

Caching of user-dependant content in Drupal

Kim Gad Thomsen

DTU



Kongens Lyngby 2012
IMM-B.Eng-2010-85

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-B.Eng-2010-85

Abstract

When developing large-scale websites with the Content management system Drupal one of the larger challenges is to make the system perform under stress. Drupal is right now one of the most popular systems among those available. This is not because it is the best to work with as a developer neither is it because of the quality of the code in the core, it is simply just the easiest system to sell because there is not a free alternative to a system with such a large community and database with available modules, free to use.

The problem with this is when Drupal needs to handle a lot of concurrent requests (e.g. in peak hours of a newspaper website), especially if those requests is from users who is logged into the system and therefore cant be served a cached version of the requested page. When this happens, at some point the server will begin to crack under the stress of serving all those requests, and the weakest link in the setup is MySQL. The problem does not lie with MySQL itself, more the way Drupal uses it. When Drupal is serving a page it needs to get every information from the database, even the cached data. So even when serving a page as fast as possible, Drupal still needs to ask the database for information, this is a problem under stress. The result of this is the database server (In this case MySQL) is not getting enough CPU time to handle all the requests, and that is what is called a Starvation Problem, and that is what I am trying to fix in this project.

Preface

This report is the result of my bachelorproject which has been done from 29. August 2011 to 30. January 2012.

This project has been done as the final project of my bachelor in engineering (B.Eng) and should be a reflection of my abilities in analyzing a problem, breaking the problem down into something easy to comprehend and then plan and execute the implementation of the solution to that problem

The project have been done in collaboration with Dwarf A/S where i did my internship as a developer working with developing web solutions based on Drupal.

My supervisor at Dwarf A/S has been Henrik Jochumsen who is the CTO at Dwarf A/S. We have been discussing different caching methods and what would suit this project best.

The background for the project was when we launched tv2sport.dk on the new version of Drupal, which is the first real high-traffic site we have done on Drupal 7, the problem with the caching system became very clear and we needed do something about it. Besides just general optimizations some of this project was also used and is now in production on <http://tv2sport.dk>, which showed that this is actually working, which gave me the idea of optimizing the user-cache and making in to a module.

Lyngby, January 2012
Kim Gad Thomsen

Acknowledgements

I would like to thank my colleagues at Dwarf A/S, Jonas Mikkelsen and Henrik Jochumsen for inspiring me to do this project and letting me use one of the development servers for testing this project.

I would also like to thank Simon Sørensen at TV2SPORT for giving me the opportunity of testing parts of my project on <http://tv2sport.dk> which gave a real-life environment to study.

Contents

Abstract	i
Preface	iii
Acknowledgements	v
1 Introduction	1
1.1 Specifying the problem	1
1.2 Structure of the report	2
2 Requirements Specification	5
2.1 Functional requirements	6
2.1.1 Anonymous and Authorized users	6
2.1.2 Site Administrator	7
2.1.3 Site Developer	8
2.2 Non-functional requirements	9
3 Project Management	11
3.1 Agile Development	12
4 Analyzing Drupal	15
4.1 Bootstrapping Drupal	16
4.1.1 The 8 phases of bootstrapping	16
4.2 Thinking Drupal	19
4.3 Default cache system	24
4.4 Tests and Benchmarks	25
4.5 Concluding the analysis	27

5	Solving the problem	29
5.1	Designing the solution	29
5.2	Implementing the module	36
5.3	Enabling cache for authorized users	37
5.4	Replacing database caching	42
5.5	Replace session handling	44
5.6	Replace Panels caching	45
5.7	Benchmarks and tests	45
6	Conclusion	49
7	Appendix A	51

Introduction

1.1 Specifying the problem

When talking about Drupal in this project, it is the newest version of Drupal there is referred to which is Drupal 7, just so there will be no confusion. Drupal 7 will now be referred to as Drupal. Drupal is a system constructed for targeting a large group of developers both professional, and people setting up a personal website. This means there are some aspects of Drupal not quite build for high-traffic websites. One of the good reasons for Drupal being designed as it is, is because of the people working with Drupal development in their spare time, not wanting to spend alot of time getting familliar with a very complex object oriented system design.

Another aspect clearly marked by the decision of targeting a very broad aspect of the web community is the caching, system which is build using the database layer. This means that even if a request to a Drupal website can be delivered with a cached page, Drupal still needs to make a connection to the database and run a couple of queries fetching the cached data.

This is exactly what is the focus of this project. The problem of using the database for caching is when handling a very high load on the webserver each request needs to connect to the database which results in the database server being bombarded with requests, especially if a user logs onto the website, then Drupal just quit caching anything and needs to deliver a fresh page, each time. This

results in each Apache worker thread is calling an already overloaded database server, so the Apache process needs to wait on the database server to process the query, this is not good when having a lot of requests per second.

The scenario specified above is of course based upon the assumption of a setup where one server is running both Apache webserver and the database server, which is a very common setup. Another solution to the problem described is just to get more hardware, meaning buy a dedicated database server, setup a Varnish HTTP cache server in front of the webserver etc. which might be an easy solution, but not very scalable. The focus in this project is to figure out a way to lower the heavy dependency on the database and also make Drupal enable caching even if a request is coming from a registered user.

This project then boils down to the following elements

- Implement a new caching layer where the cache is in XCache instead of the database
- Implement a new session handler which saves the active sessions in XCache
- Extend the page caching so it is able to cache pages for authorized users

These three points are the expected outcome of this project, bundled together into a module so it can be installed easily on any Drupal installation.

1.2 Structure of the report

This chapter (1) is meant to be an introduction to the problem this project is based on. It is also meant to give the reader an overview of what the expectations for this project is, and when it is finished. Chapter 2 continues after this section with describing what type of model was used for this project, how the timetable looked like at first and how it actually ended up being and what sacrifices had to be made to be able to keep the deadline.

Chapter 3 is about planning this project, and how that part of the project was handled.

From chapter 5 and on is about solving the problem. To be able to understand the choices there were made this report will give a good introduction to how Drupal responds to a request and where exactly the weak elements in Drupal are, before going through the implementation of the solution. Last it is the tests and benchmarkings which is shown, to see if the implementation actually optimizes anything or not.

There is a lot of diagrams and drawings in this report. Most of them is done by the author. If any drawing or diagram not done by the author, the source of

the picture is shown below with the caption.

In the sections where there should be programming code such as when discussing the actual implementation, only the most relevant parts of the code will be shown, to reinforce the understanding of how the system works. This is with the purpose of avoiding large code-listings with a lot of elements not needed when explaining how something works.

At Appendix A there is a description of how to reach the websites used in this project and also a username and password for login in at the test site.

CHAPTER 2

Requirements Specification

Based on the problem specified in chapter 1, this chapter is about actually defining the requirements for a successful outcome of this project. When defining requirements for a new system it is a good approach to look at how the system is intended to be used. This is done by defining a set of use-cases. A use-case is an entity which defines an action the user can take while interacting with the system. For example while driving a car, a use-case could be "Turn on radio" or "Stop car". Those two examples is based on the driver being the actor. A system can have more than one actor. Lets stick with the car example and look at a passenger in the car. A passenger can also have the use-case "Turn on radio", but not the "Stop car" use-case, because the passenger does not have access to the pedals. A website can have a lot of different actors to consider because

use-cases can be dependent on the actor having a specific role, like forum administrator or a regular user on a forum website. In this project though, there is only 4 actors to consider, as shown in figure 2.1 below. There is two types of

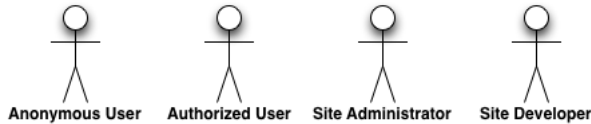


Figure 2.1: Figure with the 4 actors considered in this project

requirements for a system, Functional requirements and non-functional requirements. Functional requirements are the ones defined by use-cases and actors and are the ones specifying a specific function in the system. Non-functional requirements are more vaguely defined and is not typically something defined by how an actor interacts with the system, but more a requirement the system has to fulfill by itself. A non-functional requirement could be "Each request must be done withing 100ms" which is not defining any functionality on the website but defining that all the functionality on the website has to perform good enough to fulfill that requirement.

2.1 Functional requirements

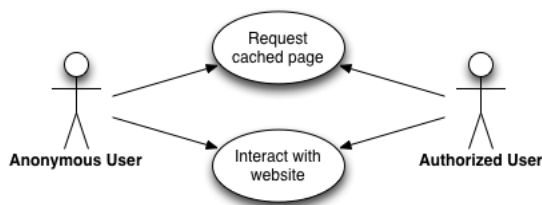


Figure 2.2: Use case diagram for anonymous and authorized users

2.1.1 Anonymous and Authorized users

This is the functional requirements for the Authorized and Anonymous users

Request cached page

Though requesting a cached page as an anonymous user is standard functionality in Drupal it is not the case for an authorized user. With the new caching system it is a requirement that an authorized user should be able to request a page without waiting for it to render everything from scratch.

Interact with website

When talking about interacting with the website it is important to understand that it is not in the sense of clicking around on different pages. Interacting with the website should be interpreted as the ability to generate content as a user, like commenting on articles and voting in a poll. When a caching system is not handling these things, a user always gets a cached version of the page until the cache expires, this means that when posting a comment the site is reloaded and if the caching system doesn't take it into account and deliver a fresh version of the page, the user can't see his own comment. That is why this is a requirement for both anonymous users and authorized users.

2.1.2 Site Administrator

The functional requirements for the administrators of the website.

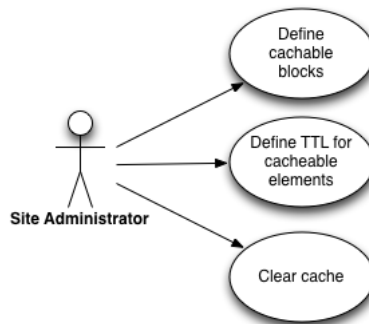


Figure 2.3: Use case diagram for a site administrator

Define cacheable blocks

Since the site administrator is the one setting up each page it is also the site administrator who defines the cache settings. This means that when the new cache system is implemented the site administrator should still be able to define the cache settings. In addition to the existing settings the new modules defines some more settings for when the system is caching for the authorized users.

These new settings is

- Enable user caching (Boolean)
- Cache pr. user (Boolean)
- Time To Live (TTL) on user cache (Integer)

Define TTL for cacheable elements

The system administrator needs to be able to set how long cached elements lives in the cache. This means both setting the cache on each element of a page, but also how long the cache should last for the whole page, now that page-caching has been enabled for authorized users.

Clear cache

When adding new content to the website or the site administrator is in need of pushing new content out to the users of a website, it is very handy to be able to clear all cache at once. Also, clearing all cache is something Drupal does at some points when an administrator changes some settings on the website, therefore it is a requirement for the new caching system.

2.1.3 Site Developer

Site developer is the actor representing the developers of the website.



Figure 2.4: Use case diagram for a site developer

Hook into cache system

In some cases there can be outside influence on the cache of a page. An outside influence could be that there is something on a rendered page which is defined by if the user has access to some content or not. This could be if a user has bought a livestream on a tv-website and there is a button saying "Buy Livestream" if user has no access and "Watch livestream" if the user has access. If the developer wants the page to be cached he needs to be able to hook into the system and modify some of the tokens.

2.2 Non-functional requirements

The non-functional requirements for this project are listed in this section

- Under no circumstances should core be hacked when installing the module into a Drupal installation. As a best-practice it is very strict that a developer should never hack the core.
- All the features in the ordinary caching system should still work. No cache settings from the original system must be dismissed, no need for administrators and developers to set all cache settings just because this module is installed.
- All content delivered through the system must never belong to another user or be inaccurate in any way. If it is not possible to deliver a cached page for the user, the system must exit and let Drupal deliver a fresh generated page.

CHAPTER 3

Project Management

When planning a project like this with a very specific timeframe it is important to be clear about how that time is spend. To do this it is necessary to break the project down to smaller tasks so it is easier to estimate how much time it should take to complete it. When having broken down the project into tasks the next step is to decide in what order the tasks needs to be done. Splitting the project into phases and deciding what phase task belongs gives the oppotunity for defining milestones for the project, more specific, the conftions for which each phase is done and when to move to the next.

This of course sounds alot like the waterfall model where when a phase or sprint is done, the team moves to the next without ever looking back and rethinking decisions based on new information. This project was planned based on the principles of agile software development. One of the principles in agile development is to revisit the design-iteration, development-iteration and test-iteration several time, one per iteration. This means that for each iteration of a project all of the above is visited to ensure each iteration has been thought throug, implemented and been tested fully. This gives a much better idea of how much time is left in the project and also gives the project manager the information to catch it early if he can see the things taking to long and if the project cant be done in time for the deadline.

3.1 Agile Development

Agile development is by itself not a specific method of running a software project. It is more a group of methodologies such as Unified Process (UP) and Extreme Programming (XP) which both are based on an iterative approach to software projects. Most agile methodologies are aimed at projects where there are more than just 1 person, but actually a team working the same tasks and milestones. Since only one person has been working on this project there is not a specific methodology used, but more a mix of the most important terms in UP and XP. Figure 3.1 shows the lifetime of an iteration. For each iteration in a project these 4 iterations are gone through. This ensures that the developers on a project using this model are informed about the requirements for this iteration, the implementation and testing and finally all members of the team are evaluating the iteration, which means everybody gets to reflect on what went wrong (It is very rare that everything in a software project, goes as planned).

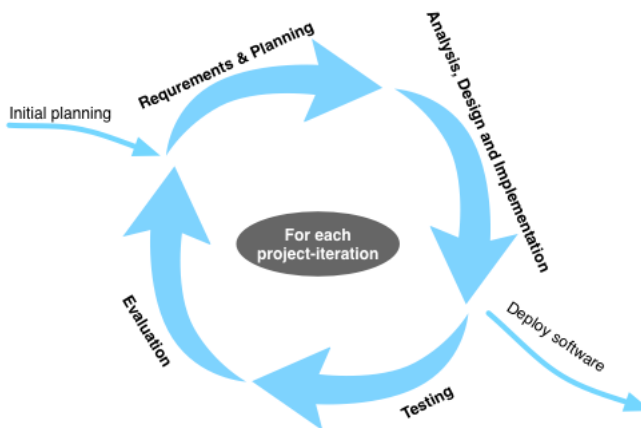


Figure 3.1: Diagram showing the idea of iterative development

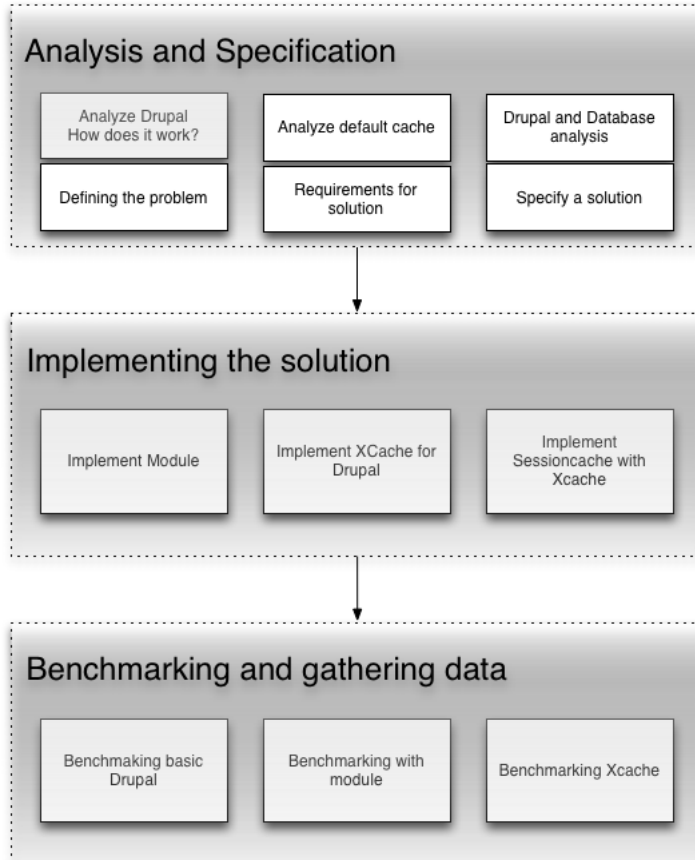


Figure 3.2: All iterations in this project

Figure 3.2 is a diagram showing the overall iterations and tasks for this project. Since the goal here is to get some kind of improvement in the performance of Drupal, naturally there is a big analysis iteration used for figuring out what exactly the problem lies and how to solve it. It is not before that has been done it is possible to actually implement anything. As shown in Figure 3.1, each of these iterations include those 4 iterations. This means that even though, there is not shown any test in Figure 3.2, there is both testing and evaluation included in each iteration.

Now when the phases has been defined it is possible to make a schedule for the project. The looks like this:

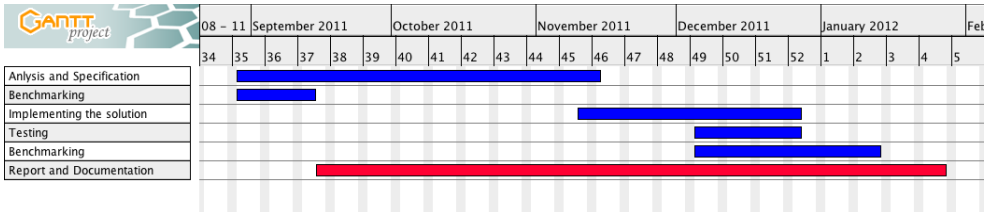


Figure 3.3: Showing project schedule. The scale is month and week numbers

Figure 3.3 shows that the analysis phase consumes a very large chunk of the time available in this project. The reason for this is Drupal is not a simple system to analyze since it is not using object oriented patterns or the object oriented features in PHP. It also shows two "Benchmarking" tasks on the schedule. This is because in the analysis phase there is also need for getting some benchmark for a Drupal installation with no alterations, this is with the purpose of getting reference data to be able to see if the new system gives any performance optimization.

How it went

The plan shown in figure 3.3 was very optimistic in the terms of development at documentation. The way it went was, it was alot harder to see how Drupal actually did the caching, and how to hook into the system and fix it. The first solution build were one where the core were hacked alot, to make it work. This of course did not go with the abition of making a module which could be downloaded from Drupal.org. This solution was removed and a new one was developer, which is the on being handed in as the product. The time used on the first module where core was hacked, should not be seen as wasted, because it meant that it now was very clear what had to be done, the problem was that it was not a part of the estimate, to do two implementations.

CHAPTER 4

Analyzing Drupal

In this chapter the main focus is to uncover how Drupal responds to a request and in that way, figure out what the issue with the caching mechanism is. When dealing with a webservice, which essentially is an application like all other things running on a computer. The only difference in this case, is the lifetime of the application which in this case is as long as it takes the code to generate its output. This means that when a request is received by Apache, the webserver spawns a worker process and that is what runs the code written for the website. When that code is done and has printed the output (The HTML in this case), Apache kills the worker thread and everything from that process is now gone. This means that it is not possible to save information for the next time the user ask for at page, inside the application as you normally would because the application is terminated as soon as apache has delivered the output to the user. To get a better understanding of how Drupal handles this, most of this chapter is dedicated to describing how Drupal is booted into a state where it is possible for it to generate the output. After running though the bootstrapping, the next section is how to setup a page in Drupal. This will explain the best practice of organizing pages and content and why this is so essential for the performance of Drupal.

4.1 Bootstrapping Drupal

Bootstrapping is the phase where Drupal is doing all the initial work. It is also here it is decided if it is necessary to bootstrap Drupal fully or if it is possible to serve cached content, since that is the topic of this project, the bootstrap phase is one of the main focuses in this project. Drupal is not written in any object oriented structure the need for defining variables and calling a lot of separate functions is huge. This means the size of the Drupal bootstrap.inc file is huge and contains just about 3300 lines of PHP code.

4.1.1 The 8 phases of bootstrapping

Each phase of the boot handles a specific task on the way to a full boot. The clever thing about the phases is that as a developer, it is possible to choose at what phase to boot, which gives the option to stop the bootstrap when the functionality needed has been loaded.

Phase 1 - Initialize configuration

This phase initializes the PHP environment with specific settings for the Drupal environment. Furthermore it loads the Drupal settings from settings.php, specifying everything from database information to the cookie and session names.

Phase 2 - Serving a cached page

This is where Drupal tries to serve a cached version of the requested page. If this is possible it will fetch the information from the location where it is saved and then serve it. After the cached information have been sent to the requesting user, Drupal ends its execution.

Phase 3 - Initialize variables

Drupal has a variable system available to the developer which makes it possible for the developer to save important variables and have them available at any time during the execution of the code.

Phase 4 - Initialize database layer

As a lot of other modern system, Drupal has a database layer available for to make it easier for developers to read and write queries for fetching data from the database. In this step of the boot that layer is initialized. The database layer of Drupal is one of the only parts of Drupal actually making use of the object oriented features in PHP. This means that the result of this phase of the boot is a database object, used in all queries in Drupal.

Phase 5 - Initialize session handling

With the variable system booted it is now possible for Drupal to load the session-handler. The session-handler can be either the custom implementation or, as in this case, a custom implementation. Drupal does not use the default PHP session-handler, when initializing the session-handler Drupal registers its own functions as replacements. This facilitates the way of having sessions stored in the database as Drupal does and also makes it possible to a user, never to get logged out of the system.

Phase 6 - The page header

When dealing with web-servers and browsers, the page headers is an important part of the process. The page headers tells the browser what type of content is served, if it is compressed and what type of compression used. Headers can also tell the browser not to keep a local cache of the served page or at least give the local cache a lifetime.

Phase 7 - Initialize the language

This is where Drupal is initializing all the multilanguage settings and functionality, but only if the site is registered to be a multilingual site. If not, the only thing this boot phase is doing is to invoke the hook `language_init`.

Phase 8 - The full boot

This is the final phase of the boot. When Drupal reaches it prepares to process a full page rendering. This means it boots everything from the path handling and theming system. Unfortunately this is also here Drupal is including a lot of core-files using the function 'require'. Of course Drupal needs all those files, but it is very slow to include them all with "require", so the only reason for booting Drupal into this phase is if in need of a full page render or need specific function, only loaded in this phase, like specific modules or core files.

Now the core of Drupal has been loaded and it is now time to load the activated modules. Drupal is bundled with a list of modules not directly associated with the core and can be deactivated if a developer or administrator decides to do so to save boot time or just do not want to make use of the module any more.

The structure of Drupal can be compared with the concept of building structure with Lego. Everything not inside of the core is made as modules which is hooked into the core system. As shown on the picture there is also different phases of the execution after the initial boot process. The diagram of course only gives an idea of how the Drupal lifecycle is, when a page is not served, cached. But it is good enough to give an idea of how the system is build.

Modules and hooks

One of the really popular elements of Drupal is the way it is possible to hook into core events. This feature gives developers the possibility to change and built almost everything they want, as long as they always use best practices and not, ever, hack the core. If a developer reaches a point where it is necessary to hack the core to implement some business logic, it is clearly a very slippery slope and he should go back and reconsider what to do.

How hooks work

When implementing a module for Drupal, the only way to interact with the system core is through hooks. Drupal has defined a large set of hooks which is invoked at a specific point in the lifecycle of the page rendering. When a module is registered as active, Drupal parses the .module file and registers which hooks implemented in the module which makes it possible for Drupal to invoke the hooks when needed.

When building a module for Drupal the basic idea is to have a .module file where

all the new functionality is contained. This means a file with a lot of function declarations.

The codeblock below shows an example of a hook implemented in a module called "examplemodule". This exact hook is used to define any blocks a developer wants to make available on the website. This shows the way to implement a hook is to name a function "modulname_hookname". It is very important to understand though that is the convention for writing hook functions, a hook is always referred to as "hook_hookname" ex. "hook_block_info".

```
function examplemodule_block_info() {
  $blocks = array();

  $blocks['exampleblock'] = array(
    'info' => "Example",
    'cache' => DRUPAL_NO_CACHE,
  );

  return $blocks;
}
```

In this project one of the key hooks is "hook_boot". This hook is important because it is called already in phase 6 (The Page header) of the boot process. This means it will be possible to implement an addition to the page-cache and handle caching for requests from users with an active session.

4.2 Thinking Drupal

Now the reader should have a basic idea of how Drupal is starting up and how developers interact with the core. The next step is to take a step back and look at the big picture. This means looking at how Drupal websites are built, how data is structured in the system and how to use that knowledge to get what is needed from the system.

Building content

Drupal is a content-based system where the basic idea is for something to exist in the system it has to be some kind of content. When defining types of content

in the system there is a certain structure the developer has to follow.

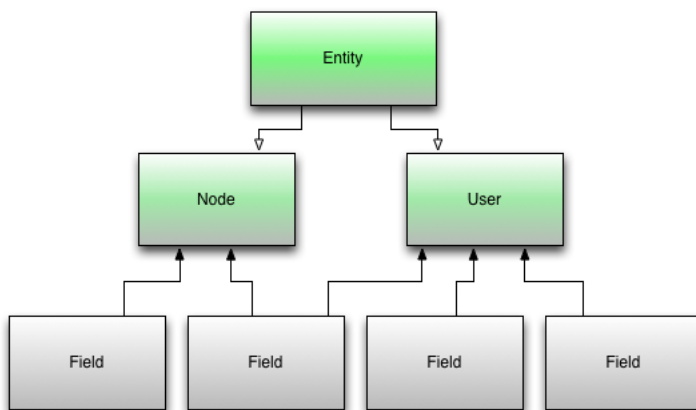


Figure 4.1: Figure showing how Entities, Nodes and Fields are related.

As seen on the figure above all content is considered entities. This means that users, taxonomy terms, nodes etc. inherit features from the basic entity type. In Object Oriented terms this would be the same as calling Entity the super class and nodes and users are children of that class. The same analogy cannot be made for the fields. The fields are attached to nodes and users by reference instead of inheritance which means that 1 field can be attached to more than 1 entity.

Figure 1.2 shows an example of a simple content-type representing an article with a heading, the article text and an image. Each field is attached to the entity and each field is specified as a type. The type represents both what datatype the contents of the field is stored as, in the database layer and the input element shown to the webmaster or journalist entering content into the system. This means that when a developer is creating a content-type no code has to be written, everything is done from the backend interface which makes it really easy and quick to add new types of content to a website, but also a little bit more tricky to keep a good structure of the data and not end up with a lot of redundancy in the datastructure.

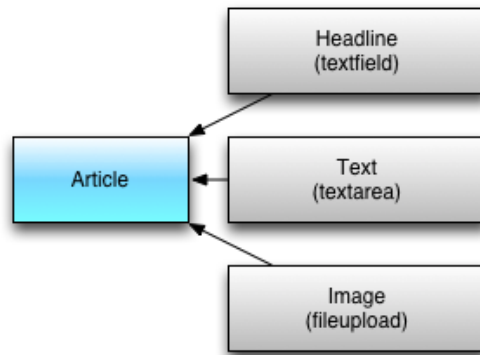


Figure 4.2: Figure showing a content-type with fields

Regions and Blocks

Although Drupal is a content-base system there will always be a need for elements on a page which is not directly content and blocks solves this problem. A block can be anything from a list of content to a facebook newsfeed. To be able to insert blocks in the website the developer of the theme needs to define some regions blocks can be inserted into. Regions is defined in the active theme. On

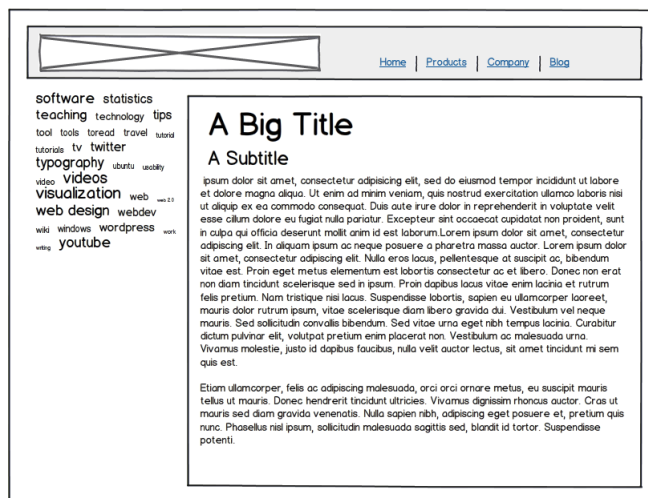


Figure 4.3: Figure showing an example article on a website

figure 4.3 there is an example of a article on a website containing an advertisement, a menu, a tagcloud and the article being read.

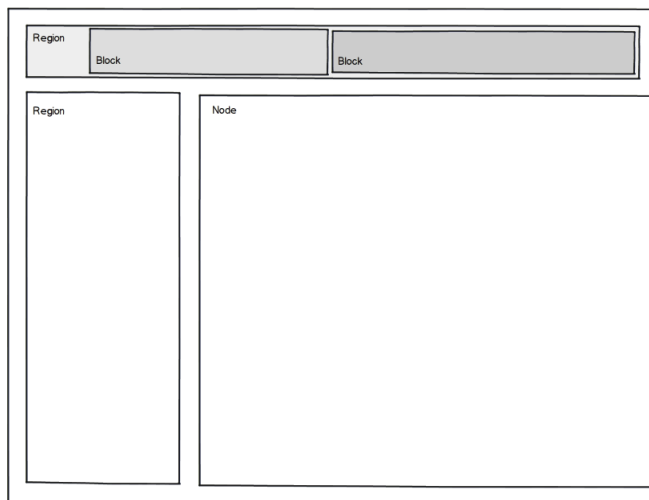


Figure 4.4: Figure showing how regions can be setup in Drupal

So, after the developer is done defining regions he can now set up the blocks needed for the website. The problem with using regions is they are defined for the whole website. This means that the blocks but in the top region in figure 4.4 is also there when looking at all other pages unless the developer is specifically telling Drupal which pages the block cannot be shown in. When doing that and having alot of differen elements for specific types of content, the regions gets filled up with alot of blocks for alot of different things, that is not a very structured way of organizing elements on a page.

From entity to page

That was that core parts of Drupal, but when developing websites for clients who is not very strong in the field of IT, it is just not good enough to rely on the basic Drupal backend functionality because the client wants to be able to rearrange elements on the frontpage or wants a page with a list of some specific content-types. That is why one of the most used modules are "Panels", which also in this projects is one of the key elements to make it work.

"Panels" is a module designed specifcly for the need to make pages with some specific content and also taking over the rendering of content-types. When

"Panels" takes over the rendering of a specific content-type it gives the opportunity for the site administrator to insert blocks to get rendered with the node content.

It is important to understand Panels does not change anything with Drupal blocks and the defined regions as shown earlier. Panel only changes the rendering of the node, so instead of rendering only the template file for the given content-type, panels gives the option of rendering other elements along with it. Though Panels does not change anything it still gives the possibility of rendering ordinary Drupal blocks inside the panel pane which is a huge advantage because then it is possible to only have element in the block system which is allowed to be shown everywhere such as website navigation which is everywhere on the website. By combining the two (Blocks and Panels) it is then possible to keep each panel clean and not be forced to insert the same elements everywhere, but only the elements important to the specific rendering.

If for example instead of only using regions and blocks the developer on the example from figure 4.3 installs panels and want to setup a panel for all the articles the solution would be to decide that the top regions should stay because that is the same all around the site, but the left region with the tagcloud is only for renderings of articles, that should be included into the panel pane. The red

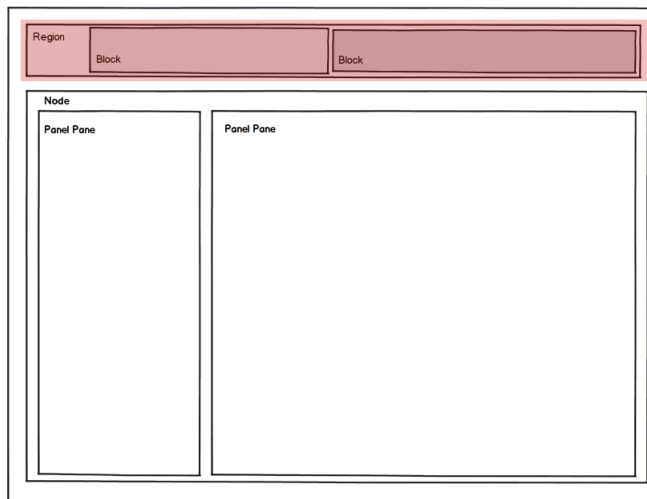


Figure 4.5: Example of a page with a panel

area on figure 4.5 is the original Drupal region which will be the same on all pages. On this version of the article page is instead of having the sidebar as a

region, it is now inside the node rendering which has been setup to be a panel. This means that the blocks inserted into this pane is only being shown in this panel, and not all around the site as with regions.

4.3 Default cache system

Now the reader should have at least a basic idea of how Drupal is initialized, how modules interact with the core and how content is structured inside Drupal. Knowing how the core of Drupal operates is needed for understanding how Drupal is caching content.

Since one of Drupals features is the possibility of install modules all the time. This means that Drupal needs to keep track of what hooks has been implemented in the different modules. This information is stored in cache along with alot of other information just like it. This means that even if Drupal cant deliver a cached page and needs to render everything, there is no need for spending alot of time searching all module to find out what hooks is available. Drupal does the same thing with the template files in the active theme. There is no need of searching through the folder every thime a page needs to be rendered, so that also exists in the cache.

In this project it is all about caching rendered output, eg. rendered HTML blocks or even whole pages. As seen in section 4.1 phase 2 of the bootstrapping is called "Page cache". The caching going on there is the one working for anonymous users. What it does is it checks if the user is not authorized. If that is the case, it looks in the cache for a version of the page requested, if there is a version not expired it just prints the cached data with the correct headers and exits the page request.

Caching of blocks currently is a part of Drupal that feature just requires the block to not depend on what user is currently viewing the page. So a block which changes based on some settings on your user cannot be cached in the default block-cache.

Where is cache stored?

Drupal stores all the cache directly into tables in the connected database. This means that every time Drupal interacts (Saves, Gets) with the cache the database has to evaluate queries to fetch the cached data. And that is not only the cached blocks and pages. It is all the cached data, even information of what hooks is implemented and what template files created in the theme. This results in quite a few queries pr. page request.

4.4 Tests and Benchmarks

This is the section where all the numbers for a clean installation will be presented. When benchmarking a website the only real interesting number is the amount of requests pr. second [Req/sec] the servers is able to get done on the given website. This means that if each request take a long time, the Req/sec is very low which is not good and vice-versa.

When benchmarking any type of application, it is very important that the parameters are the same for every type of test and if it change, there is only one parameter changing, if more than one parameter is changed at a time, there is no way to conclude anything from it.

All benchmarking have been done from a linux instance at the cluster at DTU Building 306 and the server being benchmarked is a development server owned by Dwarf A/S hosted by TDC Hosting in their datacenter.

Each benchmark have been done with the tool "Apache Benchmark" which gives the opportunity of setting the amount of concurrent requests, setting a session key which is very useful for this project and last, defining the amount of time the test should run. As clearly seen on figure 4.6 there is a very big difference

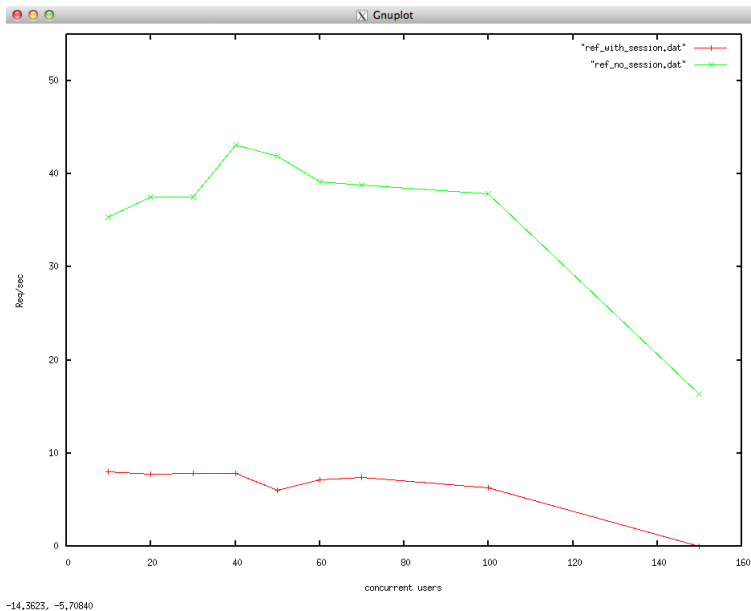


Figure 4.6: Benchmark of a clean Drupal. No optimization. Green is anonymous user, Red is authorized user. X is concurrent request and Y is Req/sec

in Req/sec when authorized users are requesting the page. In this testcase the req/sec is never above 8 for authorized users and the req/sec for anonymous users is up about 40 until the amount of concurrent request is too high.

What figure 4.6 shows is that the performance of Drupal when nothing have been done, is not very good for authorized users and when the concurrency of the request gets too much, the performance even drops very much for anonymous users. The nature of the problem being dealt with here is showed very clear here and the point of this project should be to get the red line up together with the green and perhaps fix the drop at the end, which should be the database server, not answering, making the requests very slow. These tests have been done with a timeframe of 60 seconds. This means that the only thing changing when doing the test is the amount of concurrent requests

4.5 Concluding the analysis

Following a request through Drupal is not easy because of the lack of object-oriented structure. This also means that Drupal includes a lot of files each time a page is requested, which cannot be very effective. Overall Drupal is not a very fast system, most content management systems are not very fast because it is the speed being sacrificed when the webserver has to evaluate the whole system each time, instead of just serving static HTML pages.

As clearly shown in the benchmarking part of this chapter, an authorized request is much harder to process than an anonymous. The reason for this is of course the site needs to handle and render elements according to the user. On a lot of websites it is only a small part of the website where the content is actually depending on the user logged in. This means that there is no reason for those pages not being cached and delivered quicker. Another thing also quite clear in the benchmarking chapter is all the requests made to the database, even for anonymous requests. This puts an unnecessary high pressure on the server, this can be quite a problem when the number of requests/sec increases to a point where the database server and Apache can't keep up anymore.

The results of this analysis are pointing towards the initial assumptions being correct, that it is in fact possible to make Drupal cache content for authorized users and in that way help keeping the hosting bill as low as possible for companies running a high-traffic Drupal website.

After this analysis it is now possible to propose the solution to the problem, it has been boiled down to 3 elements which are

- Implement a class with the interface "DrupalCacheInterface" caching in XCache instead of Database.
- Implement new version of the sessioncache which stores users sessions in XCache instead of in the Database.
- Extend the existing pagecache to users with a session for delivering content faster to registered users.

Solving the problem

This is the chapter where all the theory is being converted into a module to be installed into a Drupal installation. All the information needed to implement a more efficient version of the cache in Drupal has been gathered and now it is time to put it to use.

In the conclusion of the analysis chapter 3 elements were listed as a possible solution to the problem. This chapter is about how these 3 elements were implemented and what the thoughts behind it were.

5.1 Designing the solution

The first step is to evaluate all the information collected in the analysis and design a solution to solve the problem. As described earlier, the Panels module is pretty much used on every larger Drupal site to give the site administrator more flexibility when setting up pages. By taking advantage of the fact that Panels then is in charge of all rendering of blocks on the website and make it cache the rendering of each block. To do this, it will be necessary to override Panels own cache module, which also gives the opportunity to add more settings when the administrator sets up the blocks. When having a panel like the one



Figure 5.1: Example panel with one column and two blocks

in figure 5.1 it is possible to set cache settings on each block. These settings can be extended to include settings for how the system should behave when it is encountering a case where it is an authorized user requesting the block.

The following list shows what settings will be needed for the authorized-cache to work:

- Enable cacheable for users
- Define block as unique
- Set TTL on the cache for authorized users

Enable cacheable for users

A setting that defines if it is even possible to cache the block. Sometimes it is just not possible to cache a block because it needs to be live or it needs to be generated new elements at each request. This setting gives the opportunity to enable the cache for authorized users.

Define block as unique

A unique block is a block which has content based upon the user viewing it. When enabling this setting the system knows that and caches a version of that block, based on the userid of the user looking at it. So, the first time the user looks at the block, Drupal needs to render the whole page, but at soon that is done the block is stored in cache and Drupal can now deliver a cached version on the next page request.

Set TTL on the cache for authorized users

Panels have by default a TTL on the cached blocks, but although it would be possible to use that, it could be that the administrator wants to cache the user-based cache a little longer or shorter.

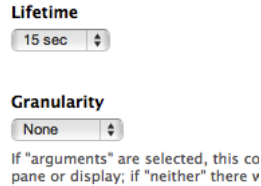


Figure 5.2: The default cache settings which should be extended

Settings and cache

At some point the system is checking if it can deliver a cached version of a page it needs to know what settings have been set for each block on the page. As it is now there is no way to get information on the blocks on the page. This is solved by using a class called "CacheTree" which is used to register the settings for each block on a page. The class in figure 5.3 has an array with all the block settings

CacheTree
Elements : array
+registerCachable (\$block) : void
+isPageCachable() : bool
+filterElements(\$data) : bool

Figure 5.3: Class handling used to save settings

for rendered blocks on a specific URL. After each time the cache tree has been loaded and changed, it is serialized into plain text and saved into XCache.

To add this feature it is necessary to implement a new version of the caching system build in to Panels. Panels luckily offers a feature of adding new caching method just by adding a file to the module. This means that it should be fairly easy to make a version which does the same thing as the original and add the functionality needed for building the cache tree. It is also in this new

implementation of the Panels cache it will be possible to add all the new settings describe earlier in this chapter.

XCache

PHP is an interpreted language like Python and Perl. When running a program written in an interpreted language, it is "indirectly" executed by an interpreter. This means that PHP is not compiled into machine code like programs written in C or C++ is. When PHP has been interpreted (parsed) it is compiled into an intermediate language also known as "operation code", which is what is being executed when running a PHP script.

Usually when compiling C or C++ code the compiler can be configured to make a lot of optimizations, and this is also the case with the interpreter. The interpreter is also configured to optimize the interpreted code when converting it to operation code. An optimization could for example be if the specific language is particularly slow in pre-incrementing an integer but fast in post-incrementing integers, the interpreter could switch out the integer incrementation if it has no influence on the execution of the program.

There is no need for interpreting the same code every time the PHP script

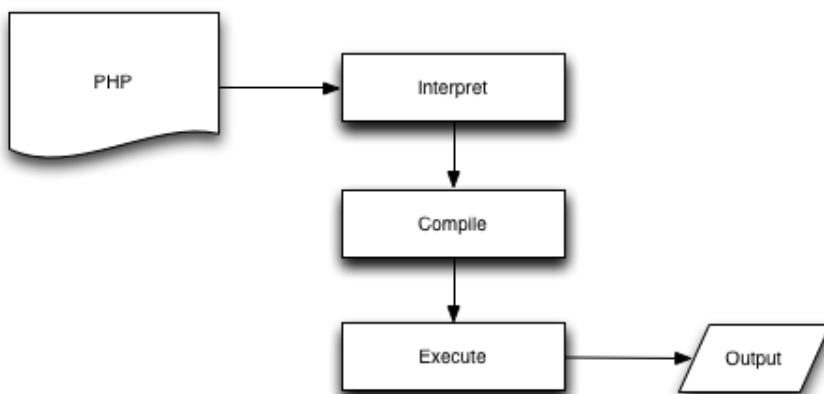


Figure 5.4: Executing php with no opcode cache

needs to be executed, and this is where XCache is used. There are other modules for opcode caching, in this project XCache is used for this purpose, because XCache also includes variable cache. This means that it's possible to save ele-

ments in to the memory of the server, which is very handy in this project. Usually when a program is done being executed on the server the compiled opcode is removed from the servers RAM so it is ready for the next program to run. When doing that, the interpreter needs to generate the opcode every time the program is executed. What XCache does is cache the opcode into its own memory. So, when the program is run again it just gets the opcode from XCache and saves all the interpreting. As long as the file is not changed the opcode is not rebuild. In this project it is not directly the opcode caching feature

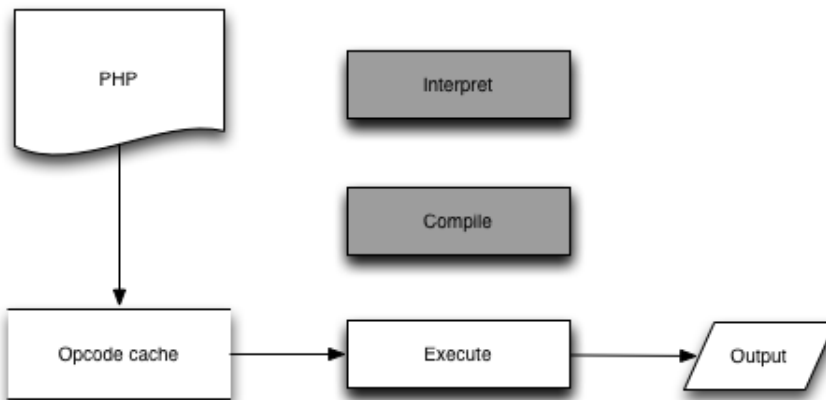


Figure 5.5: Executing php with opcode cache

of XCache but more the variable cache. The reason for choosing XCache in this project is that XCache contains both opcode caching and variable caching and because the author have been working with XCache before.

Variable Cache

Variable cache gives the opportunity of saving data directly into the memory(RAM) of the server. As ealier explained the process handling a process is killed after the output has been delivered to the user and all the utilized memory is cleaned. When saving variabed with XCache the data is stored in the memory allocated for the continuous webserver process, which also means that when the webserver is killed the xcache information is also removed from RAM, so as a developer you cannot rely on the data always being available and have to take action to prevent anything from going wrong, when the data is has been removed. Variables also has a specific amount of time they exists (TTL), which can be anything from 1 second to living indefinitely (Or, at least

as long the server is running), so whenever a developer is using variable cache it is very important to be aware of the possibility of the data not being there. The reason for removing variables is the same as with all other caching. By setting the variables lifetime to a specific amount of time, the developer ensures the renewal of the data, so no old data is hanging around in the cache. This also ensures that it is only recently used elements there actually exists in the cache.

Handling unique blocks

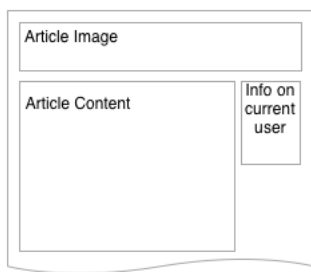


Figure 5.6: Example article page

Now all the settings is set in the system and they are available to the system when trying to deliver a cached page. The next step is to actual figure out a way to save the pages and deliver them to the users.

As seen in the analysis chapter there is some problem with just enabling page-caching for authorized users. The trick is to save as much as the page as possible by "cutting" out the elements unique to each user.

So when user A is viewing example page and there is no unique blocks on that page, he gets the cached version, just like with anonymous users. When the same user now looks at another page where ther is a unique block and there havent been cached a version of that block to user A, the whole page is rendered and the block is "cut" out and save in XCache. Since a panel is often representing a content-type, it is the same blocks for alot of nodes, which means that now, when user A is looking at the nodes of that type, the system has cached a page without that block, gets the page from XCache, gets the rendered block, pastes the block into the page and deliveres it to the user which should give a performance boost.

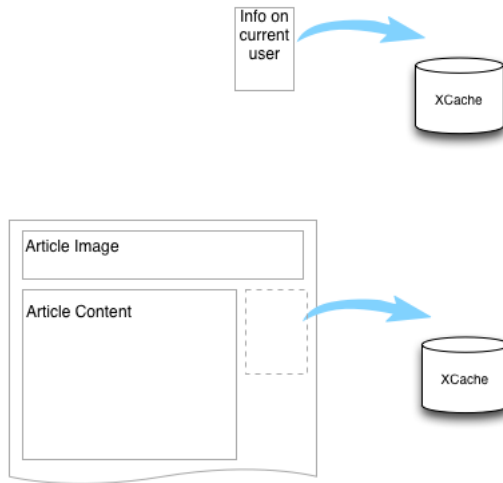


Figure 5.7: How the system would cache the example page

Saving user sessions in XCache

One of the other results of the analysis were to save the user sessions in XCache. What Drupal does is, to inform Apache of some function it needs to call when a session is opened or closed. This gived Drupal the opportunity to keep users logged in forever, because their session is saved in database. Since XCache can be emptied in any time, it would not be a good idea to only save the session here.

The solution is to keep the sessions both in XCache and in the database. This would mean that when a user is active on the website, the session is loaded from XCache at each request and then removed after som time when the user is no longer browsing the website. When the user then visits the site again, the session is fetched from the database in the first request and then stored in XCache for all the subsequent requests. A side-effect of this is the same data being stored two places, but this is OK, because the point is to use the database as little as possible, and this method helps doing that, without any risk to the data.

Final thoughts

Now it should be pretty clear what the thoughts behind the implementation were. Most of the chapter were about how to implement the page caching for

authorized users. This is because that is what the main focus is. Changing the cache layer for Drupal or caching the sessions in XCache is seen more as necessary changes so the enabling of the new page cache actually would give a performance boost. If the cache layer was kept the same, the database would still be starved of CPU time under high load, and then the new page cache would not be as effective.

5.2 Implementing the module

The module implemented for this project has been called "Booster V2". The module implements a series of hooks such as defining a configuration page in the backend where some general settings can be changed by the developer or site administrator.

```
function booster_v2_menu() {
  $items = array();

  $items['admin/config/booster_settings'] = array(
    'title' => t('Booster V2 Settings'),
    'description' => t('Settings for the booster module'),
    'properties' => "administrative",
    'page_callback' => 'booster_get_settings',
    'access_arguments' => array('access_administration_pages'),
    'type' => MENU_NORMAL_ITEM,
  );
  return $items;
}
```

The code above shows how to define a new page in the backed. This is done by making an array of elements where the URL is the key in the array and the second dimension of the array is all the settings. "page callback" defines which function should be called when the content of this page is requested. When the above defined page is rendered, it looks like this:

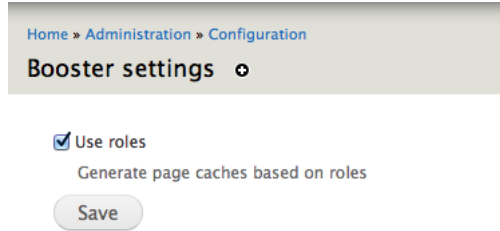


Figure 5.8: Screen capture of the settings page.

5.3 Enabling cache for authorized users

As earlier mentioned the "hook_boot" is one of the important ones in this project, this will show why.

As shown in the analysis chapter one of the bootstrap phases is the "Page header". When Drupal reaches this phase it invokes all "hook_drupal" implemented, before it continues to boot. This gives the opportunity to hook into the boot before it reaches uses anymore time to boot, which is what is the goal here.

Figure 5.9 gives an overview of how the module evaluates and decides when to cache and when not to. In this case the page is only cacheable if it is an authorized user requesting the page, and cache exists. It is very important to check if cache exists for the page and for the unique blocks.

As seen on the figure, if the page is cacheable and there is no user-unique blocks on the page, the cached content is just output to the requesting user.

If, on the other hand there is some user unique blocks on the page, the module needs to handle the insertion of these.

Is the page cacheable?

Determining if a page can be delivered from cache is relatively easy for anonymous users, but for authorized users is it a little more complicated. As earlier described, all blocks in a panel have the possibility of being unique to each user and that is what's used to determine if a page can be cached for authorized users. This has been implemented as a function in the "CacheTree" class in the func-

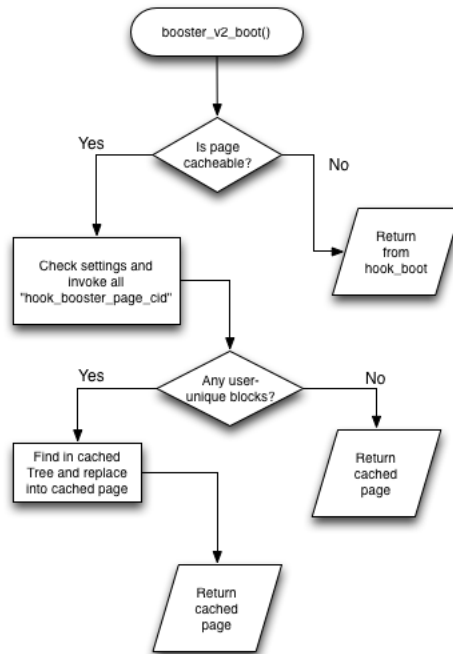


Figure 5.9: A flowchart of how implementation of "booster_v2_boot" works

tion "isPageCacheable" Here is the part that checks if an ordinary panel-page is cacheable:

```

$prefix = str_replace(".", "_", $_SERVER['HTTP_HOST']);
$prefix .= "_";
global $user;
foreach ($this->elements[$_GET['q']] as $elm) {
    if ($elm->cache['settings']['user_cache']['usr_pr_user']) {
        $cache =
        txcache_get($prefix."rendered_pane_".$elm->pid."_".$user->uid);
        if (!$cache ||
            !$elm->cache['settings']['user_cache']['usr_cacheable']) {
            return false;
        }
    }
}
}
return true;

```


What this function does it check if the block has been set to cache for authorized users and the check if there exists a version of the block for the specific user, in the cache. If the page does not exist in the CacheTree or if its just not cacheable, then the function will return boolean false and boolean true if its cacheable.

Saving a rendered block

To be able so save the rendered block from the panel it was needed to insert some code into the default display renderer. Panels have a way to create custom implementation of the display renderes, but there was not enough time to do that in this project, because Panels is very poorly documented, so the only solution was to inject it into there the default display rendered actually renderes the element. The code looks like this:

```
if (!isset($_COOKIE[session_name()]) ||
    strstr($_GET['q'], "admin") ||
    strstr($_GET['q'], "ajax") ||
    strstr($_GET['q'], "node/add") ||
    (strstr($_GET['q'], "user") &&
    (strstr($_GET['q'], "edit") ||
    strstr($_GET['q'], "register")))) {
    //do nothing
}
else {
    if ($pane->cache['settings']['user_cache']['usr_pr_user'] == 1) {
        $prefix = str_replace(".", "_", $_SERVER['HTTP_HOST']);
        $prefix .= "_";
        xcache_set($prefix."rendered_pane_" .
            $pid . "_" . $user->uid, $content);
    }
}
```

This block of code looks at the settings of a block and saves the HTML rendering of that block if the block is set to be unique, and of course if it is on a non-system page.

Saving a page

If the page has been determined to be cacheable it is now time to save a version of the page. This is done in the new cache layer for Drupal. Drupal caches page

cache by the requested URL. This means that in the cache layer it is possible to know when saving a page cache and the part checking this, looks like this.

```

if (strstr($cid, "http://")) {
    $tree = unserialize(xcache_get($prefix . "tree"));
    $data['body'] = $tree->filterElements($data);

    if (!is_string($data)) {
        $fields['data'] = serialize($data);
        $fields['serialized'] = 1;
    }
    else {
        $fields['data'] = $data;
        $fields['serialized'] = 0;
    }
    global $user;
    if ($user->uid > 0) {
        $v = variable_get("booster_use_cache");
        $d = "";
        if ($v == "1") {
            $cid.= " ".join(":",array_keys($user->roles));
        }
    }
    $addons = module_invoke_all("booster_page_cid");
    foreach ($addons as $add) {
        $cid.= " ".$add;
    }
}

```

As seen on line 2 in the listing above a function in the CacheTree class called "filterElements" which removes all unique elements and replaces them with some tokens. After that, the cache system goes on and invokes the new hook implemented to get the final cacheId (CID) and the saves the page into xcache. The page with tokens looks like this: As seen on figure 5.10 the content under "Home" has been replace with a token, because that block have been set to unique as shown on figure 5.1

Figure 5.11 shows the page when rendered where the correct blocks have been inserted. That is done like this:

```

foreach ($this->elements[_GET['q']] as $elm) {
    if ($elm->cache['settings']['user_cache']['usr_pr_user']) {
        $cache =
            xcache_get($prefix."rendered_pane_".$elm->pid."_".$user->uid);
    }
}

```

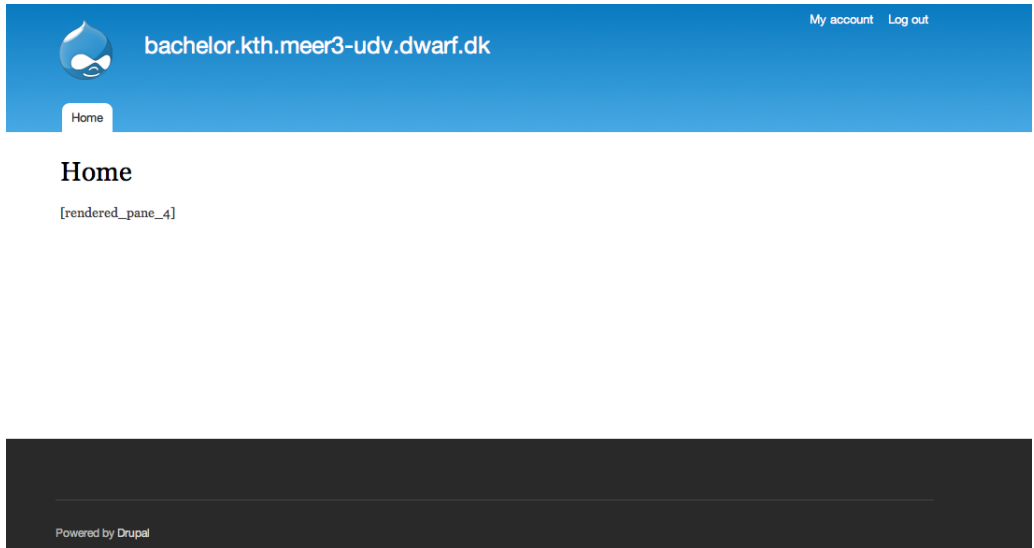


Figure 5.10: Screenshot of the page where unique blocks have been replaced with tokens

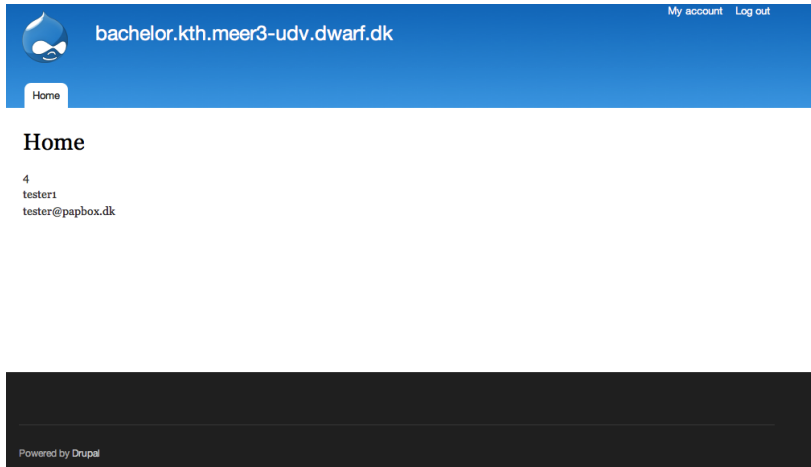


Figure 5.11: Screenshot of the page where unique blocks have been inserted to the specific user

```
$body =  
  str_replace(["." rendered_pane_ ". $elm->pid. "], $cache, $body);  
}
```

```
}
```

Shown on the code listing is the code looping through all the elements registered in the CacheTree class and inserts the renderings for the user.

5.4 Replacing database caching

When replacing a core component in Drupal it needs to be registered somewhere, so Drupal knows not to use the default implementation. When replacing the cache system with another one it is required that the developer defines what parts of the cache should be using the new system instead of the old one. Drupal has all the settings defined for a specific installation in the settings.php file. This means that when wanting to tell Drupal to use the new caching implementation it is in this file it is done. Just like this:

```
//Making sure Drupal includes the new class
$conf['cache_backends'][] =
    './sites/all/modules/booster_v2/booster_v2.inc';

//Define what class to use
$conf['cache_class_cache'] = 'DrupalXcache';
$conf['cache_class_cache_page'] = 'DrupalXcache';
$conf['cache_class_cache_field'] = 'DrupalXcache';
$conf['cache_class_cache_views'] = 'DrupalXcache';
$conf['cache_class_cache_menu'] = 'DrupalXcache';
$conf['cache_class_cache_path'] = 'DrupalXcache';
$conf['cache_default_class'] = 'DrupalXcache';
```

As soon as these settings has been set in the configuration array, Drupal uses the newly defined class as the cache system. The DrupalXcache class is implementing the interface DrupalCacheInterface. The reason for using an interface is so Drupal can be sure all the needed functions is defined. It also makes programming a new class very easy for the developer, since the interface defines the functions, the only thing the developer needs to do, is implement the functions and Drupal takes care of calling them. As figure 5.10 shows, DrupalXcache is implementing the DrupalCacheInterface interface. The only difference in this implementation from the original is when saving the cache. The original cache system is saving the data into the database with some SQL queries. DrupalXcache saves it right into XCache with a "xcache_set" function call. The same

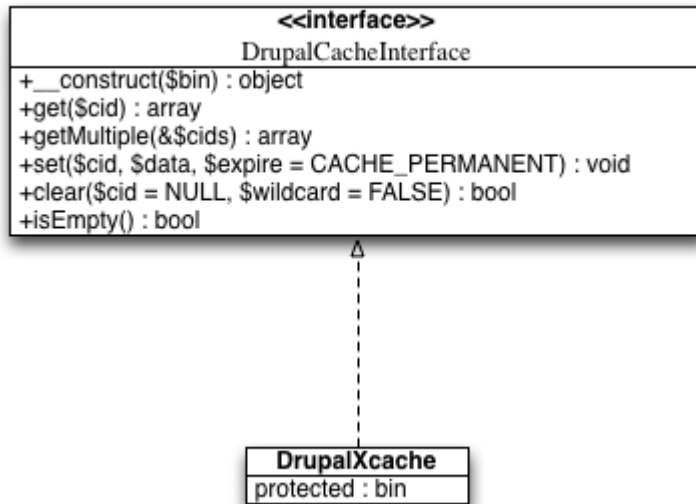


Figure 5.12: DrupalXcache and its interface

is when getting the cache from storage, the original gets it from a table in the database and DrupalXcache gets it from XCache with a "xcache_get" call.

```

function getMultiple(&$cids) {
    $cache = array();
    try {
        foreach ($cids as $cid) {
            $c = xcache_get($this->bin . "_" . $cid);
            if (!$c) {
                continue;
            }
            $c = unserialize($c);
            $item = $this->prepareItem((object)$c);

            if ($item) {
                $cache[$cid] = $item;
            }
        }
        $cids = array_diff($cids, array_keys($cache));
        return $cache;
    }
    catch (Exception $e) {
        return array();
    }
}
  
```

```

    }
}

```

The code above shows the implementation of the function "getMultiple" which is used to get multiple cache elements at once. And as shown, there is not run any queries or anything in this function, just a simple "xcache_get" call to get the elements from XCache. If cache exists it returns the elements in an array and then update the input array to tell what elements was not available in the cache.

5.5 Replace session handling

Replacing the session handling in Drupal is done in the same way when telling Drupal to use another class for handling cache and looks like this in the settings.php file:

```
$conf['session_inc'] = './sites/all/modules/booster_v2/session.inc';
```

The two main functions which is interesting in this implementation is "_drupal_session_read" and "_drupal_session_write". The first function is called when procession a request and the user has a session. Beware that anonymous users also can have a session because they might have defined som elements in the SESSION array. This is done like this:

```
function _drupal_session_read($sid) {
    global $user, $is_https;
    drupal_register_shutdown_function('session_write_close');

    $no_ssl = substr(session_name(), 1);
    if (!isset($_COOKIE[session_name()]) && !isset($_COOKIE[$no_ssl])) {
        $user = drupal_anonymous_user();
        return '';
    }

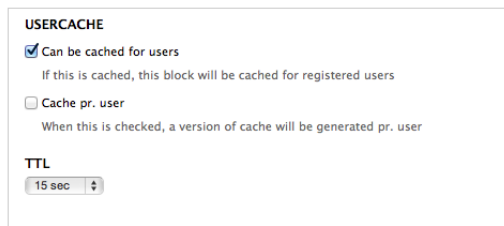
    //Load from XCache
    if ($is_https) {
        if (!$user) {
            if (isset($_COOKIE[$no_ssl])) {
                $user = unserialize(xcache_get("ssid:". $sid));
            }
        }
    }
}
```

```
}
else {
    $user = unserialize(xcache_get("sid:". $sid));
}
```

The codelisting shows how it evaluates if there is a session and then try to load from XCache. If there is found a session in XCache it continues to load the roles for the user, and then return the session to Apache. If there is no session available in XCache, it will try to load from the database and return the result of that. If there is no session at all, it will just return a anonymous session, which will be empty. To see the rest of the function, that is available on the disc handed in with this report.

5.6 Replace Panels caching

The point of implementing a new Panels cache is to be able to save the settings into panels own setting system, and ofcourse take the opportunity of saving the panel objects into XCache instead of the database, which is done by replacing the original query with an "xcache_set" function call when setting the element and an "xcache_get" call when wanting to fetch the saved element. The interesting part of this implementation is the settings, which is a crucial element of the totalt solution. Figure 5.2 shows how the panels settings was.



USERCACHE

Can be cached for users
If this is cached, this block will be cached for registered users

Cache pr. user
When this is checked, a version of cache will be generated pr. user

TTL

15 sec ▾

Figure 5.13: The settings added to the panels cache settings.

5.7 Benchmarks and tests

Everything from the analysis has now been implemented. Now it is time to see if it has actually had any effect on the performance and more important

have it had the expected effect. Of course the first thing to check, is if it is actually working. This have been done by implementing a test module called "Proofing". The only thing this module does is generating a block which prints out the information of the user, currently viewing the page where the block is added. By doing this, it is then possible to see if the user is getting the correct version of the cached block or not. An example of this can be seen in figure 5.11 which shows the frontpage where a unique block have been inserted.

Benchmarking the solution is done in exactly the same way as in the analysis chapter. It is done from a linux instance at the DTU server in 306 and the new implementation is on a development server owned Dwarf A/S hosted at TDC Hosting. All the benchmarks have been done in an timeframe of 60 seconds, which means that the only thing actually is changed from test to test is the number of concurrent requests.

To recap how the test looked like, before the new module were implemented that is shown at figure 5.14 and the new benchmark, done with the new changes is shown in figure 5.15. The new benchmark clearly shows an improvement

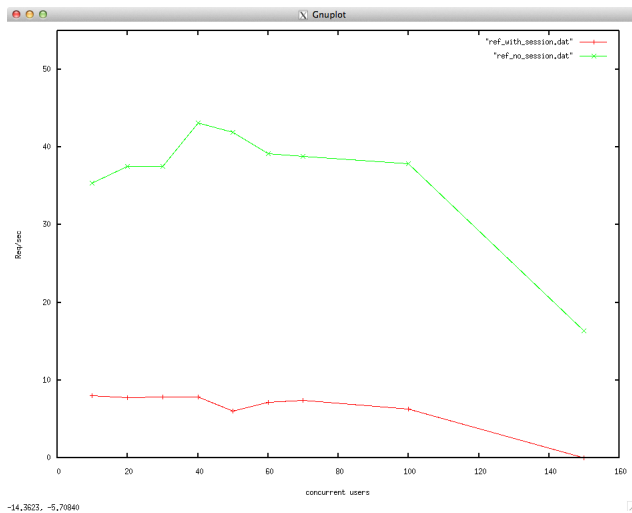


Figure 5.14: The benchmark before the changes. Green is anonymous and red is authorized users

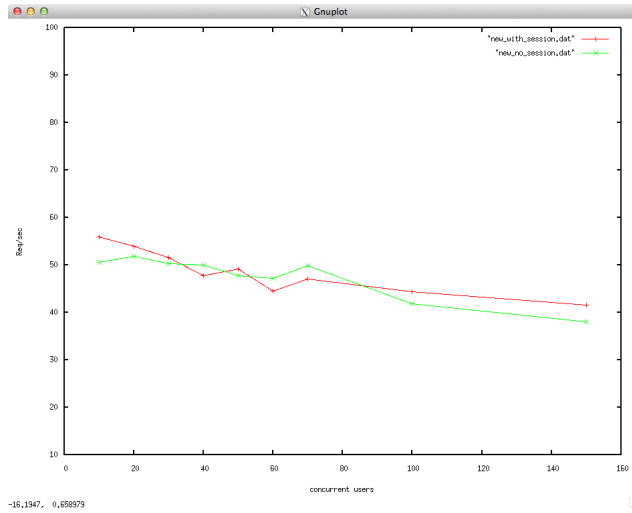


Figure 5.15: The benchmark with the changes. Green is anonymous and red is authorized users

to the performance of Drupal, also for the anonymous users. As on figure 5.14 the server can clearly better handle the more concurrent requests with the new system as predicted. The lines cross each other a couple of times so it seems like the authorized cache performs better than the anonymous. This is not the case. If anything their performance should be about the same, where the authorized cache should be a little slower overall. The reason why the lines are crossing could be because of some other processes running at either server, slowing the test down. But they are very consistent and very similar.

Conclusion

The last benchmarks shows a significant improvement in the performance of Drupal when under pressure from alot of requests. The original plan was to build a module which would be ready for publication on Drupal.org, which probably was a very ambitious goal, and did not completely succeed. The state of the product is right now that it can be installed on a Drupal installation, but there is still some minor bugs and the module is generating some notifications when a user is logging in or out, so not quite ready for production.

Although the module is not ready for production it should not take alot of time to make it ready for publication. This project have proven that the method chosen here is actually working and is improving Drupal alot. Ofcourse it is clear that not all pages can be cached because of the nature of the content, but in the terms of this project, the goal was always to prove that just because a user have logged on to a website, cache can still be used. The other very surprising element the benchmarkings are showing are the fact that when replacing the database caching with a XCache layer Drupal seems to be able to handle alot more concurrent requests.

So, what this project shows it that it definitely is possible to make Drupal perform better under pressure, also for authorized users. The module under development in this project can help Drupal scale alot better, which means that it can help companies to save money on hardware. With those results the future of the module looks promising, and with a month or two more of development it should be ready for production sites.

Appendix A

Location of websites

If you want to look around on the websites yourself the website where the module is installed is available on <http://bachelor.kth.meer3-udv.dwarf.dk> which is the site I've been using for testing. It is possible to login with username: tester1 pass: test. Which should give the opportunity to see the module in action.

For a reference test there is a Drupal which have been setup exactly the same way, just without the improvements. This can be found at http://drupal_ref.kth.meer3-udv.dwarf.dk

Those two websites will be available until the final grade have been given to this project.

There is also an administrator user available with the username: admin and password: 123wqe so you can see the backend system.

