Klarendalseweg 71
6822GB Arnhem
tel : 026 442 1950
fax : 026 443 7844
email : info@netcreators.com
web : www.netcreators.com

Optimizing TYPO3 backend responsiveness

author : Michiel Roos
date : 27 May 2008

Table of Contents

Introduction

During the last months I attended to excellent meetings. One was the TYPO3DD08 and the other was the “Dirty Handson Session” (Forelle ist geil!). During the last session we worked on HCI issues in Arnhem @ the Netcreators office together with Ben van 't Ende, Benjamin Mack, Jens Hoffmann, Patrick Broens, Steffen Kamper and me. We had a lot of fun and made a good start improving the backend step by step.

During the last session I made some minor changes in the t3skin stylesheets and found a _.htaccess file in the stylesheets directory. I wondered what it did there and it turned out to contain:

```
<FilesMatch "\.css$">
  <IfModule mod_expires.c>
    ExpiresActive on
    ExpiresDefault "access plus 7 days"
  </IfModule>
  FileETag MTime Size
</FilesMatch>
```

I wondered why this htaccess file was not enabled by default. Benjamin told me dat Dmitry put the file in but did not enable it by default for some reason. It may have to do with Apache parsing the entire document root path on every request to find out if there are any .htaccess files in the path (which might affect rendering of the requested file) and then reading all .htaccess files it finds (one filesystem access for every file). Putting a lot of .htaccess files in the source may have an adverse effect on performance.

A performance tuning page [1] explains it like this:

> *The Apache directive AllowOverride is one of the most powerful and versatile directives in the Apache arsenal. It allows for the runtime reconfiguration of Apache for a specific directory (and all its subdirectories.) If AllowOverride is enabled, Apache checks for a specific file (configurable, but the default of .htaccess is practically never changed) in each directory that it visits, and parses it for configuration directives. This feature is most commonly used to limit access to directories (ask for a password), but the .htaccess file can contain almost all Apache directives.*
>
> *Unfortunately, this level of flexibility comes at a price. In order to see the .htaccess files, Apache has to scan for it in every directory, resulting in the same number of filesystem lookups as the symlinks case previously discussed. It should be noted that checking for .htaccess and symlinks has to be done in two separate and different stat() systemcalls, so disabling only one of the two still reduces the number of lookups significantly.*
>
> *AllowOverride can be set to multiple values. Only setting it to None will turn off the checking for .htaccess files altogether. Like the FollowSymlinks and SymlinksIfOwnerMatch directives, AllowOverride can be specified on a per-directory basis. Even if the VirtualHosts want or might want to use the .htaccess functionality, it is a good idea to turn off AllowOverride for directories that will never contain .htaccess files. The default Apache configuration does this for the system root directory, only enabling AllowOverride for the server's DocumentRoot.*

This teaches us that a server wide setting of AllowOverride to None is a sane setting. It is best to only use it when absolutely necessary and then only to enable it for a specific directory. Enabling

the .htaccess files by default would make some installations send a 500 error instead of a nice TYPO3 backend.

This also puts forward an interesting optimisation issue. It is better to implement the default TYPO3 rewriting rules in the Vhost container instead of in a .htaccess file. If you have access to your apache configuration, that is the way to go. There is room for improvement in the default TYPO3 .htaccess file.

Obfuscated optimisation settings

That's a fair enough reason for not enabling this by default, but Dmitry told me that enabling the 4 or so hidden .htaccess files in the TYPO3 source is one of the first things he does after setting up a new TYPO3 installation. So optimisation options are in place in the source but disabled by default. This article is an attempt to bring these and some more optimisation tips out into the open.

In the conclusion I propose a possible enhancement to the current setup.

Why is my backend so slow?

So what does TYPO3DD08 have to do with all this? Well, during one of the talks someone (Dmitry?) mentioned a nice FireFox extension: YSlow [2]. YSlow is a Yahoo-made extension to FireBug. It check the page you load and gives it a rating (F-A and 0-100). So a F23 rating would be pretty bad and a A78 rating is “not too shabby”. Yes, go on . . . get it now. I'll wait here ;-)

NOTE Please note that there are some shots of Live HTTP Headers in this document. I am not sure if they correctly reflect reality or that YSlow interferes with the caching.

TYPO3 is fat

Let's pull up a default TYPO3 4.2 installation on a default Apache 2 installation and see what rating (and why) YSlow gives the TYPO3 backend in page mode listing a single page. YSlow lives in the bottom right corner of the browser window. Open up a page in page mode and click the YSlow icon.

YSlow in all it's glory

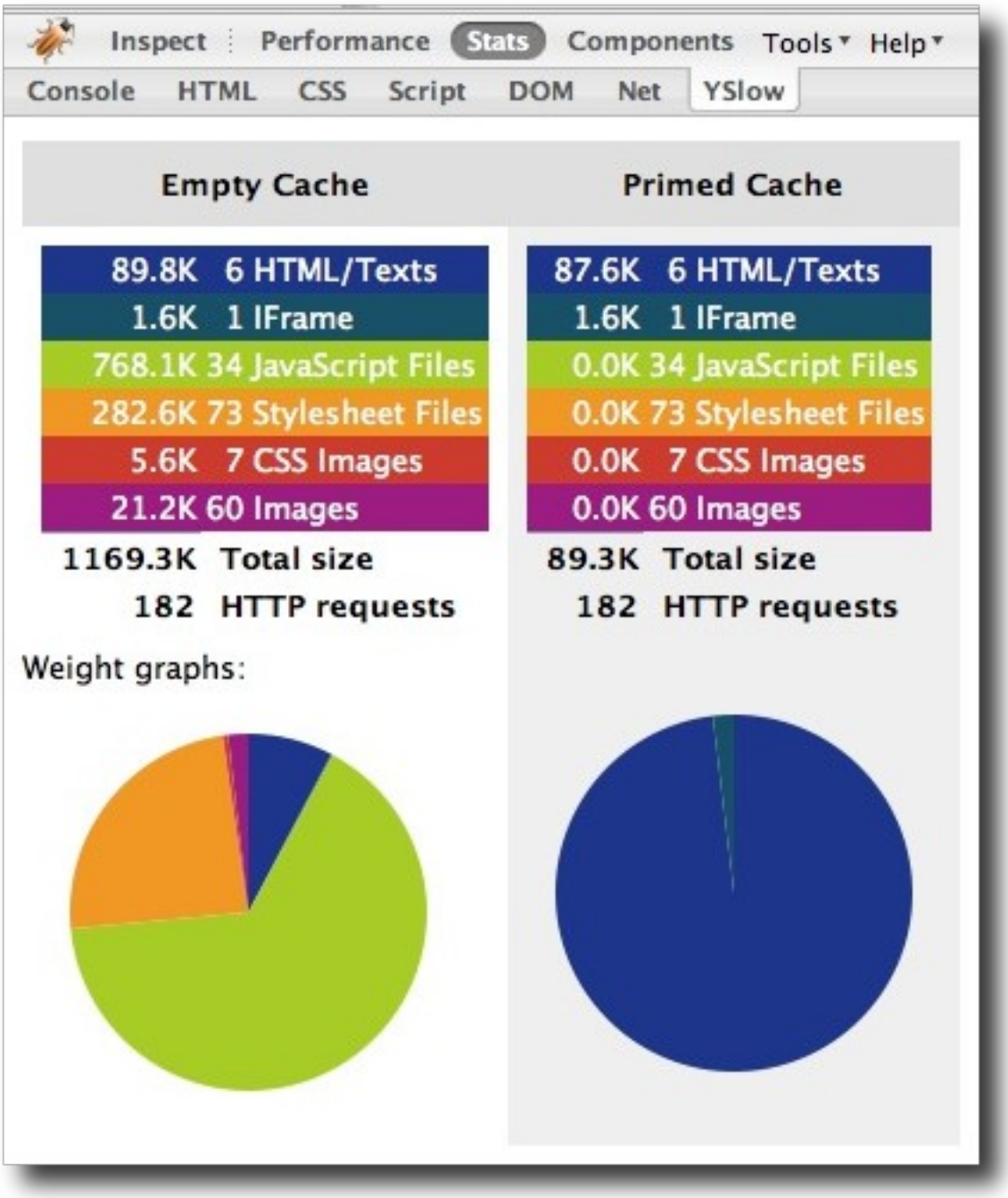
YSlow pulls in the site and inspects all the headers. Based on the result it generates lots of valuable information and a devastating rating for the page. Our job is to improve our rating and pass the test as good as we can.

Report grade and page size

Please ignore the timing information since it appears that it is calculated only from the loading time of the initial backend (without the added timing of going to page mode and clicking a page).

Judgement . . . an abominable F 32 :-(

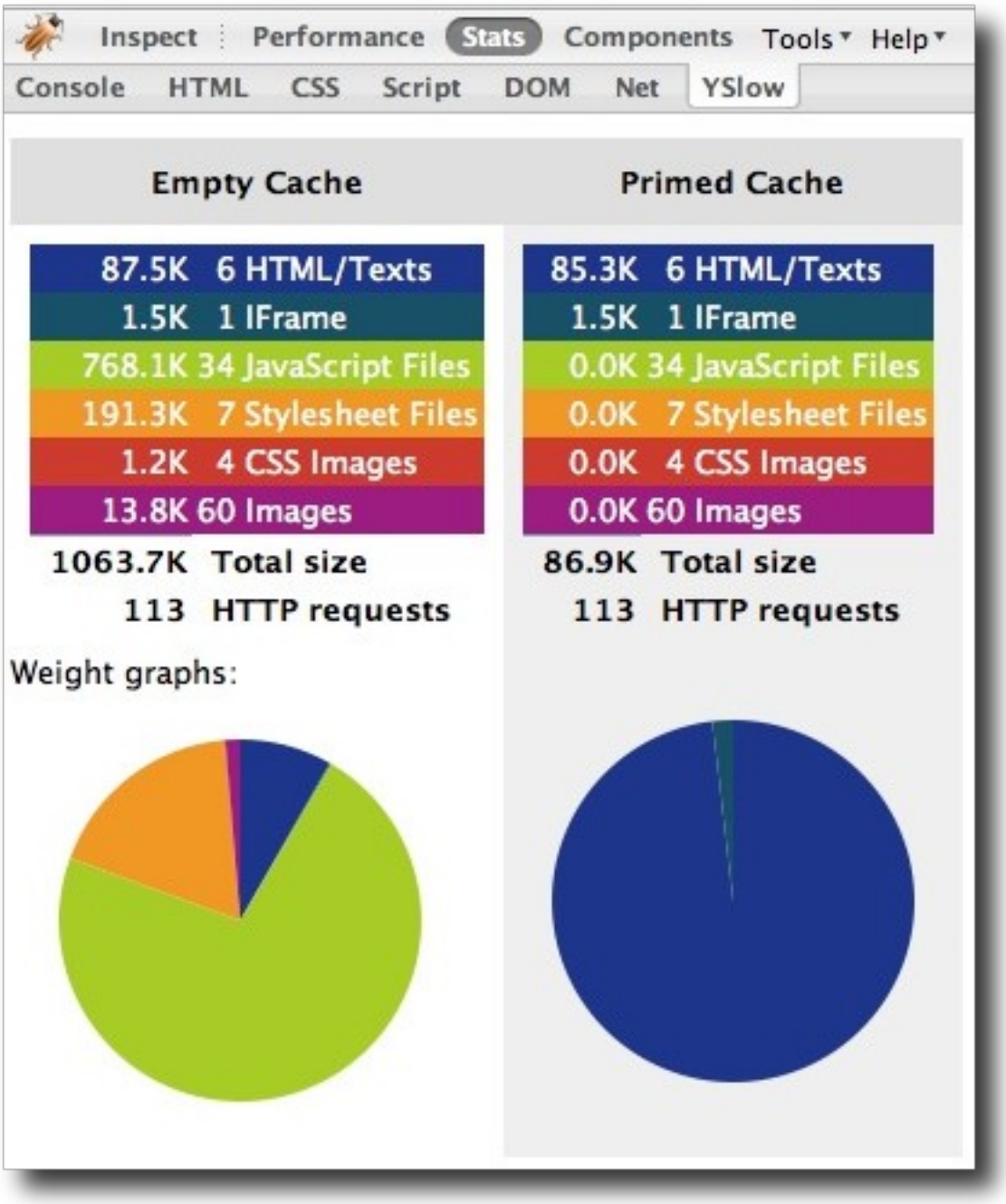
The first thing that stands out for me is the sheer size of the result we get pushed back at us. A stunning 1169.1 K! For just the backend showing us an empty page in the backend. If you wonder what information all those bits bring us, take a look at the view called; stats.



Statistical breakdown of result by meta type.

There is a lot of Javascript and CSS thrown at us. 73 CSS files and 34 Javascript files for one backend? That's amazing.

Now I have heard of some people that prefer the old brown TYPO3 skin. The first thing they do when installing a TYPO3 installation is to disable the t3skin. The advantages are (they claim) better contrast of the icons and better responsiveness. Let's investigate the last claim with our new tool.



Statistical breakdown of result by meta type for the retro skin.

So in the red corner, weighing in at 1063.7 K, using 7 CSS files, champion for more than 10 years; the retro skin. In the blue corner weighing in at 1169.1 K, using 73 CSS files, just in town; the t3skin.

Well . . . actually . . . not too pretty either. Even the retro skin gets a F32.

The report card shows you a list of criteria you are judged by. Yahoo hosts a page where you can find out all about optimising your site: http://developer.yahoo.com/performance/rules.html

The numbered criteria can fold out to give you more information. We'll go over the points one by

one and see if and how we can improve our grade. We'll skip the criteria for which we received an A. We will also skip CDN [3].

Fold out to see more information.

Minimize HTTP Requests

Good point YSlow! But not today. Of course it would be great if we could reduce the number of CSS and JS files by using something like the scriptmerger extension by Stefan Galinski. It uses the jsmin script and csstidy code to tidy up your CSS and JS before outputting it to the frontend. It would be nice to reduce the number of CSS and JS in future releases of TYPO3 and generating cached optimised JS and CSS would be great. Anybody have a personal itch already ;-)

Oh, in retrospect . . . please note the stats pictures above. Both a fresh (empty cache) page request and a second (filled cache) page request generate the same amount of requests (113). Later on we will see the number for the second request drop dramatically.

Add an Expires header

Expires headers are good. TYPO3 guru's have known this for some time (sendCacheHeaders anyone?). They tell the browser to keep a file cached for a certain time. This makes sure the browser does not re-fetch the file every time it visits the page. The TYPO3 backend does not send any expires headers as far as I can see (using Live HTTP Headers). And indeed, it would not be a good thing since content on the backend pages changes very very often.

If you open up point 3 to see what files YSlow is complaining about, you will see that a lot of static content is sent to the browser without expiry data. There are a lot of CSS, JS and image files that never change (until the next release of TYPO3) and they should be cached by the browser indefinately (until browser cache is cleared before visiting a new install).

If we use Live HTTP headers to inspect a request of one of the backend frames we can see a GET and POST request. The request shown in the shot below is not the first request. The browser already knows the so called Etag [4] of the file. The server will only transfer the file if the file does not match the Etag in the browsers cache.

This is a big step to better performance, but still the browser checks with the server for every file and tries to verify the Etag known, to the Etag the server gives back. This means that a transaction still takes place for every file not kept in cache without expiry information.

Expiring icon

If you run apache (honestly, I have not looked into getting this to work on other webservers ;-), you can install and enable mod_expires [5]. Once this is installed you can add rules to your configuration to send expiry data for the files YSlow complains about.

In words: “Keep around static JS, CSS and image files from within the TYPO3 source for a long time”. Let's say that we want to keep around the TYPO3 files for about 6 months.

In code:

```
<LocationMatch "/(typo3|t3lib)/">
  <IfModule mod_expires.c>
    ExpiresActive on
    ExpiresDefault "access plus 6 months"
```

```
    ExpiresByType image/gif "access plus 6 months"
    ExpiresByType image/png "access plus 6 months"
    ExpiresByType text/css "access plus 6 months"
    ExpiresByType application/x-javascript "access plus 6 months"
    ExpiresByType text/javascript "access plus 6 months"
    ExpiresByType image/jpeg "access plus 6 months"
  </IfModule>
</LocationMatch>
```

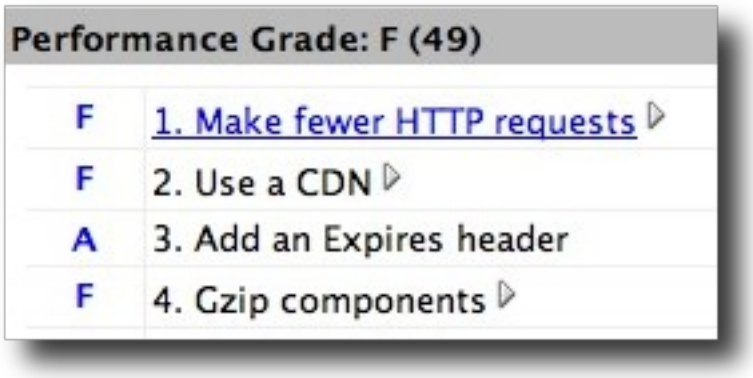
The order of the of the configuration is important. Place the most requested types at the top. The matching algorithm using this configuration will match the gif and png files soonest because they are at the top of the list.

The YSlow minimum is 48 hours, which means we can also get away with:

```
      ExpiresDefault "access plus 48 hours 1 seconds"
```

For the backend, a long period is OK, but when you are optimising for the frontend, you will have to dream up a value that is sensible for your site. An alternative would be to (when you do update your site with a new design) prefix all the css and img paths wit a version number so the clients will need to actively fetch your new styles. I know, it's not a watertight scheme for the frontend yet . . .

When we re-run YSlow . . . surprise . . . we jump from F 32 to F 49.



Our score after enabling expiry headers

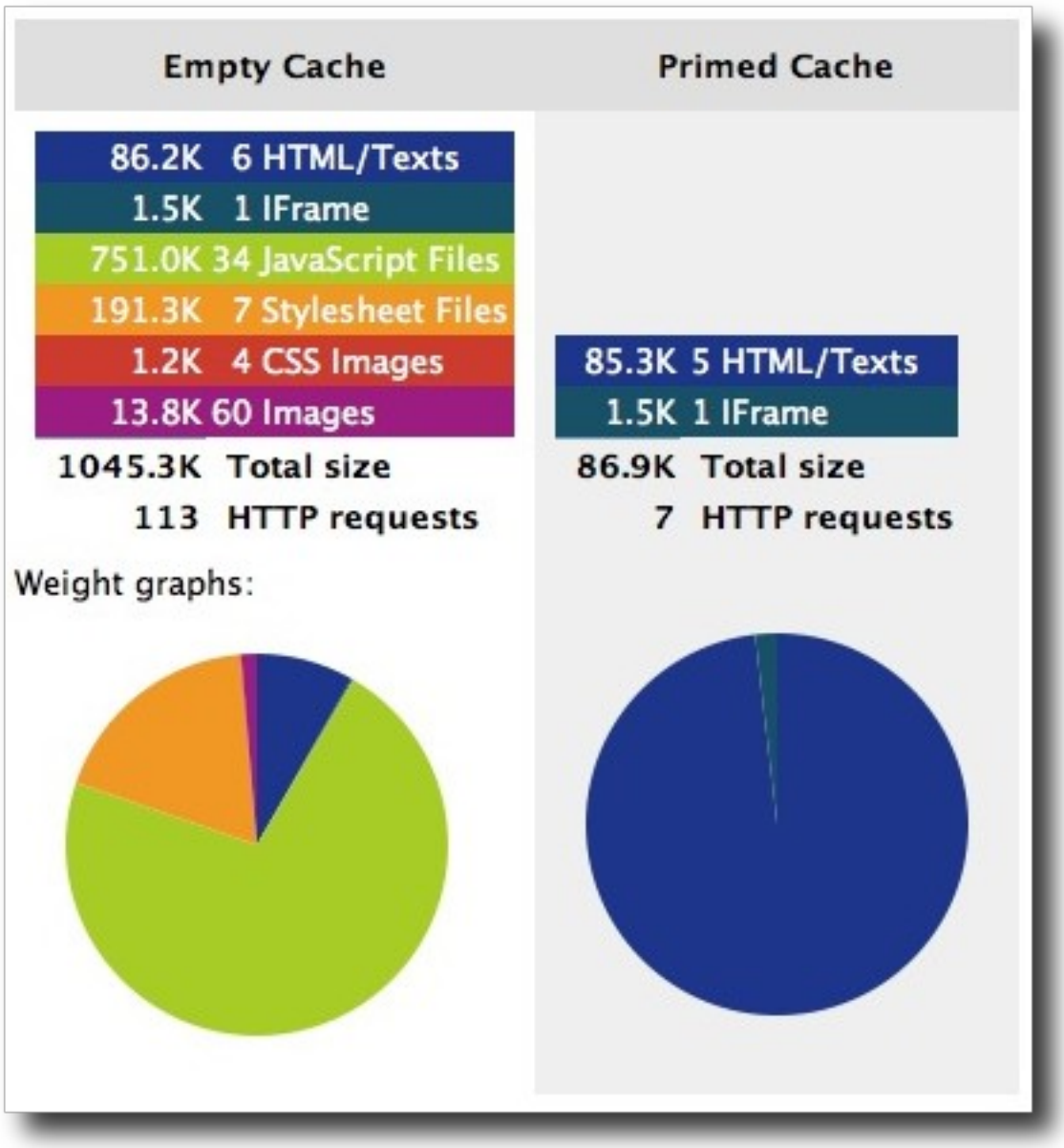
Good, so now we have made the server send expiry information which should keep those files in the browser cache (in this case 48 hours + 1 second ;-). The headers tell us that the browser still checks back with the server for every file request.

In fact, after finishing this piece I added this to my home servers config:

```
# Optimisation settings for FE files
<LocationMatch "/(fileadmin|typo3conf|typo3temp|uploads)/">
  # Send expiry headers for static stuff.
  <IfModule mod_expires.c>
    ExpiresActive on
    ExpiresDefault "access plus 48 hours 1 seconds"
```

```
  </IfModule>
</LocationMatch>
```

When you jump to the stats tab you will see the real difference the expiry headers make.



Statistics with expiry headers enabled

So there we go, we dropped from 113 requests to 7 requests.

Gzip components

When you go to YSlow's components tab, you will see a list of items retrieved from the server. There you will also see a column: Size (gzip). The total transfer size can be reduced, speeding up the packet transfer. Both the server and the client are powerful enough to compress and de-compress the files. We can enable mod_deflate [6] in apache and add a configuration snippet to

bump our grade.

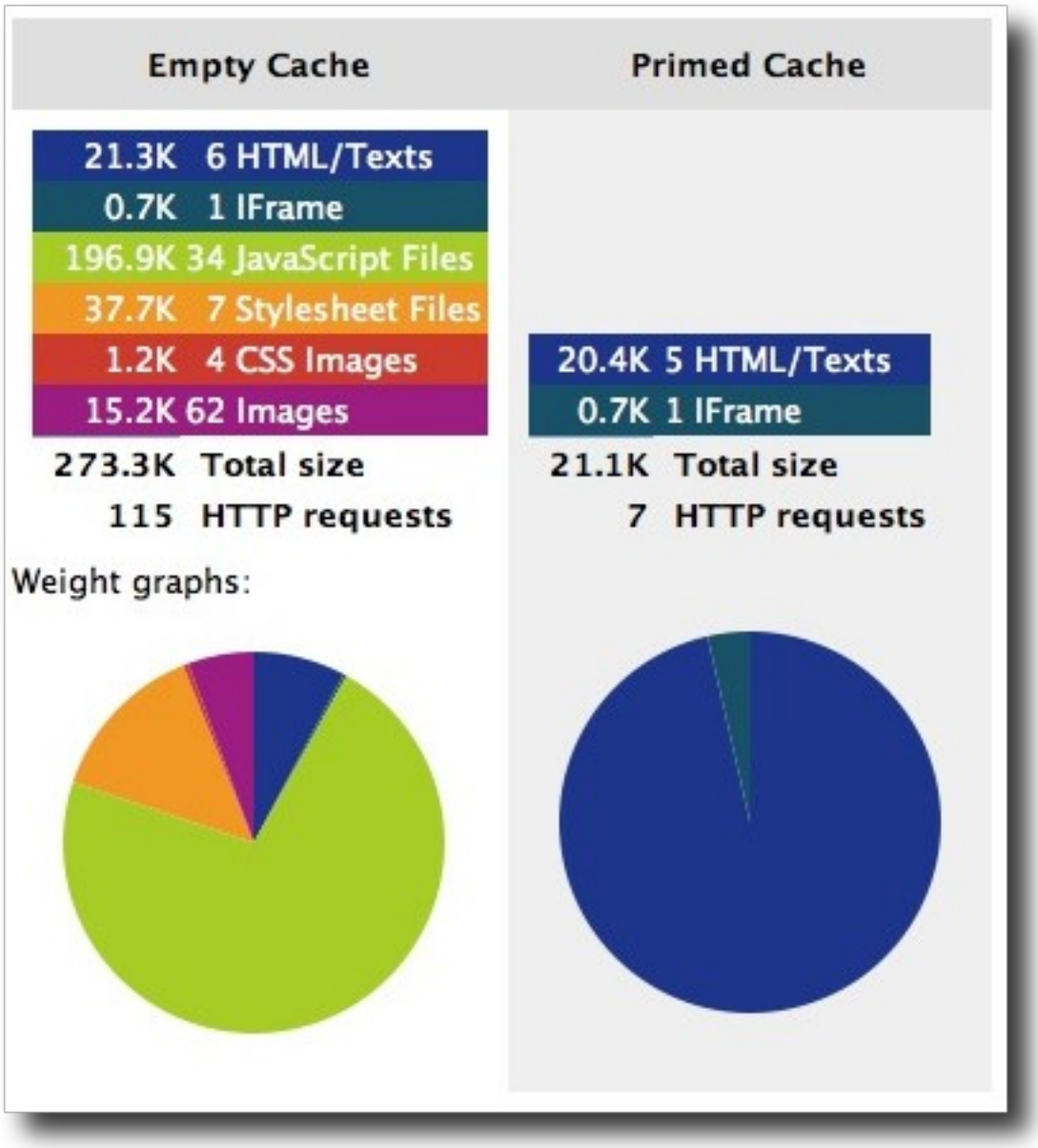
You may wonder why the module is called mod_deflate and not mod_gzip. That is because mod_deflate is the name of the default compression module of Apache 2. Mod_gzip is the name of the default compression module of Apache 1.3. Today mod_deflate even supports the compression levels that mod_gzip supports (gzip -1 to gzip -9).

The DeflateCompressionLevel directive specifies what level of compression should be used, the higher the value, the better the compression, but the more CPU time is required to achieve this. The default zlib compression level is used as the default.

```
# Enable compression (reduce traffic by 70%)
# http://httpd.apache.org/docs/2.0/mod/mod_deflate.html
<LocationMatch "/(typo3|t3lib)/">
  <IfModule mod_deflate.c>
    SetOutputFilter DEFLATE

    # Netscape 4.x has some problems...
    BrowserMatch ^Mozilla/4 gzip-only-text/html

    # Netscape 4.06-4.08 have some more problems
    BrowserMatch ^Mozilla/4\.0[678] no-gzip

    # MSIE masquerades as Netscape, but it is fine
    # BrowserMatch \bMSIE !no-gzip !gzip-only-text/html

    # NOTE: Due to a bug in mod_setenvif up to Apache 2.0.48
    # the above regex won't work. You can use the following
    # workaround to get the desired effect:
    BrowserMatch \bMSI[E] !no-gzip !gzip-only-text/html

    # Don't compress everything
    SetEnvIfNoCase Request_URI \.(?i:gif|jpe?g|png)$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \.(?i:exe|t?gz|zip|bz2|sit|rar)$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \.pdf$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \.avi$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \.mov$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \.mp3$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \.mp4$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \.rm$ no-gzip dont-vary
  </IfModule>
</LocationMatch>
```

That's it, restart apache and see the results. Our grade is still an F, but we improved from a 49 to a 55. When you look at the components tab, you can see how much every file is compressed.

Our stats view shows us we dropped from 86,9 K to a mere 21.1 K. That's pretty close to the average size (ancient rule of thumb) for a responsive web page.



Statistics with compresison enabled

Put JS at the bottom

24 external scripts were found in the document HEAD. Could they be moved lower in the page [7]?

Honestly . . . I don't know. :-)

Any Core dev care to comment on this point?

Minify JS

None of the JS and CSS files are minified [8]. There probably is a lot to gain here for TYPO3.

Web pages that refer to multiple CSS or JavaScript files often suffer from slower page loads, since the browser must request each referenced file individually. Most browsers will only make two

simultaneous requests to a single server. The latency involved in opening multiple requests and waiting for them to finish before making new requests can result in a user-visible delay, and that can make your users sad.

Minify is a PHP library that attempts to fix this problem by combining multiple CSS or JavaScript files into one download. By default, it also removes comments and unnecessary white space to decrease the amount of data that must be sent to the browser. Most importantly, it does all of this on the fly and requires only a few simple changes to your existing web pages.

CSSTidy does something similar for CSS files.

A frontend extension using Minify and CSSTidy exists. I mentioned it earlier in this document. Is that itch still bearable? You better start scratching!

Remove duplicate scripts

YSlow tells us our page has 11 redundant JS components:

- 3 typo3/contrib/prototype/prototype.js
- 1 typo3/contrib/scriptaculous/effects.js
- 1 typo3/js/common.js
- 2 typo3/js/iecompatibility.js
- 3 typo3/tab.js
- 1 typo3/js/clickmenu.js

That's right The backend pulls in prototype.js four times in total. I'm not a frames guru, but I'm sure there is room for improvement here.

Configure Etags

http://developer.yahoo.com/performance/rules.html#etags explains:

> *Entity tags (ETags) are a mechanism that web servers and browsers use to determine whether the component in the browser's cache matches the one on the origin server. (An "entity" is another word a "component": images, scripts, stylesheets, etc.) ETags were added to provide a mechanism for validating entities that is more flexible than the last-modified date. An ETag is a string that uniquely identifies a specific version of a component. The only format constraints are that the string be quoted. The origin server specifies the component's ETag using the ETag response header.*

```
HTTP/1.1 200 OK
Last-Modified: Tue, 12 Dec 2006 03:03:59 GMT
ETag: "10c24bc-4ab-457e1c1f"
Content-Length: 12195
```

> *Later, if the browser has to validate a component, it uses the If-None-Match header to pass the ETag back to the origin server. If the ETags match, a 304 status code is returned reducing the response by 12195 bytes for this example.*

```
GET /i/yahoo.gif HTTP/1.1
Host: us.yimg.com
If-Modified-Since: Tue, 12 Dec 2006 03:03:59 GMT
```

```
If-None-Match: "10c24bc-4ab-457e1c1f"
HTTP/1.1 304 Not Modified
```

> *The problem with ETags is that they typically are constructed using attributes that make them unique to a specific server hosting a site. ETags won't match when a browser gets the original component from one server and later tries to validate that component on a different server, a situation that is all too common on Web sites that use a cluster of servers to handle requests. By default, both Apache and IIS embed data in the ETag that dramatically reduces the odds of the validity test succeeding on web sites with multiple servers*

We don't use multiple servers, but then the story goes on and tells us:

> *If you're not taking advantage of the flexible validation model that ETags provide, it's better to just remove the ETag altogether. The Last-Modified header validates based on the component's timestamp. And removing the ETag reduces the size of the HTTP headers in both the response and subsequent requests. This Microsoft Support article describes how to remove ETags. In Apache, this is done by simply adding the following line to your Apache configuration file:.*

```
FileETag none
```

I can see clearly now . . . ETags must die!

Then another page [9] tells us:

> *By removing the ETag header, you disable caches and browsers from being able to validate files, so they are forced to rely on your Cache-Control and Expires header. Basically you can remove If-Modified-Since and If-None-Match requests and their 304 Not Modified Responses.*
>
> *Please don't turn off ETags and Last-Modified headers for your .html files, leave one of them ON. (I use Last-Modified for .html).*

I'm not completely sure about that last remark, but I don't think we need to worry since our backend.php sends no-cache headers to the client.

ETags can be useful for a normal website where the CSS may change from time to time, but we can safely get rid of them for the backend. Stuff only changes there every six months or so ;-). We just need to instruct our editors to clear their browser cache after installing a new TYPO3 version. If you disable ETags, then take great care for what directories you turn them off. You probably don't want to turn them off for content that changes relatively often like typo3conf, fileadmin, uploads, typo3temp etc.

After killing off ETags, our rating jumps from F 55 to F59.

Still an F :-(

Turn off Last Modified headers

The askapache page [10] also suggests to turn off Last Modified headers:

> *If you remove the Last-Modified and ETag header, you will totally eliminate If-Modified-Since and If-None-Match requests and their 304 Not Modified Responses, so a file will stay cached without checking for updates until the Expires header indicates new*

> *content is available!*
>
> *By removing both the ETag header and the Last-Modified headers from your static files (images, javascript, css) browsers and caches will not be able to validate the cached version of the file vs. the real version. By also including a Cache-Control header and Expires header, you can specify that certain files be cached for a certain period of time, and you magically (this is a really unique trick I promise) eliminate any validation requests!!*

Magic sounds good to me, so I turn this off Last-Modified:

```
<FilesMatch "\.(?i:ico|png|gif|js|css|jpe?g)">
        Header unset Last-Modified
</FilesMatch>
```

In human language: Only match the file types (case insensitive) inside the TYPO3 dirs.

Our YSlow score does not improve, but I can sleep better knowing that the headers are smaller and so the responsiveness has improved.

Although it makes our headers smaller, the legality of skipping the Last-Modified header is debatable. The HTTP spec [11] (an interesting read in itself) notes:

> *HTTP/1.1 servers SHOULD send Last-Modified whenever feasible.*

Maybe we can argue that in our case it's just 'not feasible' ;-).

I did some benchmarking runs using apachebench (ab2) to see if turning off the Last-Modified and the ETags has any effect. The transfer rate slightly increases when using the smaller headers.

Desert

There is a lot more tweaking you can do. I'll mention two more things here. One of them is caching related and one header related.

ServerToken

Reduce header size by changing the ServerToken (http://httpd.apache.org/docs/2.0/mod/core.html#servertokens). The ServerToken configures what you return as the Server HTTP response Header. The default is 'Full' which sends information about the OS-Type and compiled in modules. So instead of sending a full:

```
Server: Apache/2.0.41 (Unix) PHP/4.2.2 MyMod/1.2
```

You can also get away with a flimsy:

```
Server: Apache
```

Cache-control: public

Take care with this one! Setting an extra Cache control header to public implies that you are a-ok with any proxies storing your files.

```
Header add "Cache-Control" “public”
```

This means that if you have a proxy half way your connection to your site, then the proxy will deliver the files instead of the server. This may speed up your initial page rendering. But when the

proxy has cached one of your files for the time noted in your expiry headers, and you choose to change your files . . . . then the proxy does not know about this and happily serves the old files to the user.

So maybe you just want to add something like this:

```
<Files clear.gif>
      Header add "Cache-Control" “public”
</Files>
```

;-)

Also take care that TYPO3 also sends Cache-Control headers. So you probably want to know for sure that your settings do not conflict with or negate the settings of TYPO3.

Conclusion

I hate to be the one break the news to you, but there is room for improvement for the responsiveness of the TYPO3 backend.

By tweaking our webserver we can boost the responsiveness a great deal:

	retro skin	**t3skin**
before	F 32	F 32
after	F 59	F 52

The red corner wins. Both corners get about a 100% rating improvement (and the perceived responsiveness is a lot higher ;-).

A safe and sane backend configuration may look like this:

```
# Optimisation settings for static TYPO3 BE files.
<LocationMatch "/(typo3|t3lib)/">

  # Send expiry headers for files that do not change often
  # http://httpd.apache.org/docs/2.0/mod/mod_expires.html
  <IfModule mod_expires.c>
    ExpiresActive on
    ExpiresDefault "access plus 6 months"
    ExpiresByType image/gif "access plus 6 months"
    ExpiresByType image/png "access plus 6 months"
    ExpiresByType text/css "access plus 6 months"
    ExpiresByType application/x-javascript "access plus 6 months"
    ExpiresByType text/javascript "access plus 6 months"
    ExpiresByType image/jpeg "access plus 6 months"
  </IfModule>


  # Enable compression (reduce traffic by 70%)
  # http://httpd.apache.org/docs/2.0/mod/mod_deflate.html
```

```
  <IfModule mod_deflate.c>
    SetOutputFilter DEFLATE

    # Netscape 4.x has some problems...
    BrowserMatch ^Mozilla/4 gzip-only-text/html

    # Netscape 4.06-4.08 have some more problems
    BrowserMatch ^Mozilla/4\.0[678] no-gzip

    # MSIE masquerades as Netscape, but it is fine
    # BrowserMatch \bMSIE !no-gzip !gzip-only-text/html

    # NOTE: Due to a bug in mod_setenvif up to Apache 2.0.48
    # the above regex won't work. You can use the following
    # workaround to get the desired effect:
    BrowserMatch \bMSI[E] !no-gzip !gzip-only-text/html

    # Don't compress everything
    SetEnvIfNoCase Request_URI \.(?i:gif|jpe?g|png)$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \.(?i:exe|t?gz|zip|bz2|sit|rar)$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \.pdf$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \.avi$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \.mov$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \.mp3$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \.mp4$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \.rm$ no-gzip dont-vary
  </IfModule>


  # Disable Etags
  # http://httpd.apache.org/docs/2.0/mod/core.html#fileetag
  # Only enable this if you understand ETags and are aware that disabling them
  # means that browsers will have no way of knowing if a file on the server has
  # changed. It keeps your files in the browsers cache untill the expiry time
  # (you chose above) has passed.  Turning off ETags may be a bad idea in
  # combination with long expiry times and content that changes often.
  #FileETag none

</LocationMatch>


# Disable Last-Modified headers
# Only enable if you are really serious about optimisation and can argue that
# it's just 'not feasible' for you to send a Last-Modified header . . . ;-)
```

```
# You need mod_headers for this tweak:
# http://httpd.apache.org/docs/2.0/mod/mod_headers.html
#<FilesMatch "\.(?i:ico|png|gif|js|css|jpe?g)">
#  Header unset Last-Modified
#</FilesMatch>
```

If you have access to your Apache Vhost configuration, put these tweaks in there.

I think we should put something like this in the default .htaccess file (commented out) in the TYPO3 root. Either that or add a _.htaccess_on_steroids with these settings in them. Actually I would also like to see a separate file called something like _VirtualHost.conf which has a working realurl configuration so people can either blindly copy the information from there or enable the _.htaccess.

At the very least (if nothing else is changed) the four optimisation related _.htacces files below should be merged into the _.htaccess file in the dummy package. People should not have to hunt for them.

- ./typo3/gfx/_.htaccess
- ./typo3/mod/user/ws/_.htaccess
- ./typo3/sysext/t3skin/stylesheets/_.htaccess
- ./typo3/sysext/_.htaccess

As an added bonus, if you run multiple Vhosts with TYPO3, you can make a single TYPO3-BE-speed-tweaks.conf file and include that into your Vhost configurations using:

```
Include /etc/apache/TYPO3-BE-speed-tweaks.conf
```

Depending on how brave you are you can also apply one ore more of these rules to the TYPO3 frontend. Be carefull though, some of the settings may do more harm than good for frontend output. The highest rating I got for the frontend up to now is a A 99.

Woohoo!

I hope I have shown that there is room for improvement and that the default web server set-up is functional but by no means optimal. I am also sure that there are a lot more configuration options I missed, so feel free to share some nice tips.

Now start scratching that itch!

Sources

- [1] Thomas' tips http://www.xs4all.nl/~thomas/apachecon/PerformanceTuning.html
- [2] YSlow http://developer.yahoo.com/performance/rules.html
- [3] CDN http://developer.yahoo.com/performance/rules.html#cdn
- [4] ETag http://developer.yahoo.com/performance/rules.html#etags
- [4] ETag http://httpd.apache.org/docs/2.0/mod/core.html#fileetag

- [5] mod_expires http://httpd.apache.org/docs/2.0/mod/mod_expires.html
- [6] mod_deflate http://httpd.apache.org/docs/2.0/mod/mod_deflate.html
- [7] JS at the bottom http://developer.yahoo.com/performance/rules.html#js_bottom
- [8] Minify JS http://developer.yahoo.com/performance/rules.html#minify
- [9] Ask Apache ETag http://www.askapache.com/htaccess/apache-speed-etags.html
- [10] Ask Apache Last Modified http://www.askapache.com/htaccess/apache-speed-last-modified.html
- [11] HTTP 1.1 protocol http://www.w3.org/Protocols/rfc2616/rfc2616.html
- Apache manual http://httpd.apache.org/docs/2.0/
- CSS Tidy http://csstidy.sourceforge.net/
- Test your browser http://www.mnot.net/javascript/xmlhttprequest/cache.html
- HTTP 1.1 protocol http://www.w3.org/Protocols/rfc2616/rfc2616.html