BACHELORPROEF

Performantie van een PHP-Server verbeteren door optimalisatie van instellingen en caching methodes

Bachelor	Toegepaste Informatica
Keuzetraject	Software Engineer
Academiejaar	2019 - 2020
Student	Evert Albert
Interne begeleider	Kristien Roels
Externe promotor	Davy De Coster

howest.be

BACHELORPROEF

Performantie van een PHP-Server verbeteren door optimalisatie van instellingen en caching methodes

Bachelor	Toegepaste Informatica
Keuzetraject	Software Engineer
Academiejaar	2019 - 2020
Student	Evert Albert
Interne begeleider	Kristien Roels
Externe promotor	Davy De Coster

howest.be

Toelating tot bruikleen

De auteur(s) geeft (geven) de toelating deze bachelorproef voor consultatie beschikbaar te stellen en delen van de bachelorproef te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de bepalingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze bachelorproef.

The author(s) gives (give) permission to make this bachelor dissertation available for consultation and to copy parts of this ma bachelor ster dissertation for personal use. In all cases of other use, the copyright terms have to be respected, in particular with regard to the obligation to state explicitly the source when quoting results from this bachelor dissertation.

9/06/2020

Woord vooraf

U staat op het punt om mijn bachelorproef te lezen. Deze bachelorproef zal handelen over het verbeteren van de performantie van een PHP-applicatie dankzij optimalisatie van instellingen en caching methodes. Ik kreeg hierbij hulp van de mensen rondom mij op mijn stageplaats, school en thuis.

Als eerste wil ik Davy De Coster bedanken, ontwikkelaar bij de firma Ocular. Meneer De Coster ondersteunde mij tijdens mijn stage, waar ik voornamelijk PHP-applicaties hielp ontwikkelen. Ik wil hem ook zeker bedanken voor de controle en ondersteuning tijdens het maken van deze bachelorproef. Dankzij meneer De Coster kreeg ik een groeiende interesse in PHP-ontwikkeling.

Ik zou ook graag de firma Ocular bedanken om mij met open armen te ontvangen tijdens mijn stage en mij te betrekken in hun productieproces.

Vervolgens wil ik mevrouw Kristien Roels bedanken als stagebegeleidster en voor het opvolgen van mijn bachelorproef. Ook wil ik mevrouw Heidi Terryn bedanken voor de hulp bij het verbeteren van mijn bachelorproef.

Daarna zou ik mijn ouders willen bedanken voor de kans die ik kreeg van hen om een richting te kunnen volgen die mij kan boeien en waaruit ik met veel enthousiasme mijn toekomst zou willen opbouwen.

Ten slotte wil ik mijn verloofde Elsa Madoe bedanken. Zij motiveerde mij om door te zetten en is altijd een grote steun voor mij.

Evert Albert

Brugge, 9 juni 2020

Samenvatting

In een samenleving waar alles online staat en alles steeds sneller moet gaan, is er niets frustrerender dan wanneer je moet wachten op trage applicaties en websites. Dankzij het gebruik van een cache en de juiste instellingen van een PHP-server, kan dit laadproces drastisch versneld worden. Zo houd je gebruikers tevreden doordat ze steeds de gewenste inhoud snel te zien krijgen.

In deze bachelorproef wordt onderzocht hoe je het best aan de slag gaat met een cache in de programmeertaal PHP. Vervolgens wordt ook onderzocht welke instellingen kunnen aangepast worden op de server om een applicatie zo efficiënt mogelijk te laten werken.

Doorheen de bachelorproef wordt kennisgemaakt met de verschillende methodes en oplossingen die de PHP-gemeenschap te bieden heeft om zo met behulp van caching en gepaste instellingen een snelle en betrouwbare applicatie op te bouwen.

Sleutelwoorden: PHP – Caching – Instellingen – Opcache

Abstract

In a society where everything is available online and everything must go fast, nothing is more annoying than when you have to wait until slow applications and websites load. Thanks to the use of a cache and using the correct settings on your server, this loading process can be drastically improved. This way you can keep your users satisfied by showing them the desired content as fast as possible.

During this thesis there will be thoroughly researched how to work with caches in PHP. Research will also be done on how to use the correct settings to make an application run as smooth as possible.

Throughout the thesis many different methods of caching will be examined and there will be looked at solutions the PHP community has to offer for building a fast and reliable application.

Keywords: PHP – Caching – Settings – Opcache

Verklarende woordenlijst

Bytecode	Bytecode is een tussencode die staat tussen de, voor mensen leesbare, code en machinetaal. Deze vorm van code is onafhankelijk van platform. Dit wil zeggen dat op elk toestel of elk besturingssysteem hetzelfde resultaat zal leveren. Bytecode in PHP ontstaat wanneer de code gecompileerd wordt.
Dynamische webapplicatie	Een dynamische webapplicatie is een webapplicatie die data bevat die specifiek voor de gebruiker gegenereerd wordt en op elk moment kan wijzigen. Dezelfde webpagina ziet er niet iedere keer dat ze opgestart wordt hetzelfde uit.
Flushen	Gegevens verwijderen uit de cache.
Hit ratio	Een verhouding van het aantal cache hits tegenover het aantal cache misses. Een cache hit ontstaat wanneer data in de cache gevonden wordt. Een cache miss ontstaat wanneer iets niet in de cache werd gevonden.
Key-value pair	Een key-value paar is een combinatie van 2 waarden. De eerste waarde is een sleutel. Deze sleutel is gekoppeld aan een waarde. De waarde kan opgezocht worden aan de hand van de sleutel.
Load balancing	Verdelen van ballast door gebruikers/aanvragen over verschillende servers.
Open source	Open source wil zeggen dat de broncode openbaar beschikbaar is en dat iedereen die dat wenst de broncode kan gebruiken en toevoegingen kan doen.
Overhead	Overhead is het extra verbruik aan rekentijd, werkgeheugen, bandbreedte of andere bronnen die nodig zijn om een doel te bereiken.
Stale data	(Engelse uitspraak) Stale data is een collectie van gegevens die niet meer de meest recente versie van deze gegevens is. Dit is bijvoorbeeld een collectie gegevens uit een databank in de cache, waar al aanpassingen op gedaan werden.
Windows remote UNC	Windows Remote UNC staat voor Windows Remote Universal Naming Convention en is een naamgevingssysteem om bestanden, gedeelde mappen en printers te raadplegen in een Local Area Network

Inhoudsopgave

Overzicht tabellen

1 Inleiding (Ocular)

Ocular beoogt om hun publiek onder te dompelen in interactieve ervaringen. Ze voorzien verschillende oplossingen op vlak van IT. Dit zijn niet alleen softwareoplossingen, maar ook ondersteunen ze bij de installatie van de nodige hardware. Bijvoorbeeld in musea kan Ocular een totaalpakket aanbieden waar de projectieschermen, tablets en geluidinstallatie samen met de bijhorende software ter plaatse worden geïnstalleerd.

Wanneer een nieuw project opgestart wordt bij Ocular, kan de klant het volledige proces meevolgen en helpen sturen, zodat er een oplossing op maat kan aangeboden worden. Deze oplossing is niet alleen op maat van de klant, maar zal ook een creatieve oplossing zijn. Zo zullen de toeschouwers van het eindproduct de ervaring die hen aangeboden wordt niet snel vergeten.

Er wordt een concept uitgedacht voor de klant waar gewerkt wordt met verschillende middelen zoals visuele effecten, audio, games, virtual reality, augmented reality, etc. Het product wordt daarenboven nog eens na afwerking ondersteund zodat iedereen er zo lang mogelijk van kan genieten.

Ocular kan ingezet worden voor allerlei doelpublieken. Enkele voorbeelden hiervan zijn musea, beursstanden en showrooms. Een van de projecten waar Ocular mee bezig was op het moment van mijn stage was bijvoorbeeld ook een interactieve klimmuur.

U kan het niet zo zot bedenken, of Ocular heeft wel een oplossing voor u of uw project! Voor meer informatie over Ocular kan u natuurlijk ook terecht op hun website www.ocular.be

2 Onderzoeksvraag

Ik heb altijd interesse gehad in hoe programmeertalen werken en hoeveel vrijheid programmeertalen geven in de manier van ontwikkelen van applicaties. PHP heeft een zeer rijke geschiedenis in zijn ontwikkeling. Sinds het begin wordt er gewerkt met feedback van de gemeenschap en wordt bij iedere update gewerkt aan performantie.

Ik vroeg mij af hoe deze programmeertaal telkens met iedere update zijn performantie zo veel kan verbeteren. Hiervoor wil ik onderzoek doen naar wat er precies voor zorgt dat deze snelheden steeds verhoogd kunnen worden. Welke methodes gebruikt PHP hiervoor? Wat kan u doen als gebruiker om de snelheid van uw applicaties te perfectioneren?

Heel wat van deze verbeteringen zijn te danken aan het optimaliseren van caching methodes en de gebruiker de vrijheid te geven om zijn server in te stellen naar zijn noden en de noden van de applicatie.

Ik kwam op het idee om over dit onderwerp te schrijven dankzij een tip van mijn stagementor Davy De Coster. Hij gaf het idee om onderzoek te gaan doen op caching in PHP en aanpassen van instellingen. Dit onderzoek valt ook samen met mijn stage waar ik regelmatig in contact kom met applicaties die ontwikkeld worden in deze taal. Applicaties moeten zo efficiënt mogelijk op allerlei verschillende soorten hardware kunnen werken. Daarom koos ik als onderwerp “**Performantie van een PHP-server verbeteren door optimalisatie van instellingen en caching methodes**”.

3 Introductie in caching

3.1 Wat is caching?

Applicaties werken vaak met data. Deze data moet van ergens komen, dit kan bijvoorbeeld zijn uit een databank. Om deze gegevens op te halen en te verwerken moeten instructies uitgevoerd worden in een programmeertaal zoals SQL of PHP. Deze uitvoering van instructies kan vaak lang duren als veel data verwerkt moet worden.

Om geen tijd te verliezen kan het resultaat van de instructies tijdelijk bijgehouden worden in een specifieke opslag in onder andere het RAM-geheugen van de server waarop de instructies uitgevoerd worden. Wanneer een ander proces dezelfde data nodig heeft kan ze snel geraadpleegd worden zonder dat de instructies opnieuw uitgevoerd moeten worden. De gegevens bewaren in en tijdelijk geheugen heet caching.

3.2 Waarom wordt er gebruik gemaakt van een cache?

Wanneer geen gebruik gemaakt wordt van caching, kan data ophalen een tijdrovend proces worden. Dit zorgt voor frustraties bij de gebruiker of kan zelfs een potentiële klant wegleiden van een programma of website die iemand probeert aan te bieden.

Om een kort voorbeeld te schetsen; wanneer een webwinkel wordt geraadpleegd, en de klant zoekt een lijst op van de producten, krijgt hij deze liefst snel te zien. Als hij 10 seconden moet wachten totdat de server alle producten heeft opgehaald uit de databank en deze dan pas toont op het scherm, bestaat de kans dat de bezoeker de site al heeft verlaten op zoek naar een andere verkoper.

Caching is niet enkel handig voor de eindgebruiker, maar ook voor ontwikkelaars die een applicatie op een server moeten ontwikkelen en onderhouden. Zo kan je met caching netwerkverkeer zoals API-calls verminderen. Dit is voordelig voor de eigenaar van een server. Hij kan hierdoor meer doen met dezelfde server doordat de extra rekenkracht vrij komt op de server.

3.3 Wat is cache invalidatie?

De gegevens die worden opgeslagen in de cache zijn enkel maar een kopie van de originele data. Om ervoor te zorgen dat telkens de laatste versie van de informatie getoond wordt, moet gebruik gemaakt worden van cache invalidatie.

Invalidatie garandeert dat de data in de cache steeds de meest correctie versie is van deze gegevens. Dankzij de invalidatie weet de toepassing dat hij oude gegevens niet meer mag gebruiken en ze mag vervangen door nieuwe gegevens. Om aan cache invalidatie te doen bestaan verschillende methodes. Je kan bijvoorbeeld een tijdsperiode mee geven die bepaalt hoe lang data mag bewaard worden in het geheugen. Meer hierover kan gelezen worden in hoofdstuk 6 over de cache strategieën.

4 Cache soorten

4.1 User cache

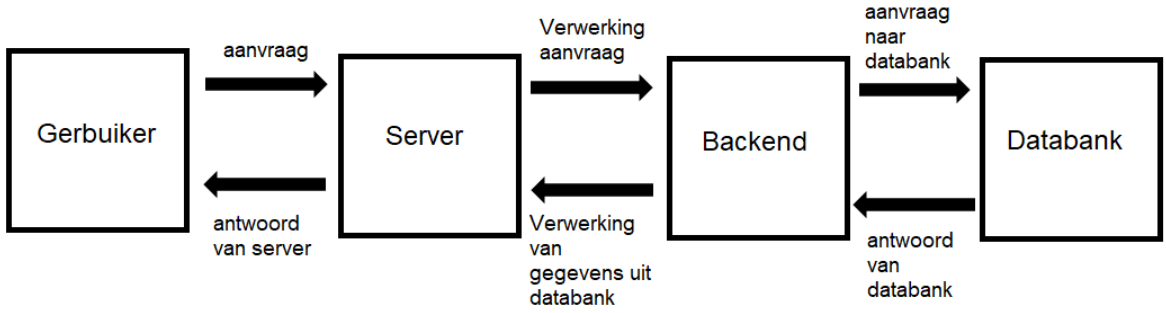
De user cache is een vorm van caching die gebruikmaakt van het geheugen van de client. Deze cache slaat lokaal informatie op die ze van de server terugkrijgt. Dit zorgt ervoor dat de gebruiker minder moet opvragen van het netwerk en dus ook zaken die eerder gecachet werden door de gebruiker sneller zullen inladen. De user cache is ideaal om specifieke gegevens van een gebruiker op te slaan.

4.2 Web cache

Web- of server side caching, is een vorm van gegevens caching die gebruikmaakt van het RAM- of diskgeheugen van de server. Het doel hiervan is om laadtijden en ballast op de server te verminderen. Dit kan gaan over data die opgehaald moet worden uit een databank. Het is mogelijk dat deze data nog eens bijgewerkt wordt in de backend en vervolgens zal ze pas doorgestuurd worden over het netwerk.

Werken zonder een web cache is niet efficiënt om verschillende redenen:

- Ten eerste moet de gebruiker telkens wachten tot zijn aanvraag door de server ontvangen werd.
- Vervolgens wacht hij tot de server zijn aanvraag verwerkt heeft.
- Daarna bestaat de kans dat de webserver gegevens moet ophalen uit de databank die gelinkt is aan de server.
- De data die opgehaald werd, moet verwerkt worden voordat ze kan getoond worden aan de gebruiker.
- En ten slotte moet de data terug verzonden worden naar de gebruiker.



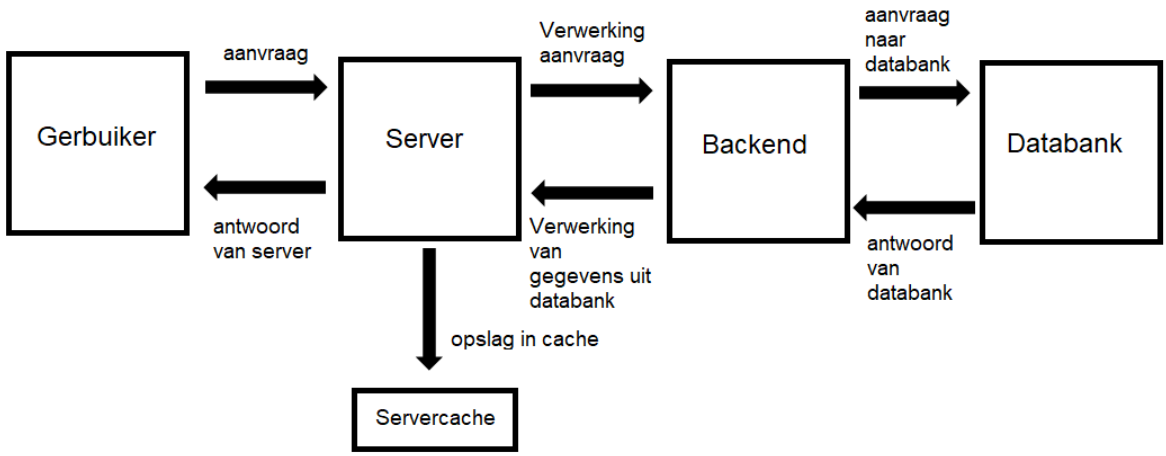
Als dit proces telkens opnieuw moet gebeuren voor elke gebruiker en je hebt duizend gebruikers die op hetzelfde moment, dezelfde aanvragen doen, dan zal de server snel overbeladen worden. Alle gebruikers zullen bijgevolg lang moeten wachten tot hun aanvraag verwerkt wordt. Als een gebruiker te lang op zijn aanvraag moet wachten, zal hij de indruk krijgen dat de site traag en inefficiënt werkt. Dit maakt dat hij zal zoeken naar een andere, alternatieve website.

Al deze ballast kan ook fouten of vertragingen op de server zelf veroorzaken. Hierdoor krijg je een administratieve last voor de eigenaar van de server. Hij zal voor al deze requests moeten kunnen garanderen dat ze allemaal correct werken. Vervolgens zullen al deze extra requests

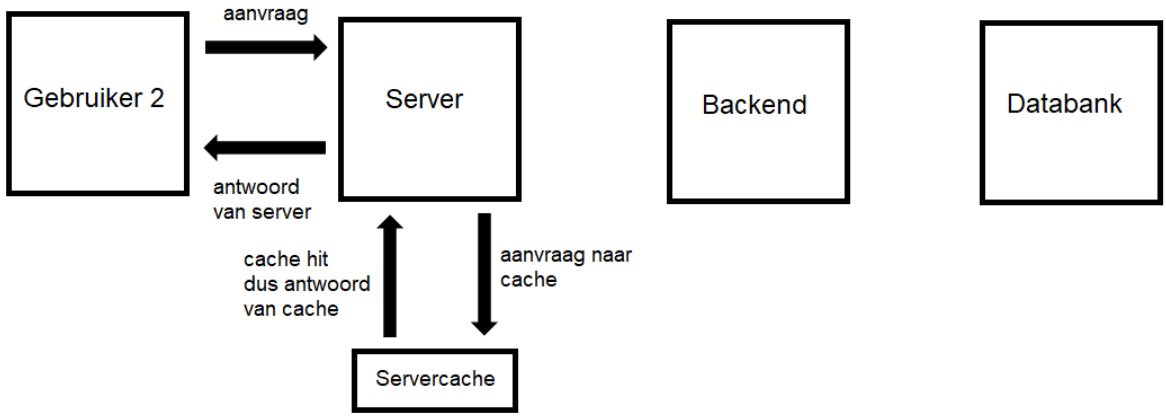
ervoor zorgen dat de servereigenaar veel geld zal moeten investeren om zijn servers uit te breiden doordat ze niet efficiënt gebruikmaken van de beschikbare hardware.

Maar wanneer gebruik wordt gemaakt van een cache, krijgt de server de eerste aanvraag binnen, verwerkt hij deze en slaat hij de gegevens op in de cache vooraleer hij ze naar de gebruiker terugstuurt. Wanneer de volgende bezoeker, dezelfde aanvraag naar de server stuurt, kan gecontroleerd worden in de cache of het resultaat van de aanvraag al in de cache zit. Wanneer dit het geval is, heb je een “cache hit” en wordt de data uit de cache teruggezonden. Dit zorgt ervoor dat er geen data meer opgehaald of verwerkt moet worden in een backend.

Gebruiker 1:



Gebruiker 2:



Deze methode van werken is handig om gegevens op te halen, maar wanneer er gegevens moeten opgeslagen worden in de databank, moet er weer een nieuwe aanvraag gebeuren om de nieuwe gegevens op te slaan en de oude gegevens up te daten. De servercache wordt dus ongeldig omdat ze niet de meest recente gegevens bevat. Het proces dat gebruikt wordt om aan te geven dat ze niet meer up-to-date zijn heet cache invalidatie.

4.3 Opcache

Opcache is een speciale vorm van serverside caching. Hier wordt niet enkel de data van een databank opgeslagen, maar ook het resultaat dat de server bekomt nadat ze de backend code uitvoerde.

Sinds versie 5.5 van PHP wordt Opcache “out of the box” meegeleverd met de installatie. Dit betekent dat Opcache niet handmatig moet geïnstalleerd worden. Deze service verhoogt de snelheid van een dynamische website door de bytecode te cachen.

Dankzij blijvende updates aan Opcache blijft ze nog steeds een relevante manier om aan caching te doen op de server. Zo werd Opcache bijvoorbeeld 100% sneller na de update van PHP 5.6 naar PHP 7.

4.3.1 Opcodes

PHP is een interpreted language. Dit wil zeggen dat wanneer PHP-code uitgevoerd wordt, ze niet rechtstreeks de processor instructies kan geven. De code kan niet zomaar uitgevoerd worden en heeft een speciale software nodig om ze uit te voeren, namelijk de interpreter. Dit is in tegenstelling tot compiled languages die rechtstreeks vertaald worden in machine code. Enkele voorbeelden van compiled languages zijn C en C++.

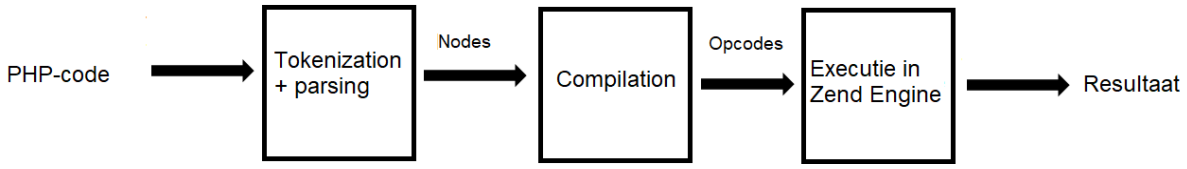
Om te kunnen begrijpen hoe Opcodes precies werken en wat het voordeel is van het cachen van deze codes, moet eerst duidelijk zijn hoe PHP als scriptingtaal uitgevoerd wordt van begin tot einde. Het uitvoeringsproces doorloopt 4 stappen.

De eerste stap is het **tokenization** proces. Hier wordt de broncode omgezet in een reeks tokens. Zij representeren de broncode als een soort ketting van instructies. Deze reeks instructies kan ook gebruikt worden door externe services. Zo maakt de PHP_CodeSniffer package gebruik van tokens. Codesniffer is een extensie voor PHP die ervoor zorgt dat de broncode voldoet aan enkele opmaakstandaarden en de code leesbaar en overzichtelijk blijft.

De volgende stap in het proces is **parsing**. Parsing maakt gebruik van de lijst van tokens die tijdens het tokenizationproces gegenereerd werden. Deze stap heeft twee grote doelen. Zijn eerste doel is om de tokens te valideren. Deze validatie is nodig om fouten in de syntax van de code op te sporen. Het tweede doel is om een “abstract syntax tree” of AST te genereren. Dit is een boomstructuur die in de compilatiestap gebruikt wordt. Dit zijn de “nodes”.

Vervolgens worden de nodes **gecompiled**. Deze nodes worden overlopen en omgezet in opcodes, een verkorting van “operation codes”. Tijdens dit omzettingsproces naar opcodes worden ook optimalisaties gedaan. Deze optimalisaties bestaan bijvoorbeeld uit het veranderen van functioncalls zoals strlen(“abc”) naar int(3).

De laatste stap in het executieproces is **interpretation**. In deze fase worden de opcodes losgelaten op de Zend Engine. Zend doet voor de uitvoering op zich niet veel speciaal. Ze gaat enkel ervoor zorgen dat de opcodes uitgevoerd worden op het systeem. De huidige versie van deze speciale virtuele machine is Zend Engine 3 en wordt gebruikt sinds PHP 7 onder de codenaam PHPNG, wat staat voor “PHP Next Generation”.



Er bestaan een tweehonderdtal opcodes in de Zend Engine. De onderstaande tabel toont enkele voorbeelden hiervan, wat hun code en nummer is en wat ze precies doen. Opgelet deze opcodes kunnen veranderen wanneer er updates worden gedaan aan de engine!

Tabel 4.3.1: Voorbeelden van opcodes en hun functie

Nr.	Code	Uitleg
1	ADD	"Add" telt twee integers met elkaar op en geeft de som terug als resultaat
8	CONCAT	"Concat" plakt twee strings aan elkaar en geeft een combinatie van de twee strings als resultaat. "Hello" + "world" → "Helloworld"
108	THROW	"Throw" is een meer ingewikkelde operation code. Wanneer de Zend Engine deze uitvoert, wordt een exceptie object teruggegeven vanuit de evaluation stack met daarbij een foutmelding.

In het onderstaand voorbeeld wordt in PHP 7.4 een variabele aangemaakt met de waarde "some text" en vervolgens met het echo commando weergegeven. We kunnen de output van deze code in opcodes bekijken aan de hand van VLD, een tool om opcodes uit de Zend VM te visualiseren. VLD staat voor Vulcan Logic Dumper.

```
1  <?php
2  $value = 1;
3  $value = 'some text';
4  echo($value);
5  ?>
```

```
Finding entry points
Branch analysis from position: 0
1 jumps found. (Code = 62) Position 1 = -2
filename:       /in/CmSjj
function name:  (null)
number of ops:  4
compiled vars:  !0 = $value
```

Line	#*	E	I	O	Op	Fetch	Ext	Return	operands
2	0	E	>		ASSIGN				!0, 1
3	1				ASSIGN				!0, 'some+text'
4	2				ECHO				!0
	3			>	RETURN				1

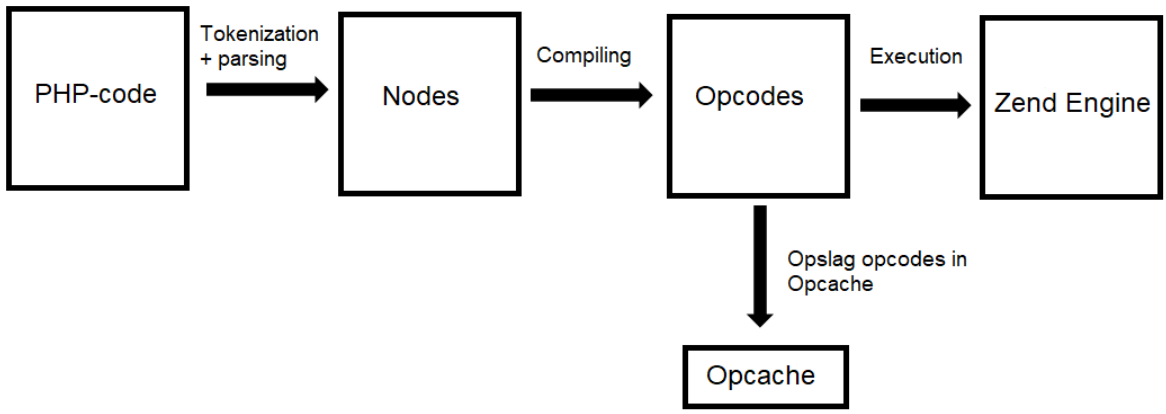
Deze code werd gegenereerd via de opcode-tool op http://www.3v4l.org op 22/05/2020 in versie 7.4 van PHP.

Dankzij de tool kan afgeleid worden dat om deze code te laten uitvoeren, 3 opcodes nodig zijn, namelijk ASSIGN, ECHO en RETURN. Assign is de opcode die ervoor zorgt dat de waarde aan de variabele wordt toegekend. Vervolgens wordt echo afgehandeld door opcode ECHO. Ten slotte krijg je een RETURN opcode die aangeeft dat het volledige programma werd uitgevoerd.

4.3.2 Opcodes cachen

Na het uitvoeren van een PHP-script worden alle instructies vergeten en verwijderd uit het geheugen. Zelfs wanneer hetzelfde bestand meerdere malen uitgevoerd wordt, zal PHP dit script verschillende malen parsen en compileren voordat hij ze uitvoert. Uit vorige voorbeelden blijkt dat dit nadelig is voor de hardware en de gebruiker omdat de hardware vaker, en onder zwaardere ballast komt.

Om te vermijden dat PHP dit overtollige werk steeds opnieuw moet doen en niet meer eerder uitgevoerde scripts opnieuw hoeft te compileren, kunnen opcodes gecachet worden aan de hand van Opcache.



Bij Opcache worden de opcodes bewaard in een gedeeld geheugen, namelijk de cache. Wanneer een ander proces hetzelfde script nodig heeft, worden de opcodes voor dit script nu uit de Opcache gehaald. Deze codes worden door de VM uitgevoerd en leveren vervolgens sneller het resultaat van het hergebruikte script. De tijd die gewonnen werd, kwam doordat het hele compilatieproces niet meer moest gebeuren.

Wanneer een Opcache gebruikt wordt, zal de Zend Engine niet zomaar alle opcodes gaan cachen en uitvoeren. Moest dit gebeuren zal de cache snel vol zitten met nutteloze code en te snel belangrijke delen van de cache moeten leeg maken om plaats te krijgen voor andere data. Het geheugen wordt maximaal benut door optimalisatiefuncties die ingebouwd zijn in Opcache en uit de Zend Optimizer. Om nog een voorbeeld te geven van optimalisatie kan bijvoorbeeld een rekensom in de code als “30 x 21 x 52” omgezet worden naar een enkele waarde van “int(32760)”.

4.3.3 Zend VM

De Zend Engine is een soort virtuele machine die aan de basis ligt van het uitvoeren van PHP-code. Ze is geen gewone virtuele machine die hardware gaat simuleren, maar is meer een programma dat instaat voor de vertaling van opcodes naar de juiste bytecode. Bytecodes zijn de instructietaal voor de hardware. Aan de hand van bytecode kunnen bijvoorbeeld instructies gegeven worden aan de CPU voor welke berekeningen en processen uitgevoerd moeten worden.

Code van de programmeur → Byte code → Machine code

De huidige versie van Zend Engine is Zend Engine 3 en wordt gebruikt in PHP 7. Ze werd oorspronkelijk aangekondigd onder de codenaam PHPNG ofwel “PHP Next Generation”. Doorheen de evolutie van Zend zijn heel wat optimalisaties gedaan. De hoge optimalisatie is te danken aan de updates die de gemeenschap rond Zend uitvoert. Zend Engine is namelijk een open source project. Dit wil zeggen dat dankzij deze talrijke aanpassingen, Zend Engine 3 de meest efficiënte versie tot nu toe is. Om dit project beschikbaar te maken voor de gemeenschap werd een Github opgesteld waar je je aanpassingen en optimalisaties aan de engine kan voorstellen.

5 Cachemethodes

Er kan op heel wat verschillende manieren gecached worden, maar uiteindelijk komen al deze methodes op enkele basisprincipes neer. Elke methode zal op zijn beurt handig zijn in specifieke situaties. Aan de hand van onderstaande onderverdeling zal duidelijk worden in welke situaties het best gekozen wordt voor een methode. De keuze die gemaakt wordt, zal invloed hebben op wat gecachet wordt, hoe lang het bewaard wordt in de cache en hoe cache-invalidatie afgehandeld wordt.

5.1 Volledige pagina cachen

Deze manier van caching is de meest simpele vorm van caching. Je gaat in dit geval de volledige HTML van de webpagina genereren, vervolgens cache je deze en ten slotte toon je de pagina aan de gebruiker.

Dit is een simpele vorm van cachen, maar is in veel gevallen niet praktisch in gebruik. De reden hiervoor is dat een website zelden statisch zal zijn. Doorheen de website zullen verschillende velden dynamisch ingeladen worden of aangepast worden als de gebruiker interact met de site. Hierdoor zal telkens wanneer de pagina verandert een nieuwe versie moeten gecachet worden en verliest de cache zijn nut.

Een van de plaatsen waar je whole page caching kan tegen komen is in een proxyserver die de whole page cache zal gaan regelen. De proxy zal bijvoorbeeld de cache 5 seconden bijhouden om ballast op de server te verminderen. Het zal zelden gebeuren dat een programmeur zelf in zijn PHP-logica whole page caching zal integreren.

Een andere plek waar full-page-caching kan voorkomen is in een webshop. Services zoals Magento maken gebruik van een full-page-cache om categoriepagina's weer te geven. Deze zullen heel zelden veranderen en vragen veel rekenkracht en databankinteractie om op te bouwen. Door meteen de gegenereerde pagina met alle categorieën te cachen, wordt al deze interactie met de server vermeden.

5.2 Gedeeltelijke pagina cachen

Gedeeltelijke pagina cachen, of partial page caching, is een vorm van caching waarbij je delen van HTML gaat bewaren als een soort widgets. Deze widgets worden invalid op het moment dat hun data verandert of kunnen bijgewerkt worden met nieuwe data, maar zo lang de data hetzelfde blijft kan de widget op elke pagina hergebruikt worden.

Deze vorm van caching wordt onder andere bij Wordpress gebruikt. Hier kunnen statische blokken hergebruikt worden op verschillende locaties terwijl bijvoorbeeld de rest van de pagina dynamisch is.

5.3 Databasequery cachen

Het kan gebeuren dat er data moet opgehaald worden uit een databank. Door de complexiteit van de query en de hoeveelheid data die opgehaald moet worden, kan deze query uitvoeren heel wat tijd innemen. Om te voorkomen dat de gebruiker te lang moet wachten op zijn opgevraagde, complexe data, kan het resultaat van een query opgeslagen worden in de cache.

Hierdoor kunnen de gegevens veel vlotter teruggestuurd worden naar de gebruiker wanneer een andere pagina dezelfde query opnieuw uitvoert.

Dit principe is voor een ontwikkelaar simpel te implementeren. Dit kan gedaan worden door en wrapper rond de query's te schrijven die automatisch de resultaten gaat cachen. Zo hoeft de programmeur die de volgende keer dezelfde request gaat maken, zich geen zorgen maken over de cache en wordt alles automatisch geregistreerd in de cache.

5.4 Biggest-smallest reusable cache

"Biggest-smallest Reusable" is een principe in caching waar je verwerkte data probeert in zijn meest herbruikbare vorm te gaan cachen. Het doel hier is om zo'n groot mogelijk blok gegevens te cachen die tegelijk nog altijd zo herbruikbaar mogelijk is. Dit is moeilijker om te implementeren want de oplossing zal heel specifiek zijn voor de applicatie. Wanneer dit echter lukt, zal de cache een praktisch werkmiddel worden voor de applicatie.

5.5 Cachen in verschillende lagen of "layered cache"

Een cache kan opgedeeld worden in lagen. Deze cache in lagen of "layered cache" is een onderverdeling van caches naargelang hoe "stale" de data mag zijn en hoe snel de data geleverd moet worden na een request. Data die niet lang stale zal zijn, zal je terugvinden in de bovenste, meest beschikbare laag. Stale data is een Engelstalige term voor data die verouderd is. "Stale" wil zeggen dat er in het hoofdgeheugen al een aanpassing gebeurde aan de gegevens die in de cache zitten en ze dus verouderd is.

Een vaak gebruikte techniek is door 3 lagen te gebruiken. De eerste laag kan een cache zijn in de PHP-instantie zelf. Dit betekent dat wanneer twee of meerdere keren dezelfde data opgevraagd wordt in eenzelfde request, er niet gewacht moet worden tot alle requests verwerkt werden in de instantie. Het resultaat kan in de instantie hergebruikt worden om het resultaat van de aanvraag op te bouwen.

De volgende laag kan een Opcache zijn. Als de data niet werd teruggevonden in de instantie, kan gecontroleerd worden of de data niet uit een Opcache kan gehaald worden. Is dit niet het geval, dan zou een cache zoals Memcache een alternatief kunnen zijn. Memcache is gespecialiseerd in het cachen van database query's. (zie hoofdstuk 7.2).

En ten slotte komt de request toe in de laatste laag, de database zelf. Wanneer de data nergens kon teruggevonden worden in de cache is het logisch om een standaard aanvraag te doen naar de gegevens door ze, zoals je zonder cache zou doen, op te zoeken in de databank. Wanneer uit de databank de gegevens opgehaald werden, zal de data weggeschreven worden naar de bovenliggende lagen. Dit voorkomt onnodig werk bij een volgende keer dat hetzelfde request gedaan wordt en dat het hele proces weer doorlopen moet worden.

5.6 Cache Warming

Cache warming, of cache voorverwarming is een methode die gebruikt wordt om te vermijden dat zelfs de eerste persoon die gegevens aanvraagt, zal moeten wachten. Wanneer de server opgestart wordt, wordt een script uitgevoerd. Dit speciaal script zorgt ervoor dat op voorhand enkele fundamentele functionaliteiten van de webapplicatie aangesproken worden. Doordat ze aangesproken worden, wordt een cache gegenereerd en staat ze al klaar wanneer de eerste gebruiker zijn aanvraag doet.

Facebook maakte gebruik van deze methode om de cache voor te verwarmen. Ze gebruikten de functionaliteit wanneer ze nieuwe serverclusters toevoegden aan hun netwerk. Hier werd opgemerkt dat wanneer Facebook de gebruikers omleidde naar de nieuwe servers, hun performantie het eerste uur zeer slecht was. Dit kwam doordat eerst alle nieuwe caches opnieuw moesten gegenereerd worden op de nieuwe server. Wanneer de cache volledig opnieuw opgebouwd was, werkten de servers naar behoren.

De socialmediagigant deed vervolgens een onderzoek om het probleem op te lossen en bestudeerde welke gegevens het vaakst werden geraadpleegd. De volgende keer dat er nieuwe servers werden toegevoegd, laadden ze deze specifieke gegevens in de cache voordat ze de gebruikers toelieten op de toegevoegde servers. Hierdoor moest de cache niet meer door de requests van de gebruikers opgebouwd worden, maar werkte de server vanaf het eerste moment optimaal.

5.7 Opcache preloading

PHP-functies en classes kunnen vooraf ingeladen worden. Dit moet maar een keer gedaan worden en vervolgens zullen ze in de toekomst gebruikt worden in de context van de applicatie zonder extra overhead. Deze functionaliteit is nieuw in PHP 7.4. Preloading maakt deel uit van Opcache en start op nog voor de rest van de applicatiecode. De manier waarop ze opgestart wordt, is volgens hoe de administrator ze instelt. Preloaden voorkomt dat er extra overhead is. De overhead is het extra werk dat moet gebeuren om de cache uit shared memory in te laden in het applicatieproces. Vervolgens blijft deze gepreloade code permanent aanwezig zo lang de applicatie uitgevoerd wordt. Deze methode zorgt er uiteindelijk voor dat de functies die werden geschreven door de programmeur even snel zullen uitgevoerd worden als standaard ingebouwde functies van PHP.

In de PHP.ini file is er, in PHP 7.4, maar 2 instelling die horen bij **Opcache preloading**. De eerste instelling is **opcache.preload** en heeft als parameter een URL nodig die naar een bestand verwijst. Het bestand waarnaar de URL wijst, zal gecompileerd en uitgevoerd worden op het moment dat de server opstart. Na het uitvoeren van zijn functionaliteiten zullen pas requests in de applicatie aanvaard worden.

De volgende instelling is **opcache.preload_user**. Met deze instelling kan het preloadbestand door een andere gebruiker uitgevoerd worden moest de administrator dat wensen. Wegens veiligheidsredenen wordt niet toegelaten dat code als root wordt gepreload.

In dit preloadbestand moet een lijst gemaakt worden van alle files die vooraf ingeladen moeten worden. Vervolgens moet deze lijst in Opcache bewaard worden. Dit wordt gedaan met de functie **opcache_compile_file()**. Alle files die door dit proces verwerkt zijn, zullen beschikbaar zijn uit cache. Het is belangrijk dat bestanden altijd ingeladen worden in de volgorde van hun afhankelijkheden. Als een bestand functionaliteit gebruikt uit een ander script, moet het script waarvan ze afhankelijk is eerst gecachet worden. Wordt dit niet gedaan, dan zal ze op de normale manier gecompileerd worden en vullen de gecachete bestanden de cache zonder enig voordeel te bieden.

Preloading is een nieuw concept in PHP. Het is dus zeker aangeraden om op voorhand te onderzoeken of de applicatie waaraan gebouwd wordt wel voordeel haalt uit deze vorm van caching voordat ze ingesteld wordt.

5.8 Opcache cachen naar bestanden

Opcache kan weggeschreven worden naar bestanden. In een normale situatie wordt de cache geladen uit het RAM-geheugen. Dit zorgt ervoor dat als de server afgesloten wordt, de volledige cache verloren gaat. Om te voorkomen dat alles verloren gaat, kan de cache opgeslagen worden op de harde schijf van een server in een file. Deze file zal alle gecachete opcodes bevatten en kan ze dan weer inladen in het RAM-geheugen op het moment dat de server weer opstart. Dit voorkomt dat de cache opnieuw moet opgebouwd worden. Dit is nuttig voor websites met veel bezoekers die meteen na opstarten moeten kunnen reageren.

6 Cache invalidatie strategieën

Cache invalidatie zorgt ervoor dat oude data die niet meer relevant is nooit getoond wordt aan de gebruiker. Deze “oude data’’ is data waarvan er een meer recente, aangepaste versie bestaat.

6.1 Cache invalidatie met tijdsperiode

Een eerste manier om cache te invalideren is door de cache een **time to live** te geven, ofwel ttl. De “time to live” is een tijdsperiode waarin de data van de cache getoond blijft worden aan de gebruiker. Als deze tijdsperiode is verlopen wordt de oude key ongeldig en moet ze, na een nieuwe request van de gebruiker, opnieuw ingeladen worden in de cache voordat ze weer getoond wordt. Dit proces herhaalt zich totdat de ingestelde tijd weer verstrijkt

Bijvoorbeeld: Een gebruiker haalt data op uit een databank. De ttl op de server is 5 minuten. Voor de volgende 5 minuten zal aan elke gebruiker, die dezelfde data opvraagt, de gecachete data te zien krijgen. Na 5 minuten is deze data in de cache invalid en zal de volgende persoon die deze aanvraag doet aan de server een nieuwe versie van de gegevens opwekken in de cache.

Cache hammering is een probleem dat ontstaat bij deze vorm van caching. De situatie ontstaat wanneer er bijvoorbeeld 100 aanvragen binnenkomen op het moment dat de data in de cache vervalt. Doordat de cache verlopen is, moet een nieuwe aanvraag gedaan worden naar de server om de cache opnieuw op te vullen. Maar omdat al deze requests op hetzelfde moment binnenkwamen, zullen ze allemaal tegelijk verstuurd worden naar de server. Hierdoor zal de server al deze aanvragen verwerken en opnieuw in de cache stoppen. De cache wordt dan telkens overschreven wanneer iedere aanvraag verwerkt is. In dit voorbeeld zal dit dus 100 keer na elkaar zijn.

Cache hammering kan opgelost worden met complexe logica die bijvoorbeeld de aanvragen gaat vergelijken en vraagt veel werk om op te lossen. De oplossing hiervoor is echter simpel. Het probleem kan in sommige gevallen worden genegeerd. In een tijdsperiode van 5 minuten worden 100 aanvragen verwerkt. Terwijl er zonder cache misschien 100 acties per seconde verwerkt moeten worden en 100 aanvragen in 5 minuten is nog steeds efficiënter.

6.2 Update invalidatie

Vervolgens kan je cache ook laten vervallen wanneer de data bijgewerkt wordt. In het geval van update invalidatie, wordt de cache manueel ongeldig gemaakt op het moment dat er een update is op de data.

Deze vorm van caching is voordelig in situaties waar data weinig verandert. De data blijft in de cache bewaard totdat er een aanpassing aan gebeurt of totdat de cache vol zit. Als de cache vol zit, moet er plaats vrijgemaakt worden door de delen van de cache te flushen.

Een alternatieve manier van de cache updaten is om nieuwe data op te slaan in de databank. Terwijl dit plaatsvindt, worden dezelfde gegevens in de cache bijgewerkt. Zo hoeft de cache niet meer te wachten op een nieuwe aanvraag om de data te invalideren en bij te werken in een nieuwe request.

Nadelig aan deze methode is dat de cache “out of sync” kan raken. Door een storing of een update die onopgemerkt gebeurt, kan het gebeuren dat de data in de databank niet meer hetzelfde is zoals deze in de cache. Om te voorkomen dat dit gebeurt, wordt een time to live ingesteld. Deze limiet kan langer zijn dan wanneer je enkel met tijdsverval werkt. Zo kan bijvoorbeeld in plaats van om de 30 seconden, om het uur de cache vervallen worden. Zo kan gegarandeerd worden dat er geen extreme afwijkingen in de gegevens van de cache ontstaan.

7 Cache oplossingen

7.1 APC en APCu

APC staat voor *Alternative PHP Cache* en is de voorloper van Opcache. APC kan kleine scripts bijhouden in de cache. Aan de hand van opcode caching worden laadtijden drastisch verminderd. Wanneer APC in gebruik is, kunnen voor veelgebruikte scripts de parsing en compiling stap overgeslagen worden.

APC wordt sinds PHP 5.5 niet meer gebruikt voor opcaching en werd vervangen door Opcache. APC zorgde niet enkel voor opcode caching, maar ook voor user cache. Omdat APC enkele basisfunctionaliteiten heeft en APCu nog steeds beschikbaar is in PHP zal APC kort besproken worden.

APCu is het gedeelte van APC met user cache. Zoals in hoofdstuk 4 vermeld werd, is de usercache een cache die lokaal, bij de gebruiker bewaard wordt. Doordat ze aan de userside werkt, wil dit zeggen dat wanneer APCu nodig is, dit expliciet moet vermeld zijn in de PHP-scripts. De usercache kan gebruikt worden wanneer het programma taken moet uitvoeren die veel data vragen van het netwerk van de gebruiker, zoals een grote file ophalen. Het resultaat van deze actie kan gecachet worden bij de user en vervolgens zorgt de cache ervoor dat deze actie maar eenmalig uitgevoerd moet worden over het netwerk voor zolang het bestand in de cache bewaard blijft.

APC werkt aan de hand van enkele functies. De belangrijkste zijn:

- **apc_add**, om nieuwe gegevens op te slaan in de cache. Apc_add heeft drie verschillende invoerwaarden. Namelijk een **key**, om de cache gegevens te kunnen opzoeken. De volgende waarde is de effectieve **data** die gecachet wordt en ten slotte kan een "**time to live**" of ttl meegegeven worden. Deze integer bepaalt hoelang de data in de cache bewaard mag blijven, maar is een optionele variabele. Wanneer de functie geen specifieke tijd meekrijgt, worden de gegevens bewaard in de cache totdat ze manueel verwijderd worden, de server herstart wordt of de cache manueel, volledig geleegd wordt.
- De volgende functie is **apc_store** en wordt gebruikt om variabelen op te slaan in de cache. Deze functie heeft ook als waarden een **key** nodig, een **waarde** en ten slotte een optionele **time to live**. Het verschil tussen apc_add() en apc_store() is dat de addfunctie enkel maar de waarde zal cachen als ze nog niet in de cache zit terwijl de storefunctie de oude waarde zal overschrijven.
- Vervolgens is **apc_fetch** nodig om gegevens op te vragen. Aan de hand van de **sleutel** die als parameter meegegeven wordt, kan data opgevraagd worden.
- Ten slotte heb je **apc_delete**. De deletefunctie heeft als waarde ook een **key** nodig. Aan de hand van deze key wordt de data opgezocht en verwijderd uit de cache.

Voorbeeldcode:

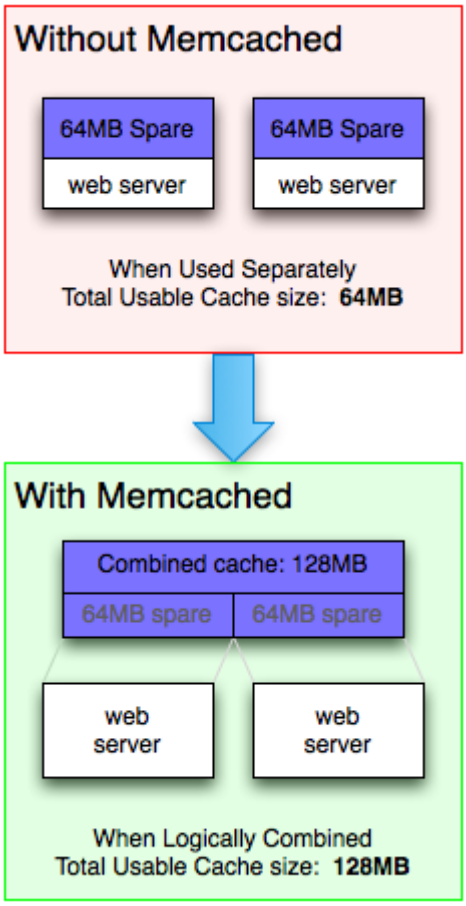
```
1  <? Php
2  $bar = 'BAR';
3  apc_store('foo', $bar);
4  var_dump(apc_fetch('foo'));
5  apc_delete('foo');
6  var_dump(apc_fetch('foo'));
7  ?>
```

Output in console

```
String(3) "BAR"
false
```

In bovenstaande code wordt een variabele 'bar' aangemaakt. Deze variabele wordt in de apc-cache bewaard. Daarna wordt ze opgezocht aan de hand van apc_fetch en met var_dump() wordt de waarde van de variabele in de console getoond. Nadat ze werd getoond, wordt de waarde verwijderd uit de cache aan de hand van apc_delete().

7.2 Memcache en Memcached



Memcache is een caching systeem dat data kan opslaan in het geheugen en werd ontworpen om laadtijden van database-interacties te verlichten bij dynamische webapplicaties. Dankzij Memcache kunnen SQL-query's bewaard worden. Dit zorgt ervoor dat als een query al eerder uitgevoerd werd, het resultaat uit het geheugen kan ingeladen worden. Zo voorkomt Memcache dat er te veel database-interacties uitgevoerd worden en kunnen resultaten sneller ingeladen worden. Memcache kan niet werken zonder Memcached. Want Memcache is een object georiënteerde interface op Memcached, ontwikkeld voor gebruik in PHP.

Memcached is een open source project dat in het werkgeheugen een key-value pair gaat maken met strings en objects. Deze key-value pairs zijn gegenereerd uit resultaten van database calls, api calls of pagina rendering. Een voordeel aan Memcached is ook dat ze overbodig cachegeheugen vanop verschillende locaties kan combineren.

Zoals geïllustreerd in onderstaand schema, kan je zien dat wanneer Memcached niet gebruikt wordt, je maar 64MB cache per server hebt. Met Memcached ingeschakeld, kan je cache combineren. Zo ontstaat uit 2 losse caches van 64MB, een enkele centrale cache van 128MB. Vervolgens kan het aantal servers makkelijk uitgebreid worden. Als er bijvoorbeeld 10 servers bijkomen van elk 64MB, zal de cache met 640MB groeien. Met een uitbreiding van 100 servers met een cache van 64MB wordt een cache van bijna 7GB bekomen.

7.2.1 Hoe werkt Memcached

Memcached maakt gebruik van key-value combinaties. De keys zijn stringwaarden die gekoppeld worden aan een value. Deze value bevat de nuttige data en de keys worden gebruikt om de data op te zoeken.

Memcached heeft 3 basisfuncties:

- Set(key, value)
- Get(key) met als output de value
- Delete(key)

De set functie bewaart een value of resultaat uit de database onder de key die meegegeven werd. Vervolgens kan via de get-functie data uit de key opgehaald worden. Ten slotte kan met de delete-functie een key verwijderd worden uit de cache.

De gegevens die bewaard en bewerkt worden, bevinden zich in het werkgeheugen en is dus volatile. Volatile betekent dat wanneer de server uitgeschakeld wordt, alle data die in het geheugen zat, verloren gaat.

7.2.2 Statistieken bekijken

Memcached heeft een ingebouwde functionaliteit om statistieken te genereren over de cache. Hier kan teruggevonden worden hoeveel geheugen de cache verbruikt. Ook kan bepaald worden uit deze gegevens wat de hit ratio is van de cache. Hit ratio is een percentage dat weergeeft hoeveel keren iets uit de cache wordt gehaald en hoeveel keer de data niet in de cache werd gevonden. De ratio zou geleidelijk aan moeten stijgen naar 100%. Als na lange tijd de ratio nog altijd niet 100% nadert, kan dit een indicatie zijn dat de cache te klein is.

Om te verbinden met Memcached moet een telnetverbinding gemaakt worden met de server. Wanneer er verbinding is met de server kunnen statistieken gegenereerd worden met het commando “stats”. Dit commando toont een aantal gegevens.

In onderstaand voorbeeld wordt verbinding gemaakt met memcached. Enkele zaken die te zien zijn is bijvoorbeeld de **uptime** in seconden. Uptime toont aan hoe lang de server actief is. **Get_hits** en **get_misses** toont het aantal cache hits en het aantal cache misses. Om de hit ratio te verbeteren moet de get_hits gedeeld worden door de som van get_hits en get_misses. Deze berekening toont dat het voorbeeld een hit ratio heeft van 69,5%. Dit is nog geen ideaal ratio. Voor een heel efficiënte cache moet voor de hit ratio gemikt worden op 95% of hoger.

```
Hits / (hits + misses) = hit ratio
78926 / (78926 + 34556) = 0.69543
```

Curr_items toont het aantal bestanden in de cache. Ten slotte is ook **limit_maxbytes** te zien. Dit is de hoeveelheid bytes die de server mag gebruiken voor cache. In dit voorbeeld is de cache gelimiteerd tot 64 Megabytes of 67108864 Bytes.

```
shell> telnet localhost 11211
Trying ::1...
Connected to localhost.
Escape character is '^]'.
stats
STAT pid 23599
STAT uptime 675
STAT time 1211439587
STAT version 1.2.5
STAT pointer_size 32
STAT rusage_user 1.404992
STAT rusage_system 4.694685
STAT curr_items 32
STAT total_items 56361
```

STAT bytes 2642

STAT curr_connections 53

STAT total_connections 438

STAT connection_structures 55

STAT cmd_get 113482

STAT cmd_set 80519

STAT **get_hits** 78926

STAT **get_misses** 34556

STAT evictions 0

STAT bytes_read 6379783

STAT bytes_written 4860179

STAT **limit_maxbytes** 67108864

STAT threads 1

END

(Voorbeeld gehaald van https://docs.oracle.com/cd/E17952_01/mysql-5.6-en/ha-memcached-stats.html)

Via deze site is een handige template te vinden om het resultaat van “stats” te helpen interpreteren: https://lzone.de/cheat-sheet/memcached

7.2.3 Memcache en/of APC

APC was voor PHP 5.5 voornamelijk een opcode cache en wordt nu enkel nog gebruikt voor user cache. Memcache dient meer om gegevens zoals database query’s op te slaan. Het is ook aangeraden om Memcache te gebruiken wanneer je applicatie uitgevoerd wordt over verschillende servers. Dit omdat ze de cache van alle servers kan combineren. Elke soort cache is geoptimaliseerd voor zijn eigen doeleinden. Ze kunnen dus in combinatie met elkaar gebruikt worden, Opcache voor opcode caching, APCu voor user cache en Memcache voor load balancing.

7.3 Redis

Redis staat voor “Remote Dictionary Server” en is een open source, **NoSQL** databasesysteem. Dit betekent dat ze een databank is die niet werkt met rijen en kolommen zoals bijvoorbeeld MySQL. In de plaats werkt Redis met een **key-value** store. Er kunnen gegevens opgeslagen worden met keys. De data die gekoppeld wordt aan de keys zijn bijvoorbeeld strings, lists, sets, sorted sets of hashes.

Wat Redis voordelig maakt in gebruik is dat ze een databank is die kan bestaan in de cache. Dit zorgt voor een databank met hoge snelheden in de applicatie. Ze zal veel sneller gegevens kunnen wegschrijven en ophalen uit de databank dan SQL. Redis laat ook toe om zijn databank uit cache weg te schrijven uit het werkgeheugen naar de disk.

7.3.1 Commando's en voorbeelden

Redis werkt niet met query's zoals in SQL, maar maakt gebruik van commando's. Om een kort voorbeeld te schetsen van de meest gebruikte commando's is hieronder een lijst te vinden met de meest voorkomende commando's, een voorbeeld en uitleg van hun functionaliteit.

Tabel 7.3.1: overzicht van Rediscommando's

Commando	**Uitleg**
SET foo "hello world"	Het set-commando heeft 2 parameters. De eerste parameter is de naam van de key. In dit geval zal de key "foo" zijn. De tweede waarde die wordt meegegeven met het commando is het "hello world" gedeelte. Dit is de data die bij de key hoort en samen vormen zij een key-value pair.
GET foo	Met "get" kan de data die aan een key werd gekoppeld opgevraagd worden. In dit voorbeeld wordt de data opgevraagd die bij de key "foo" hoort. Als we dit voorbeeld combineren met de gegevens die in het vorig voorbeeld met "set" ingevoerd werden, zal "GET foo" als output "hello world" geven. Voorbeeldoutput: "hello world"
DEL foo	"del" staat voor delete en is het commando dat wordt gebruikt om een key te verwijderen. Zoals te zien was in het vorig voorbeeld, kon data opgevraagd worden met GET. Als "DEL foo" gebruikt wordt en vervolgens weer "GET foo", toont de console een een leeg resultaat omdat de data uit "foo" verwijderd werd.
SETEX foo 30 "hello world" **SET foo "hello world" EX 10**	Setex wordt gebruikt om een waarde en een time to live mee te geven aan een key. In het voorbeeld wordt de "expiration time" of verlooptijd op 30 seconden geplaatst voor de waarde "hello world" met als key "foo". Wanneer **SETEX foo 30 "hello world"** uitgevoerd wordt en vervolgens na 30 seconden ***GET foo*** wordt gebruikt, zal het commando een leeg resultaat tonen, want ze zal verwijderd zijn. De verlooptijd kan ook meegegeven worden met het "SET" commando. Door na het standaard set commando "EX 10" te zetten, krijgt "foo" een bewaartijd van 10 seconden.
EXPIRE foo 30	Als een key oorspronkelijk geen verlooptijd had, kan dit later nog aangepast worden met "EXPIRE". Het commando in het voorbeeld zorgt ervoor dat "foo" 30 seconden na het uitvoeren van dit commando verwijderd zal worden.
TTL foo	Ttl staat zoals eerder al vermeld voor "time to live". Het commando TTL vraagt als invoerwaarde de naam van een key en zal de resterende tijd die een key heeft in de command line tonen in seconden.

	Voorbeeldoutput: (integer) 30
PERSIST foo	Voor een key die geen time to live meer mag hebben, maar oorspronkelijk wel een expiration time meekreeg, kan het commando "**persist"** gebruikt worden. De key die na PERSIST geschreven wordt, zal na het uitvoeren van het commando, permanent in het geheugen bewaard blijven.
RENAME foo bar	"**Rename**" heeft 2 invoerwaarden nodig, de oude key om deze aan te kunnen spreken en als tweede waarde de nieuwe naam die de key moet krijgen. In het voorbeeld wordt de key "foo" hernoemd naar "bar".
FLUSHALL	Om het geheugen van Redis volledig leeg te maken typ je "**flushall**" in de command line. Dit zal alle gegevens verwijderen uit het geheugen.

Voor meer informatie over de commando's is de documentatie beschikbaar op https://redis.io/commands. Via de documentatie kunnen commando's uitgetest worden in een console die beschikbaar is op de website.

Voorbeeld:

```
redis>  Set key "test" EX 10
"OK"
redis>  ttl key
(integer) 7
redis>
```

7.3.2 Redis en cachen in Laravel

Redis is ook beschikbaar in het populaire PHP-framework Laravel. Laravel is een framework dat MVC naar PHP brengt. Om Redis op zijn meest efficiënte manier te gebruiken wordt in de Laravel documentatie aangeraden om PhpRedis te installeren, een extensie die ervoor zal zorgen dat Redis beter en sneller zal werken met Laravel.

In Laravel kunnen Redis-commando's gebruikt worden aan de hand van facades. Ze zullen hetzelfde werken zoals in de command prompt, maar kunnen op deze manier gemakkelijk gebruikt worden in scripts.

Zoals te zien is in onderstaand voorbeeld, wordt een get-commando gebruikt in Laravel om een gebruiker op te halen aan de hand van zijn sleutel. Deze sleutel zal een id zijn die door de gebruiker meegegeven wordt aan de functie.

```
<?php
namespace App\Http\Controllers;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Redis;
```

```
class UserController extends Controller
{
    /**
     * Show the profile for the given user.
     *
     * @param  int  $id
     * @return Response
     */
    public function showProfile($id)
    {
        $user = Redis::get('user:profile:'.$id);

        return view('user.profile', ['user' => $user]);
    }
}
```

Voor meer informatie over Redis in Laravel 7 kan verwezen worden naar de Laravel documentatie: https://laravel.com/docs/7.x/redis. Deze voorbeeldcode werd opgehaald van https://laravel.com/docs/7.x/redis.

7.4 MySQL query cache

MySQL heeft zijn eigen ingebouwde cache. Het is niet aangeraden om volledig te vertrouwen op deze cache om een applicatie sneller te maken. Het nadeel van de MySQL-cache is dat ze in haar cache de exacte query's cachet samen met de exacte resultaten. Dit wil zeggen dat wanneer de query verandert, er een nieuwe cache moet gemaakt worden.

Het is niet aangeraden om een MySQL-cache te gebruiken met een databank die veel wijzigt. Als data in een tabel verandert, zullen cachegegevens van alle data die deze tabel gebruikt, geflusht worden. Dit wil zeggen dat wanneer een gebruiker data opvraagt, de query wordt uitgevoerd en gecachet. Wanneer de volgende gebruiker zijn gegevens opvraagt, krijgt hij zijn gegevens en wordt ook zijn query gecachet voor hem. Als plots een derde gebruiker zijn e-mailadres aanpast, zullen alle caches die gebruik maken van de gebruikersgegevenstabel, geflusht worden. Wat ervoor zorgt dat de volledige cache van de gebruikerstabel opnieuw moet aangemaakt worden.

De cache is niet gebonden aan sessies van gebruikers. Dit zorgt ervoor dat elke databankgebruiker dezelfde databankcache gebruikt. De tabel die gecachet werd door een aanvraag van gebruiker 1 kan ook gebruikt worden door gebruiker 2, die dezelfde gegevens wil raadplegen. Dit kan zo lang de query van gebruiker 2 exact hetzelfde is als de query van gebruiker 1.

7.5 NGINX

Nginx, uitgesproken als "Engine-X", is zowel een HTTP-server zoals bijvoorbeeld Apache, maar ook een reverse proxyserver. Ze staat bekend om haar hoge snelheden en efficiënt gebruik van beschikbare resources. Een grote hoeveelheid van de meest bezochte websites ter

wereld maakt gebruik van deze servertechnologie. Je kan Nginx aantreffen in sites zoals Dropbox, Netflix en Wordpress. Het voordeel van Nginx is dat ze een totaalpakket is van load balancer en webserver.

Een van de services die Nginx biedt, is dat ze een reverse-proxy is en zorgt voor load balancing aan de serverkant. Dit betekent dat het netwerkverkeer door Nginx kan herleid worden naar achterliggende servers om de ballast te verdelen. Hierdoor kan voor elke gebruiker een vlot gebruik gegarandeerd worden.

Nginx benut middelen zoals RAM en CPU zo goed mogelijk. Dankzij deze efficiëntie met resources, kunnen servers meer gebruikers accepteren op hetzelfde moment met gelijke middelen. Wat dan weer zorgt voor lagere operationele kosten en hogere responssnelheden voor de eindgebruiker.

Nginx biedt een web cache aan die kan worden ingeschakeld. Ze zorgt ervoor dat requests niet telkens opnieuw moeten verstuurd worden via de proxy naar de achterliggende servers wanneer dezelfde content meerdere keren nodig is. Wat bovenop de load balancing nog een winst in performantie is.

7.6 Varnish

Varnish is een cache die voornamelijk voor de webserver staat, maar niet lokaal bij de gebruiker. Ze is een cache die statische en dynamische content kan opslaan. Wanneer gebruikgemaakt wordt van Varnish is het mogelijk dat een website 300 tot 1000 keer sneller reageert op requests. Daarenboven is Varnish heel configureerbaar dankzij de configuratietaal waarmee Varnish werkt. Varnish kan namelijk geprogrammeerd worden met “Varnish Cache Configuration Language” ofwel VCL. Deze taal laat toe om specifieke regels voor requests te zetten en het cachebeleid aan te passen.

Varnish is ideaal voor het gebruik met websites die grote delen complexe of gepersonaliseerde content moeten aanbieden. Zo gebruikt bijvoorbeeld Wikipedia, Twitter en The New York Times website allemaal Varnish.

7.7 Wincache

Wincache is de PHP-cache van Windows Server en dient apart als extension geïnstalleerd te worden op de server. De Wincache extensie biedt verschillende functionaliteiten aan. Zo kan Wincache een soort Opcache zijn voor Windows Server die zoals gewoonlijk bij Opcache, PHP-bytecodes opslaat in cache na executie.

Met Wincache kan ook data opgeslagen worden. De verwerkte data kan daarna hergebruikt worden door PHP scripts. Doordat deze data niet telkens opnieuw verwerkt moet worden, wordt er weer tijd en ballast op de server verminderd.

De caching software van Windows Server bevat een file system cache die interacties van PHP-scripts op remote UNC locaties efficiënter maakt doordat er minder file system operaties gemaakt moeten worden.

Opgelet, voorlopig worden enkel PHP-versies 5.2 tot 7.1 ondersteund in Wincache.

8 PHP-configuratie

8.1 PHP.ini

PHP.ini is het configuratiebestand van PHP. Dit bestand wordt gebruikt om variabelen te configureren die helpen de applicatie/service te sturen op de server. Deze file wordt ingeladen wanneer de PHP-service wordt gestart. De volgende punten zullen meer duidelijkheid geven in wat de PHP.ini file precies doet.

Sommige instellingen beschrijven een hoeveelheid in bytes, kilobytes, megabytes of gigabytes en worden met een integerwaarde weergegeven in combinatie met een letter. Deze manier van instellen heet **Shorthand Notation**. Shorthand Notation is niet hoofdlettergevoelig, maar vaak zullen de waarden genoteerd worden met hoofdletter.

Overzicht van Shorthand Notations:

- K staat voor kilobytes.
- M staat voor megabytes.
- G staat voor gigabytes.

Wanneer geen extra letter bij de integer staat, zal de waarde in bytes omgezet worden.

Meer over *Shorthand Notation* is te lezen in de PHP-documentatie via https://www.php.net/manual/en/faq.using.php#faq.using.shorthandbytes

8.2 Standaard PHP-instellingen

Het PHP.ini bestand maakt gebruik van variabelen om een PHP-omgeving op de server in te stellen. Aan de hand van de waarden die de gebruiker ingeeft bij deze variabelen zal de server een specifiek gedrag hebben op het moment dat ze opstart. In onderstaande hoofdstukken worden de instellingen besproken die de grootste impact zullen hebben op de performantie van de server.

8.2.1 Memory_limit

De waarde die ingegeven wordt bij “Memory_limit” zal een integerwaarde zijn in Shorthand Notation. Deze waarde bepaalt voor de server wat de maximale hoeveelheid werkgeheugen is in megabytes die de server mag reserveren om scripts te verwerken. Als er enkel een getal staat, zal de waarde het aantal bytes bepalen die mag gegeven worden aan een script.

Om de instelling uit te schakelen moet je een waarde -1 bij de instelling plaatsen.

Tabel 8.2.1: Voorbeeld en uitleg

Instelling	**Uitleg**
memory_limit = 128M	Hier wordt de waarde van memory_limit op 128MB gezet. Dit wil zeggen dat maximum 128MB zal mogen gebruikt worden op de server om zijn scripts uit te voeren.

8.2.2 Max_input_time en max_execution_time

Bij “Max_input_time” wordt ingesteld hoe lang elk script **maximaal mag wachten** op aangevraagde data. Als deze tijd verloopt, zal een time-out plaats vinden en zal het proces stoppen. Dit is een handige instelling om te voorkomen dat de server te lang moet wachten op input. Dit voorkomt dat andere processen te lang moeten wachten op een ander proces.

Max_execution_time zal vervolgens het hoogst aantal seconden aangeven waaraan de PHP-server tijd mag **spenderen voor de executie**. Verloopt deze tijdsperiode, dan wordt weer een time-out teruggegeven door de server.

Beide instellingen kunnen uitgeschakeld worden door ze een waarde van -1 te geven. Hierdoor weet de server dat ze oneindig lang mag wachten op input of tot een script uitgevoerd wordt.

Tabel 8.2.2: Voorbeelden van max_input_time en max_execution_time

Instelling	**Uitleg**
max_input_time = 300	In dit voorbeeld krijgt max_input_time een waarde van 300 seconden ofwel 5 minuten. De server zal dus 5 minuten wachten op input en vervolgens een time-out geven.
max_execution_time = -1	Hier wordt in het voorbeeld de max_execution_time uitgeschakeld. Dit kan gedaan worden wanneer een server regelmatig moet back-uppen. Back-ups kunnen veel tijd in beslag nemen en de tijd die ze nodig hebben hangt vaak af van de hoeveelheid data die geback-upt moet worden. Als er tijdens het back-upproces een time-out gegeven wordt, kan dit voor problemen zorgen en de back-up laten falen.

8.2.3 Upload_max_filesize en post_max_size

Om te regelen hoe groot upgeloade files mogen zijn op de server kunnen de instellingen “upload_max_filesize” en “post_max_size” gebruikt worden. De beide instellingen hebben Shorhand Notation nodig die bepaalt hoe groot een file of request mag zijn.

Upload_max_filesize bepaalt hoe groot een file maximum mag zijn wanneer ze upgeload wordt naar de server.

Post_max_size bepaalt een limiet op hoe groot de volledige uploadaanvraag maximaal mag bedragen. Om zeker te zijn dat er geen onverwachte errors plaatsvinden, moet de waarde bij post_max_size altijd groter dan of gelijk zijn aan upload_max_filesize.

Het is aangeraden na te gaan welk soort files upgeload moeten worden op de server. Als bijvoorbeeld vaak grote pdf’s of afbeeldingen met een hoge resolutie moeten upgeload worden, is het best dat je deze instellingen een hoge waarde geeft.

Tabel 8.2.3: Voorbeelden van upload_max_filesize en post_max_size

Instelling	**Uitleg**
upload_max_filesize = 2M	Met de instelling op deze waarde kunnen enkel bestanden van maximum 2 megabytes upgeload worden naar de server.

Post_max_size = 8M	Als post_max_size staat op “8M” wil dit zeggen de totale grootte van upgeloade bestanden 8 megabytes mag zijn. Dit wil dus zeggen dat in combinatie met bovenstaande instelling 4 files van elk 2 megabytes kunnen upgeload worden naar de server.

8.2.4 max_input_vars

Het aantal invoerwaarden dat tegelijk verzonden wordt naar de server per pagina kan ook gelimiteerd worden. Dit kan de administrator instellen met “max_input_vars”. Deze instelling heeft als waarde een integerwaarde. De integer bepaalt het maximale aantal variabelen die met een form meegegeven mogen worden naar de server per webpagina.

Tabel 8.2.4: Voorbeeld van max_input_vars

Instelling	**Uitleg**
max_input_vars = 1000	Als deze instelling gebruikt wordt, zullen bijvoorbeeld maximum 1000 elementen van een form meegegeven worden naar de server per webpagina.

8.3 Opcache inschakelen met php.ini

Zoals de basisinstellingen van PHP gemakkelijk kunnen gewijzigd worden, kan Opcache ook ingesteld worden in PHP.ini. De instelling “**opcache.enable**”, kan Opcache inschakelen (opcache.enable=1) of uitschakelen (opcahce.enable=0).

Vervolgens kan ook ingesteld worden hoeveel geheugen vrijgemaakt moet worden voor Opcache. Dit gebeurt met “**opcache.memory_consumption**” en staat standaard op 128. De waarde 128 wil zeggen dat de cache 128MB groot zal zijn.

Opcache instellingen zullen vaak ook na het initieel instellen van een project aangepast worden. Dit wordt gedaan aan de hand van de problemen die zich voordoen tijdens het gebruik van de applicatie, zoals het cachegeheugen dat vol zit of de keystore die vol zit. Om deze problemen op te lossen kunnen de Opcache-opties bijgewerkt worden.

Opgelet, bij bijvoorbeeld opcache.memory_consumption wordt Shorthand Notation niet gebruikt, maar wordt standaard gebruikgemaakt van een integer die de grootte bepaalt in megabytes.

In de onderstaande punten worden nog enkele nuttige instellingen meegegeven met uitleg over wat ze kunnen doen voor de server.

8.3.1 Opcache.max_file_size

Deze instelling kan aangepast worden om de maximale grootte van gecachete files te bepalen. Deze instelling gebruikt een integerwaarde om in bytes te bepalen hoe groot de file mag zijn. Wanneer de waarde 0 is, worden alle files gecachet ongeachte de grootte. Dit is ook de standaardwaarde van de instelling. **Opcache.max_file_size** = 256 zal ervoor zorgen dat enkel files kleiner dan, of gelijk aan 256 bytes gecachet zullen worden.

Tabel 8.3.1: Voorbeelden van opcache.max_file_size

Instelling	**Uitleg**
Opcache.max_file_size = 256	Met deze instelling zal Opcache enkel files cachen die kleiner zijn dan, of gelijk aan 256 bytes
Opcache.max_file_size = 0	Met deze instelling zal Opcache niet controleren op de grootte van een file en zal ze alle files cachen. Dit is de standaardinstelling.

8.3.2 Opcache.validate_timestamps en opcache.revalidate_freq

Opcache.revalidate_freq is een instelling die hoort bij **opcache.validate_timestamps**. Wanneer opcache.validate_timestamps ingeschakeld wordt, zal ze om de zoveel seconden zoeken naar geüpdatete scripts. De frequentie van deze controles wordt ingesteld volgens het aantal seconden dat bij opcache.revalidate_freq staat.

Wanneer in productie gewerkt wordt aan een applicatie, is het best dat het tijdsverloop op 0 wordt gezet. Dit zorgt ervoor dat de server steeds de laatste nieuwe scripts heeft. Voor productieservers daarentegen, wordt timestampvalidatie best uitgezet. Dit zorgt ervoor dat wanneer een server updates krijgt, ze eerst moet herstart worden.

Tabel 8.3.2: Voorbeelden van opcache.validate_timestamps en opcache.revalidate_freq

Instelling	**Uitleg**
Opcache.validate_timestamps = 0 Opcache.revalidate_freq = 30	Hier wordt Opcache niet gerefresht totdat de server herstart wordt. Opcache.revalidate_freq wordt hierdoor genegeerd.
Opcache.validate_timestamps = 1 Opcache.revalidate_freq = 30	Hier worden scripts die ouder zijn dan 30 seconden gerefresht.
Opcache.validate_timestamps = 1 Opcache.revalidate_freq = 0	In dit script, worden timestamps gevalideerd, maar staat de frequentie op 0 seconden. Doordat de frequentie op 0 staat, zullen alle nieuwe scripts meteen gecachet worden.

8.3.3 Opcache.max_accelrated_files

De hoeveelheid bestanden in de cache kan ook gelimiteerd worden. Om ervoor te zorgen dat alle projectbestanden gecachet kunnen worden, zet je **opcache.max_accelerated_files** op een aantal dat hoger is dan het aantal bestanden in de applicatie. In Linux gebaseerde besturingssystemen kan een *find* commando gebruikt worden om het aantal PHP-bestanden te tellen.

```
find . -type f -print | grep php | wc -l
```

Tabel 8.3.3: Voorbeeld van opache.max_accelerated_files

Instelling	Uitleg
Opcache.max_accelerated_files = 7500	Deze instelling laat toe dat 7500 php bestanden kunnen bewaard worden in de Opcache. De standaardinstelling laat toe dat 10 000 bestanden bewaard worden.

8.3.4 Opcache.memory_consumption

De instelling **opcache.memory_consumption** heeft als waarde een integer. Deze integer bepaalt hoe veel RAM-geheugen mag toegekend worden aan Opcache in Megabytes. Dit staat standaard op 128MB sinds PHP-versie 7.0.0.

Wanneer de cache veel code moet verwerken kan de hoeveelheid geheugen verhoogd worden. Om te zien hoeveel Opcache verbruikt kan gebruik gemaakt worden van de functie **opcache_get_status()**. Hieruit kan afgeleid worden hoeveel geheugen de Opcache verbruikt.

Tabel 8.3.4: Voorbeeld van opcache.memory_consumption

Instelling	Uitleg
Opcache.memory_consumption = 128	Hier wordt Opcachegeheugen gelimiteerd op 128 Megabytes. Dit is ook de standaardinstelling.

8.3.5 Opcache.preload

Met **opcache.preload** kan een script ingesteld worden dat gecompileerd moet worden bij het opstarten van de server. Doordat dit script al gepreload wordt bij het opstarten van de server, zullen de gegevens meteen beschikbaar zijn in de cache. Deze instelling neemt als invoerwaarde een string met het URL naar het script.

Tabel 8.3.5: Voorbeeld van opcache.preload

Instelling	Uitleg
Opcache.preload = /route/naar/preload.php	Dit is een voorbeeld van hoe naar een bestand kan verwezen worden in de instellingen.

Het scriptbestand dat in het voorbeeld gebruikt werd, zal minimum onderstaande code bevatten om te kunnen werken.

```
1  $files = /* array van bestanden om te preloaden */
2  foreach ($files as $file) {
3     opcache_compile_file($file);
4  }
```

8.3.6 Besluit Opcache instellingen

Opcache is een goeie manier om aan caching te doen en is standaard ingeschakeld in PHP. Sinds versie 5.5 van PHP is de cacheoplossing steeds beter geworden. Met de toevoeging van preloading in de laatste versie wordt nogmaals bewezen dat ze fundamenteel is voor de verhoging van rekensnelheden in PHP. Het is dus ook belangrijk dat instellingen correct gebeuren. Correcte instellingen kunnen een waardevolle bijdrage zijn voor een efficiënte werking van een project.

Een kort overzicht van alle mogelijke instellingen van Opcache is terug te vinden in de handleiding van PHP via https://www.php.net/manual/en/opcache.configuration.php

9 Conclusies

Caching komt voor in verschillende vormen. Elke methode of oplossing heeft zijn eigen voordelen. Uit deze bachelorproef kunnen onderstaande zaken besloten worden.

Er kan op verschillende momenten in de applicatie gecachet worden. De cache kan zich bevinden bij de client of de server. Tijdens deze bachelorproef werd vooral gefocust op de cache aan de kant van de server om de beleving van de client te verbeteren en om de onderhouder van de server geld en werk te besparen.

Vervolgens kan geconcludeerd worden dat dit een topic is in PHP die zeer actueel is. Dit was af te leiden uit de introductie van Opcache in PHP 5.5 en de verbeteringen die hier verder uit volgden. Ook de optimalisaties aan de Zend Engine spelen hier een rol in.

Uit het hoofdstuk over cachemethodes kan besloten worden dat met caching, data sneller beschikbaar kan gemaakt worden voor eindgebruikers. Deze data is niet enkel informatie uit een databank, maar ook delen webpagina's of zelfs volledige pagina's. Om dit te kunnen doen bestaan verschillende methodes zoals cache warming en de cache inladen uit bestanden bij het opstarten van de server.

Om ervoor te zorgen dat een cache nuttig blijft terwijl de data verandert, wordt gebruik gemaakt van invalidatie. Invalidatie zorgt ervoor dat de cache altijd up-to-date blijft en zijn nut behoudt. Om dit zo efficiënt mogelijk te laten verlopen bestaan methodes zoals update invalidatie en controles die tijdsperiodes volgen.

Uit onderzoek blijkt dat gespecialiseerde oplossingen bestaan om applicaties te ondersteunen bij caching. Zo blijkt dat APC nog deels gebruikt wordt in PHP voor user caching en een technologie genaamd Memcached kan gebruikt worden om caches te combineren over verschillende servers. Opcache vervangt de rol van APC om opcode te cachen en kreeg hiervoor extra functionaliteiten bij zoals preloading.

Verder kan uit het hoofdstuk over technologieën afgeleid worden dat NGINX een efficiënte serverhost kan zijn en standaard ook een web cache ingebouwd heeft. Ze kan in combinatie met Varnish gebruikt worden om een web cache te bekomen die heel specifiek kan ingesteld worden. Terwijl ook te zien is dat Windows Server zijn eigen oplossing ontwikkelt in de vorm van Wincache, maar dat deze technologie soms niet helemaal up-to-date kan zijn met de laatste versies van PHP.

Om het meeste uit een server te halen die een PHP-oplossing host, kan de instellingsfile, PHP.ini, geconfigureerd worden. De file zal tijdens het opstarten van de server ervoor zorgen voor een geconfigureerde omgeving. Dankzij deze file kan ingesteld worden hoeveel geheugen een server mag gebruiken, hoe groot een geüploade file mag zijn of hoe lang ze over het uitvoeren van een script mag doen.

In het instellingsbestand zijn ook instellingen te vinden om Opcache te configureren. Uit onderzoek blijkt dat Opcache standaard ingeschakeld is en de administrator best enkele instellingen optimaliseert voordat hij de server inschakelt. Instellingen zoals de grootte van de cache of de hoeveelheid files die in de cache mogen bestaan, zijn zaken die performantie kunnen beïnvloeden.

10 Blik op de toekomst/ Opportuniteiten

Caching is en blijft een grote factor in het versnellen van applicaties. Er zullen in de toekomst nog veel verbeteringen aan caching komen met hun relevante instellingen. Preloading bijvoorbeeld werd in de laatste versie van PHP geïntroduceerd en kan nog heel wat verbeteringen gebruiken zodat in de toekomst bijna iedereen hier zijn voordeel uit kan halen door gebruik te maken van deze functionaliteit.

Opcache blijft ook steeds verbeteren dankzij de aanpassingen die gedaan worden aan de Zend Optimizer. Opcache kan in de toekomst van PHP zorgen voor optimalisaties en verbeteringen in compilatietijden en heeft veel potentieel op groei.

Het viel op dat bij de instellingen van Opcache in php.ini niet alles met Shorthand Notation genoteerd werd. Om te voorkomen dat de documentatie te veel geraadpleegd moet worden, kan bijgewerkt worden dat Opcache-instellingen ook Shorthand kunnen verstaan.

Bronnen- & literatuurlijst

Adobe. (2020, 06 04). *Full-Page Cache*. Opgehaald van Magento Documantatie: https://docs.magento.com/user-guide/system/cache-full-page.html

Akala, D. (2018, 04 20). *Learn about PHP OPCodes*. Opgehaald van Xteam: https://x-team.com/blog/learn-about-php-opcodes/

Brent. (2019, 06 05). *Preloading in PHP 7.4*. Opgehaald van Stitcher.io: https://stitcher.io/blog/preloading-in-php-74

Castro, K. (2018, 08 16). *What is Caching?* Opgehaald van Tutorialspoint: https://www.tutorialspoint.com/What-is-caching

Copes, F. (2014, 08 18). *Basics of PHP Caching*. Opgehaald van FlavioCopes: https://flaviocopes.com/php-caching/

Corona, S. (sd). *Scaling PHP Book*. Opgeroepen op 06 06, 2020, van Best Zend Opcache Setting/Tuning/Config: https://www.scalingphpbook.com/blog/2014/02/14/best-zend-opcache-settings.html

Corporation, O. (sd). *The MySQL Query Cache*. (Oracle) Opgeroepen op 05 31, 2020, van MySQL Documentation: https://dev.mysql.com/doc/refman/5.7/en/query-cache.html

Coudenys, J. (2019, 12 10). *What's new in php 7.4*. Opgehaald van Speakerdeck: https://speakerdeck.com/coudenysj/whats-new-in-php-7-dot-4

Coudenys, J. (2020, 01 24). *PHP OPCache, Realpath Cache and Preloading*. Opgehaald van Joindin: https://joind.in/event/phpbenelux-conference-2020/php-opcache-realpath-cache-and-preloading

Encyclo.nl. (2020, 05 24). *Bytecode definitie*. Opgehaald van Encyclo: https://www.encyclo.nl/begrip/Bytecode

F5. (2020, 05 30). *Welcome to Nginx Wiki*. Opgehaald van nginx.com: https://www.nginx.com/resources/wiki/

Feryn, T. (2019, 12 10). *'t Oncachebare cachen*. Opgehaald van Speakerdeck: https://speakerdeck.com/thijsferyn/t-oncachebare-cachen

GbmbOrg. (sd). *Bytes to MB Conversion*. Opgeroepen op 06 06, 2020, van gbmb.org: https://www.gbmb.org/bytes-to-mb

Gogia, P. (2018, 02 14). *Redis: what and why?* Opgehaald van codeburst: https://codeburst.io/redis-what-and-why-d52b6829813

Google Developers. (2013, 02 05). *Memcache basics*. Opgehaald van Youtube: https://www.youtube.com/watch?v=TGl81wr8lz8

Group, T. P. (2020, 05 24). *OPcache Runtime Configuration*. Opgehaald van PHP Documentation: https://www.php.net/manual/en/opcache.configuration.php

Izone. (sd). *Memcached Cheat Sheet*. Opgeroepen op 06 06, 2020, van Izone: https://lzone.de/cheat-sheet/memcached

Jackson, B. (2020, 05 25). *The Definitive PHP 5.6, 7.0, 7.1, 7.2, 7.3, and 7.4 Benchmarks (2020)*. Opgehaald van Kinsta Blog: https://kinsta.com/blog/php-benchmarks/

Jung, J.-B. (2020, 02 20). *How to Create a Simple and Efficient PHP Cache*. Retrieved from Dzone: https://dzone.com/articles/how-to-create-a-simple-and-efficient-php-cache

Laracademy. (2020, 02 20). *Laravel - Caching Made Easy*. Retrieved from Youtube: https://www.youtube.com/watch?v=34zQR21MIoU

Limpalair, C. (2015, 05 13). *How to Cache Database Queries with Redis in Laravel - Redis Series Episode 4*. Opgehaald van Youtube: https://www.youtube.com/watch?v=zCMypIqZQzo

Media, T. (2017, 03 19). *Redis Crash Course Tutorial*. Opgehaald van Youtube: https://www.youtube.com/watch?v=Hbt56gFj998

Meetup. (2020, 02 20). *Meetup PHP-WVL*. Retrieved from https://www.meetup.com/php-wvl/

Memcached. (2020, 05 22). *About Memcached*. Opgehaald van Memcached.org: http://www.memcached.org/about

Microsoft. (2020, 05 30). *Wincache extension for PHP*. Opgehaald van iis.net: https://www.iis.net/downloads/microsoft/wincache-extension

Mitchel, B. (2020, 05 29). *Working With the Universal Naming Convention (UNC Path)*. Opgehaald van Lifewire: https://www.lifewire.com/unc-universal-naming-convention-818230

Nasser, H. (2020, 02 02). *What is Nginx and what are its use cases?* Opgehaald van Youtube: https://www.youtube.com/watch?v=WHv_t_yK-QM

Oracle. (sd). *Getting Memcached Statistics*. Opgeroepen op 06 06, 2020, van Oracle Documentation: https://docs.oracle.com/cd/E17952_01/mysql-5.6-en/ha-memcached-stats.html

Oracle. (sd). *Memcached General Statistics*. Opgeroepen op 06 06, 2020, van Oracle Documentation: https://docs.oracle.com/cd/E17952_01/mysql-5.6-en/ha-memcached-stats-general.html

Petters, J. (2020, 03 29). *What is a Proxy Server and How Does it Work?* Opgehaald van Varonis: https://www.varonis.com/blog/what-is-a-proxy-server/

PHP UK Conference. (2014, 03 21). *PHP UK Conference 2014 - Eli White - Caching Best Practices*. Opgehaald van Youtube: https://www.youtube.com/watch?v=bsZQcbBcXuQ

PHP UK Conference. (2014, 03 21). *PHP UK Conference 2014 - Julien Pauli - PHP Opcache Explained*. Opgehaald van Youtube: https://www.youtube.com/watch?v=pxW5WdKmprs

Popov, N. (2017, 04 14). *PHP 7 Virtual Machine*. Opgehaald van nikic.github.io: https://nikic.github.io/2017/04/14/PHP-7-Virtual-machine.html

Presta Shop. (2013, 06 27). *How to optimize the PHP.ini file for your website*. Opgehaald van Presta Shop: https://www.prestashop.com/en/blog/php-ini-file

Punt, T. (2017, 01 02). *How PHP executes - from Source Code to Render*. Opgehaald van Sitepoint: https://www.sitepoint.com/how-php-executes-from-source-code-to-render/

Redislabs. (sd). *Redis documentation*. Opgeroepen op 06 02, 2020, van Redis.io: https://redis.io/commands

Section. (2020, 05 30). *Comparing Varnish Cache and Nginx Caching*. Opgehaald van section.io: https://www.section.io/blog/varnish-cache-versus-nginx/

Sklar, D. (2004). Debugging, Caching and Optimizing. In D. Sklar, *Essential PHP Tools: Modules, Extensions, and Accelerators* (p. 331). Apress.

Tarvainen, J. (2019, 12 09). *PHP 7.4 OPCache Preloading benchmark results + note on database extension crashes*. Opgehaald van Ibexa: https://www.ibexa.co/blog/php-7.4-opcache-preloading-benchmark-results-note-on-database-extension-crashes

The PHP Group. (2020, 03 05). *ADD PHP Code*. Opgehaald van PHP Manual: https://www.php.net/manual/en/internals2.opcodes.add.php

The PHP Group. (2020, 03 05). *CONCAT PHP Code*. Opgehaald van PHP Manual: https://www.php.net/manual/en/internals2.opcodes.concat.php

The PHP Group. (2020, 02 27). *Magic Methods*. Opgehaald van PHP Manual: https://www.php.net/manual/en/language.oop5.magic.php

The PHP Group. (2020, 02 20). *PHP*. Retrieved from https://www.php.net/

The PHP Group. (2020, 05 22). *Runtime Configuration*. Opgehaald van PHP Documentation: https://www.php.net/manual/en/apc.configuration.php

The PHP Group. (2020, 05 31). *Stream introduction*. Opgehaald van PHP Manual: https://www.php.net/manual/en/intro.stream.php

The PHP Group. (2020, 03 13). *The configuration file*. Opgehaald van PHP Manual: https://www.php.net/configuration.file

The PHP Group. (2020, 03 05). *THROW PHP Code*. Opgehaald van PHP manual: https://www.php.net/manual/en/internals2.opcodes.throw.php

Tony. (2019, 08 27). *PHP 5.6 vs PHP 7 Performance Comparison*. Opgehaald van Gbksoft: https://gbksoft.com/blog/php-5-vs-php-7-performance-comparison/

Udacity. (2015, 02 23). *Caching techniques - Web Development*. Opgehaald van Youtube: https://www.youtube.com/watch?v=RgPf5RDv4-s

Varnish Software. (2015, 10 15). *What is Varnish Cache?* Opgehaald van Youtube: https://www.youtube.com/watch?v=fGD14ChpcL4

Varnish Software. (2020, 05 30). *How it works*. Opgehaald van Varnish-software.com: https://www.varnish-software.com/how-it-works/?_gl=1*wxd2ph*_gcl_aw*R0NMLjE1OTA4NTg2MTYuRUFJYUlRb2JDaE1Ja0lmYmo0cmM2UUlWR29mVkNoMC04UTEwRUFBWUFTQUFFZ0s1QmZEX0J3RQ..

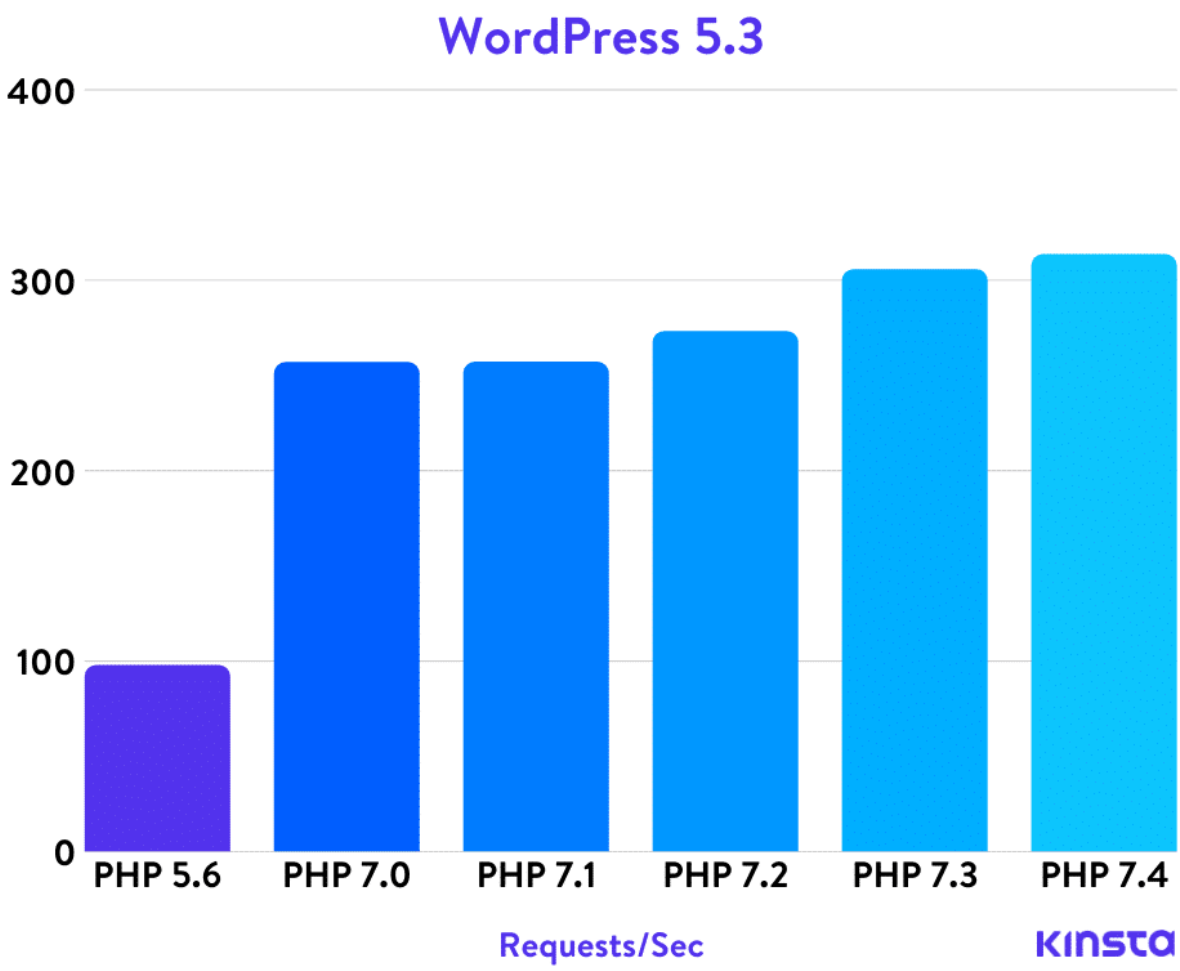
Wikibooks. (2020, 02 20). Retrieved from https://en.wikibooks.org/wiki/PHP_Programming/Caching

Zientek, S. (2018, 11 06). *PHP 7 + OPcache speed up websites noticeably*. Opgehaald van Ionos: https://www.ionos.com/community/hosting/php/php-7-opcache-speed-up-websites-noticeably/

Overzicht van de bijlagen

1 Wordpress performantie over verschillende versies van PHP
2 Wordpress performantie verbetering met caching plugin

Bijlage 1: Wordpress performantie over verschillende versies van PHP



Benchmark resultaten van https://kinsta.com/blog/php-benchmarks/.

In dit diagram werd in Wordpress 5.3 geïnstalleerd met verschillende versies van PHP. Het "*Twenty Twenty"*-thema werd gebruikt en er waren 15 bezoekers op de website. Er werd naar een pagina */hello-world/* gesurft. Deze pagina bevatte 1 comment en een navigatiebalk met enkele menu items.

Op het schema is duidelijk te zien dat na de update van PHP 7 het aantal requests per seconde stijgt. Er is duidelijk te zien dat in PHP 7.4 het aantal requests per seconde meer dan het dubbele is dan het aantal requests in versie 5.6

Dit resultaat is te danken aan de optimalisaties die in versie 7 gedaan werden aan de Zend Engine met de upgrade van Zend Engine 2 naar PHPNG, ofwel Zend Engine 3. In Zend Engine 3 werd opcaching veel efficiënter.

Bijlage 2: Wordpress performantie verbetering met caching plugin

Op volgende website https://gbksoft.com/blog/php-5-vs-php-7-performance-comparison/ werden op 27/08/2019 benchmarks gedaan op 2 identiek ingestelde Wordpress websites. Aan de hand van de Apache benchmark tool werden voor beide sites onderstaande resultaten bekomen.

Ook werden tests gedaan met een caching plugin voor Wordpress genaamd Super Cache. Een plugin die onder andere HTML-pagina's cachet. Deze statistieken bewijzen nogmaals de kracht van caching.

	PHP 5.6	**PHP 7.0**
Time taken for the test	45.35 s	22.95 s
Request per second	22.05 r/s	43.56 r/s
With Super Cache plugin		
Time taken for the test	1.95 s	1.02 s
Request per second	512.25 r/s	974.26 r/s