数据中心操作系统（DCOS），其最核心功能在于集群管理，计算资源管理，任务分发，原本用作Hadoop等分布式任务管理。从0.20版开始，Mesos支持Docker形式的任务调度（主要看中Docker的任务隔离，资源限制，随镜像发布）。在Mesos之上，Marathon可以用作为Docker编排和生命周期管理。

为保证云计算领域的领先地位，2015年5月份温哥华峰会上，OpenStack提出了“集成引擎”的思路。其实说白了就是在Kubernetes、Swarm和Mesos的上面套一层，用OpenStack Magnum的接口来管理它们。OpenStack的优势在于在虚拟化上积累的多租户，编排，存储以及网络能力。笔者认为这个方案有点另类，OpenStack不会甘心仅仅做个"集成商"的。笔者大胆猜测，凭借强大的内生能力，OpenStack Magnum一定会慢慢学习和消化Kubernetes、Swarm和Mesos的优势，并最终替代它们变成另一个"Nova"，这个"Nova"操作和管理Docker。

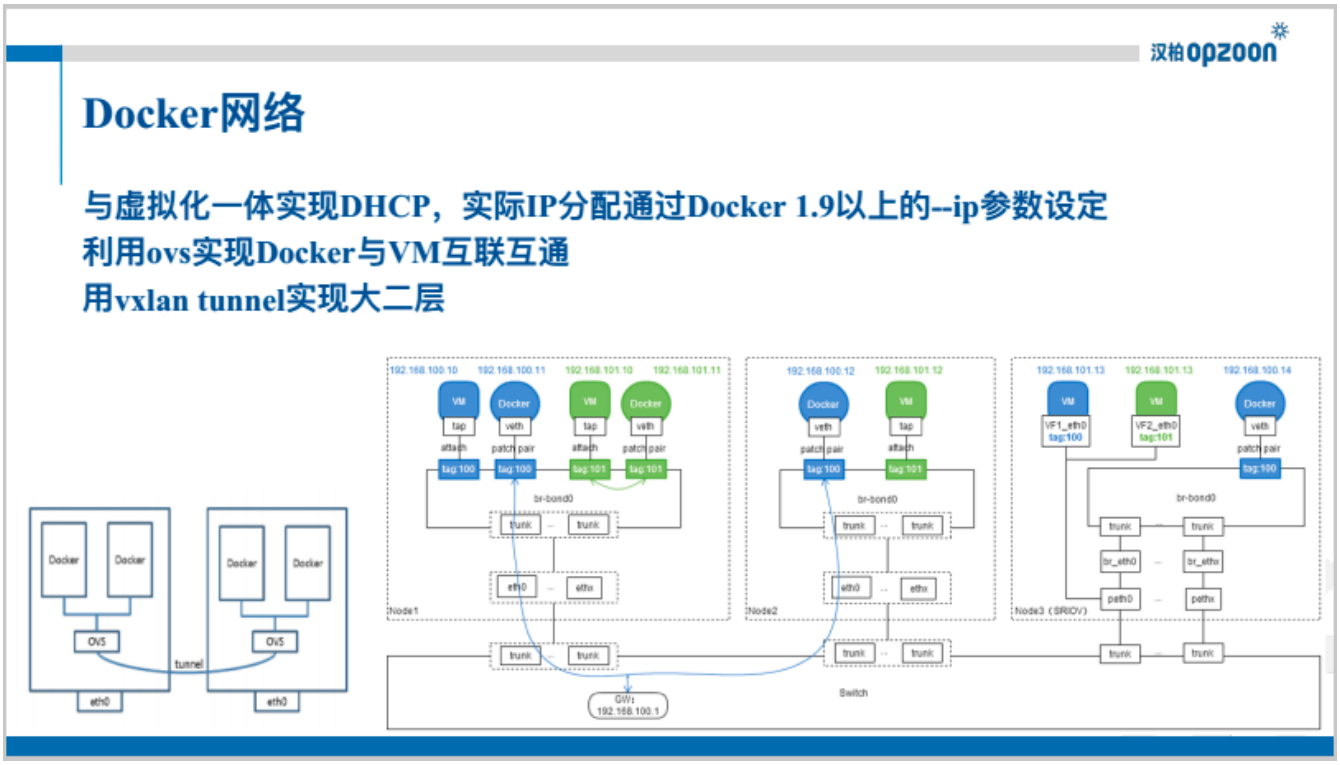
对四种方案进行全方位的比较，按照0（没有此项功能），1（有，但不完善），2（有，且比较完善）。结果是Kubernets稍稍胜出，个人认识，不一定正确。



(http://dockerone.com/uploads/article/20160530/8a3419acc5654aeb71f57a6b4f08d3c0.PNG)

关于存储

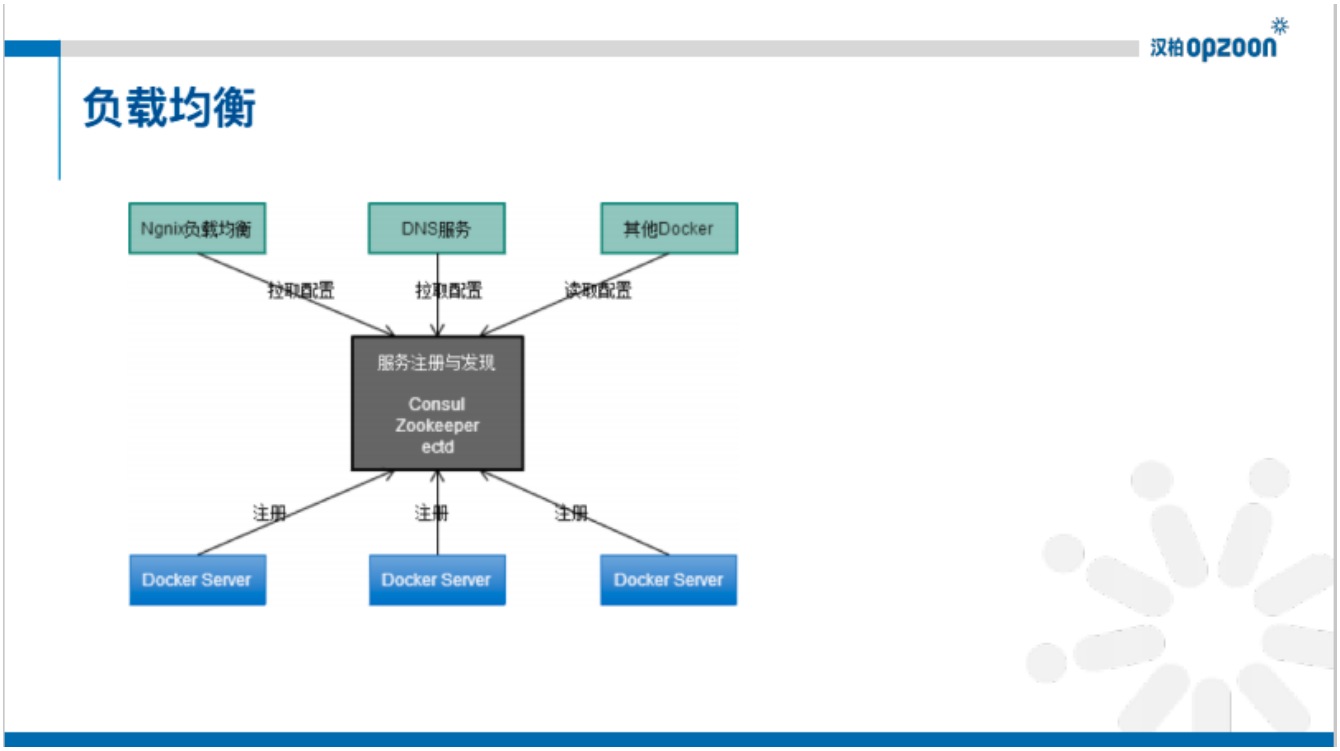
1. 将Docker的rootfs以及Volume跑在Ceph集群中。据说Docker社区即将实现Ceph RBD Graph Driver。
2. Ceph模块Docker化，与Docker节点统一起来，实现超融合理念。



(http://dockerone.com/uploads/article/20160530/842c67c31497317cceae7720964f0649.PNG)

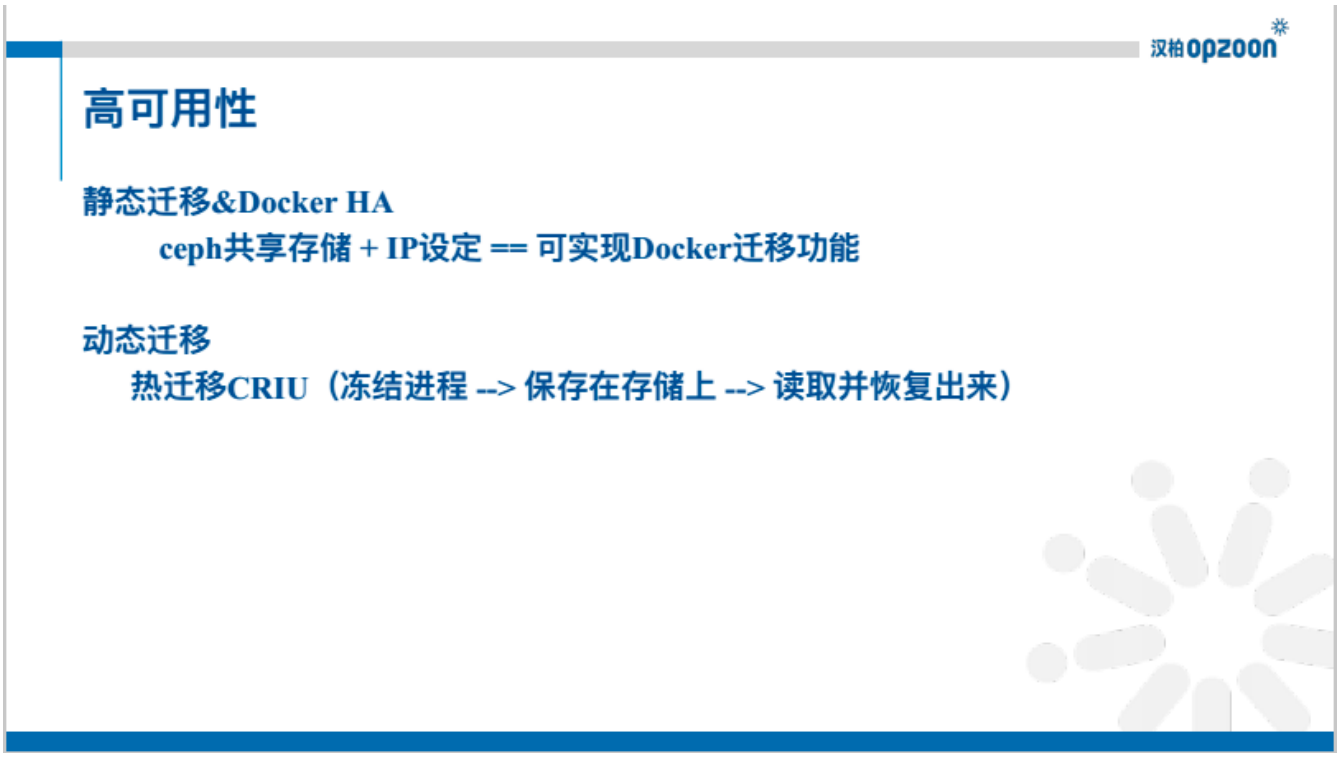
关于网络

1. 与虚拟化一体实现DHCP，实际IP分配通过Docker 1.9以上的--ip参数设定。
2. 利用OVS实现Docker与VM互联互通。



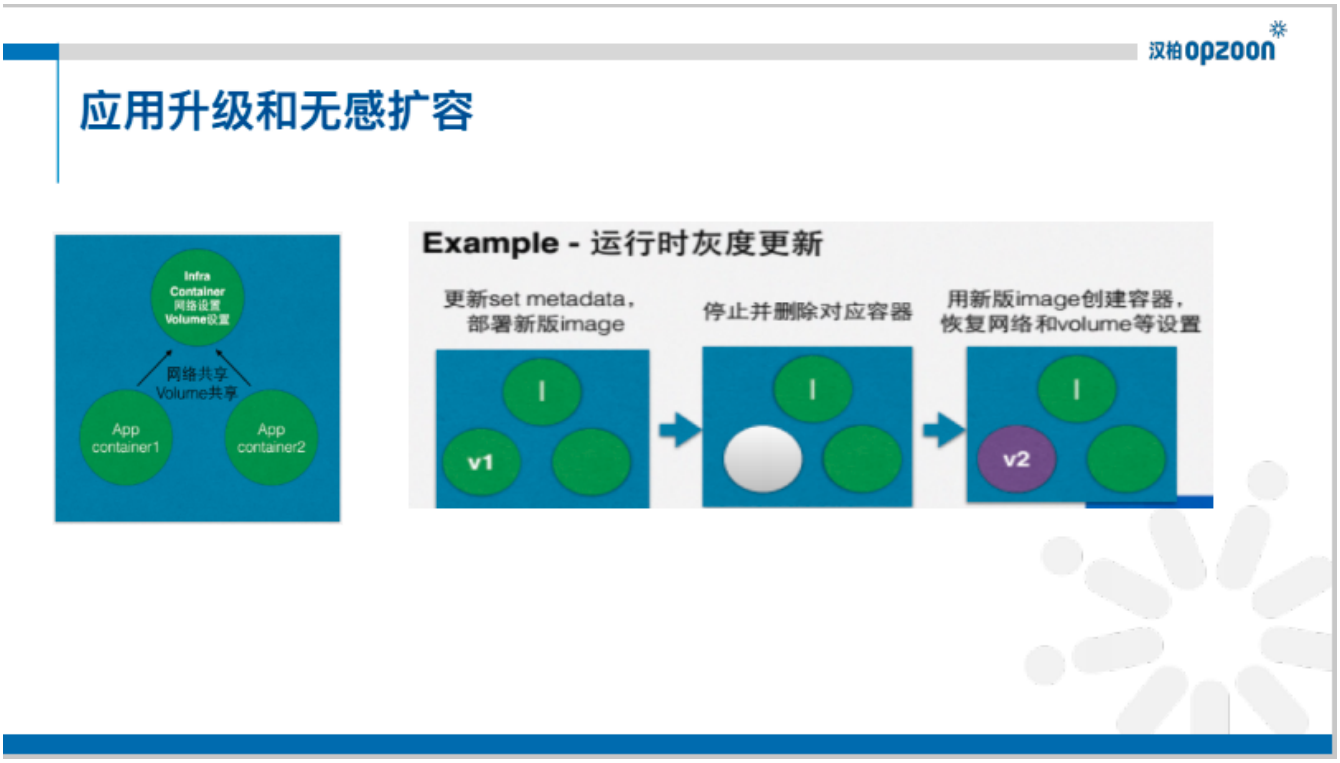
(http://dockerone.com/uploads/article/20160530/2250762c874ff50c8d819bd4b964a880.PNG)

服务发现可以让一个应用或者组件发现其运行环境以及其它应用或组件的信息。当一个服务启动时，注册自身信息，例如，一个MySQL数据库服务会在这注册它运行的ip和端口。负载均衡的策略可根据负载在Ngnix和DNS上体现，不如发现某个负载太高，就Ngnix或者DNS到另外的Docker Server上。



(http://dockerone.com/uploads/article/20160530/f826435a4063494623ed2c3a438fa112.PNG)

1. 静态迁移&Docker HA：Ceph共享存储 + IP设定 == 可实现Docker迁移功能。
2. 动态迁移： 热迁移CRIU（冻结进程 --> 保存在存储上 --> 读取并恢复出来）。

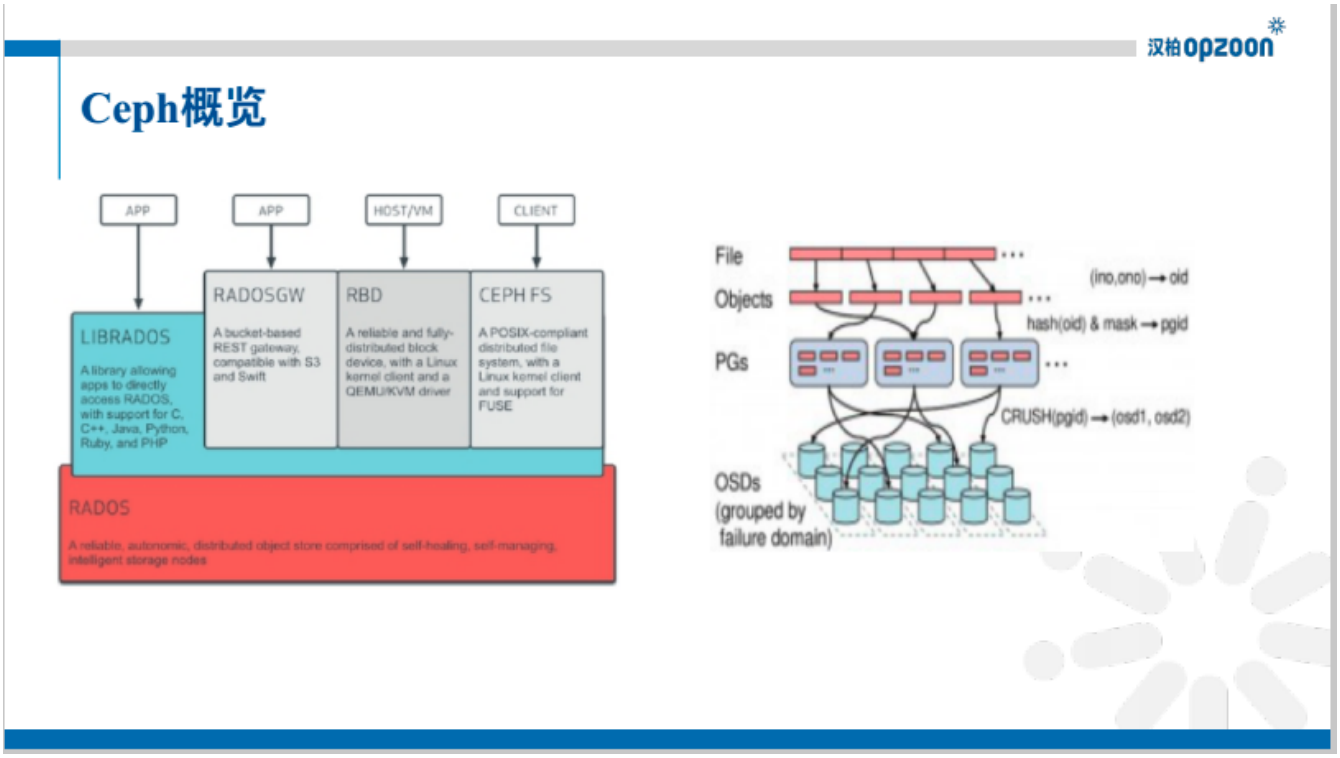


(http://dockerone.com/uploads/article/20160530/8d6e3f223d6832b781e09528cf5e947e.PNG)

一个Set跑一个业务，由多个Docker组成，支持灰度升级，如果监控到负载太高，也可在Set内增加Docker数量。

(http://dockerone.com/uploads/article/20160530/fec1ebf31d65f580bdb091e1212e1d78.PNG)

这里说的MixSAN就是Ceph。

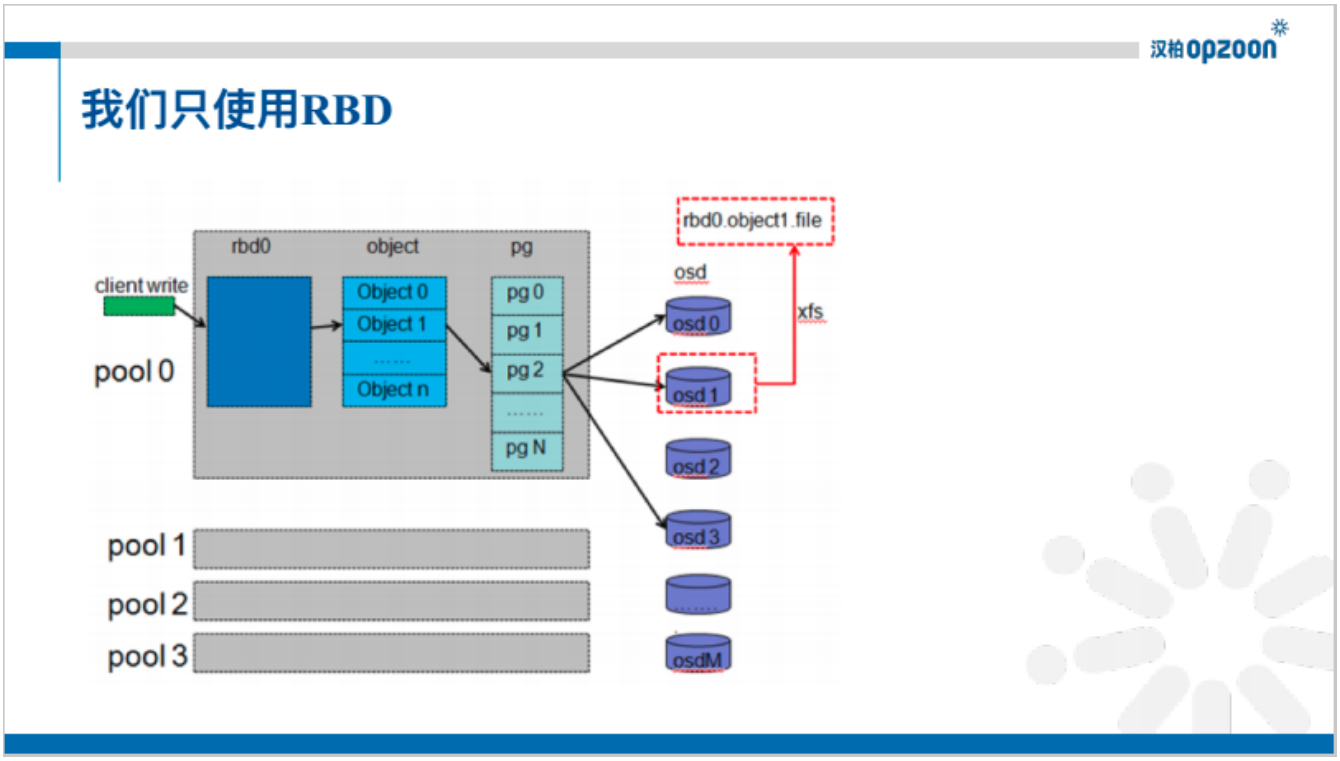


(http://dockerone.com/uploads/article/20160530/dd9e9721c690028f63017e4e80dd31b8.PNG)

Ceph是一种为优秀的性能、可靠性和可扩展性而设计的统一的（同时提供对象存储、块存储和文件系统存储三种功能）、分布式的存储系统（高可靠性；高度自动化；高可扩展性）。

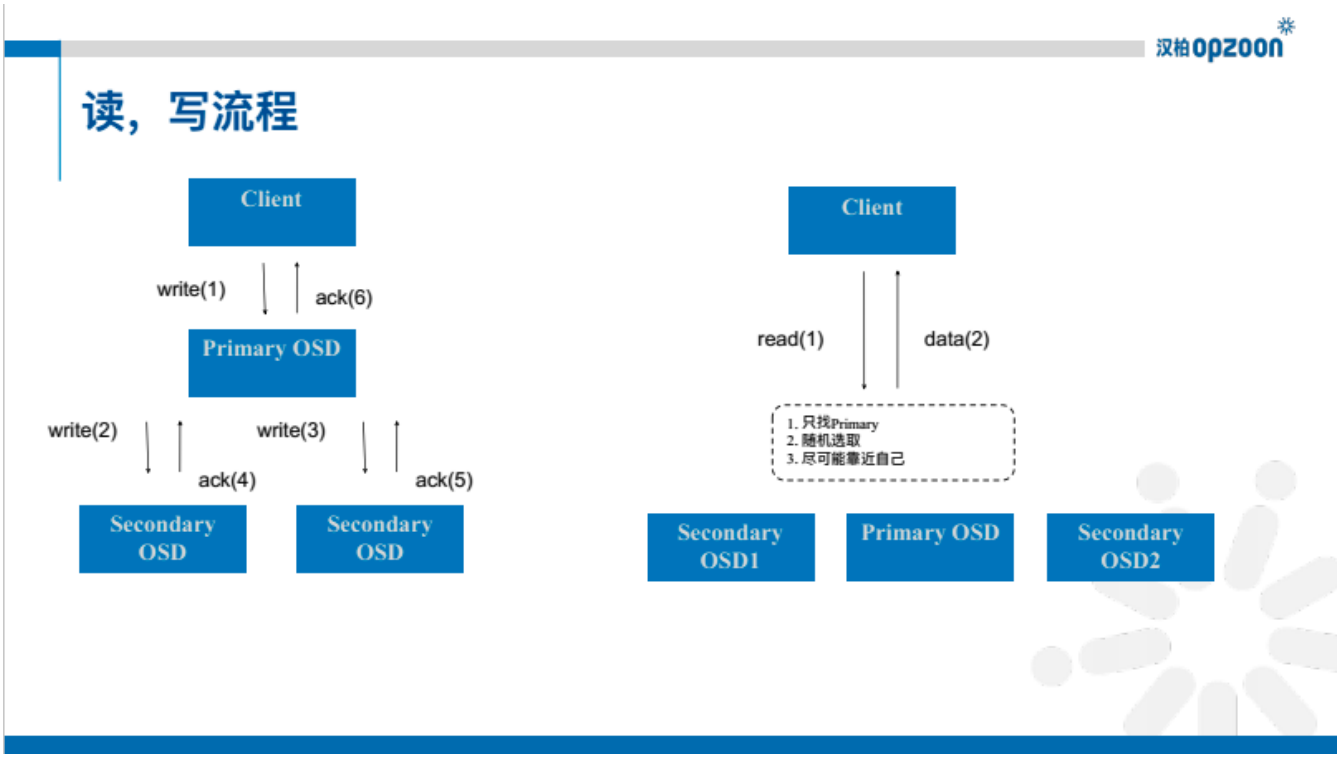
RADOS的系统逻辑结构如右图所示：

1. OSD和monitor之间相互传输节点状态信息，共同得出系统的cluster map。
2. 客户端程序通过与OSD或者monitor的交互获取cluster map，然后直接在本地进行计算，得出对象的存储位置后，便直接与对应的OSD通信，完成数据的各种操作。



(http://dockerone.com/uploads/article/20160530/2c731ec718c82c74ca767b780198ef3c.PNG)

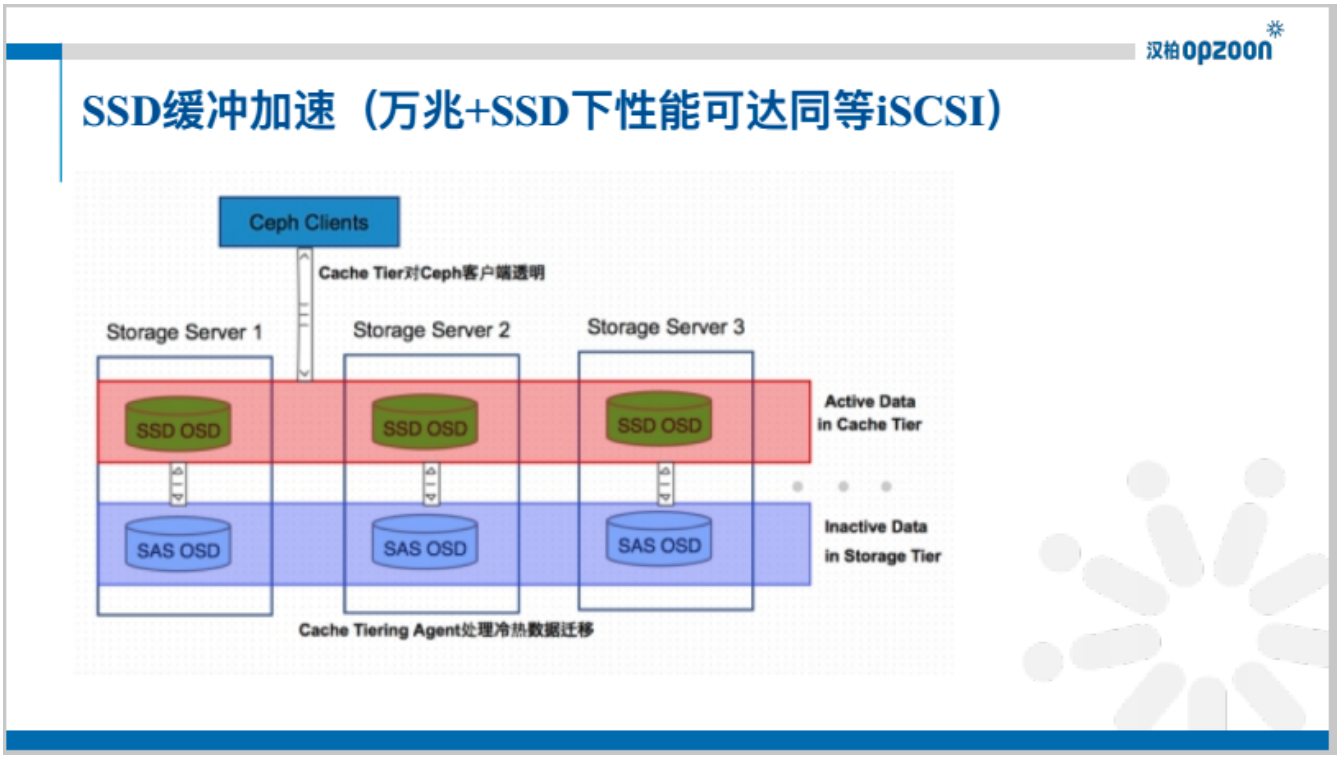
pool，rbd卷，pg，osd之间的关系如图所示。在Pool内建立rbd卷，按照4M分为许多object，一个object对应一个pg，一个pg分发到不同的osd上把数据存储起来，一般就是存储在一个文件中。



(http://dockerone.com/uploads/article/20160530/2fd36d5c84b0fbaafead9f3e9480e2a8.PNG)

左图可以明显看出，一次写会写三次，所以有写放大的问题。

读的话分为几种算法，一种是只去找primary，一种是在三个osd上随机找一个去读，另一个是尽可能靠近自己去读。



(http://dockerone.com/uploads/article/20160530/3c15ec358da339aa7665fe9611774377.PNG)

这种情况我们在万兆情况下，测试出来的性能等同于万兆iSCSI存储性能。

这么做的好处：冷热数据分离，用相对快速/昂贵的存储设备如SSD盘，组成一个Pool来作为Cache层，后端用相对慢速/廉价的设备来组建冷数据存储池。

tier Cache策略有两种模式：

1. Write-back模式 Ceph客户端直接往cache-pool里写数据，写完立即返回，Tiering Agent再及时把数据flush到base-pool。当客户端要读的数据不在cache-pool时，Tiering Agent负责把数据从base-pool迁移到cache-pool。