Jeff Geerling

Skip to Navigation ↓

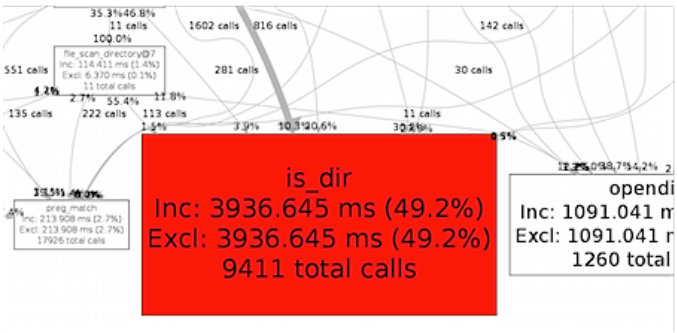
NFS, rsync, and shared folder performance in Vagrant VMs

November 24, 2014

It's been a well-known fact that using native VirtualBox or VMWare shared folders is a terrible idea if you're developing a Drupal site (or some other site that uses thousands of files in hundreds of folders). The most common recommendation is to switch to NFS for shared folders.

NFS shared folders are a decent solution, and using NFS does indeed speed up performance quite a bit (usually on the order of 20-50x for a file-heavy framework like Drupal!). However, it has it's downsides: it requires extra effort to get running on Windows, requires NFS support inside the VM (not all Vagrant base boxes provide support by default), and is not actually all that fast—in comparison to native filesystem performance.

I was developing a relatively large Drupal site lately, with over 200 modules enabled, meaning there were literally thousands of files and hundreds of directories that Drupal would end up scanning/including on *every page request*. For some reason, even simple pages like admin forms would take 2+ seconds to load, and digging into the situation with XHProf, I found a likely culprit:



There are a few ways to make this less painful when using NFS (since NFS incurs a slight overhead for every directory/file scan):

- Use APC and set stat=0 to prevent file lookups (this is a non-starter, since that would mean every time I save a file in development, I would need to restart Apache or manually flush the PHP APC cache).
- Increase PHP's `realpath_cache_size` ini variable, which defaults to '16K' (this has a small, but noticeable impact on performance).
- Micro-optimize the NFS mounts by basically setting them up on your own outside of Vagrant's shared folder configuration (another non-starter... and the performance gains would be almost negligible).

I wanted to benchmark NFS against rsync shared folders (which I've discussed elsewhere), to see how much of a difference using VirtualBox's native filesystem can make.

For testing, I used a Drupal site with about 200 modules, and used XHProf to measure the combined Excl. Wall Time for calls to `is_dir`, `readdir`, `opendir`, and `file_scan_directory`. Here are my results after 8 test runs on each:

NFS shared folder:

- 1.5s* (realpath_cache_size = 16K - PHP default)
- 1.0s (realpath_cache_size = 1024K)
- Average page load time: 1710ms (realpath_cache_size = 1024K, used admin/config/development/devel)

*Note: I had a two outliers on this test, where the time would go to as much as 6s, so I discarded those two results. But realize that, even though this NFS share is on a local/internal network, the fact that every file access goes through the full TCP stack of the guest VM, networking issues can make NFS performance unstable.

Native filesystem (using rsync shared folder):

- 0.15s (realpath_cache_size = 16K - PHP default)
- 0.1s (realpath_cache_size = 1024K)
- Average page load time: 900ms (realpath_cache_size = 1024K, used admin/config/development/devel)

Tuning PHPs `realpath_cache_size` makes a meaningful difference (though not too great), since the default 16K cache doesn't handle a large Drupal site very well.

As you can see, there's really no contest—just as NFS is an order of magnitude faster than standard VirtualBox shared folders, native filesystem performance is an order of magnitude faster than NFS. Overall site page load times for the Drupal site I was testing went from 5-10s to 1-3s by switching from NFS to rsync!

I've updated my Drupal Development VM and Acquia Cloud VM to use rsync shares by default (though you can still configure NFS or any other supported share type), and to use a `realpath_cache_size` of 1024K). Hopefully Drupal developers everywhere will save a few minutes a day from these changes :)

Note that other causes for abysmal filesystem performance and many calls to is_dir, opendir, etc. may include things like a missing module or major networking issues. Generally, when fixing performance issues, it's best to eliminate the obvious, and only start digging deeper (like this post) when you don't find an obvious problem.

Notes on using rsync shared folders

Besides the comprehensive rsync shared folder documentation in Vagrant's official docs, here are a few tips to help you get up and running with rsync shared folders:

- Use `rsync__args` to pass CLI options to `rsync`. The defaults are `["--verbose", "--archive", "--delete", "-z"]`, but if you want to preserve the files created within the shared folder on the guest, you can set this option, but without `--delete`.
- Use `rsync__exclude` to exclude directories like `.git` and other non-essential directories that are unneccessary for running your application within the VM. While not incredibly impactful, it could shave a couple seconds off the rsync process.

Not all is perfect; there are a few weaknesses in the rsync model as it is currently implemented out-of-the-box:

1. You have to either manually run `vagrant rsync` when you make a change (or have your IDE/editor run the command every time you save a file), or have `vagrant rsync-auto` running in the background while you work.
2. rsync is currently one-way only (though there's an issue to add two-way sync support).
3. Permissions can still be an issue, since permissions inside the VM sometimes require some trickery; read up on the `rsync__chown` option in the docs, and consider passing additional options to the `rsync__args` to manually configure permissions as you'd like.

Further reading

- rsync in Vagrant 1.5 improves file performance and Windows usage
- Vagrant - NFS shared folders for Mac/Linux hosts, Samba shares for Windows
- Vagrant web development - is VMware better than VirtualBox?

drupal | drupal planet | drupal 7 | php | performance | xhprof | nfs | vagrant | rsync | virtualbox | filesystem

Add new comment

Comments

David Lohmeyer – 2 years ago

Does this work on Windows? I seem to remember having issues trying to setup rsync with Vagrant on Windows, even with the Gnu rsync for windows installed.

reply

Jeff Geerling – 2 years ago

I've used this successfully with cygwin on Windows... don't know about other solutions though.

reply

David Corbacho – 2 years ago

I played once with Unison project (2-way sync) and it worked pretty well. But the process is brittle and requires constant attention. Finally I gave up, because it shaves only 1-3 seconds compared to NFS.
But maybe someone would like to continue researching this option. It's promising, and it will be even more important with D8

reply

fago – 2 years ago

Thanks, interesting findings.

I've been running into the same issue with slow NFS and came up with simple bash script (linux only) for auto-rsyncing changes, see http://drunomics.com/en/blog/syncd-sync-changes-vagrant-box. To avoid 2-way sync troubles I'm setting up shares via nfs for sites/default/files or private files directories, what allows stream-lined syncing / changing of files on the host.

I never really tried vagrant auto-rsync as I came up with it earlier and it just works now, but I'm wondering whether you are auto-rsync and it's working fine now given a considerable huge drupal site? I've read that there has been a performance issue (which I encountered testing guard/listen as well) - did you run into that? https://github.com/mitchellh/vagrant/issues/3249

reply

Jeff Geerling – 2 years ago

I do use rsync auto quite a bit, and it generally performs very well, once you give it 2-3 minutes to get the initial process going. I've left it running for full days, sometimes 2-3 days without issue (updating hundreds of files and doing git operations that change bunches of files as well).

Every now and then, the auto ruby process will hang and start taking up 60+% CPU, so I just ctrl-c it and start it over again.

reply

fago – 2 years ago

I see, thanks. 2-3min for startup seems to be a while though, not sure why it takes so long? The inotify-based watching is ready near instantly and works fine with switching Git branches etc. as well while watching multiple projects at the same time.

However, sounds like it's usable and if vagrant 1.7 comes with a listen upgrade it might be even better then :)

reply

mikeytown2 – 2 years ago

To help with slow NFS mounts I recommend using http://drupal.org/project/imageinfo_cache & https://www.drupal.org/project/advagg/ as these where built with a slow file system in mind. If you have the files mounted on S3 (really slow access) then this advagg setting `$conf['advagg_fast_filesystem'] = FALSE` will help a lot if you have https://drupal.org/project/httprl/ enabled as well.

reply

Chris Hall – 2 years ago

I have usually set up a Samba share on the VM when running on Windows, it is hardly ever mentioned as an option but always works very well for me.

reply

mradcliffe – 2 years ago

Great to see some metrics on NFS vs rsync shared folder support. Did you run any tests where the VM is the NFS Server and the Host machine mounts instead.

I prefer -rtlDPvc --delete as my rsync options. I run into problems with just using archive.

Finally, The rsync-back vagrant plugin was pretty much made for Drupal and features by smerrill.

reply

Joseph Rawson – 2 years ago

The rsync solution has problems. FWIW, nfs is an excellent solution if you use the guest VM as an nfs server and add /vagrant to /etc/exports. I'm sure the samba solution mentioned in the comments is just as good in the windows environments. Running the service from the guest VM keeps all the files on the native file system.

reply

Jeff Geerling – 2 years ago

This setup definitely solves some of the pain points, however, it's not supported out of the box with Vagrant (you have to configure it manually, either via CM or by hand), and you then end up with the same latency problem, but in reverse—that is, if you edit files

on your master workstation (e.g. in an IDE or something like Sublime), operations will have inherent delays.

I like to use full-project search quite often, and it takes minutes (instead of seconds) to search entire large codebases if mounted this way.

reply

Rhys Arkins – 2 years ago

Hi Jeff, FYI I found this article useful and mentioned it in a blog post about why/how we use unison with Vagrant. Tried linking it from here but got blocked as spam, but I assume you can find it if you desire!

reply

Jeff Geerling – 2 years ago

Sorry about that! Here's the aforementioned post: https://keylocation.sg/blog/vagrant-and-unison-without-a-plugin/

reply

Clint Beacock – 1 year ago

Thanks for the writeup, Jeff. I inherited a drupal site this week that was killing me on local performance, and it was nfs issues for me too.

I've switched over to rsync, but found that it was painfully slow and cpu intensive for watches. I then found smerrill's 'vagrant-gatling-rsync' plugin and it's now performing great.

github.com/smerrill/vagrant-gatling-rsync

reply

All content copyright Jeff Geerling. Top of page.