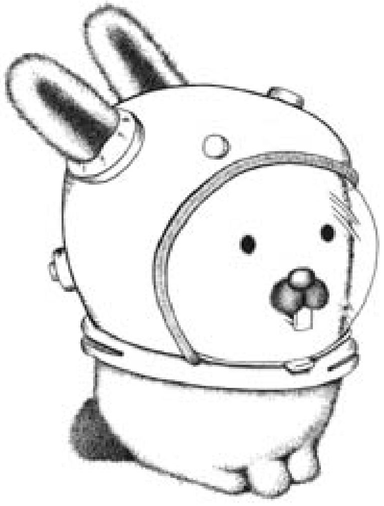
IBM Research

Paravirtualized File Systems



Eric Van Hensbergen
IBM Austin Research Lab
(bergevan@us.ibm.com)

KVM Forum 2008

Agenda

- Motivation: Why File Systems?
- Overview of Approach (9P over Virtio)
- 9P Basics
- Using 9P/virtio with KVM
- Preliminary Performance
- Future Work

Traditional Storage Options

- Virtual Block Devices
  - Generally best performance
  - Built-in mechanisms for copy-on-write, snapshots, etc.
  - Generally exclusive to a guest
    - No easy way to maintain consistency between two guests sharing a block device read/write
  - Can be somewhat cumbersome to manage content out of guest
- Network File System
  - Provide mechanisms for consistency (mostly) so enable sharing
  - Can be used with special servers or stackable file systems (ie. Unionfs) to provide copy-on-write, snapshots, etc.
  - Incurs extra management and performance overhead
  - NFS (and many others) are quite latency sensitive and don't seem to do well over virtualized networks

Goal

- Provide a direct file system proxy mechanism built on top of the native host<->guest I/O transport
  - Avoid unnecessary network and device overhead
  - Opportunity to optimize sharing and management
- Use cases
  - Access to host provided file hierarchy for consistent sharing with host and/or other guests
  - Access to transitive mount on host to traditional distributed or parallel file system
  - Use as an alternative to virtual disk for root file system so that content can be managed more effectively on the host
  - Use as a front-end to a file-system based content addressable storage system (such as Venti)

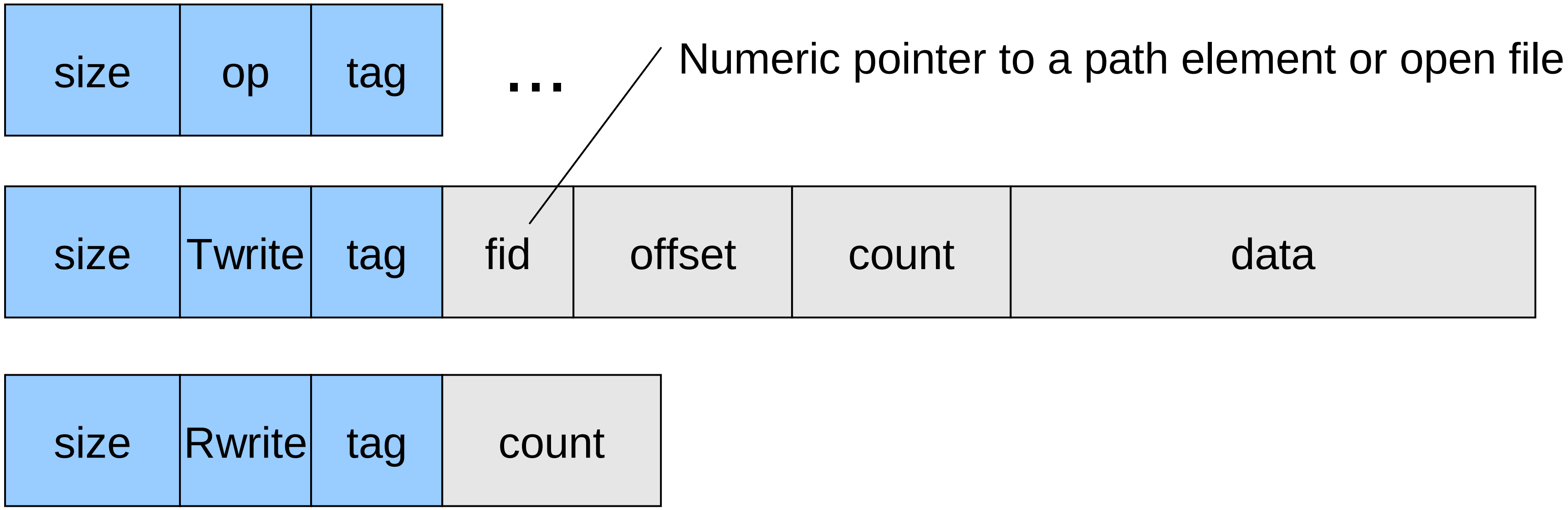
9P Basics: Overview

- Pure request/response RPC model
- Transport Independent
  - only requires reliable, in order delivery mechanism
  - can be secured with authentication, encryption, & digesting
- By default, requests are synchronous in nature avoiding coherence problems and race conditions
- Design stresses keeping things simple resulting in small and efficient client and servers

9P in the Linux Kernel

- Since 2.6.14
- Small Client Code Base
  - fs/9p: VFS Interface ~1500 lines of code
  - net/9p
    - Core: Protocol Handling ~2500 lines of code
    - FD Transport (sockets, etc.): ~1100 lines of code
    - Virtio Transport: ~300 lines of code
- Small Server Code Base
  - Spfs (standard userspace server): ~7500 lines of code
  - Current KVM-qemu patch: ~1500 lines

9P Basics: Protocol Overview



size	op	tag	…

Numeric pointer to a path element or open file

size	Twrite	tag	fid	offset	count	data

size	Rwrite	tag	count

Protocol Specification Available: http://v9fs.sf.net/rfc

9P Basics: Operations

- Metadata Management
  - Stat: retrieve file metadata
  - Wstat: write file metadata
- File I/O
  - Create: atomic create/open
  - Open, Read, Write, Close
  - Directory read packaged w/read operation (Reads stat information with file list)
  - Remove
- Session Management
  - Version: protocol version and capabilities negotiation
  - Attach: user identification and session option negotiation
  - Auth: user authentication enablement
  - Walk: hierarchy traversal and transaction management
- Error Management
  - Error: a pending request triggered an error
  - Flush: cancel a pending request

9P Basics: Unix Extensions

- Existing Support:
  - UID/GID support
  - Error ID support
  - Stat mapping
  - Permissions mapping
  - Symbolic and Hard Links
  - Device Files
- Limitations which need to be overcome
  - File locking
  - Extended Attributes
  - Writable mmap
  - ioctl(?)

9P/Virtio: Overview

- New transport module built for 9P which uses virtio
- 9P packet buffers are marshaled into 4k chunks and shoved into ring buffers
- Client can handle multiple outstanding transactions, but server is currently single threaded
- Server is implemented within kvm-qemu (thanks to Anthony) and handles service of requests.
- Multiple channels supported, but currently no way of specifying which channel you want
- Lguest virtio also supported, but lguest server gateways packets from virtio to tcp/ip connection to server

9P/Virtio: Basic Usage

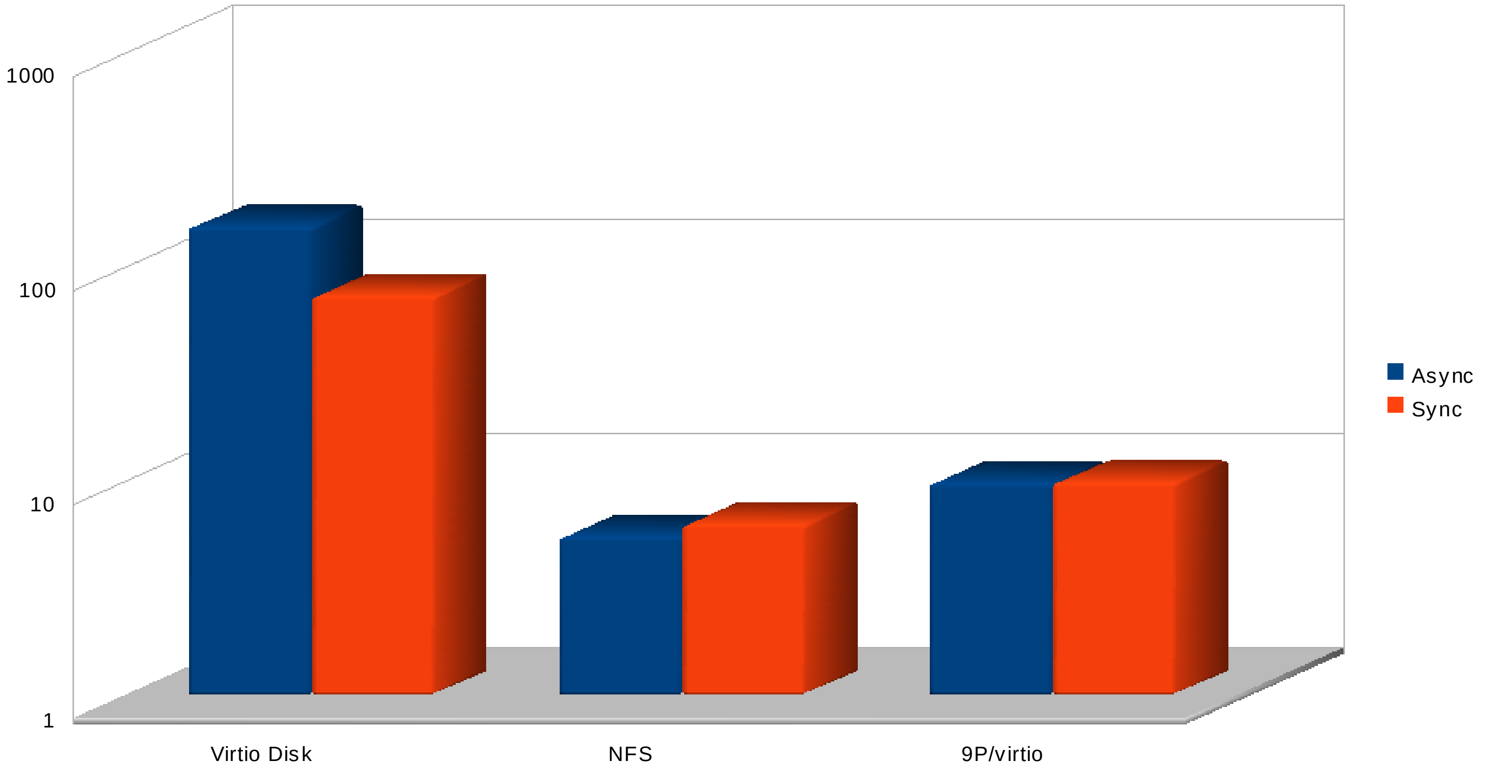
- Make sure your guest kernel has:
  - 9p, 9pnet and 9pnet_virtio kernel modules
- Patch kvm-qemu w/Anthony's patch
- Use -share argument to specify directory hierarchy to export when starting kvm-qemu
- Once guest is started:
  - mount -t 9p nodev /mnt -o trans=virtio
- Another Option:
  - Use 9P/virtio to provide root file system

Preliminary Performance

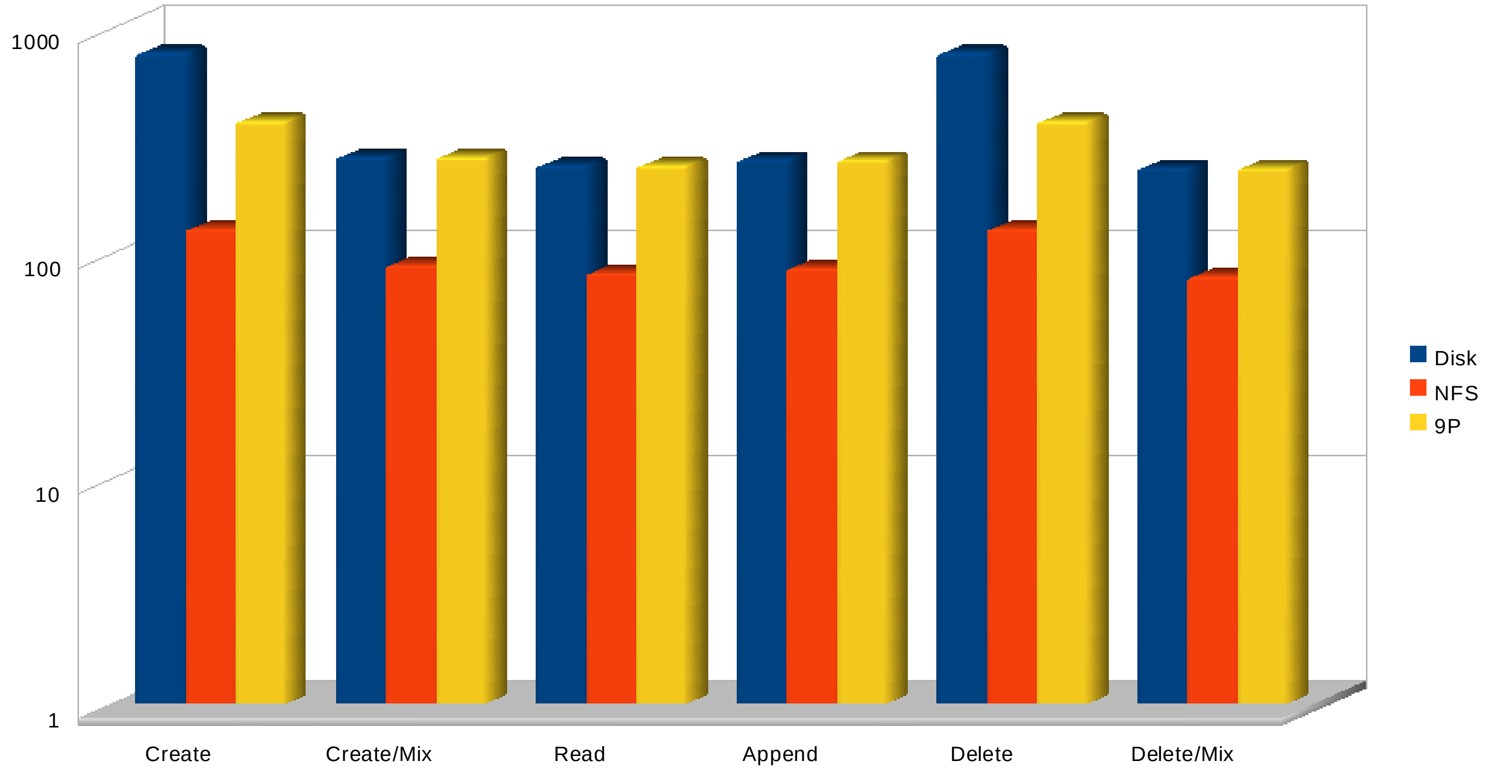
- Running on my Thinkpad T60p
  - 2.16 GHz Core Duo
- Host Kernel: Ubuntu 2.6.24-18-generic
- KVM Userspace: kvm-69
- Guest Kernel: Ubuntu 2.6.24-18-generic w/patched virtio drivers
- KVM Initialized w/1 CPU and 128MB Memory

```
qemu-system-x86_64 -share / -drive "file=/images/kvm-
  8.04.img,if=virtio,boot=on" -append "console=ttyS0" -serial stdio -
  nographic -net nic,module=virtio -net tap,script=/etc/kvm/qemu-ifup
```

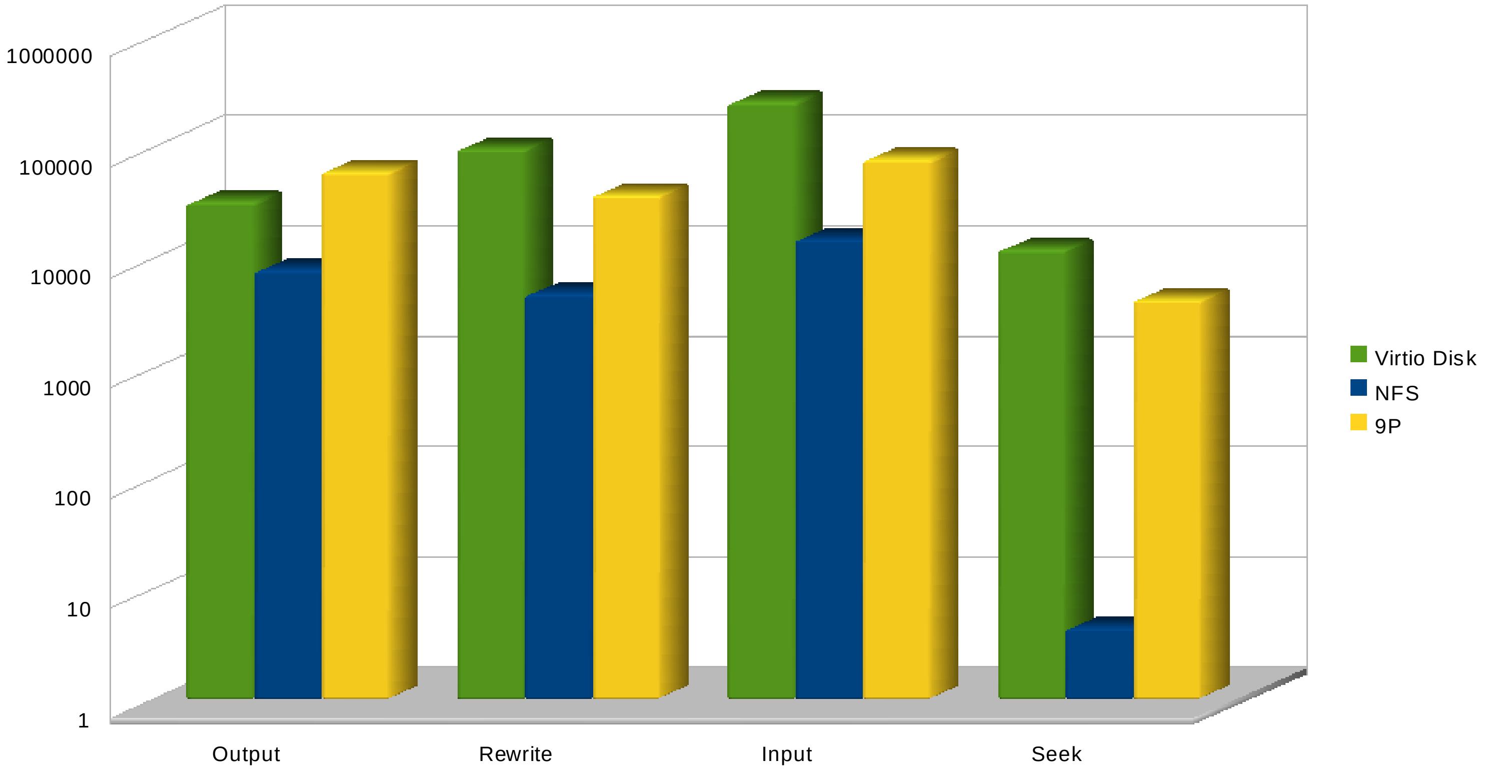
Preliminary Performance: Dbench



Preliminary Performance: Postmark



Preliminary Performance: Bonnie (256MB)



Future Work: Optimization

- Existing KVM server solution is synchronous
  - Use thread pools
  - Use async i/o
  - Use some combination of both
- More aggressive zero-copy support (2.6.27 merge window)
  - Current zero-copy only when cache is enabled
  - No write-cache support, so writes aren't zero-copy
- In-kernel server (2.6.28 merge window)
  - Should allow more aggressive zero-copy on the server side
  - Opens up potential for tricks such as guests sharing the server's page cache

Future Work: Cache Mechanisms

- Linux/9P currently supports a loose read cache
  - No read-based coherence, any write-through invalidates the entire file's cache
- Write-cache alternatives (2.6.27 merge window)
  - Support write-back as well as write-through caching
- Lease based coherent caching part of a parallel project road map

Future Work: .L extension series

- The 9P protocol is a network mapping of the Plan 9 file system API
- Many mismatches with Linux/POSIX
- Existing .U extension model is clunky
- Developing a more direct mapping to Linux VFS
  - New opcodes which match VFS API
  - Linux native data formats (stat, permissions, etc.)
  - Direct support of extended attributes, locking, etc.
- Should be able to co-exist with legacy 9P and 9P2000.u protocols and servers.

Future Work

- Support multiple 9P connections with string-identifier based lookup
- Support guest to guest direct networking
- Use 9P as transport for other devices (block, network, audio, graphics)
- Native UID/GID mapping mechanisms
- Better packaging for server and utilities
- Kernel based built-in extensions
  - Composable exportable name space
  - Copy-on-write support & snapshot support
- Bridge based server support
  - Allow gateway of 9P from virtio to network based server

Related Work

- Power Virtualization 9P Support
- Xen 9P support (currently orphaned)
- Lguest 9P support (virtio gateway to spfs)
- Envoy: Hierarchical Cache for Xen (Cambridge)
- Kvmfs: synthetic file system to control kvm (LANL)
- Plan 9 Kernel KVM and Lguest Support (Sandia)
- Foundation: Venti for storage content-addressable backend for vmware (MIT)

Thanks

- Virtio development now part of mainline v9fs development repository
  - kernel.org: /pub/scm/linux/kernel/git/ericvh/v9fs.git#v9fs-devel
- Lguest and KVM server patches have been posted to mailing list
  - http://www.kernel.org/~ericvh/virtio
- Thanks to Anthony Ligouri for providing KVM server
- Thanks to Lucho Ionkov (LANL) for contributions to 9P Linux client, providing the server infrastructure, and early work on PCI-based 9P client for KVM

Backup Slides

9P Client/Server Support

- Comprehensive list: http://9p.cat-v.org/implementations
- C, C#, Python, Ruby, Java, Python, TCL, Limbo, Lisp, OCAML, Scheme, PHP and Javascript
- FUSE Clients (for Linux, BSD, and Mac)
- Native Kernel Support for OpenBSD
- Windows support via Rangboom proprietary client

9P Packet Trace

```
<<< (0x8055650) Tattach tag 0 fid 2 afid -1 uname  aname  nuname 266594
>>> (0x8055650) Rattach tag 0 qid  (0000000000000002 48513969 'd')
<<< (0x8055650) Twalk tag 0 fid 1 newfid 3 nwname 1 'test'
>>> (0x8055650) Rwalk tag 0 nwqid 1 (000000000000401a 48613b9d 'd')
<<< (0x8055650) Tstat tag 0 fid 3
>>> (0x8055650) Rstat tag 0 'test' 'ericvh' 'root' '' q  (000000000000401a 48513b9d 'd') m d777 at 1213278479 mt 1213283229 l 0 t
    0 d 0 ext ''
<<< (0x8055650) Twalk tag 0 fid 3 newfid 4 nwname 1 'hello.txt'
>>> (0x8055650) Rwalk tag 0 nwqid 1 (000000000000401b 4851379d '')
<<< (0x8055650) Tstat tag 0 fid 4
>>> (0x8055650) Rstat tag 0 'hello.txt' 'ericvh' 'ericvh' '' q  (000000000000401b 4851379d '') m 644 at 1213283229 mt 1213283229 l
    12 t 0 d 0 ext ''
<<< (0x8055650) Twalk tag 0 fid 4 newfid 5 nwname 0
>>> (0x8055650) Rwalk tag 0 nwqid 0
<<< (0x8055650) Topen tag 0 fid 5 mode 0
>>> (0x8055650) Ropen tag 0 (000000000000401b 4851379d '') iounit 0
<<< (0x8055650) Tstat tag 0 fid 4
>>> (0x8055650) Rstat tag 0 'hello.txt' 'ericvh' 'ericvh' '' q  (000000000000401b 4851379d '') m 644 at 1213283229 mt 1213283229 l
    12 t 0 d 0 ext ''
<<< (0x8055650) Tread tag 0 fid 5 offset 0 count 8192
>>> (0x8055650) Rread tag 0 count 12 data 68656c6c 6f20776f 726c640a

<<< (0x8055650) Tread tag 0 fid 5 offset 12 count 8192
>>> (0x8055650) Rread tag 0 count 0 data

<<< (0x8055650) Tclunk tag 0 fid 5
>>> (0x8055650) Rclunk tag 0
<<< (0x8055650) Tclunk tag 0 fid 4
>>> (0x8055650) Rclunk tag 0
<<< (0x8055650) Tclunk tag 0 fid 3
>>> (0x8055650) Rclunk tag 0
```