VirtFS

A virtualization aware File System pass-through

Venkateswararao Jujjuri (JV)
jvrao@us.ibm.com

Linux Plumbers Conference 2010

Outline

- VirtFS Overview
- Why not traditional n/w FileSystems?
- Use Cases
- Architecture
- Progress
- Performance
- Short/Long Term Challenges/Vision
- Conclusions.

Overview – What is it?

- Moves up the virtualization beyond the physical layer.
- File system pass-through between the KVM host and guest.
- Uses 9P Protocol between Client and Server.
  - New extension 9P2000.L is being defined in addition to existing 9P2000.u and 9P2000.
- Server is part of QEMU, uses VirtIO transport.
- Client is part of the Kernel.
- Server runs on KVM hypervisor and client mounts exported File system on the guest.
- Virtualization intelligence at system services layer.

Overview – What can it offer?

- Guest access to Host File systems at close to native performance.
- Secure window of host file system on the guest.
- Multi-tenancy.
- Knowledge about the guest activity enables hypervisor to offer variety of services.
- Further exports on guest through NFS/CIFS.
- Better utilization of system resources.

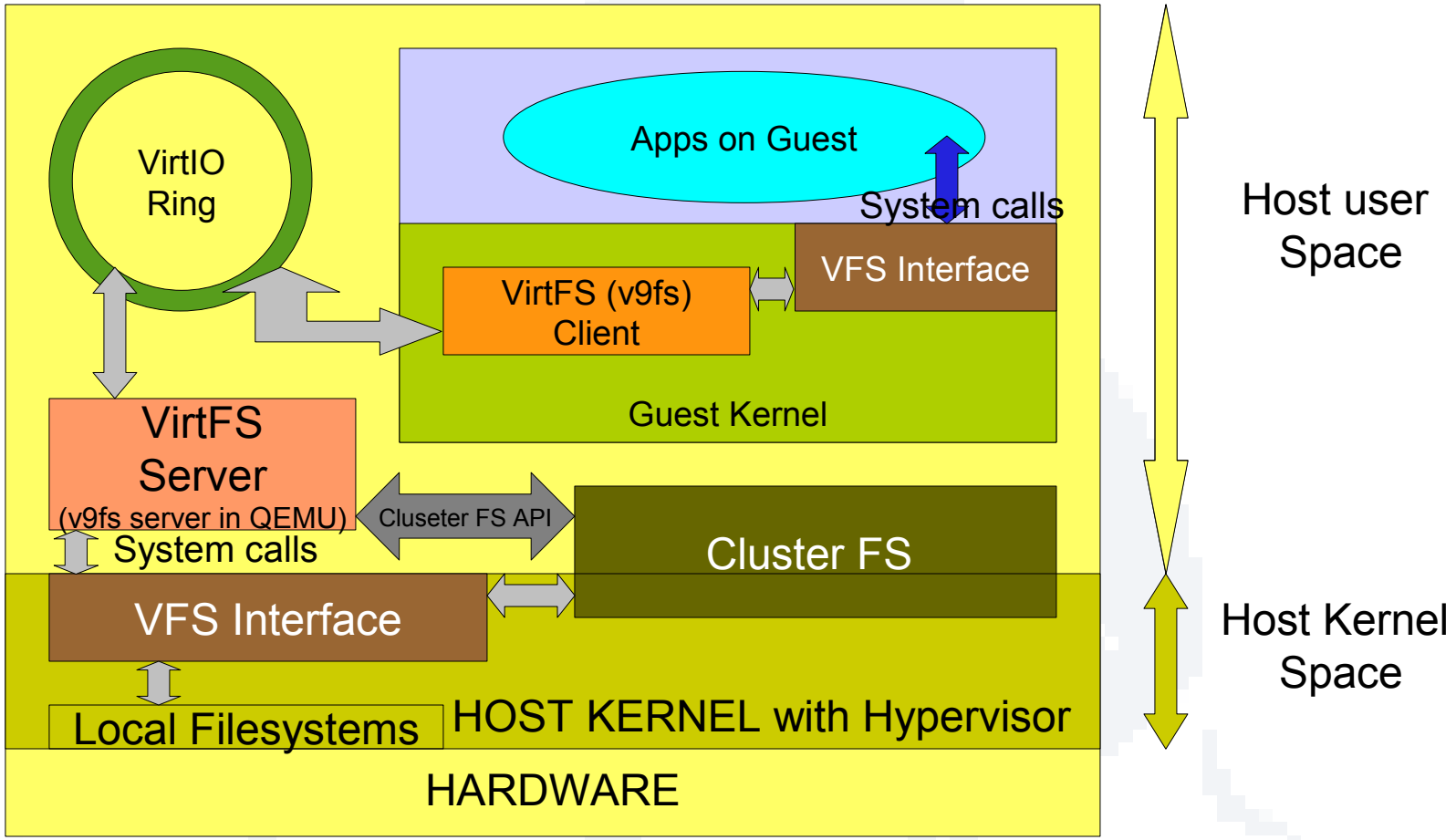
Why not traditional N/W File systems?

- Configuration, management and encapsulation overheads.
- Double caching.
- Different semantics for different File Systems.
- Security - Non IP based transport.
- Most Important: Lacks the knowledge that both are running on the same hardware.

Use Cases

- Replace virtual disk as the root filesystem.
  - Rapid cloning, Easy management, secure.
- Securely sharing parts of the host File Systems(Local FS or Cluster FS) to guests.
- Sharing guest File System to hosts. (Synthetic FS like /proc, /sys etc)
- Cloud computing
  - Secure window of host file system on the guest.
  - Different portions of the same file system shared among different guests.
  - Knowledge about the guest activity enables hypervisor to offer services like de-dup, snapshots etc.
- Better utilization of system resources.

VirtFS Internals



VirtFS Implementation

- KVM, QEMU, and VirtIO presents an ideal platform for the VirtFS server.
- Two types of virtual devices
  - virtio-9p-pci, used to transport protocol messages and data between host and the guest.
  - Fsdev, used to define the exported file system characteristics like fs type and security model etc.
- Command line options
  - *-fsdev local,id=myid,path=/share_path/,security_model=mapped -device virtio-9p-pci,fsdev=myid,mount_tag=v_tag*
  - *-virtfs local,path=/share_path/,security_model=passthrough,mnt_tag=v_tag*
  - On Client: *mount -t 9p -o trans=virtio -o version=9p2000.L v_tag /mnt*

Security

- Three models of security enforcement
- One with complete isolation of guest user domain from that of the host.
  - Eliminates the need for root squashing
  - No setuid/setgid exposures.
  - Complete isolation enhances security.
- Second shares user domains between host and guest.
  - Follows traditional network file system model.
  - If not careful, it is susceptible to security holes.
- Third is a small variation to the second model.
- Client based security enforcement.
- Server makes sure that the client control never crosses the exported portion.

Security Model - Mapped

- VirtFS server intercepts and maps all file object create and get/set attribute requests from client.
- Files are created with VirtFS server's user credentials.
- Client user credentials are stored in extended attributes.
- Extended user attributes are allowed for regular files and directories only.
- For special files, corresponding regular files are created on file server and appropriate mode bits are added to extended attributes.
- This enhances security.
  - Guest user domain is completely isolated from host's.
  - Symlinks can't be followed locally on the file server.
- File system will be VirtFS'ized.

Security Model – Mapped (Cont...)

- On Host (ls -l output)

drwx------. 2 virfsuid virtfsgid 4096 2010-05-11 09:19 adir
-rw-------. 1 virfsuid virtfsgid 0 2010-05-11 09:36 afifo
-rw-------. 2 virfsuid virtfsgid 0 2010-05-11 09:19 afile
-rw-------. 2 virfsuid virtfsgid 0 2010-05-11 09:19 alink
-rw-------. 1 virfsuid virtfsgid 0 2010-05-11 09:57 asocket1
-rw-------. 1 virfsuid virtfsgid 0 2010-05-11 09:32 blkdev
-rw-------. 1 virfsuid virtfsgid 0 2010-05-11 09:33 chardev
-rw-------. 1 virtfsuid virtfsgid 6 2010-05-11 09:20 asymlink

- On Guest (ls -l output)

drwxr-xr-x 2 guid ggid 4096 2010-05-11 12:19 adir
prw-r--r-- 1 guid ggid 0 2010-05-11 12:36 afifo
-rw-r--r-- 2 guid ggid 0 2010-05-11 12:19 afile
-rw-r--r-- 2 guid ggid 0 2010-05-11 12:19 alink
srwxr-xr-x 1 guid ggid 0 2010-05-11 12:57 asocket1
brw-r--r-- 1 guid ggid 0, 0 2010-05-11 12:32 blkdev
crw-r--r-- 1 guid ggid 4, 5 2010-05-11 12:33 chardev
lrwxrwxrwx 1 guid ggid 6 2010-05-11 12:20 asymlink -> afile

Security Model – Passthrough

- All the requests are passed directly to underlying file system without any interception.
- File system objects on the fileserver will be created with client-user's credentials.
- Two ways to achieve this:
  - setuid/setgid during the creation.
  - chmod/chown immediately after creation.
- All special files are created as-is.
- Portable between NFS/CIFS.
- Security model similar to NFSv3

Security Model – Passthrough (Cont...)

- On Host

```
# grep 611 /etc/passwd
hostuser:x:611:611::/home/hostuser:/bin/bash
# ls -l
-rwxrwxrwx. 2 hostuser hostuser 0 2010-05-12 18:14 file1
-rwxrwxrwx. 2 hostuser hostuser 0 2010-05-12 18:14 link1
srwxrwxr-x. 1 hostuser hostuser 0 2010-05-12 18:27 mysock
lrwxrwxrwx. 1 hostuser hostuser 5 2010-05-12 18:25 symlink1 -> file1
```

- On Guest

```
$ grep 611 /etc/passwd
guestuser:x:611:611::/home/guestuser:/bin/bash
$ ls -l
-rwxrwxrwx 2 guestuser guestuser 0 2010-05-12 21:14 file1
-rwxrwxrwx 2 guestuser guestuser 0 2010-05-12 21:14 link1
srwxrwxr-x 1 guestuser guestuser 0 2010-05-12 21:27 mysock
lrwxrwxrwx 1 guestuser guestuser 5 2010-05-12 21:25 symlink1 ->file1
```

Security Model – None

- Slight variation of 'passthrough' model.
- Best effort attempt to set client user's credentials on the fileserver.
- No need for QEMU to run as root.
- Useful in desktop environments.

Symlink security hole - TOCTTOU

- Security whole if symlinks can be followed on host.
  - TOCTTOU exposes window of vulnerability.
  - Only in passthrough model. Mapped model is immune.
- Use chroot functionality to address this.
- Patch is ready. Will hit the mailing list soon.

Scalability work - Threadlets

- Introduced treadlet infrastructure in QEMU
- A reserved pool of worker threads.
  - Min long living; upto max limit for short living threads.
- Only syscalls(blocking) are offloaded to threadlets.
- VCPU threads are left to do what they are intended for.
  - No need for additional locking.
- Drastically reduced systime from guest's perspective.
- Improved scalability.

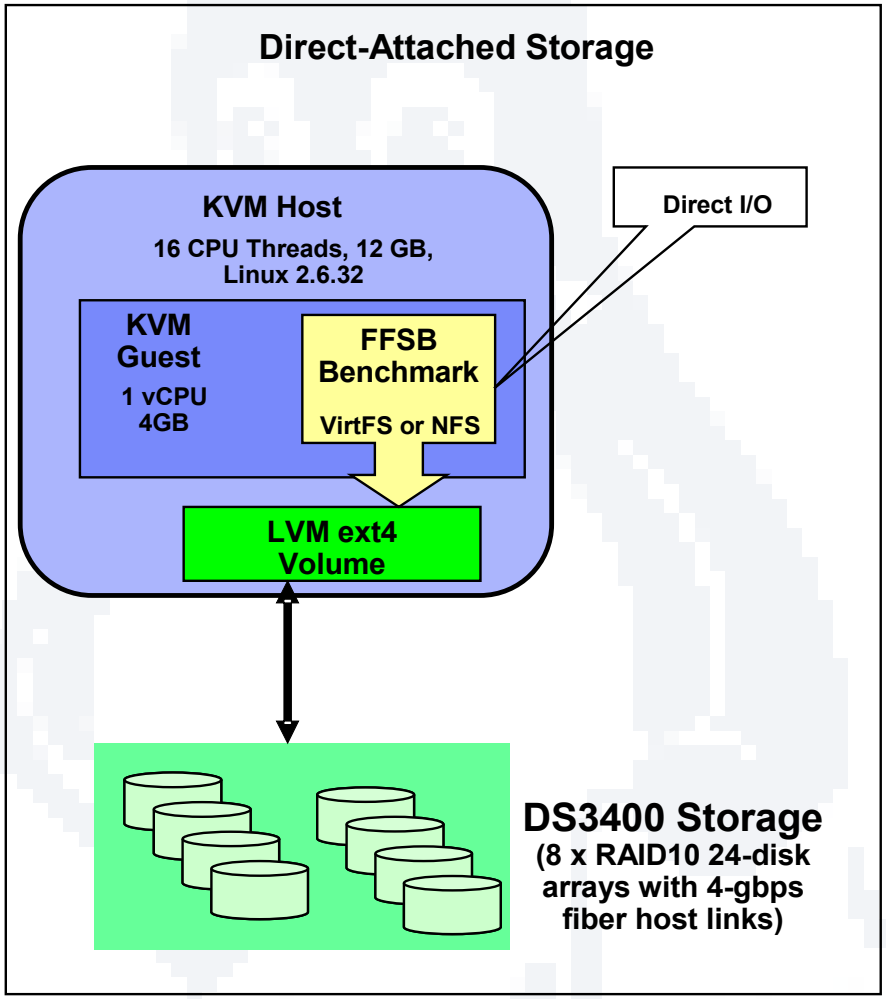
Additional Functionality

- Byte Range locking
  - POSIX lock semantics pose challenge to user-level fileservers.
    - Locks are bound to processes, not file descriptors.
    - locks are automatically released if a process calls close() on any of its open file descriptors for that file.
  - Implemented byte-range locks at QEMU/Fileserver.
  - Works fine if the FS is exported to only to one client.
- ACLs
  - Added POSIX Locks.
  - In the case of mapped security model “user.virtfs.” prefixed to the default and access ACLs.

Zero Copy Transport

- No copies between guest kernel and QEMU.
- Guest user buffer is GUP'd onto virtio ring.
- VirtFS server directly reads/writes into these pages.
- Enhances performance
- Patches on the mailing list not merged into upstream.
- Challenges
  - GUP logic should be moved into transport layer.
  - Transport layer doesn't have protocol knowledge.
  - Only msize is negotiated between client and server. This poses challenge for different size messages.
  - Working out a design.

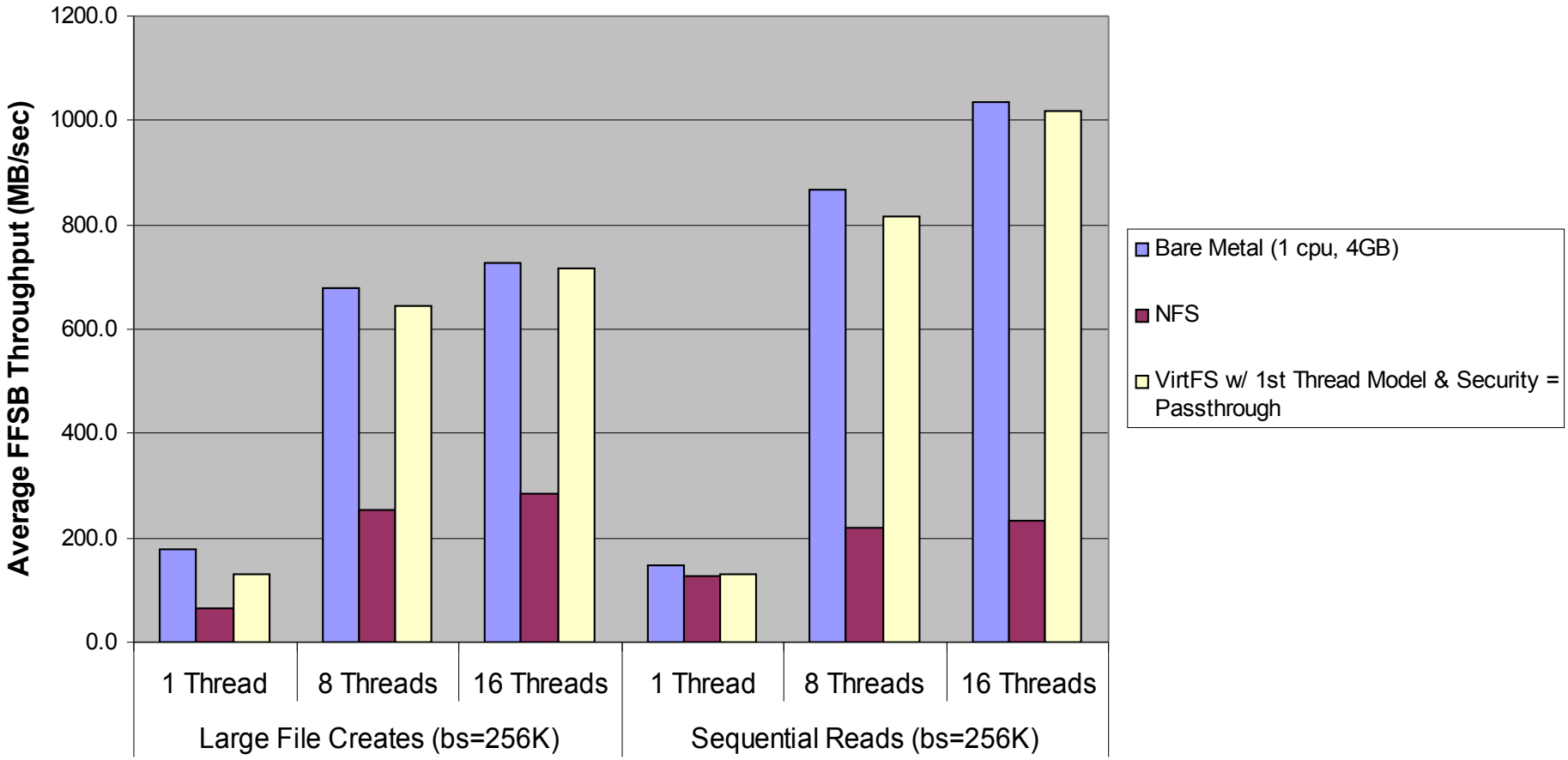
Performance Test Setup



- KVM Host: x3650 M2 (8 x E5530 @ 2.40GHz, 16 CPU Threads (8 cores), 12 GB memory, Chelsio 10-GbE, Broadcom 1-GbE)

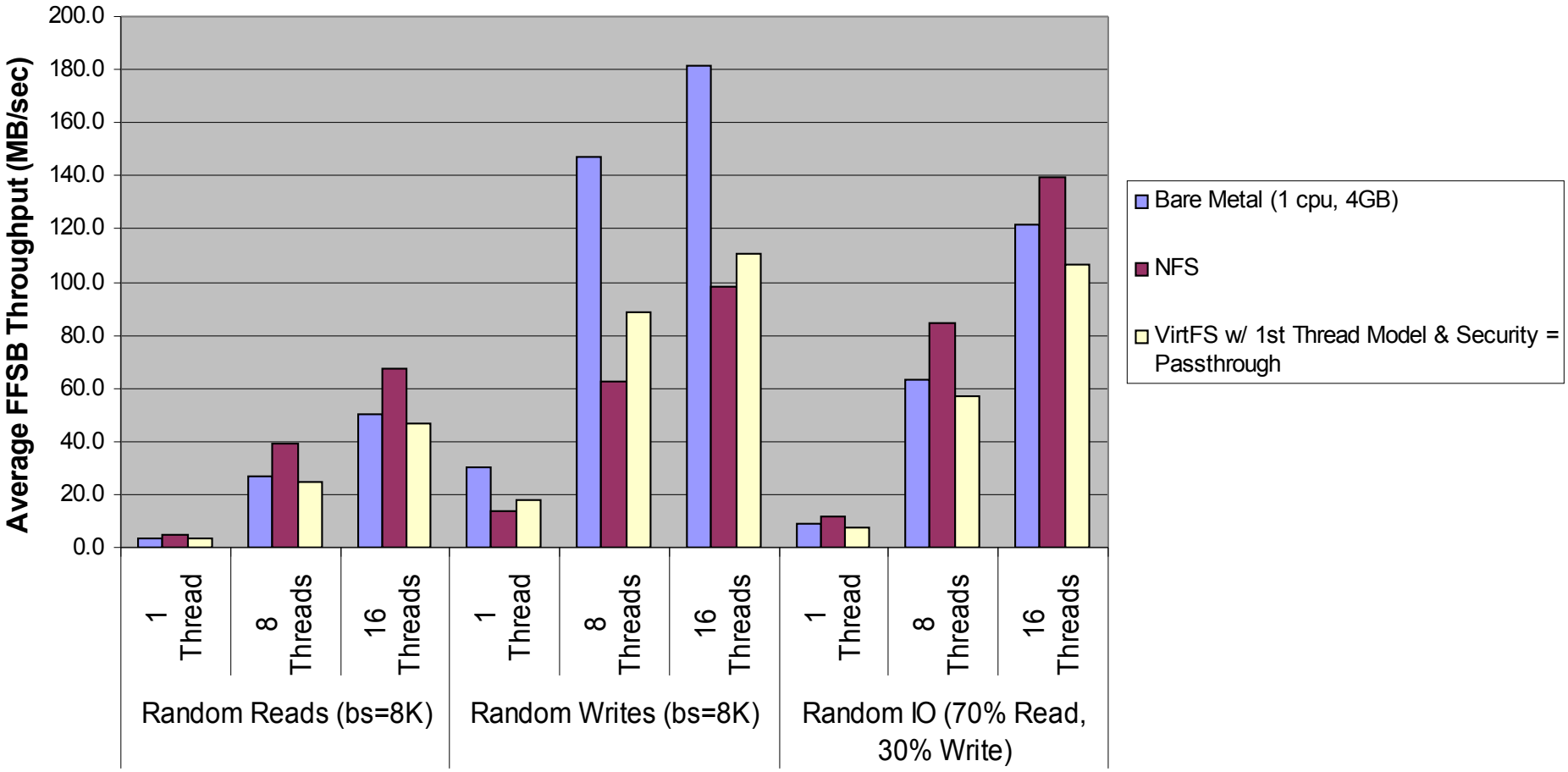
VirtFS Preliminary Performance - Data Streaming

KVM Guest = 1 vcpu, 4GB; Host = 16 cpus, 12GB; Linux 2.6.32; Direct I/O



VirtFS Preliminary Performance - Random I/O Workloads

KVM Guest = 1 vcpu, 4GB; Host = 16 cpus, 12GB; Linux 2.6.32; Direct I/O



Short Term Goals/Challenges

- Fully functional and stable 9P2000.L protocol.
- Writable mmaps (only on one guest)
- Enterprise level applications on VirtFS. Ex:DB2, WAS, Oracle, etc.
- Inclusion into distros.
- Community involvement and participation.
- Threadlets into upstream.
- Zero Copy patches into mainline.
- VirtFS as root filesystem.
- Exporting out VirtFS mount through NFS/CIFS.
- Ongoing stability and scalability and performance improvements.

Mid to Long Term Goals

- Sharing byte-range locks across multiple exports through central database (ex. CTDB).
- Distributive Cache Consistency.
  - A filesystem can be exported out of different fileservers and achieve NFS level of consistency.
- Performance improvements to bring VirtFS close to bare-metal.
  - Page Cache sharing between host and guest(s)
  - dcache sharing between host and guests(s)
- Interfacing with 3rd party filesystem APIs.
- Rich ACL support.

Conclusions

- Huge Potential for specialized filesystems in the virtualization space.
- Growth in the cloud space will be a major catalyst.
- Lot of scope for innovation.
- A step towards paravirtual system services.

References

- VirtFS Setup
  - http://wiki.qemu.org/Documentation/9psetup
- OLS 2010 Paper
  - http://www.sciweavers.org/publications/virtfs-virtualization-aware-file-system-pass-through
- USENIX paper on overview of 9P protocol:
  - http://www.usenix.org/events/usenix05/tech/freenix/hensbergen.html

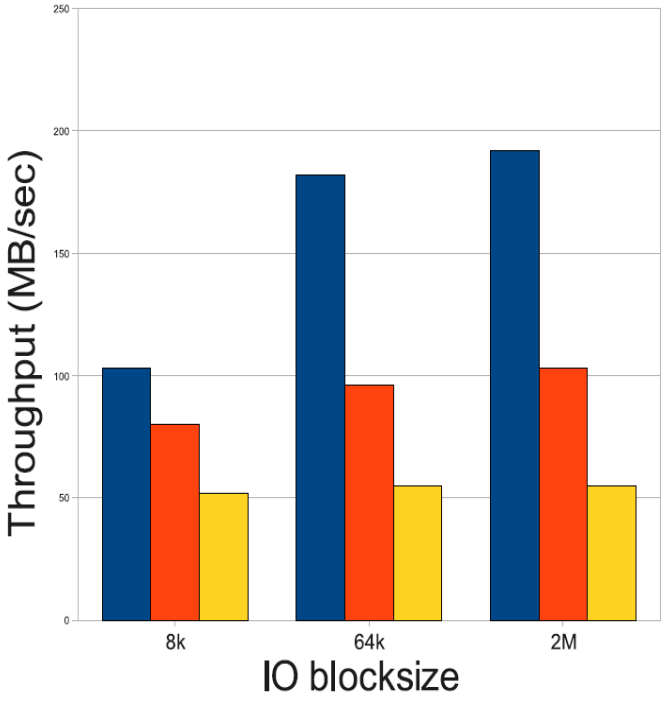
Questions?

Backup

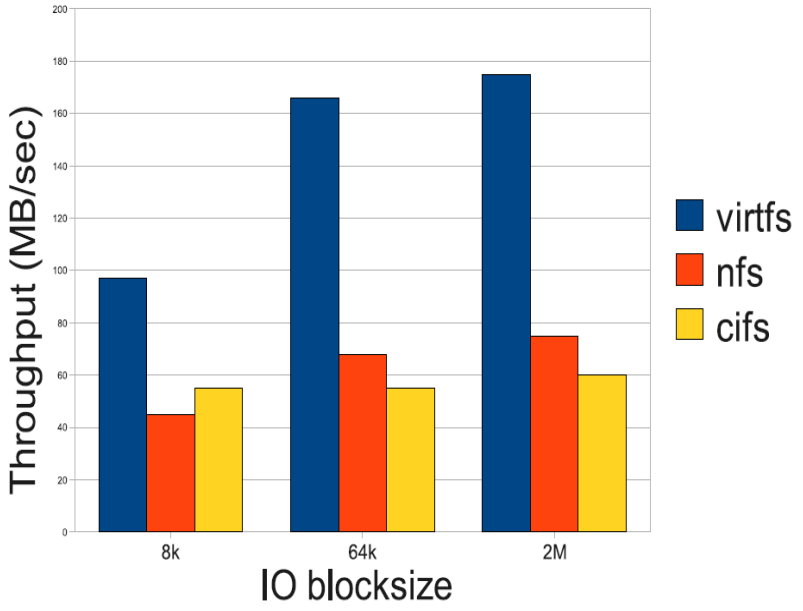
Performance Tests

- Plain & Simple; dd to compare.
- Write
  - dd if=/dev/zero of=/mnt/fileX bs=<blocksize> count=<count>
- Read
  - dd if=/mnt/fileX of=/dev/null bs=<blocksize> count=<count>
- blocksize - 8k, 64k, 2M.
- Count = Number of blocks to do **8GB** worth of IO
- All tests are conducted on Guest.

Comparison with NFS and CIFS

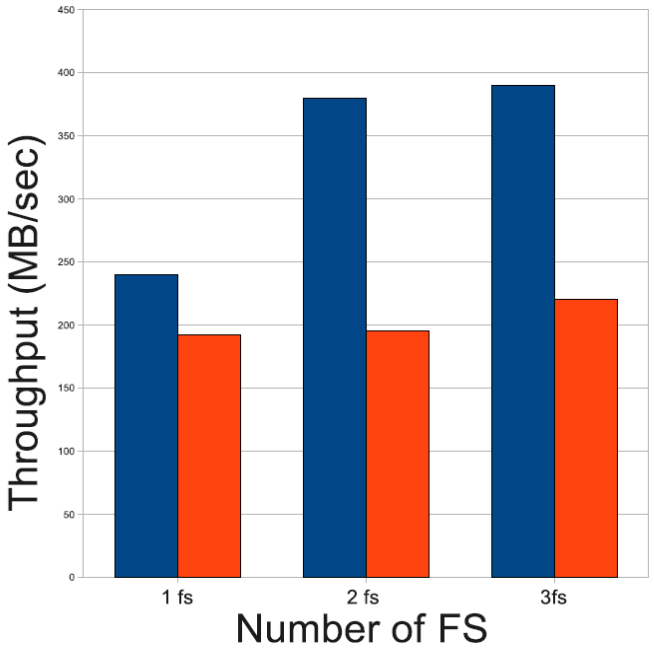


Sequential Read

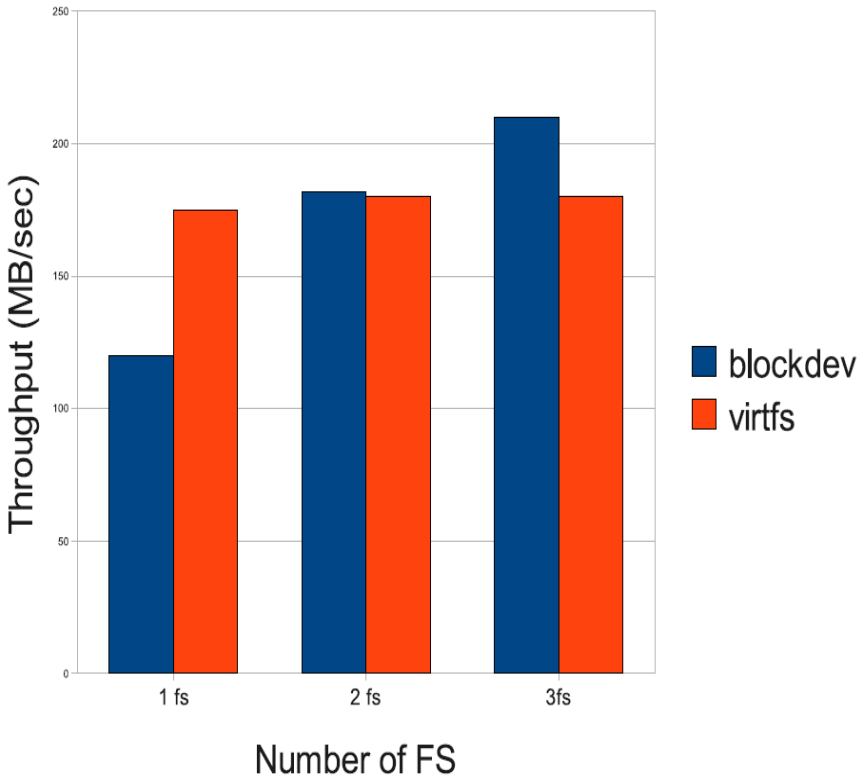


Sequential Write

Comparison with blockdev



Sequential Read



Sequential Write