[转载]VirtFS 虚拟化技术简介


Zero
Kernel、Storage

原文链接：blog.csdn.net/xuriwuyun...

背景

在虚拟化领域中，向虚拟机提供虚拟化设备的技术从系统层次上，可以分为三种：硬件层虚拟化、驱动层虚拟化、系统层虚拟机。这三种技术也是按照这种先后排序依次出现的，后一种技术是为弥补前一种技术的不足而被发明出来。不过有时为了达到新的要求，需要在原本的优点之间进行取舍。

硬件层虚拟化，即完全虚拟一个物理设备，提供一个完整设备指令集。对于这类虚拟化技术，使用自带的设备驱动程序，操作系统可以不经任何修改直接运行在虚拟环境中。如：虚拟IDE设备、虚拟E1000网卡等。虚拟机对该种虚拟化设备的操作需要经过多次转换，才能最终完成：首先调用虚拟系统服务接口，其次由虚拟设备驱动处理，最后转换成虚拟设备指令执行。执行虚拟设备指令时触发异常，由Hypervisor捕获，并将虚拟设备操作信息交给宿主机系统监控程序，监控程序调用宿主机系统服务接口完成真正的设备操作。这里需要进行多次转换，而且需要多个VM_IN\VM_OUT操作，性能损耗较大。优点在于可以模拟出多种物理设备，运行未经修改的操作系统。

驱动层虚拟化，这类虚拟化不提供对设备指令集的模拟，而是在硬件驱动层进行虚拟化。使用这类虚拟化技术，需要虚拟机操作系统提供这类虚拟化驱动。使用该虚拟化技术的有：KVM上基于VirtIO架构的net、disk和XEN上基于split-driver架构的虚拟设备等。该虚拟化技术无需对虚拟设备指令集进行解析和转换，并且减少了VM_IN\VM_OUT操作，性能上有较大提升。缺点在于虚拟机操作系统需要有相应的驱动支持。

系统层虚拟化，这类虚拟化直接将宿主机的系统服务接口交给虚拟机系统使用，减少了虚拟化过程中的多次转换和VM_IN\VM_OUT操作，性能上有进一步的提升。而且由于虚拟机直接使用宿主机系统服务，这使得Hypervisor能够更加精准的监控和分析虚拟机内部操作和负载情况。同样，这类虚拟化技术要求虚拟机系统具有相应的内核支持模块，并且这类技术出现时间不长，现在操作系统对其支持还不够成熟。

由于虚拟机使用的块虚拟设备是基于硬件层虚拟化技术和驱动层虚拟化技术，从理论上分析，其性能还可以提升，并且块虚拟设备无法满足虚拟机与宿主机之间共享文件的需求。而一般网络文件系统，如NFS、CIFS，用于实现在虚拟机与宿主机之间共享文件时，性能上有较大损耗。加上网络文件系统并非专为虚拟化设计，所以其支持的功能也不够完善。

VirtFS，系统层虚拟化技术的一种实现，就是为解决共享文件系统在虚拟化环境中所遇到的一些问题：提供完备的功能支持和良好的性能。

VirtFS

VirtFS是针对虚拟化环境，定制的类虚拟化文件系统，属于系统层虚拟化技术。它通过向虚拟机系统提供系统层的服务接口，与Hypervisor交互，能够获得比直接访问设备驱动更高的性能。

VirtFS是基于QEMU、KVM、VirtIO技术和9P2000.L协议实现的，由VirtFSServer和VirtFSClient两部分组成。通过VirtFSServer分享在宿主机上指定的文件目录，虚拟机系统使用mount指令通过9P2000.L协议挂载VirtFS，虚拟机用户使用VirtFS如同使用本地文件一般。使用VirtFS的系统结构

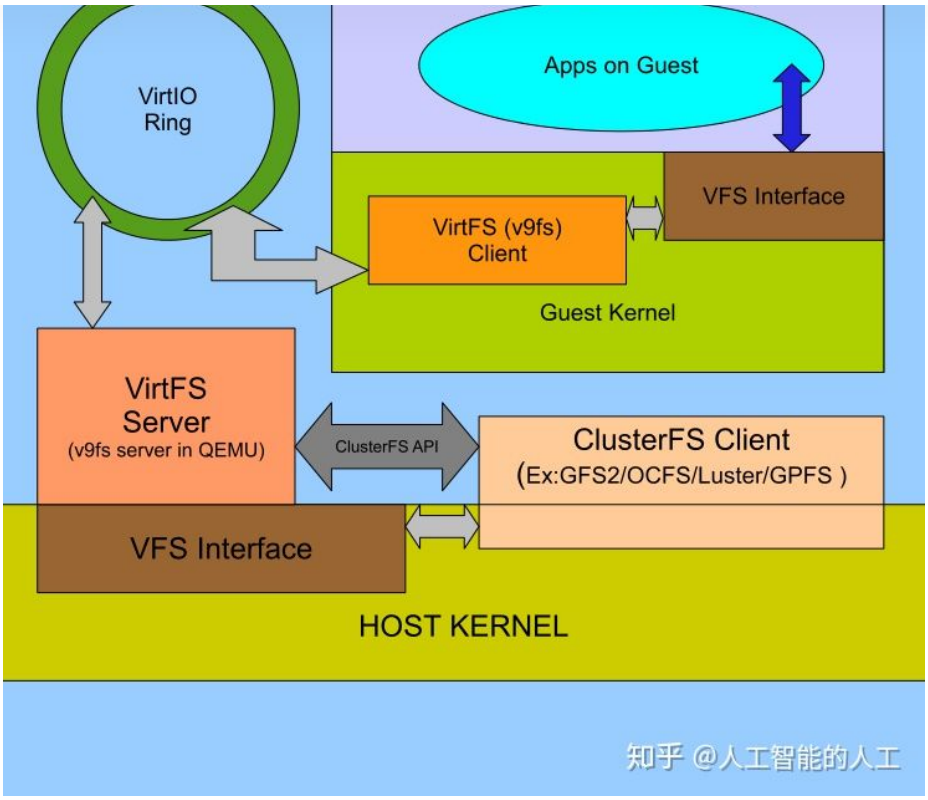

图1 VirtFS系统结构图

由图1可知，VirtFSServer以QEMU进程的一个子功能形式运行在宿主机上，VirtFSClient以内核模块的形式运行在虚拟机系统上。Server与Client之间的接口设计遵循9P协议[6]，并在9P协议的基础上进行了扩展，形成了9P2000.L协议[8]，以更好的支持Linux的VFSAPI。使用QEMU-KVM的VirtIO技术，作为Server与Client之间消息传递机制，保证良好的传输性能。对于9P协议和VirtIO下面将会进行详细介绍。

VirtFS相比传统虚拟块设备，它具有以下优势：

- 不用将虚拟文件系统操作转换成虚拟设备驱动操作，再由虚拟设备驱动操作转换为宿主机文件系统操作；
- Hypervisor能够更加容易监控虚拟机的文件操作，了解其负载情况。进而能够更合理的利用IOcache，和其它系统机制；
- 可以在宿主机与虚拟机、虚拟机与虚拟机之间共享文件系统。而虚拟块设备不能被多个虚拟机共享，除非它是只读的，因为传统文件系统在内存保存了大量文件、diskcache和pagecache的状态。并且宿主机与客户机重复cache；
- 多种用户管理模式，解决多操作系统共享同一文件系统的用户权限问题。传统分布式文件系统，存在对两个用户域的管理问题，同样存在重复cache。并且多种分布式系统，可能存在不同语义问题；
- 简单，针对虚拟化定制。

9P协议

9P协议是由贝尔实验室制定，在Plan9项目中产生出来的[6]。Plan9是贝尔实验室研发一种新型的操作系统，以弥补Unix在网络方面表现的不足，不以网络上孤立的操作系统存在，而是作为一个安全整合网络资源的无缝衔接的分布式系统。Plan9中，所有的系统资源和服务接口都是以文件的形式存在，9P协议就是关于文件访问的抽象接口定义。

Rxxxx表示返回值的消息类型，tag[2]是为每次请求生成唯一标示id，其对应返回消息的tag应与请求的tag保持一致，其它的为每个操作接收的参数：

```
size[4] Tversion tag[2] msize[4] version[s]

size[4] Rversion tag[2] msize[4] version[s]

size[4] Tauth tag[2] afid[4] uname[s] aname[s]

size[4] Rauth tag[2] aqid[13]

size[4] Rerror tag[2] ename[s]

size[4] Tflush tag[2] oldtag[2]

size[4] Rflush tag[2]

size[4] Tattach tag[2] fid[4] afid[4] uname[s] aname[s]

size[4] Rattach tag[2] qid[13]

size[4] Twalk tag[2] fid[4] newfid[4] nwname[2] nwname*(wname[s])

size[4] Rwalk tag[2] nwqid[2] nwqid*(wqid[13])

size[4] Topen tag[2] fid[4] mode[1]

size[4] Ropen tag[2] qid[13] iounit[4]

size[4] Tcreate tag[2] fid[4] name[s] perm[4] mode[1]
```

```
size[4] Tread tag[2] fid[4] offset[8] count[4]

size[4] Rread tag[2] count[4] data[count]

size[4] Twrite tag[2] fid[4] offset[8] count[4] data[count]

size[4] Rwrite tag[2] count[4]

size[4] Tclunk tag[2] fid[4]

size[4] Rclunk tag[2]

size[4] Tremove tag[2] fid[4]

size[4] Rremove tag[2]

size[4] Tstat tag[2] fid[4]

size[4] Rstat tag[2] stat[n]

size[4] Twstat tag[2] fid[4] stat[n]

size[4] Rwstat tag[2]
```

9P协议本身没有规定相关权限设置，需要借助额外的认证系统。它规定了server端保存状态信息，和事务操作。支持以session方式操作，支持在同一个传输通道上同时执行多个事务操作。Linux2.6.14支持基于TCP/IP和命名管道的9P协议，之后的linux支持以模块化方式加载对9P支持的传输协议。后面我们加载VirtFS系统时，使用的就是以模块形式加载v9fs模块。

9P在Plan9系统系统中类似于linux中VFS，不管是本地还是远程的资源都是通过9P访问。但是9P只实现了VFS的一个功能子集，而且某些参数定义并不一样（如权限设置、文件打开模式）。并且以字符串id来管理用户和组。

9P2000.u—为Unix定制的9P协议，支持以数字ID来管理用户和组[7]。为兼容POSIX标准，还定义了permission位、额外的文件操作模式、扩展属性结构。它的缺点是并未完全支持VFS功能，重要

```
size[4] Tversion tag[2] msize[4] version[s]

size[4] Rversion tag[2] msize[4] version[s]

size[4] Rerror tag[2] ename[s] errno[4]

size[4] Tauth tag[2] afid[4] uname[s] aname[s]

size[4] Rauth tag[2] aqid[13]

size[4] Tattach tag[2] fid[4] afid[4] uname[s] aname[s]

size[4] Rattach tag[2] qid[13]

size[4] Tcreate tag[2] fid[4] name[s] perm[4] mode[1] extension[s]

size[4] Rcreate tag[2] qid[13] iounit[4]

size[4] Tstat tag[2] fid[4]

size[4] Rstat tag[2] stat[n]

size[4] Twstat tag[2] fid[4] stat[n]

size[4] Rwstat tag[2]
```

9P2000.L—9P2000.u的扩展，为支持Linux的VFS而设计[8]。添加对目录、节点、连接、锁文件、扩展属性等一系列操作。下面列出9P2000.L扩展的一部分操作，并未全部列出。

```
size[4] Tsetattr tag[2] fid[4] valid[4] mode[4] uid[4] gid[4] size[8]atime_sec
```

```
size[4] Tfsync tag[2] fid[4]

size[4] Rfsync tag[2]

size[4] Tlock tag[2] fid[4] type[1] flags[4] start[8] length[8]proc_id[4] clier

size[4] Rlock tag[2] status[1]

size[4] Tgetlock tag[2] fid[4] type[1] start[8] length[8] proc_id[4]client_id[s

size[4] Rgetlock tag[2] type[1] start[8] length[8] proc_id[4]client_id[s]

size[4] Trenameat tag[2] olddirfid[4] oldname[s] newdirfid[4]newname[s]

size[4] Rrenameat tag[2]

size[4] Tmkdir tag[2] dfid[4] name[s] mode[4] gid[4]

size[4] Rmkdir tag[2] qid[13]

size[4] Tlink tag[2] dfid[4] fid[4] name[s]

size[4] Rlink tag[2]
```

VirtIO

VirtIO最初由澳大利亚的一个天才级程序员RustyRussell编写，是一个在hypervisor之上的抽象API接口，让客户机知道自己运行在虚拟化环境中，从而与hypervisor根据VirtIO标准协作，从而在客户机中达到更好的性能（特别是I/O性能）。目前，有不少虚拟机都采用了VirtIO半虚拟化驱动来提高性能，如KVM和Lguest。QEMU/KVM中，VirtIO的基本结构框架如图2所示。

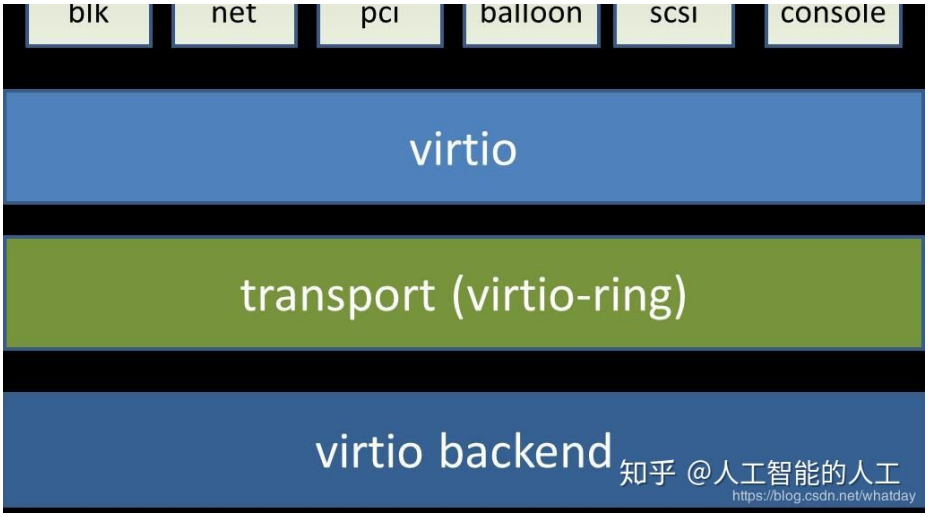

图2 KVM中virtio基本架构

其中前端驱动（frondend，如virtio-blk、virtio-net等）是在客户机中存在的驱动程序模块，而后端处理程序（backend）是在QEMU中实现的[2]。在这前后端驱动之间，还定义了两层来支持客户机与QEMU之间的通信。其中，“virtio”这一层是虚拟队列接口，它在概念上将前端驱动程序附加到后端处理程序。一个前端驱动程序可以使用0个或多个队列，具体数量取决于需求。例如，virtio-net网络驱动程序使用两个虚拟队列（一个用于接收，另一个用于发送），而virtio-blk块驱动程序仅使用一个虚拟队列。虚拟队列实际上被实现为跨越客户机操作系统和hypervisor的衔接点，但它可以通过任意方式实现，前提是客户机操作系统和virtio后端程序都遵循一定的标准，以相互匹配的方式实现它。而virtio-ring实现了环形缓冲区（ringbuffer），用于保存前端驱动和后端处理程序执行的信息，并且它可以一次性保存前端驱动的多次I/O请求，并且交由后端去动去批量处理，最后实际调用宿主机中设备驱动实现物理上的I/O操作，这样做就可以根据约定实现批量处理而不是客户机中每次I/O请求都需要处理一次，从而提高客户机与hypervisor信息交换的效率。

VirtIO半虚拟化驱动的方式，可以获得很好的I/O性能，其性能几乎可以达到和native（即：非虚拟化环境中的原生系统）差不多的I/O性能。所以，在使用KVM之时，如果宿主机内核和客户机都支持VirtIO的情况下，一般推荐使用VirtIO达到更好的性能。当然，VirtIO的也是有缺点的，它必须要客户机安装特定的VirtIO驱动使其知道是运行在虚拟化环境中，且按照VirtIO的规定格式进行数据传输，不过客户机中可能有一些老的Linux系统不支持VirtIO和主流的Windows系统需要安装特定的驱动才支持VirtIO。不过，较新的一些Linux发行版（如RHEL6.3、Fedora17等）默认都将VirtIO相关驱动编译为模块，可直接作为客户机使用VirtIO，而且对于主流Windows系统都有对应的VirtIO驱动程序可供下载使用。

挂载VirtFS

VirtFS文件系统相对于其它虚拟文件系统而言，挂载较为复杂。它即需要QEMU对virtio-9p-pci虚拟设备支持，同时也需要客户机操作系统对9P文件系统的支持。

1. 编译QEMU

Centos6.5系统中，官方源下载安装的QEMU0.12.1没有提供对virtio-9p-pci设备的支持。如果需要使用VirtFS，需重新编译安装QEMU。下载最新版本，编译安装步骤如下：

```
git clone git://git.qemu-project.org/qemu.git
cd qemu
./configure –enable-virtfs
make –j 24
```

dev，导致QEMU没有编译对VirtFS的支持（在没有加—enable-virtfs参数情况下，即使没有libattr1-dev，运行也不会报错）。安装过程中可能会出现问题，一般因为依赖包未安装，根据错误提示安装相关依赖包即可。使用QEMU的virtio-9p-pci功能，无需对宿主机的内核进行特殊配置。

1. 编译客户机内核

对于客户机操作系统，要求内核提供对9P协议文件系统的支持。内核2.6.36.rc4及以上版本，已经提供对9P文件系统的支持。在编译内核时，选定如下配置项，对进行重新编译安装即可：

CONFIG_NET_9P=y

CONFIG_NET_9P_VIRTIO=y

CONFIG_NET_9P_DEBUG=y (Optional)

CONFIG_9P_FS=y

CONFIG_9P_FS_POSIX_ACL=y

运行如下命令，以配置安装：

```
make oldconfig
make menuconfig
make -j 24
make modules_install
make install
```

对于现已使用的操作系统，可以查看/boot/config文件，确定对应配置项是否打开。Centos6.5系统的默认安装内核没有打开这些配置项，需要重新编译内核。Ubuntu12.04系统内核的相关配置项已经打开，可以直接使用。

1. 启动虚拟机

启动带有VirtFS的虚拟机，指令如下：

qemu-system-x86_64 -m 4096 -drivefile=centos6.5.qcow2 \

-fsdev@var{fsdriver},id=@var{id},path=@var{path},[security_model=@var{security_model}][,writeout=@var{writeout}][,readonly][,socket=@var{socket}|sock_fd=@var{sock_fd}]\

-device virtio-9p-pci,fsdev=@var{id},mount_tag=@var{mount_tag}

-fsdev定义新的文件系统设备，它支持如下参数：

fsdriver –指定该文件系统设用后端驱动，目前支持的选项有：local、handle、proxy。

id – 用于给QEMU进行标示该设备的唯一IP，客户机不可见。

path –指定通过VirtFS系统共享的宿主机上的文件路径，虚拟机通过挂载VirtFS系统，即可访问该路径上的文件。

security_model — 配置VirtFSserver使用的安全模型，由于虚拟机与宿主机需要共享同一文件系统，那么需要通过安全模型来解决，不同操作系统用户使用同一文件的访问属性问题。目前支持的安全模型有"passthrough" "mapped-xattr" "mapped-file" and "none"

readonly -- 指定改参数后，VirtFS挂载将只读，默认为可读写。

-devicevirtio-9p-pci参数与-fsdev成对使用，支持参数如下：

id –指定前面通过-fsdev配置的id，与fsdev进行关联。

mount_tag –设置该VirsFS文件的挂载标示，这在虚拟机挂载VirtFS系统时会用到。

1. **挂载VirtFS**

当虚拟机启动完成后，登录虚拟机系统，执行如下命令，挂载VirtFS。

mount -t 9p -otrans=virtio,version=@var{version},msize=@var{msize}@var{mount_tag} /mnt

该命令将VirtFS挂载到/mnt目录，即宿主机上通过-fsdev参数指定path文件路径。其读写与其它本地文件操作一模一样。它支持很多参数，这里指简单讲解两个。参考文献[4]对其做了详细说明。

version –挂载VirtFS文件系统所使用的协议，可用的有：9p2000、9p2000.u、9p2000.l，默认值为9p2000.u。

msize– 该参数指定VirtFSserver与client之间交互消息的最大长度。文件读写操作时，如果文件块过大，那么每次读写请求会才分成多个消息进行发送。所以需要将该值尽量增大，以免影响性能。

mount_tag– 即为通过-device参数配置的mount_tag。

性能测试

硬盘测试分别在虚拟机（VirtFS）、虚拟机（VirtIO）、虚拟机（IDE）三种环境上进行。由于VirtFS系统的挂载不能禁用Cache（ps：VirtFS文件系统有一个writeout=immediate设置，它依然使用Cache来进行数据读写，但是保证每次写入数据时，只有Cache同步到硬盘时，才返回响应消息）。而且Cache功能本为VirtFS的一大优势，如若不用，VirtFS也可以不用了。为公正比较，测试时，VirtFS、VirtIO设备、IDE设备都加上了宿主机的Cache功能，而禁用虚拟机系统内部的Cache功能。

使用virtio硬盘驱动的虚拟机采用如下命令启动：

kvm-smp 24 -m 4096 -drivefile=centos6.5.qcow2,if=virtio

完全虚拟化虚拟机采用如下命令启动：

kvm-smp 24 -m 4096 -drivefile=centos6.5.qcow2,if=ide

使用VirtFS的虚拟机采用如下命令启动：

qemu-system-x86_64--enable-kvm -smp 4 -m 4096 -drive file=centos6.5.qcow2 -fsdevlocal,security_model=passthrough,id=fsdev0,path=/tmp/share -devicevirtio-9p-pci,fsdev=fsdev0,mount_tag=hostshare

测试分别以4KB、64KB、1MB文件块大小对于磁盘进行顺序读、顺序写、随机读、随机写，每次测试IO大小为4GB。下面是磁盘IO的测试结果，单位为KB/s。

为防止9P协议的消息长度对VirtFS的性能造成影响，虚拟机在挂载VirtFS时，将msize参数设置成足够大，挂载命令如下：

mount-t 9p -o trans=virtio,version=9p2000.L,msize=1000000000hostshare /mnt

单位（KB/s）顺序读写随机读写

顺序读（moreis better）

单位（KB/s）	虚拟机（VirtFS）	虚拟机（VirtIO）	虚拟机（IDE）
4KB	36650	19613	8979
64KB	335956	130027	86349
1MB	672491	354147	245257

表1磁盘io连续读性能对比

顺序写（moreis better）

单位（KB/s）	虚拟机（VirtFS）	虚拟机（VirtIO）	虚拟机（IDE）
4KB	34917	19610	8974
64KB	334810	129472	85950
1MB	643574	358321	245017

表2磁盘io连续写性能对比

随机读（moreis better）

单位（KB/s）	虚拟机（VirtFS）	虚拟机（VirtIO）	虚拟机（IDE）
4KB	32829	9786	5622
64KB	328884	85979	69278
1MB	576954	196024	233620

表3磁盘io随机读性能对比

随机写（moreis better）

单位（KB/s）	虚拟机（VirtFS）	虚拟机（VirtIO）	虚拟机（IDE）
4KB	32811	9793	5648
64KB	329045	86058	68773
1MB	602878	194688	240097

表5默认msize值下，VirtFs读写性能对比

结论

- 在不使用虚拟机系统cache的情况下，虚拟机（VirtFS）> 虚拟机（VirtIO）> 虚拟机（IDE）：

（1）小文件随机读写性能，VirtFS是VirtIO的3.35倍，是IDE的5.81倍；

（2）小文件顺序读写性能，VirtFS是VirtIO的1.87倍，是IDE的4.08倍；

（3）大文件随机读写性能，VirtFS是VirtIO的3.10倍，是IDE的2.51倍；

（4）大文件连续读写性能，VirtFS是VirtIO的1.90倍，是IDE的2.74倍；

- 当9P协议的消息长度足够长时（即mount命令的参数msize足够大时），读写文件块的大小对VirtFS的读写性能有明显影响：1MB文件块的读写速度是4KB文件块的18.35倍，64KB文件块的读写速度是4KB文件块的9.17倍。
- 当9P协议的消息长度采用默认值时，读写文件块的大小对VirtFS的读写性能影响不大：1MB文件块的读写速度是4KB文件块的1.11倍，64KB文件块的读写速度是4KB文件块的1.09倍。
- 在VirtFS中，连续读写和随机读写速度基本相等。

[1] linux-kvm.org/page/9p_v...

[2] wiki.qemu.org/Documenta...

[3] kernel.org/doc/Document...

[4] en.wikipedia.org/wiki/9...

[5] Jujjuri V, Van Hensbergen E, Liguori A, et al. VirtFS—Avirtualization aware File System passthrough[C]//Ottawa LinuxSymposium (OLS). 2010: 109-120.

[6] man.cat-v.org/plan_9/5/...

[7] ericvh.github.io/9p-rfc...

[8] code.google.com/p/diod/...

[9] plan9.bell-labs.com/mag...

编辑于 2020-03-25 19:02

VMware（威睿）

知乎 首发于 落雪的之乎者也