JANUARY 10, 2014

Comparing Filesystem Performance in Virtual Machines

For years, the primary bottleneck for virtual machine based development environments with Vagrant has been filesystem performance. CPU differences are minimal and barely noticeable, and RAM only becomes an issue when many virtual machines are active[1]. I spent the better part of yesterday benchmarking and analyzing common filesystem mechanisms, and now share those results here with you.

I'll begin with an analysis of the results, because that is what is most interesting to most people. The exact method of testing, the software used, and the raw data from my results can be found below this analysis.

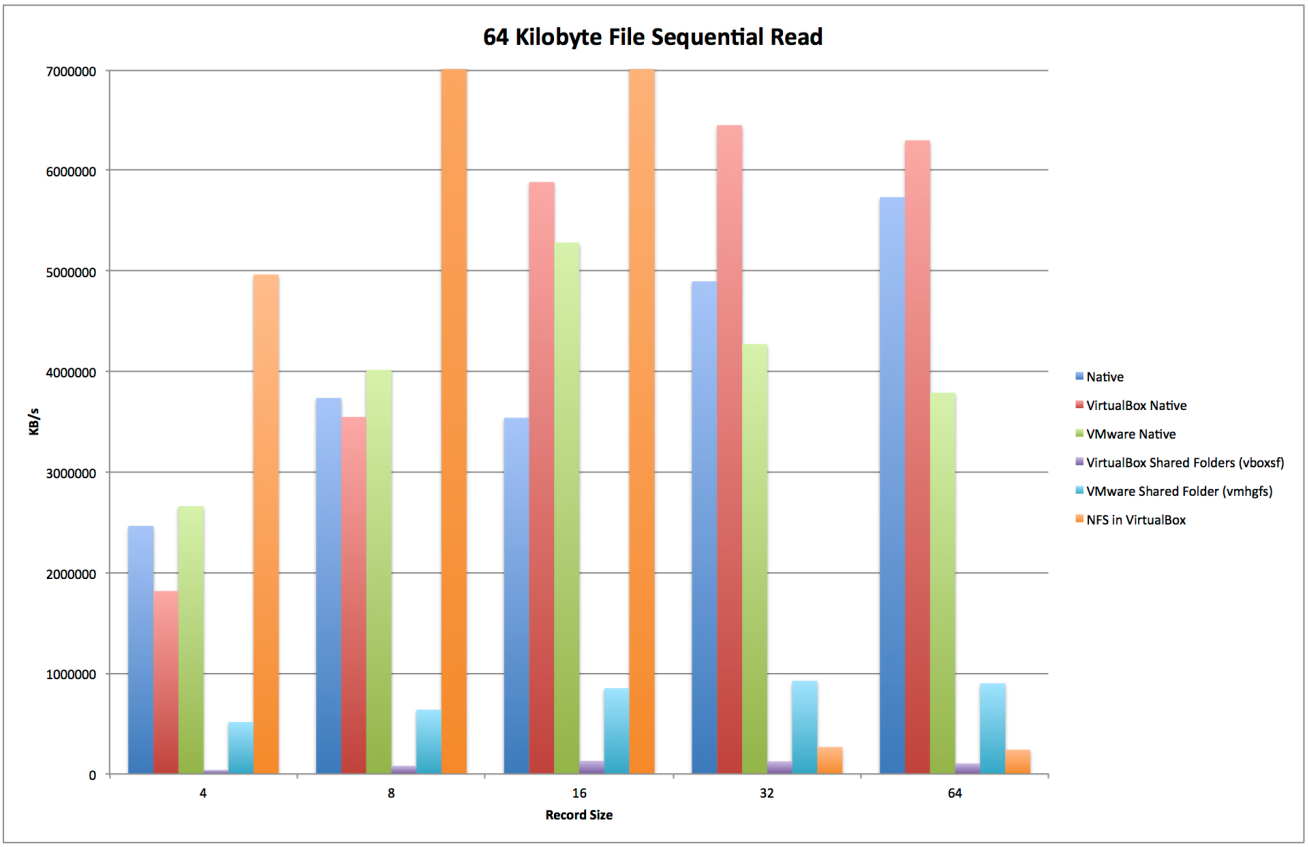
In every chart shown below, we test reading or writing a file in some way. The total size of the file being written is fixed for each graph. The Y axis is the throughput in KB/s. The X axis is "record size" or the size of the chunks of data that are being read/written at one time, in KB.

The different test environments are native, VirtualBox native, VMware native, VirtualBox shared folders (vboxsf), VMware shared folders (vmhgfs), and NFS. The "native" environments refer to using the filesystem on its own in that environment. "native" is on the host machine, "VirtualBox native" is on the root device within the VirtualBox virtual machine, etc. NFS was only tested on VirtualBox because the performance characteristics should be similar in both VirtualBox and VMware.

For every chart, higher throughput (Y axis) is better.

Small File Sequential Read

First up is a sequential read of a 64 KB file, tested over various read record sizes. A real world application of a small sequential read would be loading the source files of an application to run, compile, or test.



The first thing you can't help but notice is that the read performance of NFS is incredible for small record sizes. NFS is likely doing some heavy read-ahead operations and caching to get this kind of performance. I don't have a good theory on how NFS outperforms the native virtual filesystems.

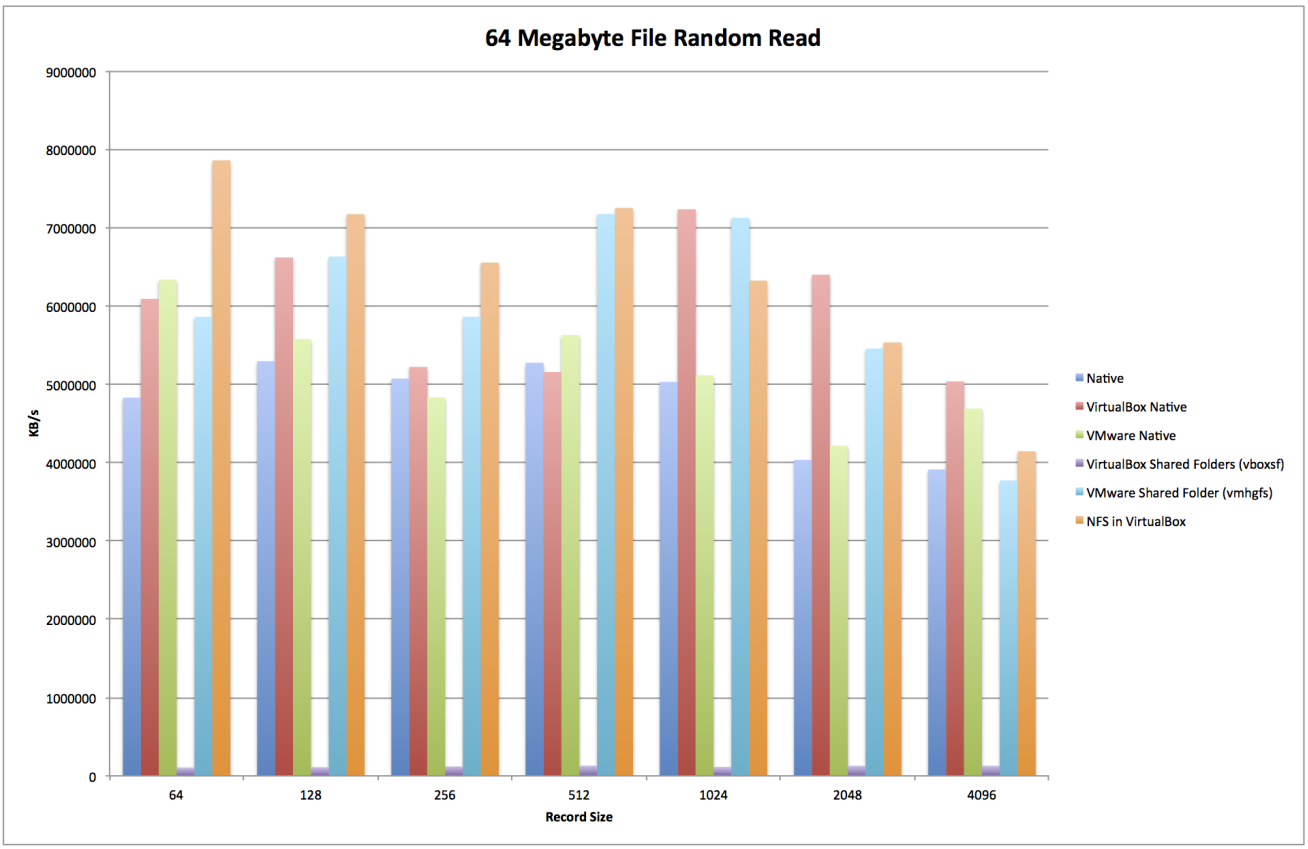
VMware shared folders just crush VirtualBox shared folders here. VirtualBox shared folder read performance is just abysmal. If you look at the raw data, you'll see the throughput actually never goes above 100 MB/s, while VMware never goes below 500 MB/s, and peaks at over 900 MB/s.

It is interesting that sometimes the native filesystem within the virtual machine outperforms the native filesystem on the host machine. This test uses raw `read` system calls with zero user-space buffering. It is very likely that the hypervisors do buffering for reads from their virtual machines, so they're seeing better performance from not context switching to the native kernel as much. This theory is further supported by looking at the raw result data for `fread` benchmarks. In those tests, the native filesystem beats the virtual filesystems every time.

Large File Random Read

This tested the throughput of randomly reading various parts of a 64 MB file, again tested with various read record sizes. This file is 1000x larger than in

our previous test. This sort of behavior might be seen when dealing with database reads that hit the filesystem.



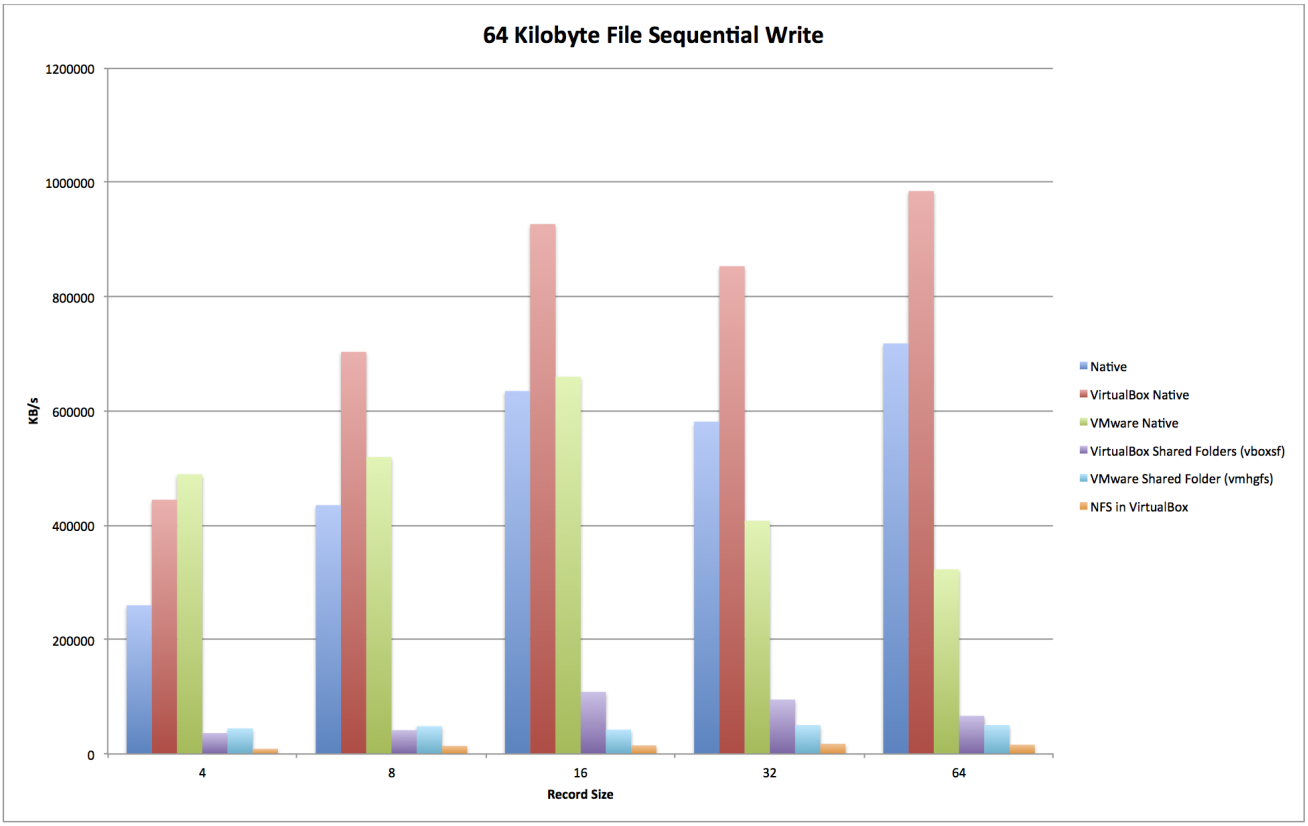
The gap between VMware shared folders and VirtualBox shared folders is widened considerably compared to small sequential reads. VirtualBox performs so badly you can barely see it. Again, VirtualBox throughput never peaks above 100 MB/s. VMware, on the other hand, is peaking at 7 GB/s. Because the deviation of the VirtualBox throughput is so small across various test cases, I theorize that there is a single hot path of code in the VirtualBox shared folder system limiting this. They're clearly doing something wrong.

NFS is less dominating, most likely because the read-ahead benefits are not seen in this test case. Still, NFS performs very well versus other options.

And, just as with the small sequential read, we're still seeing better performance within the virtual machine versus outside. Again, this can be attributed to the hypervisor doing clever buffering, whereas the raw syscalls to the host machine don't allow this to happen.

Small File Sequential Write

Let's look at the performance of sequential writes of small files. This most accurately describes storing session state, temporary files, or writing new source files.



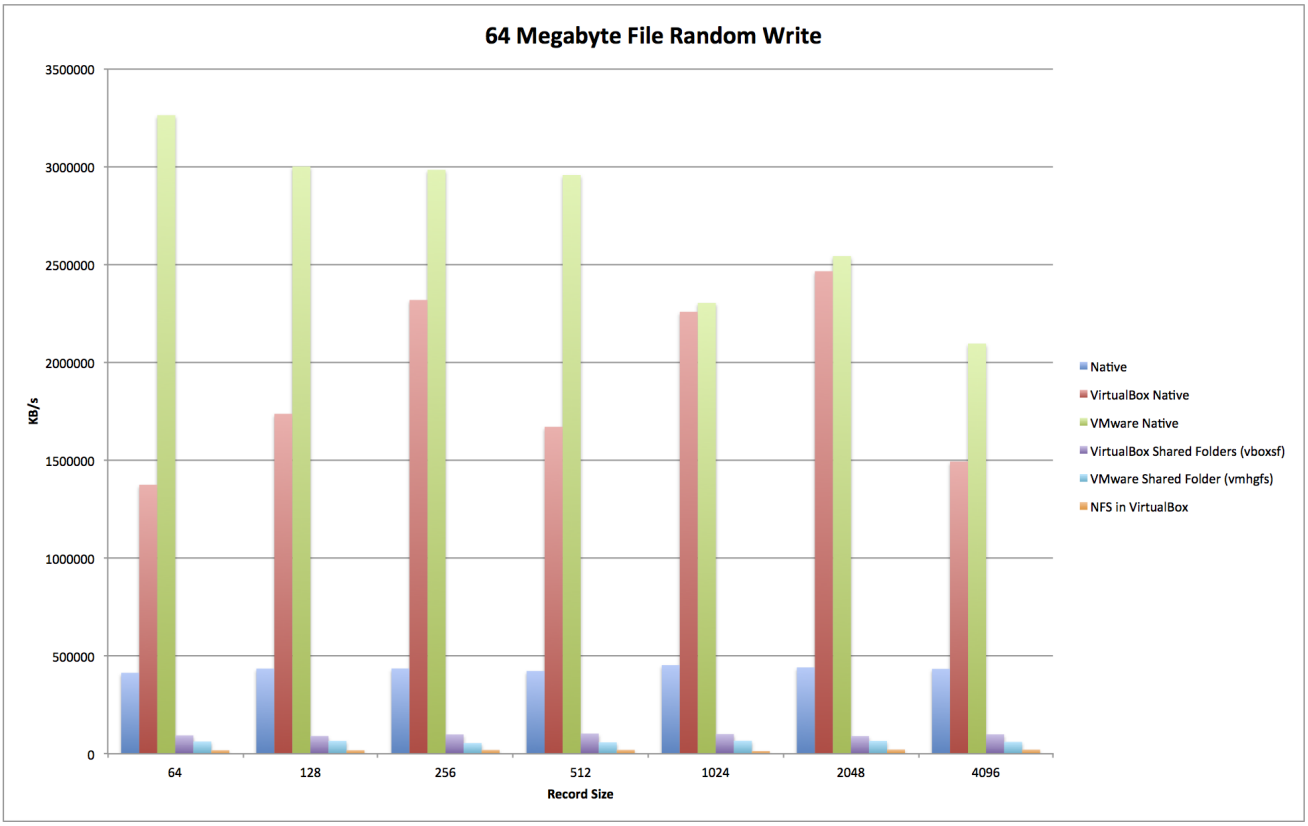
The first noticeable thing is that NFS performance is terrible for these kinds of writes. There is no real caching that NFS can do here, so you must pay the full penalty for network overhead, then writing to disk on the host side, and finally waiting for an ack again the VM that the write succeeded. Ouch.

The various "native" filesystems perform very well. Again, the virtual machines outperform the host. And again, this can be attributed to buffering in the hypervisors.

The shared filesystems are close, but VirtualBox clearly outperforms VMware in this test case.

Large File Random Write

The final chart we'll look at are the results of testing random writes to a large (64 MB) file. Just like our large random read test, this is a good test for how a database would perform.



There really isn't much of a difference here versus small sequential writes. Because it is a large file, the gaps between the different test environments is larger, but other than that, the results are mostly the same.

NFS continues to be terrible at writes. VirtualBox shared folders continue to outperform VMware on writes, and the hypervisors outperform the host machine.

Hypervisors outperforming the host machine is the most interesting to me. The results of this test clearly show that the hypervisors must be lying about synced writes for performance. This corroborates what I've seen with Packer as well, where if the virtual machine is not cleanly shut down, committed writes are lost. `fsync()` in a virtual machine does not mean that the data was written on the host, only that is is committed within the hypervisor.

Overall Analysis

With regards to shared filesystems, VMware has the behavior you want. Loading web pages, running test suites, and compiling software are all very read heavy. VMware shared folder read performance demolishes VirtualBox, while the write performance of VirtualBox shared folders is only marginally better than VMware.

If you have the option to use NFS, use it. Again, the read performance is far more valuable than the write performance.

Hypervisor read/write performance is fantastic (because they cheat). Thanks to this data, I'm definitely going to be focusing more on new synced folder implementations in Vagrant that use only the native filesystems (such as rsync, or using the host machine as an NFS client instead of a server).

More immediately applicable, however: if you use virtual machines for development, move database files out of the shared filesystems, if possible. You'll likely see huge performance benefits.

Finally, I don't think there are any huge surprises in these results. Vagrant has supported NFS synced folders since 2010 because we realized early on that performance in the shared folders was bad. But having some hard data to show different behaviors is nice to have, and provides some interesting insight into what each system might be doing.

Test Software, Configuration, and Raw Data

The host machine is a 2012 Retina MacBook Pro with a 256 GB SSD running Mac OS X 10.9.1. Tests here were labelled **native**.

VirtualBox is version 4.3.4 running a virtual machine with Ubuntu 12.04 with the VirtualBox guest additions installed and enabled. The root device under test in VirtualBox is formatted with an ext3 filesystem. Tests on the root device were labelled **VirtualBox native** while tests on the shared folder system (vboxsf) were labelled **VirtualBox shared folders (vboxsf)**.

VMware Fusion is version 6.0.2 running a virtual machine with Ubuntu 12.04 with VMware Tools 5.0 installed and enabled. The root device under test in VMware is formatted with an ext3 filesystem. Tests on the root device were labelled **VMware native** while tests on the shared folder system (vmhgfs) were labelled **VMware shared folders (vmhgfs)**.

The NFS server used is the one built and shipped with OS X 10.9.1. The NFS client is what comes with the `nfs-common` package in Ubuntu 12.04. NFS

protocol 3 was used over UDP. Tests using NFS were labelled **NFS in VirtualBox**.

The benchmarking software used was the Iozone Filesystem Benchmark compiled from source for 64-bit Linux. The same binary was used in every instance except for the native test, which used a 32-bit OS X binary of Iozone compiled from source. The flags passed to Iozone were `-Racb`.

Unless otherwise noted, the remaining settings not mentioned here were defaults or not touched.

The raw data for the tests can be found in this Excel workbook. There are also more graphs for this data in this imgur album.

Footnotes:

[1]: This is in the case of a standard web development environment. They are typically neither CPU bound nor RAM-intensive.