Wim Leers
Bachelor thesis proposal
October 18, 2008

Proposal: Improving Drupal's page loading performance

Introduction

I've been an active member of the Drupal[1] community for almost 2 years[2]. Drupal is an open source Content Management System (of Belgian foil!), but the term Content Management Framework would be a better fit. It's a very solid and extensive framework with APIs[3] for various kinds of functionality: Forms API, Menu System, Batch API, Taxonomy, database abstraction layer, theme system, etc. It's written in PHP. For cross-browser JavaScript functionality, the jQuery[4] library is used.

To indicate the maturity of the Drupal project: it's being used by an endless list of big names. A short selection: Amnesty International, MTV UK, United Nations, Harvard Science.
Recently the jQuery team announced that both Nokia and Microsoft were adopting jQuery as part of their development platforms.
It should be clear by now that Drupal has become one of the big names on the market.

Exactly because of its popularity, Drupal is being used for bigger and more high-traffic web sites than before. Scalability then becomes a very important factor.
If you know that the largest part – 80 to 90%[5] – of the response time (as observed by the end user) is spent on downloading the components of a web page, then that's also the part where optimizations have the largest effect: the page loading performance (not the page *rendering* performance).
And that's exactly the area in which Drupal, like virtually every CMS/F, fails completely.

There are many aspects to page loading performance. See the basic analysis that I've written for details[6]. Most problems are fairly trivial to fix. The most complex part however, is using a CDN. For static web sites, this is relatively easy to support, but for dynamic web sites, where files are uploaded (and even altered) all the time, the process of synchronizing files to a CDN, must be automated to be maintainable.

A CDN is a Content Delivery Network; a network of (static file or streaming media) servers that are located around the globe. These servers all mirror each others' files. When a user requests a certain file from the CDN, the server that is the closest to the user will serve the file.
By using a CDN, your web site will load much faster: the latency will be lower and the throughput will be greater.

---

[1] http://drupal.org

[2] http://drupal.org/user/99777

[3] http://api.drupal.org

[4] http://jquery.com

[5] http://developer.yahoo.com/performance/rules.html#feperf

[6] http://wimleers.com/article/improving-drupals-page-loading-performance

Problem

The goal I'd like to set: provide easy CDN integration for Drupal web sites. Drupal core will probably have to be patched, to be able to alter the URLs, to make it possible to serve files from another domain, but also to make file URLs unique. I will write a Drupal module that synchronizes files to CDNs, and a daemon that cooperates with this module for big-scale web sites. This daemon could then also be used by other projects, such as WordPress, Joomla, Plone, Django, and so on.

Now, it would seem obvious that the tools necessary to synchronize files to a CDN are already available. Unfortunately, that's not the case. The most important reason is that CDNs have only become cheaper very recently. Most web developers don't know about it yet. At least one closed-source tool exists[7], but it's written specifically for one particular CDN service.
Most CDNs assume that they are going to be used for the distribution of big files, and therefore a manual upload – typically via (S)FTP – of each file is acceptable. Finally, page loading performance analysis is a fairly recent development.
The combination of the above factors explains why there are no CDN synchronization tools available yet.

Most CDNs support file uploads via FTP and SFTP, some support rsync (for details, see "CDNs and supported upload methods"). Synchronization through rsync could fairly easily be automated, but not so easily through FTP and SFTP. You would still need to do a fair amount of coding, if you'd want to be able to specify which files to synchronize. Even if you would write all of the above, it wouldn't be scalable, because you'd have to recursively scan each directory for new and changed files.
The daemon I propose to write would take advantage of the present filesystem event monitor (inotify on Linux, FSEvents on Mac OS X, WMI on Windows).

Important: I don't have to start from scratch!

I want to make it crystal clear from the beginning that *I'm exploring something I'm already fairly familiar with*, as I've already said in my initial e-mail.

I've already written a basic analysis and written a module that implements the essence of the required functionality[8]. Ironically, this module has serious design issues, which prevents it from being scalable. I will rewrite this module from scratch, with a lot of additional functionality.

Research & implementation details

Wherever feasible, unit & functional tests should be written, both for PHP and C++/Qt code.

Items marked with an asterisk (*****) have already been implemented before, so I can look at my old code for inspiration when writing the new implementation.

Complex problems are marked with (**§**).

---

[7] http://www.us.cdnetworks.com/technology/content_Synchronization.php

[8] http://drupal.org/project/cdn

1. CDN integration
    1. Synchronization
        1. Small scale (PHP/shell scripts)
            1. FTP *****
            2. (extra) SFTP
            3. (extra) Amazon S3
        2. Big scale (cross-platform daemon written in C++/Qt)

            For maximum scalability, you don't want PHP or shell scripts to be running during cron every x minutes, scanning directories for newly added or changed files and then synchronizing them to a CDN (*****).
            That's why you want an actual *daemon* (**§**) to handle that: this daemon would run at all times instead of at specified intervals (cron), could take advantage of the filesystem event monitor (inotify[9] on Linux, FSEvents[10] on Mac OS X, WMI[11] on Windows) to detect new/changed files and can open parallel network connections for synchronizing (which will result in huge synchronization speedups).

            1. Linux support (inotify)
            2. FTP
            3. SFTP
            4. rsync
            5. (extra) Amazon S3
            6. (extra) Mac OS X support (FSEvents)
            7. (extra) Windows support (WMI)
    2. Tracking (of synchronized files)

        At all times, we must maintain a list of synchronized files; 1) to know which files still have to be synchronized, 2) to know the unique URL of a file. Typically, most files must have unique URLs; only files that are not cached on the client side don't need this (like streaming video).
        You could argue that you really only need a *blacklist* (track files that aren't synchronized yet), but that wouldn't work! You would have to regenerate the unique file URL each time, which is vastly slower than keeping them stored somewhere (i.e. a *whitelist*).
        So, how do we maintain a list of files that could grow up to (and beyond) 100,000 files (**§**)? And how do we actually use this effectively, i.e. without slowing down the page *rendering* time

---

[9] http://en.wikipedia.org/wiki/Inotify

[10] http://en.wikipedia.org/wiki/FSEvents

[11] Windows Management Instrumentation, http://en.wikipedia.org/wiki/Windows_Management_Instrumentation

(**§**)? This is one of the most important aspects on which I still have to do research. (Possibilities: MySQL, file-based, SQLite, Memcached.)

3. Issues

    1. You must be able to use multiple CDNs, and route different groups of files to different CDNs.

    2. All "standard" URLs must be altered, so that clients will request files from the CDN instead of the web server. In 99% of the cases (e.g. CSS, JS files), it's necessary that the URLs are *unique*, so you can set a far future *Expires* header (e.g. +10 years) (*****).

    3. Even URLs in CSS/JS files must be altered! The reason: because the URLs must be unique, you can no longer depend on the relative paths used in CSS/JS files (*****).

    4. Page caching is problematic: ensure that URLs referenced in cached pages persist. This means that we must keep files on the CDN for at least as long as pages can be cached.

    5. CDN propagation is a problem, too. Because a CDN typically is a globally distributed network of servers, you have to ensure that the file has been propagated to *all* servers, before you can start using (linking) to it.

4. Optimizations

    1. Since proper usage of a CDN demands unique filenames for each version of a file, we can simplify the necessary tests. All we need to check, is if the file 1) exists and 2) has the correct size.

    2. Automatically minify CSS and JS files that are being synchronized, by using the YUI compressor.[12] Will only be available when running the daemon.

    3. (extra) Ignore files of disabled modules and themes.

5. Basic analysis tools

    1. Status report (time, duration, etc. of last synchronization)

    2. Statistics (ratio synchronized files, file size, # of files synced per day, etc.)

    3. As soon as the average duration required to synchronize the files to the CDN when using the small scale features reaches a certain threshold, the site owner should automatically be warned that he should upgrade.

    4. (extra) Which files should be served from a CDN, but aren't? I.e. which pages have most unsynchronized files (**§**)?

2. (extra) JS should be loaded as late as possible (before the closing body tag), but *not* JS that affects the styling of the page, because this results in annoyingly late style updates ("flicker" after the page seems to have finished loading). Therefore, "critical" JS should be loaded in the head tag. Clear guidelines for this should be written.

---

[12] http://developer.yahoo.com/yui/compressor

3. (extra) An Expiration API. This would allow the different entities in Drupal (Taxonomy vocabularies, Menus, Nodes, etc.) to keep track of when it was last updated, i.e. when the cache should be invalidated. It should also support hierarchies, because you wouldn't want an entire cached tree to be marked as invalidated, but only the relevant subtree (**§**).
   One could then make all JSON/AJAX replies of the server set the *Last-Modified* header, which would greatly improve the responsiveness of pages that include AJAX functionality. It would even allow for more aggressive and effective caching techniques.

CDNs and supported upload methods

CDN	**FTP**	**SFTP**	**rsync**	**other**
Akamai	Y	N	Y	SCP
BitGravity	Y	N	Y	WebDAV
CacheFly	Y	Y	Y	
CDNetworks	N	N	N	custom
Edgecast	Y	N	Y	web interface
Level3	Y	Y	Y	SOAP/REST, API
Limelight	Y	Y	N	WebDAV, Aspera
SimpleCDN	Y	Y	N	automatic offloading, web interface