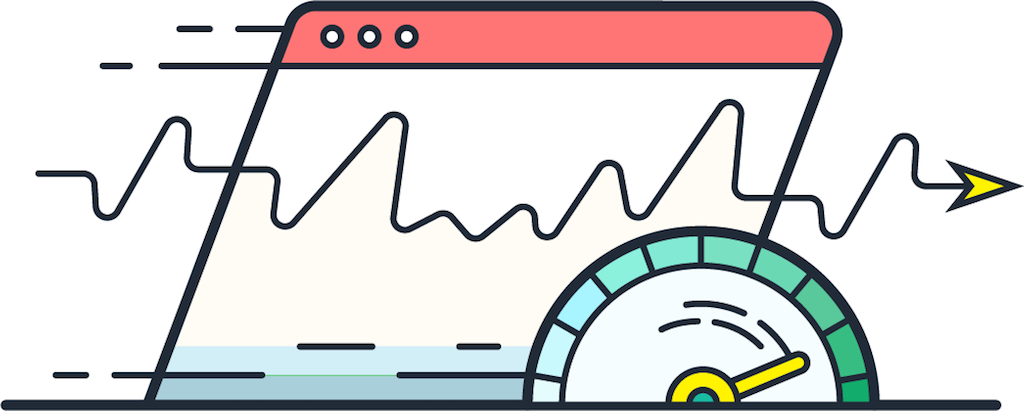
Performance meting van distributed caching

Op welke wijze kan Divide distributed caching inzetten om performance problemen binnen hun E-commerce omgevingen tegen te gaan?



**Thomas Vieveen**15088405

**Afstudeerbedrijf**Divide B.V.  
Koraalrood 27  
2718 SB, Zoetermeer

**De Haagse Hogeschool**HBO ICT - Software Engineering

**Afstudeerbegeleiders**Arie Toet (eerste examinator)  
Alwine Lousberg (tweede examinator)

**Inhoudelijke begeleider**Tom Bina

# 

Voorwoord

Het afstudeerdossier wat voor u ligt is de afsluiting van mijn afstudeerperiode bij de Haagse Hogeschool en in opdracht van stagebedrijf Divide. Het is geschreven in de periode van augustus 2018 tot januari 2019 ter afronding van mijn deeltijdopleiding ICT – Software Engineering.

Het afstudeeronderwerp is mij aangeboden door het bedrijf Divide, waar ik tijdens mijn schoolperiode werkzaam ben. Divide zat met een uitdaging omtrent snelheidsverbetering van E-commerce websites bij grotere omgevingen, waarbij caching een mogelijke oplossing kan zijn. Toen ik aangaf een afstudeeropdracht nodig te hebben, is het onderzoeken van distributed caching als onderwerp voorgesteld.

Caching is over het algemeen een van de lastigste onderwerpen in software engineering, daarom vond ik het interessant om dit tijdens mijn afstudeerperiode te mogen onderzoeken. Ik heb veel over caching kunnen leren, wat ook later in mijn carrière van pas kan komen.

Zonder hulp had ik dit onderzoek niet uit kunnen voeren. Ten eerste wil ik mijn afstudeerbegeleider Tom Bina bedanken voor de begeleiding tijdens het afstuderen. Daarnaast wil ik mijn vader Jan Vieveen en collega Alexander Streng bedanken voor het ondersteunen in het opzetten van het afstudeerdossier en het taalkundig corrigeren van de inhoud.

Ook wil ik mijn afstudeerbegeleiders Alwine Lousberg en Arie Toet bedanken voor het geven van waardevolle feedback. Ten slotte wil ik klasgenoot Victor Elsenaar, mijn enige andere klasgenoot die afstudeerde, bedanken voor het adviseren over bepaalde onderdelen in het afstuderen.

Nootdorp, januari 2019

Thomas Vieveen

Samenvatting

Divide maakt momenteel gebruik van in-memory caching. Deze vorm van caching is momenteel nog acceptabel, maar Divide ervaart bij grotere klanten dat deze traditionele vorm van caching niet altijd voldoende performance biedt.

Het doel van dit onderzoek is om inzichtelijk te maken hoe Divide door middel van distributed caching haar performanceproblemen bij E-commerce websites kan verminderen. Hiervoor is de volgende onderzoeksvraag opgesteld: ‘Op welke wijze kan Divide distributed caching inzetten om performance problemen binnen hun E-commerce omgevingen tegen te gaan?’

Om antwoord te geven op de onderzoeksvraag wordt eerst een onderzoek uitgevoerd naar de werking van (distributed) caching en hoe distributed caching systemen zijn te karakteriseren. Met behulp van de gevonden karakteristieken is er een marktonderzoek uitgevoerd naar het voor Divide meest geschikte distributed cache systeem.

Vervolgens is er onderzocht hoe de performance van een E-commerce omgeving gemeten en beïnvloed kan worden. Dit resulteerde in een lijst van parameters waarmee de performance van een E-commerce omgeving beïnvloed kan worden. Met deze parameters zijn een aantal testen opgesteld om uit te zoeken in welke situaties distributed caching wel en niet goed presteert.

Op basis van het onderzoek is het meest voor de hand liggende distributed cache systeem gekozen, op basis waarvan een test-applicatie is ontwikkeld. Met de testapplicatie zijn vervolgens de opgestelde testen uitgevoerd, wat resulteerde in een grote hoeveelheid metingen over diverse situaties.

Aan de hand van de testresultaten wordt aanbevolen wanneer distributed caching het beste ingezet kan worden. Daarnaast is op basis van de test-applicatie een op Divide gericht advies geschreven over het behalen van de beste performance bij het implementeren van distributed caching.

Afkortingen en definities

|  |  |
| --- | --- |
| Afkorting/Definitie | Betekenis |
| CMS | Content Management Systeem |
| Code First Database | Een database die wordt gegenereerd aan de hand van de aanwezige modellen. |
| Single channel | Een winkelier die alleen producten verkoopt in een fysieke winkel. |
| Omnichannel | Afgestemde ervaring voor de klant op basis van fysiek en online. Alle content is op elkaar afgestemd, onafhankelijk van welk kanaal de klant gebruikt. |
| ACU | Azure Compute Units, de maatstaf die Azure gebruikt om App Service’s performance aan te geven. |
| DTU | Data Transaction Unit, de maatstaf die Azure gebruikt om database performance aan te geven. |

Tabel 0.1: Afkortingen en definities

Figuren- en tabellenlijst

In dit hoofdstuk staan alle verwijzingen naar de gebruikte figuren en tabellen.

**Figuren**

[Figuur 1.1: De planning in Azure DevOps 3](#_Toc534571295)

[Figuur 2.1: Actuele planning zoals deze in de praktijk is verlopen 10](#_Toc534571296)

[Figuur 5.1 Metingen bij statische pagina's (He & Yang) 23](#_Toc534571297)

[Figuur 6.1: De pricing tier van App Services 30](#_Toc534571298)

[Figuur 6.2: De pricing tier van SQL-databases 30](#_Toc534571299)

[Figuur 6.3: De pricing tier van Redis Cache 31](#_Toc534571300)

[Figuur 7.1: Het analysis class diagram van de testapplicatie 34](#_Toc534571301)

[Figuur 7.2: Werking applicatie in Azure omgeving 36](#_Toc534571302)

[Figuur 8.1: DataServiceRetreiver properties 37](#_Toc534571303)

[Figuur 8.2: De homepagina 38](#_Toc534571304)

[Figuur 8.3: Keuze uit alle typen dataopslag 38](#_Toc534571305)

[Figuur 8.4: Productoverzicht pagina 39](#_Toc534571306)

[Figuur 8.5: Redis benchmarktest met 50.000 verzoeken van 1.5 kB 42](#_Toc534571307)

[Figuur 8.6: Redis benchmarktest met 1000 verzoeken van 75 kB 42](#_Toc534571308)

[Figuur 8.7: Loadtest project in Visual Studio 42](#_Toc534571309)

[Figuur 8.8: Loadtest samenvatting in Visual Studio 43](#_Toc534571310)

[Figuur 9.1: Het aanmaken van een nieuwe load test in Azure 47](#_Toc534571311)

[Figuur 9.2: Testresultaten in Azure 48](#_Toc534571312)

[Figuur 9.3: Alle testscenario’s met 10 gelijktijdige gebruikers en 50 getoonde producten 49](#_Toc534571313)

**Tabellen**

[Tabel 0.1: Afkortingen en definities iii](#_Toc534571314)

[Tabel 2.1: De onderzoeksmethoden 7](#_Toc534571315)

[Tabel 2.2: Geplande en behaalde deadlines van het gewenste resultaat 9](#_Toc534571316)

[Tabel 2.3: Risico's en maatregelen 10](#_Toc534571317)

[Tabel 9.1: Voorbeeld opgestelde testen 45](#_Toc534571318)

[Tabel 9.2: Alle testresultaten 50](#_Toc534571319)

[Tabel 10.1: Testresultaten van de distributed cache 53](#_Toc534571320)

Inhoudsopgave

[1 Inleiding en achtergrond 1](#_Toc534571321)

[1.1 Inleiding 1](#_Toc534571322)

[1.2 De organisatie – Divide 1](#_Toc534571323)

[2 Projectdefinitie 4](#_Toc534571324)

[2.1 Probleemstelling 4](#_Toc534571325)

[2.2 Aanvulling op probleemstelling 4](#_Toc534571326)

[2.3 Doelstelling 6](#_Toc534571327)

[2.4 Onderzoeksvragen 6](#_Toc534571328)

[2.5 Onderzoeksmethoden 7](#_Toc534571329)

[2.6 Afbakening project 8](#_Toc534571330)

[2.7 Gewenste resultaat 8](#_Toc534571331)

[2.8 Planning 9](#_Toc534571332)

[2.9 Risico’s en maatregelen 10](#_Toc534571333)

[3 Onderzoek naar distributed caching 11](#_Toc534571334)

[3.1 Wat is caching? 11](#_Toc534571335)

[3.2 De werking van een distributed cache 11](#_Toc534571336)

[3.3 Karakterisering van een distributed cache systeem 12](#_Toc534571337)

[4 Marktonderzoek distributed cache systemen 17](#_Toc534571338)

[4.1 Selectie distributed cache systemen 17](#_Toc534571339)

[4.2 Vergelijking van de distributed cache systemen 19](#_Toc534571340)

[4.3 Resultaat 20](#_Toc534571341)

[5 Performance aspecten bij E-commerce omgevingen 21](#_Toc534571342)

[5.1 Performance en eventuele oorzaken van performanceproblemen 21](#_Toc534571343)

[5.2 Parameters om performance te beïnvloeden 23](#_Toc534571344)

[5.3 Resultaat 25](#_Toc534571345)

[6 Proces testapplicatie 26](#_Toc534571346)

[6.1 Softwareontwikkelmethode 26](#_Toc534571347)

[6.2 Gebruikte methodieken en tools 26](#_Toc534571348)

[6.3 Hosting en hardware 28](#_Toc534571349)

[7 Ontwerp 32](#_Toc534571350)

[7.1 Requirements 32](#_Toc534571351)

[7.2 Architectuur 33](#_Toc534571352)

[7.3 Ontwerp applicatie architectuur 33](#_Toc534571353)

[7.4 Applicatie in de praktijk 35](#_Toc534571354)

[8 Bouw 37](#_Toc534571355)

[8.1 De bouw van de applicatie 37](#_Toc534571356)

[8.2 Uitdagingen 40](#_Toc534571357)

[8.3 Testen van de applicatie 42](#_Toc534571358)

[9 Testen 44](#_Toc534571359)

[9.1 Opzetten van testscenario’s 44](#_Toc534571360)

[9.2 Testscenario’s uitvoeren 45](#_Toc534571361)

[9.3 Uitlezen testresultaten 47](#_Toc534571362)

[9.4 Bespreking testresultaten 48](#_Toc534571363)

[10 Conclusies en aanbevelingen 51](#_Toc534571364)

[10.1 Resultaten 51](#_Toc534571365)

[10.2 Conclusies 52](#_Toc534571366)

[10.3 Advies en aanbevelingen 53](#_Toc534571367)

[11 Evaluatie 55](#_Toc534571368)

[11.1 Product 55](#_Toc534571369)

[11.2 Proces 56](#_Toc534571370)

[11.3 Beroepstaken 56](#_Toc534571371)

[12 Literatuurlijst 58](#_Toc534571372)

[Bijlage 1: De testresultaten 61](#_Toc534571373)

[Bijlage 2: De testresultaten omgezet in grafieken 65](#_Toc534571374)

[Bijlage 3: Afstudeeropdracht 69](#_Toc534571375)

[Bijlage 4: Plan van aanpak 75](#_Toc534571376)

# Inleiding en achtergrond

Om een introductie te geven voor dit afstudeerproduct wordt de aanleiding van het project besproken en wordt de afstudeerorganisatie geïntroduceerd met rol die de student hierin heeft gespeeld.

## Inleiding

De afgelopen jaren heeft E-commerce een enorme vlucht genomen. Steeds meer consumenten verleggen hun bestedingspatroon naar online of oriënteren zich uitgebreid online. Er is al jaren een sterke groei in online bestedingen. Consumenten raken verwend en eisen steeds meer van de webshops die zij bezoeken. Denk bijvoorbeeld aan een aantrekkelijke presentatie, hoge snelheid van de website, een ruim en overzichtelijk aanbod, goede mogelijkheden om te vergelijken en een responsive website met de mogelijkheid om de website op alle apparaten te kunnen gebruiken. E-commerce op mobiele telefoons en tablets realiseert een zeer sterke groei in de laatste jaren.

E-commerce is vandaag de dag een supply chain. Dit betekent dat er een keten is vanaf de producent via eventuele tussenhandel naar de consument. Het optimaliseren van deze supply chain is steeds belangrijker: om kosten te reduceren moeten voorraden zo laag mogelijk gehouden worden terwijl de consument een snelle levering van alle producten wenst. Vandaag de dag is het voor een succesvolle E-commerce website absoluut noodzakelijk de supply chain te integreren. Grote bekende ketens zoals Amazon, Coolblue en BOL hebben hierin geïnvesteerd en een eigen Content Management Systeem (CMS) ontwikkeld.

De klanten van Divide zijn niet groot genoeg om een eigen CMS te realiseren, maar willen wel over dezelfde mogelijkheden beschikken zoals de eerdergenoemde grote partijen. Divide heeft een generiek CMS ontwikkeld waarmee haar klanten in staat zijn te concurreren met de grotere concurrenten.

Zoals eerdergenoemd is de performance van een webshop van groot belang, onafhankelijk van de grootte van de webshop en het aantal gelijktijdige gebruikers. Dit onderzoek gaat in op distributed caching als instrument om de performance van de E-commerce webshop te verbeteren. Distributed caching is een uitbreiding van het traditionele concept van caching. In tegenstelling tot de traditionele cache, kan een distributed cache meerdere servers omvatten, zodat deze eenvoudig kan groeien in omvang en in transactionele capaciteit.

De hieruit volgende hoofdvraag luidt dan ook: ‘**Op welke wijze kan Divide distributed caching inzetten om performance problemen binnen hun E-commerce omgevingen tegen te gaan?**’

## De organisatie – Divide

Divide bestaat sinds 2000 en is opgericht door drie jeugdvrienden. Anno 2018 bestaat Divide uit ongeveer 35 internet professionals en is Divide marktleider op het gebied van webshops in de kleding- en schoenenindustrie. Divide legt de focus op grotere retailers met fysieke winkels en helpt single channel organisaties tot de weg naar omnichannel organisaties. Bekende ketens zoals Only for Men, The Stone en Christine le Duc zijn klanten van Divide.

E-commerce is sterk gegroeid in het afgelopen decennium. Voor de online verkoop van kleding is de groei nog veel groter dan andere sectoren. Daarnaast is het aantal producten enorm toegenomen en worden collecties in een steeds hoger tempo vernieuwd. Daarbij zijn de consumenten verwend geraakt: zij willen een snelle, interactieve website met mooie vormgeving, veel informatie, advies, veel functies (wensenlijstjes, vergelijkingen, snel zoeken, adviezen voor alternatieven) en een snelle navigatie. Divide heeft deze ontwikkelingen intensief gevolgd en heeft geïnvesteerd in een eigen Content Management Systeem (CMS) met als doel deze wensen uit de markt beschikbaar te maken voor haar opdrachtgevers en hiermee voorop te lopen.

Divide is sinds 2018 onderdeel van Total Specific Solutions (TSS). TSS is een bedrijf wat verliesgevende bedrijven overneemt en winstgevend maakt. Divide is niet verliesgevend maar is overgenomen samen met het partnerbedrijf ACA. TSS biedt met haar contacten een opstap naar het buitenland. Zo zijn de eerste webshops van Divide nog in 2018 in Duitsland gelanceerd. Dit is een opstap naar verdere groei in binnen- en buitenland voor de komende jaren.

**Teams**

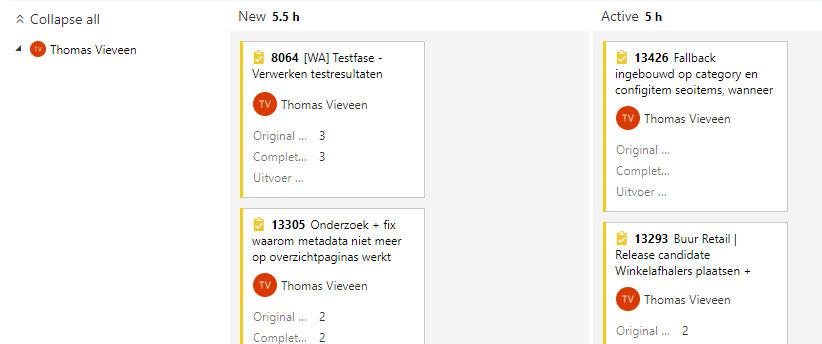
Bij Divide werken op het moment van schrijven ongeveer 35 werknemers en is verdeeld in vijf teams:

* General & Administration (G&A), voert functionaliteiten uit om de dagelijkse bedrijfsvoering van Divide te ondersteunen. Hierbij kan gedacht worden aan facilitaire zaken (bijvoorbeeld het bestellen van nieuwe computers), het beheren van de algemene planning en zorgt dat de administratieve zaken op orde zijn;
* Customer Support (CS), alle eventuele problemen die in de software van Divide worden aangetroffen door klanten, worden bij CS aangemeld. CS lost de problemen en zet deze indien nodig door naar een ander team;
* Research & Development (R&D), het R&D-team is bezig met het constant verbeteren van de bestaande software en realiseert daarnaast ook nieuwe software. Het R&D-team doet onderzoek naar nieuwe technologieën en implementeert deze in de bestaande software wanneer dat mogelijk is;
* Professional Services (PS), PS doet alles rondom het onderhouden van de webshops van klanten. Zij implementeren nieuwe en reeds gebouwde functionaliteiten in de bestaande webshops en zorgen ervoor dat de klanten up-to-date blijven;
* Sales & Marketing (S&M), S&M zorgt ervoor dat er voldoende wordt verkocht. Zij gaan langs bij klanten en verkopen bestaande en/of nieuwe functionaliteiten en eventueel ook complete webshops. S&M voorziet het R&D- en PS-team van werk.

**Planning**

Divide is zich op het moment van schrijven aan het innoveren door de Scrum softwareontwikkelmethode te adopteren. Scrum is een Agile softwareontwikkelmethode, waarin de ontwikkelteams in grote mate zelfsturend zijn. (Wat is Scrum?, 2018)

Divide hanteerde voorheen individuele planningen waarbij een document werd bijgehouden met alle activiteiten per persoon, per week. Tegenwoordig maakt Divide gebruik van ‘Azure DevOps’, een door Microsoft aangeboden online platform waarin een Agile planning kan worden bijgehouden. (Figuur 1.1)



Figuur 1.1: De planning in Azure DevOps

**Bedrijfscultuur**

Binnen Divide heerst een no-nonsense cultuur met de nadruk op open en informele omgangsvormen. Daarnaast is Divide een platte organisatie en is iedereen gelijk aan elkaar. Iedereen staat open voor kleine verzoeken en wil elkaar graag helpen.

**Mijn rol binnen het bedrijf**

Als deeltijd student ben ik aan het begin van mijn studie op zoek gegaan naar een ICT-bedrijf in de softwareontwikkeling en ben ik bij Divide uitgekomen. Ondertussen ben ik al meer dan 3 jaar fulltime werkzaam bij Divide als webdeveloper. Ik zit in het Research & Development team en mijn taken bij dit team bestaan uit onder andere:

* Het ontwikkelen van nieuwe software en het uitbreiden van bestaande software;
* Werkzaamheden omtrent de database (raadplegen & muteren van data en tabellen);
* Het oplossen van 3e-lijns support tickets;
* Het inschatten van nieuwe projecten.

# Projectdefinitie

In dit hoofdstuk wordt op basis van het plan van aanpak (Bijlage 4: Plan van aanpak) de huidige situatie beschreven en daarnaast het projectonderwerp. Tevens wordt het project afgebakend zodat de relevante aspecten vooraf vastgesteld zijn. Tenslotte worden de onderzoeksmethoden aangegeven en toegelicht waarna potentiële risico’s besproken worden.

## Probleemstelling

De webshops van Divide maken momenteel gebruik van in-memory caching. In-memory caching houdt in dat alle data in het geheugen van de server wordt opgeslagen. Dit maakt in-memory caching heel snel, maar niet schaalbaar genoeg. Er kan alleen opgeschaald worden door extra geheugen aan de server toe te voegen.

Divide host voor elke klant hun webshop en het CMS. Beide applicaties maken gebruik van een core-project, waarin alle hoofdfunctionaliteiten staan om redundante code te voorkomen. Het core-project stopt alle data die het nodig heeft in de cache. Dit betekent dat de webshop en het CMS allebei individueel alle data cachen. Hier kan Divide niets aan verbeteren, omdat elke applicatie alleen bij zijn eigen cache kan.

Wanneer een klant een groot assortiment heeft kan de omvang van de cache heel groot worden. Dit loopt op het moment van schrijven in de praktijk soms op tot 16 GB-geheugen. Dit betekent dat een klant op één server met de webshop en het CMS tot wel 32 GB-geheugen gebruik kan maken.

Klanten met een hoog aantal bezoekers maken gebruik van meerdere servers om zo de druk van de bezoekers te verdelen. Voor elke extra server die een klant afneemt is dan ook 16 GB aan geheugen nodig. De klant betaalt niet alleen voor het geheugen, maar ook voor de andere specificaties van de server. Dit kost de klant relatief veel geld aan hosting.

Naast dat elke uitbreiding dezelfde server specificaties nodig heeft, kan het voorkomen dat er meer dan het aanwezige geheugen gevraagd wordt en daardoor de server crasht. Hierdoor moet op die server de cache opnieuw opgebouwd worden. Dit proces kost tijd en levert tijdens het opbouwen een extra belasting op voor de database. De gebruikers worden op dat moment verdeeld over de overige servers die daarop weer een hogere belasting ervaren dan verwacht. Dit kan een domino-effect veroorzaken. In de praktijk levert dit voor klanten time-outs op en is niet gewenst.

## Aanvulling op probleemstelling

Om meer context te bieden aan de probleemstelling, is de uitgangssituatie nader uitgewerkt.

**Caching bij Divide**

Divide is ooit begonnen met het gebruik van de cache voor data waar zware berekeningen voor nodig waren, zodat deze berekeningen niet telkens opnieuw uitgevoerd hoeven te worden. De data die gecached wordt is in de loop der tijd uitgelopen tot een steeds grotere hoeveelheid, waardoor de cache nu bijna alle tabellen uit de database bevat.

De reden hiervoor is simpelweg om de performance te verbeteren. Wanneer een bepaalde functionaliteit traag bleek te zijn, kwam dit vaak doordat de data nog niet in het geheugen stond, wat veel queries naar de database veroorzaakte. Het cachen van data zou in principe geen probleem moeten zijn, echter door de grote hoeveelheid data die gecached wordt ontstaan er nieuwe bottlenecks.

Op de productoverzichten van Divide’s webshops wordt momenteel alles omtrent de producten al gecached. Denk hierbij aan de componenten als de titel, de omschrijving, de prijs, de kleuren en maten. In sommige webshops worden nog queries uitgevoerd om productfilters op te halen om te tonen op het overzicht. Deze queries bevatten weinig data en hebben geen impact op de performance.

Direct bij het opstarten van de website wordt de gehele cache ingeladen. Dit inladen duurt kost veel CPU en geheugen en duurt vaak een aantal minuten. Wanneer het inladen klaar is, zal de webshop snel zijn en ervaart de klant weinig extra wachttijden. Hoewel dit fijn is voor de gebruiker, brengt het inladen van de gehele cache echter wel andere problemen met zich mee.

**Het product (in de database)**

Het product bestaat uit veel properties die allemaal gecached worden. Veel van deze properties worden niet voor elke webshop gebruikt omdat deze niet voor elke webshop nodig zijn. Het kan ook voorkomen dat properties helemaal niet meer gebruikt worden (obsolete). Deze properties moeten dan alsnog blijven bestaan om breaking changes te voorkomen. Ook kan het zijn dat bij oudere webshops deze properties nog steeds gevuld zijn, ook al worden ze niet meer gebruikt. Dit zou de performance eventueel kunnen beïnvloeden omdat het product dan groter wordt. De ‘product class’ bestaat momenteel uit 24 kolommen.

Een product kan meerdere varianten hebben. Een basis voor een variant is bijvoorbeeld een combinatie van maat en kleur. Wanneer een product beschikbaar is in twee kleuren met allebei 5 maten, zijn er dus 10 varianten aanwezig. Alle varianten zijn ook gecached. Een individueel product zonder varianten is ongeveer 1 kB groot. Dit is gecontroleerd door het product uit te lezen in JSON en dit om te zetten in Bytes.

**Pagina inhoud**

De inhoud die op de pagina’s van de webshop getoond worden is gedeeltelijk statisch en grotendeels dynamisch. Statisch houdt in dat het altijd dezelfde inhoud is, dynamisch heeft variërende inhoud. De belangrijkste pagina in dit onderzoek is het productoverzicht. De productoverzichten zijn volledig dynamisch omdat deze aan de hand van filters worden opgebouwd en weergegeven aan de gebruiker.

**Hosting database**

Divide maakt voor het opslaan van de data gebruik van één database. Deze database staat altijd gehost op een andere server en haalt daarmee druk weg van de webapplicatie. Dit zorgt echter wel voor netwerkvertraging, omdat alle data van een andere server moet worden opgehaald.

De database wordt gehost op een gedeelde server met databases van andere webshops. Wanneer de klant extra betaalt kan de database op een dedicated server worden gehost. Dit wordt bij klanten met grote hoeveelheden data en bezoekers aangeraden.

**Hosting webshop**

De hosting van de webshops is vergelijkbaar met de hosting van de databases. Alle webshops van kleinere klanten worden gehost op een gedeelde server. Klanten kunnen ook hun webshops tegen extra betaling op een dedicated server plaatsen. Dit komt vaak voor bij webshops die veel bezoekers ontvangen of klanten die dit zelf aanvragen.

## Doelstelling

De doelstelling van dit onderzoek is om inzichtelijk te maken hoe Divide door middel van distributed caching de performanceproblemen voor E-commerce websites kan verminderen.

Daarmee kan deze doelstelling opgedeeld worden in de volgende onderdelen:

* Inzicht krijgen hoe Divide distributed caching kan inzetten om haar performanceproblemen tegen te gaan;
* Voor Divide inzichtelijk maken in welke situaties distributed caching het best ingezet kan worden.

## Onderzoeksvragen

Om een volledig antwoord te kunnen geven op de hoofdvraag zijn deelvragen opgesteld. In deze paragraaf wordt besproken hoe de deelvragen tot stand zijn gekomen en welk resultaat van elke deelvraag verwacht wordt.

Om antwoord te geven op de hoofdvraag moeten een aantal onderzoeken uitgevoerd worden. Deze onderzoeken gaan over de volgende onderwerpen:

* **Performance**, wat is het, hoe is het te meten en hoe is het te beïnvloeden;
* **Distributed caching**, wat is het, hoe kan het gekarakteriseerd worden, welke systemen zijn beschikbaar en welk systeem is voor Divide het meest geschikt;
* **Testen**, welke testen moeten uitgevoerd worden en hoe moeten de testen uitgevoerd worden.

Op basis van deze onderwerpen zijn tijdens het opzetten van het plan van aanpak (Bijlage 4: Plan van aanpak) een aantal deelvragen opgesteld. Deze deelvragen leken goed afgebakend te zijn, maar naarmate de tijd vorderde bleken twee deelvragen niet voldoende te zijn.

Twee deelvragen waren te veel gericht op één aspect en kwamen niet goed van de grond. Deze deelvragen konden beter samengevoegd worden om één geheel te vormen. Dit waren de volgende deelvragen:

* Wat zijn de grootste oorzaken van performanceproblemen bij E-commerce websites?
* Met welke parameters kunnen verschillende soorten websites gekarakteriseerd worden?

Dit samenvoegen resulteerde in de deelvraag: ‘Wat wordt verstaan onder performance op het gebied van E-commerce en welke parameters hebben invloed hierop?’

Uiteindelijk is nog een deelvraag toegevoegd voor de informatieverzameling omtrent de testapplicatie: het ontwerpen, bouwen en testen. Er is daarom een nieuwe deelvraag toegevoegd: ‘*Op welke wijze kan een testapplicatie waarin performance parameters verwerkt zijn inzicht geven op de effecten bij het toepassen van een distributed cache systeem?*’.

De uiteindelijk opgestelde deelvragen met een korte toelichting zijn als volgt:

* **Wat wordt verstaan onder een distributed cache systeem en hoe is dit te karakteriseren?** Voor het testen van een distributed cache systeem, moet eerst een keuze gemaakt worden uit een type caching systeem. Deze deelvraag beantwoord welke karakteristieken voor Divide belangrijk zijn bij het kiezen van een distributed cache systeem.
* Na het vaststellen van relevante karakteristieken, beantwoorden we de volgende vraag: **Welke distributed cache systemen zijn momenteel beschikbaar en welke karakteristieken hebben deze?** Aan de hand van deze informatie wordt een keuze gemaakt uit het voor Divide meest van toepassing zijnde cache-systeem, om op basis hiervan performance aspecten te testen middels een testapplicatie.
* **Wat wordt verstaan onder performance op het gebied van E-commerce en welke parameters hebben invloed hierop?** Om duidelijk te maken wat performance is en hoe we performance van de testapplicatie kunnen beïnvloeden, wordt onderzoek gedaan naar performance en worden relevante parameters opgesteld.
* Om een duidelijk beeld te krijgen wat de testapplicatie moet gaan doen, wordt de volgende vraag gesteld: **Op welke wijze kan een testapplicatie waarin performance parameters verwerkt zijn inzicht geven op de effecten bij het toepassen van een distributed cache systeem?**
* Zodra bekend is welke karakteristieken de testapplicatie moet gaan genereren, zullen de tests uitgevoerd worden en zal de performance-informatie verwerkt worden. Door deze testen te interpreteren wordt bepaald op welke wijze een distributed cache het beste ingezet kan worden: **Welke adviezen kunnen gegeven worden voor praktische toepassing binnen Divide op basis van de testresultaten gegenereerd door de testapplicatie?** Met dit advies kan Divide een optimale strategie bepalen om distributed caching te implementeren.

## Onderzoeksmethoden

|  |  |  |
| --- | --- | --- |
| Deelvraag | Onderzoeksmethode | Motivatie |
| 1: Wat wordt verstaan onder een distributed cache systeem caching en hoe is dit te karakteriseren? | Beschrijvend onderzoek | Het in kaart brengen van de werking van distributed caching en hoe deze opgedeeld kan worden in karakteristieken. |
| 2: Welke distributed cache systemen zijn momenteel beschikbaar en welke karakteristieken hebben deze? | Marktonderzoek | Een aantal relevante systemen vinden en deze vergelijken op de gevonden karakteristieken. |
| 3: Wat wordt verstaan onder performance op het gebied van E-commerce en welke parameters hebben invloed hierop? | Deskresearch en exploratief onderzoek | Data verzameling uit reeds uitgevoerde onderzoeken. Wat verstaan anderen onder performance en met welke parameters kan deze performance beïnvloed worden? Parameters uit onderzoek en persoonlijke ervaringen. |
| 4: Op welke wijze kan een testapplicatie, gebaseerd op de performance parameters, inzicht geven op de effecten bij het toepassen van een distributed cache systeem? | Beschrijvend onderzoek | Alle informatie omtrent de testapplicatie verzamelen. Hoe wordt de applicatie ontworpen, hoe is het gebouwd, welke testen worden uitgevoerd, etc. |
| 5: Welke adviezen kunnen gegeven worden voor praktische toepassing binnen Divide op basis van de resultaten van de testapplicatie? | Pilotstudie | Een onderzoek om voor Divide de meest relevante strategie op te stellen, gebaseerd op een testapplicatie en de informatie uit voorgaande deelvragen. |

Tabel 2.1: De onderzoeksmethoden

## Afbakening project

In het volgende stuk wordt het project afgebakend zodat de scope en reikwijdte vooraf bekend zijn. In dit onderzoek wordt een testapplicatie ontwikkeld die duidelijk moet maken in welke mate distributed caching voor Divide effectief is om in te zetten en wanneer niet. Om de testapplicatie te realiseren en hier succesvolle tests mee uit te voeren worden vooraf randvoorwaarden opgesteld.

Om een distributed cache systeem te gaan testen met de testapplicatie, moet eerst een type cache gekozen worden. Dit wordt gedaan op basis van twee onderzoeken:

* Een onderzoek op welke wijze een distributed cache systeem te karakteristeren is;
* Een vergelijkend onderzoek om op de markt verkrijgbare distributed cache systemen te vergelijken met elkaar. Hieruit volgt een keuze die tijdens het bouwen van de testapplicatie ingezet zal worden.

De testapplicatie moet beïnvloedbaar zijn op basis van bepaalde parameters. Deze parameters zullen uit het eerste literatuuronderzoek komen. Daarnaast moet uit dit onderzoek duidelijk worden hoe performance te meten is, zodat dit later tijdens het uitvoeren van de tests bijgehouden kan worden.

Ten slotte zal een testplan worden opgesteld, waar alle uit te voeren testen in opgenomen zijn. Aan de hand van deze testen zal het praktijkonderzoek uitgevoerd worden. Uit deze testen wordt afgeleid wanneer en hoe het inzetten van een distributed cache voor Divide’s klanten profijtelijk is.

## Gewenste resultaat

1. Een literatuuronderzoek naar performance in de omgeving van E-commerce websites en hoe deze te beïnvloeden is op basis van parameters;
2. Een literatuuronderzoek naar de werking van distributed caching op welke deze dit te karakteriseren is;
3. Een vergelijkend onderzoek naar de diverse distributed caching systemen die momenteel op de markt zijn en een onderbouwde keuze voor een systeem;
4. Een praktijkonderzoek met behulp van een testapplicatie naar de mate van snelheidsverbetering in E-commerce websites onder toepassing van verschillende karakteristieken op basis van een geselecteerd distributed cache systeem bij E-commerce websites;
5. Een advies over het toepassen van distributed caching voor E-commerce websites op basis van verschillende parameters.

## Planning

Het gewenste resultaat is uit de zetten in verschillende acties en resultaten. In deze paragraaf wordt besproken in hoeverre deze planning is gehaald en of afwijkingen zijn opgetreden.

|  |  |  |  |
| --- | --- | --- | --- |
| Resultaat / Actie | Aantal uren | Deadline | Voldaan |
| Voorbereidend onderzoek, afstudeeropdracht, plan van aanpak, oriëntatie op werk | 96 uur | 07-09-18 | 07-08-18 |
| Literatuuronderzoek naar de oorzaken van performanceproblemen bij E-commerce websites | 80 uur | 30-09-18 | 01-09-18 |
| Literatuuronderzoek naar de achtergronden en werking van distributed caching | 80 uur | 16-10-18 | 21-09-18 |
| Een marktonderzoek naar diverse distributed caching systemen | 64 uur | 30-10-18 | 07-10-18 |
| Testplan opzetten en uitwerken voor prototype | 24 uur | 04-11-18 | 10-10-18 |
| Onderzoeken waar de testapplicatie gehost moet worden en deze omgeving opzetten | 48 uur | 11-11-18 | 21-10-18 |
| Uitzoeken welke type ontwerpen voor het prototype te maken en deze maken | 48 uur | 18-11-18 | 30-10-18 |
| Met behulp van de gemaakte ontwerpen, het prototype bouwen | 48 uur | 25-11-18 | 07-11-18 |
| Met behulp van het prototype tests uitvoeren | 80 uur | 13-12-18 | 01-12-18 |
| Scriptie schrijven, alle details uitwerken en één geheel van maken | 112 uur | 07-01-19 | 01-01-19 |

Tabel 2.2: Geplande en behaalde deadlines van het gewenste resultaat

In Tabel 2.2 zijn de (tussen)producten opgesteld met de gewenste deadlines. Naast de primaire producten (opgesteld in hoofdstuk 2.7), zijn hier ook de secundaire producten in opgenomen die belangrijk waren om de primaire producten te realiseren.

De datum bij het kopje voldaan is niet altijd exact aangezien na het voltooien nog kleine wijzigingen kunnen zijn doorgevoerd. Dit geeft een indicatie hoe het project op dat moment in de planning ervoor stond.

Zoals in de tabel is te zien, duurde het onderzoek gedeelte 2 weken langer dan gepland (zie Tabel 2.2). De oorzaak hiervan is dat onderzoek niet een van mijn sterkste punten is. Het uitlopen hiervan heeft geen negatieve impact op de planning gehad, omdat er marge in de planning zat. Deze marge is ontstaan omdat ik al iets eerder begonnen was met het afstuderen dan in eerste instantie was gepland.

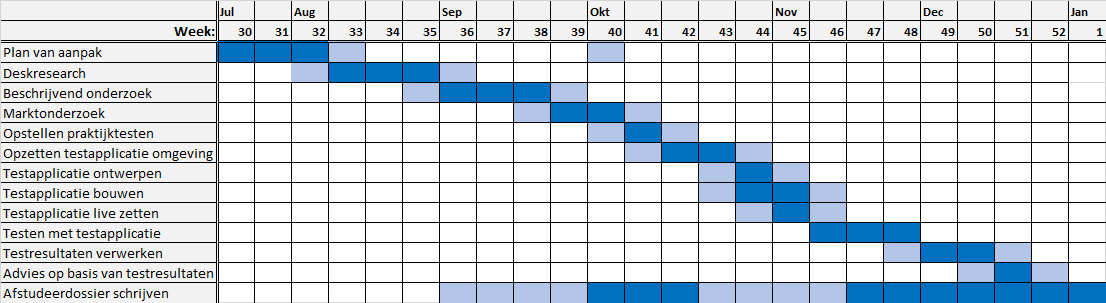
Nadat alle taken zijn voltooid is nog veel tijd gespendeerd om het afstudeerdossier compleet te maken. Deze tijd bestond vooral uit het schrijven van de stukken die nog niet compleet waren of überhaupt nog niet geschreven waren, het verwerken van feedback en het afronden van de laatste to-do’s.

**Planningsmethodiek**

Oorspronkelijk was het de bedoeling om met de Scrum-methodiek het project uit te voeren. De Agile-aanpak verliep uiteindelijk niet helemaal zoals het gepland was, omdat Divide nog in de beginfase zit van de implementatie hiervan. Dit zorgde voor veel onduidelijkheid waardoor de tussenproducten niet voldoende gecontroleerd werden.

Om dit probleem op te lossen is gekeken naar een aanpak waar het project zonder al te veel problemen op overgezet kon worden. Uiteindelijk is gekozen om de werkzaamheden in Excel bij te houden. (Figuur 2.1). De planning in Excel was een typische planning, waarbij de te voltooien tussenproducten verticaal werden opgesteld en de weeknummers horizontaal werden opgesteld. Dit alternatief werkte goed en gaf een mooi overzicht van het verrichte werk en de progressie in tussenproducten.

Wanneer een tussenproduct was afgerond, werd deze opgeleverd aan de inhoudelijk begeleider voor feedback. In de meeste gevallen werd de feedback mondeling besproken, waarna dit direct weer verwerkt werd. Het verwerken van de feedback is terug te zien in Figuur 2.1, waar nadat een oplevering is geweest, vaak de week erna nog aan het tussenproduct gewerkt wordt.



Figuur 2.1: Actuele planning zoals deze in de praktijk is verlopen

De gekozen softwareontwikkelmethode voor het bouwen van de applicatie wordt in hoofdstuk 6.1 toegelicht.

## Risico’s en maatregelen

In het project kunnen ongewenst risico’s ontstaan. Om deze risico’s vooraf te identificeren is een tabel opgesteld met risico’s en bijbehorende maatregel(en).

|  |  |
| --- | --- |
| Risico | Maatregel |
| Optreden van ziekte | Er is tijdens het project vanuit gegaan dat er geen ziekte optreedt, mocht dit toch gebeuren zullen afspraken gemaakt worden om eventuele achterstand in te halen. |
| Te grote hoeveelheid testen | In het geval van een te grote hoeveelheid aan testen, kunnen de testen geprioriteerd worden. Alleen de testen met de hoogste prioriteit worden nog uitgevoerd, zodat het onderzoek nog steeds relevant blijft. |
| Onvoldoende kennis | Wanneer het blijkt dat onvoldoende kennis aanwezig is, kan dit opgelost worden door de inhoudelijke begeleider en/of andere collega’s te raadplegen. |
| Kwaliteit onderzoek onder de maat | Om te voorkomen kwaliteit van het onderzoek onder de maat is, zal het onderzoek periodiek aan de inhoudelijke begeleider worden voorgelegd om van feedback voorzien te worden. |

Tabel 2.3: Risico's en maatregelen

# Onderzoek naar distributed caching

Wat wordt verstaan onder een distributed cache systeem en hoe is dit te karakteriseren? In dit hoofdstuk wordt beschreven hoe distributed cache systemen onder zijn te verdelen in karakteristieken. Deze karakteristieken worden uitgebreid toegelicht zodat duidelijk is wat voor invloed zij hebben bij het inzetten van een distributed cache systeem. Vervolgens wordt een selectie gemaakt van een aantal distributed cache systemen die daarna vergeleken worden in een matrix. Het systeem dat als beste naar voren komt zal uiteindelijk gebruikt worden bij de bouw van de testapplicatie. Deze testapplicatie wordt vervolgens in het onderzoek gebruikt om de resultaten in verschillende situaties en met verschillende parameters te bepalen.

## Wat is caching?

Caching zorgt ervoor dat verzoeken waarbij data opgevraagd wordt, sneller uit te voeren zijn. De gegevens die in de cache zijn opgeslagen, kunnen het resultaat zijn van een eerdere berekening of een kopie van elders opgeslagen gegevens. Een ‘cache hit’ vindt plaats wanneer de gevraagde gegevens zich in de cache bevinden. Bij een ‘cache miss’ zijn de gegevens niet aanwezig en moet de data alsnog in de oorspronkelijke bron (database) worden opgevraagd. Een cache miss is niet per se ernstig, aangezien het sneller is om te kijken of de gegevens aanwezig zijn, dan deze telkens opnieuw op te halen. (Han, et al., 2013) (Cache (computing), 2018)

Het RAM-geheugen kan ongeacht de fysieke locatie van de gegevens in het geheugen, de gegevens in vrijwel dezelfde tijd lezen of schrijven. Dit is in tegenstelling tot bijvoorbeeld de harde schijf aanzienlijk sneller omdat een harde schijf mechanische beperkingen kent zoals rotatiesnelheden en verplaatsing van de arm. Een SSD heeft deze mechanische beperkingen niet, maar is nog steeds minder snel als RAM-geheugen. (Random-access memory, 2018)

Caching is mogelijk sinds het bestaan van het random-access-memory (RAM-geheugen) en wordt bij veel software en hardware ingezet om performance te winnen. De eerste commerciële vorm van RAM-geheugen kwam in oktober 1970 op de markt, uitgebracht door Intel. (Intel 1103, 2018) Deze vorm van RAM wordt dynamic random access memory (DRAM) genoemd. DRAM maakt gebruik van condensators die constant dynamisch worden ververst om data op te slaan.

Naast DRAM maakt de computer ook gebruik van Static RAM (SRAM). SRAM is het interne geheugen van de CPU en wordt gebruikt om kopieën van data en instructies van RAM op te slaan die door de CPU uitgevoerd gaan worden. SRAM is nog sneller dan DRAM omdat het niet ververst hoeft te worden. Het is daarentegen duurder dan DRAM waardoor het slechts in kleine hoeveelheden ingezet wordt. Wanneer de opgevraagde data niet aanwezig is op het SRAM, wordt het opgehaald van het DRAM. Dit betekent dat een cache in meerdere niveaus achter elkaar gebruikt kan worden.

## De werking van een distributed cache

Bij in-memory caching wordt de data opgeslagen in het geheugen van de server waar de applicatie op dat moment draait. In het geval van distributed caching kan het cachen van data worden geabstraheerd naar een andere server. De distributed cache is een netwerk van één of meerdere servers, waar clients van kunnen consumeren. Op deze manier kan de cache eenvoudig groeien in omvang en transactionele capaciteit door middel van scaling-up én scaling-out. (Distributed cache, 2018)

Een distributed cache is schaalbaar en daarom aantrekkelijk om in te zetten voor applicaties met veel data en gebruikers. Toch zijn een aantal nadelen waar aandacht aan besteed moet worden.

Een distributed cache kan meerdere servers tegelijkertijd gebruiken. Wanneer de distributed cache meerdere servers omvat, kunnen problemen optreden die bij gedistribueerde systemen vaker optreden:

* Een storing bij een cache server;
* Een cache server die overbelast raakt;
* Netwerk congestieproblemen die het communiceren moeilijk maken.

In het ‘Distributed Cache System’ patent (Han, et al., 2013) door Xiao Han, Kun Liu, Peter Zhao, Edward Lu, Jessica Yang en Tim Ke wordt een oplossing voor dit soort problemen beschreven. Het distributed cache systeem is namelijk voorzien van een functie voor het intelligent toewijzen van cacheverzoeken tussen caches in een distributed cache.

Deze functie bestaat uit het alloceren van cacheverzoeken naar een eerste andere server, wanneer dit verzoek niet bij een van de andere caches doorkomt. Wanneer een vertraging wordt gedetecteerd zal de cache in een veilige modus worden gezet. Om te bepalen of deze cache weer verder mag, zal periodiek een verzoek worden gestuurd totdat deze cache weer opgang komt. Daarna zal de veilige modus weer uitgezet worden en vervolgt de cache zijn activiteiten. Het wacht totdat het de verzoeken alsnog kan sturen.

## Karakterisering van een distributed cache systeem

Een distributed cache is onder te verdelen in verschillende (niet)-technische karakteristieken. In deze paragraaf worden deze karakteristieken benoemd en toegelicht. De gekozen karakteristieken zijn gebaseerd op eigenschappen die bij elke distributed cache systeem aanwezig zijn. De geselecteerde karakteristieken zijn afkomstig van websites waar distributed caching wordt besproken, websites van de distributed cache systemen zelf en van de ervaring binnen Divide.

Een website die de verschillende distributed cache systemen indexeert en informatie over verzamelt is DB-engines (solid IT, sd). De website biedt veel informatie aan over relationele- en NoSQL-databasemanagementsystemen. Niet alle karakteristieken van deze website zijn voor dit onderzoek interessant. Er is daarom met de inhoudelijke begeleider een selectie gemaakt van de karakteristieken die voor Divide relevant zijn.

Naast de huidige selectie karakteristieken is ook aan collega’s uit hetzelfde team gevraagd welke karakteristieken belangrijk kunnen zijn. Hier zijn een aantal niet-technische karakteristieken uitgekomen, zoals de grootte van de community en de ingebruikname van het systeem.

De uit dit onderzoek gevonden karakteristieken worden in een vervolg marktonderzoek ingezet om een aantal distributed cache systemen met elkaar te vergelijken. Een aantal van deze karakteristieken zijn belangrijk en moeten dus minstens van een bepaald niveau zijn. Wanneer dit het geval is, wordt dit aangegeven in de individuele beschrijving van de karakteristiek. De overige karakteristieken wegen allemaal hetzelfde mee in het onderzoek. De gevonden karakteristieken zijn:

* Ondersteunt .NET Framework
* Ondersteuning datatypes
* Data partitionering
* Data replicatie
* Data consistentie
* Data concurrency
* Data durability
* In-memory opslag
* Ondersteuning cloud-oplossingen
* Beschikbare API voor .NET
* Back-up mogelijkheden
* Server besturingssysteem
* Licentie
* Ingebruikname systeem
* Grootte community

**Ondersteunt .NET Framework**Divide’s applicaties worden gebouwd in het .NET Framework. Het is dan ook belangrijk dat het systeem dat uitgekozen wordt dit ondersteunt. Wanneer het systeem dit framework niet ondersteund, kan gebruik gemaakt worden van een API die wel met het .NET Framework werkt. Wanneer geen API van een derde partij aanwezig is, is het systeem voor Divide onbruikbaar. Dit is een voorwaarde om het systeem te gaan gebruiken.

**Ondersteuning datatypes**Zoals eerder aangegeven is een distributed cache een key-value store. Hierin is de key-type vaak een string. De datatypes die een distributed cache systeem ondersteunt, verschilt per systeem. Een aantal distributed cache systemen ondersteunen enkel strings, andere systemen ondersteunen alle types. Divide programmeert in het .NET Framework, een aantal voorbeelden van datatypes zijn strings, lists en floats.

Divide heeft bij deze karakteristiek geen eisen voor een specifieke set aan types, maar haar voorkeur gaat uit naar een systeem wat in ieder geval strings ondersteund. In de optimale situatie ondersteunt het systeem alle types. Dit maakt het raadplegen van de data eenvoudiger, tevens hoeft niet telkens een conversie van string naar een ander type gedaan te worden.

**Data partitionering**Partitionering houdt in dat alle data die in het systeem opgeslagen zijn in stukjes gedeeld worden. De gehele dataset wordt gelijk verdeeld in partities en alle partities worden gelijk verdeeld tussen de deelnemende nodes. Een node is in deze een server. Dit zorgt ervoor dat de gehele dataset gedistribueerd is opgeslagen. Met deze aanpak is het mogelijk zoveel data op te slaan als past in het totale geheugen wat beschikbaar is op alle nodes. Hoe meer nodes, hoe meer data opgeslagen kan worden. Partitionering is tevens de meest schaalbare manier van dataopslag. (Apache Ignite, sd)

Wanneer een systeem partitionering niet ondersteunt, betekent dit dat de dataopslag gelimiteerd is aan één server. Voor Divide gaat de voorkeur naar een systeem wat wel kan partitioneren.

**Data replicatie**Replicatie is ongeveer het tegenovergestelde van partitionering. Bij partitionering wordt alle data in stukjes verdeeld over alle beschikbare nodes. In het geval van replicatie wordt deze data beschikbaar op elke node. Omdat de data redundant is kan dit kan voordelen en nadelen hebben. Het voordeel hiervan is dat wanneer een node uitvalt, deze direct vervangen kan worden met een andere node, het systeem zal dus geen storing ervaren. Het nadeel hiervan is dat wanneer de data van één van de nodes corrupt raakt, deze data over alle nodes corrupt zal raken.

**Data consistentie**Consistentie is voor Divide een belangrijk karakteristiek. Wanneer data inconsistent is, kunnen in de webwinkel van Divide acties plaatsvinden die niet mogelijk zouden moeten zijn. Denk bijvoorbeeld aan het afboeken van voorraad: een gebruiker besteld een broek waar nog één stuk voorraad van is. Wanneer een andere gebruiker deze broek ook aan het afrekenen is, kan deze broek in het geval van inconsistente data dubbel verkocht worden en eindigt dit in een nee-verkoop. (EhCache, n.d.)

Data consistentie kan op verschillende manieren gerealiseerd worden:

* Immediate consistency, nadat data is bijgewerkt, retourneert elke opvolgende aanvraag gegarandeerd de meest recente bijgewerkte data;
* Weak consistency, nadat data is bijgewerkt, kan niet gegarandeerd worden dat elke opvolgende aanvraag de meest recente data retourneert;
* Eventual consistency, is een vorm van weak consistency welke gebruikt wordt om een hoge availability te behalen. Wanneer het opslagsysteem garandeert dat geen nieuwe bijgewerkte data aanwezig is, zullen de onderliggende nodes de master node (bovenste node) kopiëren en het systeem bij alle opvolgende aanvragen hetzelfde antwoord retourneren;

**Data concurrency**Concurrency betekent dat delen van een proces buiten-volgorde uitgevoerd kunnen worden, zonder de uiteindelijke uitkomst te beïnvloeden. Bij een distributed cache systeem betekent dit dat meerdere gebruikers dezelfde data gelijktijdig kunnen opvragen en wegschrijven, zonder dat dit in fouten resulteert. Er zijn diverse manieren om concurrency te bereiken. In het geval van een distributed cache systeem wordt gebruik gemaakt van een speciaal lock-mechanisme. De eerste gebruiker die de data bijwerkt plaatst een lock op de gegevens, waardoor andere gebruikers het tijdelijk niet meer kunnen bijwerken. Wanneer de eerste gebruiker klaar is, wordt de lock opgeheven en kan de volgende gebruiker zijn data wegschrijven. De andere gebruikers zien de nieuwe data pas, zodra de vorige gebruiker klaar is met het bewerken van de data. (Oracle, n.d.) (Techopedia, n.d.)

**Data durability**Durability, ook wel de duurzaamheid van de data, zorgt ervoor dat wanneer een systeem crasht, de aanpassingen die gedaan zijn bewaard zullen blijven. De duurzaamheid van de data is voor Divide een belangrijk punt. Een voorbeeld bij een webshop is een bezoeker die op een webwinkel een bestelling plaatst. Een goede duurzaamheid zou ervoor moeten zorgen dat wanneer de website crasht, deze bestelling nog steeds doorgevoerd wordt en niet kwijtraakt.

Duurzaamheid van de data kan bereikt worden door de logs van de databewerkingen naar non-volatile memory door te spoelen (flushen), voordat de commit wordt bevestigd. Non-volatile memory is geheugen wat blijft bestaan, zelfs nadat de server is uitgezet. Zo kunnen deze bewerkingen na een crash alsnog worden uitgevoerd en gaat niets verloren.

**In-memory opslag**Deze karakteristiek houdt in dat de data die in het distributed cache systeem opgeslagen staat, wordt opgeslagen in het RAM-geheugen. Het opslaan van de data in het RAM-geheugen is voor Divide belangrijk, omdat het veel sneller is dan dataopslag op de harde schijf. Een cache systeem zou ook gebruik kunnen maken van dataopslag op een harde schijf. De voorkeur gaat uit naar een systeem wat data in het geheugen opslaat. Het RAM-geheugen is namelijk tot wel 6 keer sneller in het lezen en tot wel 4.5 keer sneller in het schrijven, dan een Solid State Drive (SSD). (Betts, 2015)

**Ondersteuning cloud-oplossingen**Een voorbeeld van een cloud platform wat distributed caching ondersteund is Azure. De gegevens zijn gehost in de cloud en kunnen in het dashboard van Azure bekeken worden. Wanneer de cache te groot wordt, kan deze eenvoudig opgeschaald worden. Aan cloud hosting zitten kosten verbonden, het is onbekend hoeveel dit is. Wanneer gebruik gemaakt wordt van cloud hosting, gaat de voorkeur van Divide uit naar Microsoft Azure. Divide bouwt al haar applicaties met behulp van Microsoft’s .NET Framework. Deze twee systemen zijn daarom eenvoudig te combineren. Eventuele alternatieven voor cloud hosting zijn Amazon Web Services (AWS) en Google Cloud.

Divide heeft nog geen concrete plannen om over te stappen op cloud hosting, maar vind het idee wel interessant. Deze karakteristiek zal daarom niet zwaar meewegen in de uiteindelijke vergelijking van systemen.

**Beschikbare API voor .NET**Een aantal distributed cache systemen biedt van zichzelf al een API aan voor .NET verzoeken naar toe te sturen. Een API bevat functies die men kan aanroepen om met het systeem te praten. Het kan gezien worden als een doorgeefluik tussen de applicatie en de cache. Niet alle distributed cache systemen bieden een API aan. In deze gevallen kan het wel zo zijn dat een persoon of bedrijf uit de community reeds een API voor het betreffende systeem gebouwd heeft.

Het is voor Divide belangrijk dat het distributed cache systeem een API aanbiedt waar Divide mee kan werken. Wanneer zij dit niet aanbieden, is het mogelijk dat wordt gekeken naar een API vanuit de community. Deze API moet wel goed onderhouden worden. Dit is veelal af te leiden uit het aantal sterren dat de API heeft gekregen en het aantal downloads van de API. Daarnaast kan gekeken worden naar de hoeveel commits die het afgelopen jaar zijn gedaan en de hoeveelheid code die is toegevoegd of aangepast. Divide gaat niet met een systeem aan de slag wat geen API beschikbaar heeft, dit zou een showstopper zijn.

**Back-up mogelijkheden**Het vullen van een cache is vaak een intensief proces voor de processor en harde schijf. Wanneer een cache corrupt raakt of crasht moet alle data opnieuw opgehaald en berekend moet worden. Dit zorgt voor downtime van het systeem en is zeer onwenselijk. Een aantal distributed cache systemen bieden hier een oplossing voor, namelijk ‘data persistence’. Dit houdt in dat periodiek point-in-time snapshots van de dataset naar een tekstbestand worden weggeschreven. Wanneer de cache dan crasht of corrupt raakt, kan deze data worden teruggezet zonder alles opnieuw op te hoeven halen en berekenen. Daarnaast kunnen deze bestanden worden gebruikt als logbestanden, wanneer data ingezien moet worden van een specifiek moment in het verleden. (Pfeil, 2010)

**Server besturingssysteem**Op welke besturingssystemen kan het distributed cache systeem draaien? De ondersteuning voor server besturingssysteem is een belangrijk karakteristiek omdat Divide momenteel enkel met Windows servers werkt. Wanneer het systeem Windows niet ondersteunt is het niet praktisch omdat beperkt kennis aanwezig is voor andere besturingssystemen. Toch bestaan technieken als Docker en Kubernetes die het mogelijk maken om applicaties die bijvoorbeeld enkel op Linux draaien ook op Windows te kunnen draaien. Wanneer deze technieken ondersteund zijn kan het systeem alsnog ingezet worden.

**Licentie**Bij licenties zijn twee opties mogelijk, namelijk ‘open source’ of ’commercieel’. In het geval van open source betekent dit dat mensen de code van de software kunnen inzien, aanpassen en verbeteren, omdat het publiekelijk beschikbaar is. Omdat iedereen de software kan bewerken, heeft dit een aantal mogelijke voordelen:

* Participatie door samenwerking;
* Rapid prototyping (het snel produceren van nieuwe testapplicaties);
* Transparantie;
* Gemeenschapsgerichte ontwikkeling.

Open source software is meestal gratis, maar kan ook worden aangeboden in bijvoorbeeld de vorm van een abonnement. Open source is onder te verdelen in licenties en elk van deze licenties heeft eigen condities. Deze condities gaan bijvoorbeeld over aspecten als het distribueren, verkopen en gebruiken van de software.

In het geval van commerciële software betekent dit dat het is ontworpen en ontwikkeld is voor licentieverlening of verkoop aan eindgebruikers of dat een commercieel doel dient. Voorheen werd commerciële software gezien als bedrijfseigen software. Tegenwoordig is dit niet meer zo, omdat open source software ook verkocht wordt. De meest bekende commerciële software zijn bijvoorbeeld het Windows Operating System (OS) en de meeste games die tegen betaling beschikbaar zijn.

Voor Divide gaat de voorkeur naar open source. Hier zijn een aantal redenen voor, namelijk:

* Omdat de broncode van het systeem door iedereen ingezien kan worden, zullen veel bugs en problemen vaak sneller gevonden worden;
* Het is in de meeste gevallen gratis, zo kan makkelijker getest worden of het systeem relevant is voor Divide. Wanneer dit niet het geval is, kan eenvoudiger een ander systeem gekozen worden.

**Ingebruikname systeem**Voor Divide is het belangrijk dat het systeem al veel gebruikt wordt. Wanneer een systeem veel gebruikt wordt, geeft dit een meer vertrouwelijk gevoel dan wanneer het systeem net nieuw is en nog niet veel gebruikt wordt. Daarnaast zal naar verwachting het aantal bugs beperkt zijn omdat anderen die al gesignaleerd hebben. In het geval van commerciële software is dit een minder zorgwekkend punt, aangezien dan gegarandeerde ondersteuning is. Bij open source is dit niet het geval. Daarnaast zijn bij veelgebruikte systemen op het internet vaak veel reviews te vinden en deze kunnen belangrijk zijn bij het zoeken van een dergelijk systeem. De waarde van deze karakteristiek wordt gemeten door het aantal downloads van de .NET API te bekijken in de NuGet store - dit is een algemeen verzamelpunt voor .NET API’s.

**Grootte community**De grootte van de community is voor Divide ook van belang. Met een grote community is de kans groter dat de ondersteuning van het systeem beter is. Daarnaast vinden bij een grote community vaak meer updates plaats omdat meerdere mensen aan het systeem werken. Tevens kijken meer mensen elkaars werk na en zal de aangepaste code in de updates ook secuurder zijn. Dit wordt gemeten door te kijken naar het aantal bijdragers op een project. De grootte van een community is lastig te meten, daarom nemen we nu het aantal personen wat heeft bijgedragen aan het project als maatstaf.

# Marktonderzoek distributed cache systemen

In dit hoofdstuk wordt een klein marktonderzoek gedaan om een goed distributed cache systeem te vinden. Er wordt een selectie gemaakt van een aantal distributed cache systemen op basis van bepaalde criteria. Deze selectie wordt vervolgens uitgezet tegen de gevonden karakteristieken uit hoofdstuk 3.3 in een matrix.

## Selectie distributed cache systemen

Om deze vergelijking op te zetten, wordt als eerste een selectie gemaakt uit het grote aanbod van distributed cache systemen. Dit aanbod bestaat uit meer dan 50 verschillende systemen. (solid IT, sd) Er moet uit deze systemen een globale selectie worden gemaakt met systemen die zich voor de situatie van Divide het beste lenen. Deze globale keuze is gebaseerd op een aantal punten. Een keuze maken uit een grote selectie kan lastig zijn. Daarom zijn de volgende selectiepunten met de inhoudelijke begeleider opgesteld:

* Bestaande kennis over caching systemen binnen Divide;
* Online reviews, wat zegt men online over het systeem;
* Gebruik door E-commerce concurrenten;
* Jaartal eerste uitgave;
* Activiteit op GIT-platform.

Er is in diverse bronnen gezocht en gekeken naar distributed cache systemen. Wanneer een systemen zich op basis van de bovenstaande punten goed scoort, wordt het systeem meegenomen in de vergelijking. De keuze bestaat uit maximaal vijf systemen om het marktonderzoek af te bakenen. Zo wordt het onderzoek niet te klein noch te groot. Het zoeken naar systemen duurde zolang het aantal nog niet op vijf stond. Nadat 5 potentiële systemen geselecteerd waren is het zoeken gestaakt, zodat de selectie afgebakend is.

De systemen zijn geselecteerd op basis van zoekopdrachten in zoekmachines (bijvoorbeeld Google). Er is gezocht op review websites om te kijken hoe de systemen worden beoordeeld. Daarnaast is gezocht naar wat webshop aanbieders (bijvoorbeeld Magento, Wix, etc) voor systemen beschikbaar stellen en ten slotte is gekeken naar de GitHub repositories van deze systemen om vast te stellen of de systemen actief beheerd worden.

Op basis van deze criteria is een aantal systemen gekozen. Voor elk systeem volgt een korte toelichting waaruit deze keuze blijkt. De geselecteerde systemen zijn als volgt:

* Memcached
* Redis
* NCache
* Aerospike
* Riak KV

**Memcached**

Memcached is een algemeen, distributed cache systeem. Het wordt vaak gebruikt om dynamische database-gestuurde websites te versnellen door gegevens en objecten in het RAM-geheugen te plaatsen om het aantal verzoeken naar externe gegevensbronnen (zoals een database of API) te verminderen (Memcached, 2018). Het systeem bestaat sinds 2003 en wordt nog steeds gebruikt door platforms zoals Magento. Magento is een open-source E-commerce platform, wat ondersteuning biedt voor session caching met behulp van Memcached. Tevens is Memcached op GitHub nog steeds actief met meer dan 100 bijdragers en krijgt het van 8000 personen een ster. (Memcached, sd)

**Redis**

Redis is een al enigszins bekend systeem binnen Divide. Redis is een gedistribueerd cache systeem uit 2009 en wordt online goed beoordeeld. Twee voorbeelden hiervan zijn peer-to-peer review website G2Crowd, waar Redis met een 8,8 wordt beoordeeld (G2Crowd, sd) en op TrustRadius, waar professionals software reviews met elkaar delen een 8.9 (TrustRadius, sd). Divide concurrent Magento biedt haar klanten de mogelijkheid Redis te gebruiken. Op GitHub wordt Redis beoordeeld met circa 32000 sterren en zijn meer dan 250 personen die aan het project bijdragen. (Redis, sd)

**NCache**

Binnen Divide werd gevraagd naar NCache te kijken, omdat NCache al een langere tijd op de markt beschikbaar is. NCache is uitgebracht in 2004 en was voorheen de marktleider op het gebied van .NET distributed caching. (DB-Engines, sd) NCache is ook beschikbaar op GitHub en krijgt 300 sterren. Het is nog redelijk actief met een aantal updates per maand van 15 bijdragers. (NCache, sd)

**Aerospike**

Door naar vergelijkbare systemen te kijken op review websites als G2Crowd en TrustRadius, is Aerospike naar voren gekomen. Aerospike behaalt gemiddeld een 8 op deze websites en is daarom interessant om te onderzoeken. Aerospike is een flash-optimized in-memory NoSQL database, uitgebracht in 2010. Er wordt nog steeds aan Aerospike gewerkt, met een klein aantal aanpassingen per week. (Aerospike, sd)

**Riak KV**

Riak KV is een gedistribueerd fouttolerant cache systeem dat in 2009 voor het eerst geïntroduceerd werd. Riak KV is fouttolerant omdat data nog steeds gelezen en geschreven kan worden zodra een node uitvalt. Dit is mogelijk omdat Riak KV geen gebruik maakt van master nodes. Riak is actief op GitHub met meer dan 60 bijdragers en 400 sterren. Riak KV wordt op website DB-Engines hoog aangeschreven en wordt daarom onderzocht. (solid IT, sd)

## Vergelijking van de distributed cache systemen

Wanneer alle systemen worden uitgezet met de bijbehorende waarden voor de karakteristieken resulteert dat in de volgende tabel:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Memcached | Redis | NCache | Aerospike | Riak KV |
| Ondersteuning datatypes | Strings | Alle types | Alle types | Alle types | Strings |
| Ondersteunt .NET Framework | Ja | Ja | Ja | Ja | Ja |
| Data partitionering | Nee | Ja | Ja | Ja | Ja |
| Data replicatie | Ja | Ja | Ja | Ja | Ja |
| Data consistentie | Nee | Immediate eventual consistency en eventual consistency | Eventual consistency en immediate consistency | Eventual consistency en immediate consistency | Eventual consistency |
| Data concurrency | Ja | Ja | Ja | Ja | Ja |
| Data durability | Nee | Ja | Ja | Ja | Ja |
| In-memory opslag | Ja | Ja | Ja | Ja | Ja |
| Ondersteuning cloud-oplossingen | Ja (Azure, AWS) | Ja (Azure, AWS, Google Cloud) | Ja (Azure, AWS) | Ja (Azure, AWS) | Ja (Azure, AWS) |
| Beschikbare API voor .NET | Ja | Ja | Ja | Ja | Ja |
| Back-up mogelijkheden | Nee | Ja | Ja | Ja | Ja |
| Server besturingssysteem | FreeBSD, Linux, OSD, OS X, Unix en Windows | BSD, Linux, OS X en Windows (met Docker) | Windows | Linux | BSD, Linux, OS X, Windows |
| Licentie | OS – BSD license | OS – BSD 3-Clause, commercial | OS – Apache 2.0 | OS – Apache en AGPL, commercial Enterprise | OS – Apache 2.0 commercial Enterprise |
| Ingebruikname systeem | ±700k | ±13M | ±15k | ±100k | ±35k |
| Grootte community | Gemiddeld (100+) | Groot (275+) | Klein (15+) | Klein (5+) | Gemiddeld (50+) |

## Resultaat

Met de karakteristieken uit paragraaf 3.3 is een vergelijking opgesteld van diverse systemen. De systemen zijn geselecteerd op basis van opgestelde criteria in paragraaf 4.1. Deze criteria zijn bijvoorbeeld de bestaande kennis over systemen binnen Divide en het vergelijken op review websites.

Vervolgens zijn alle systemen in de matrix uitgezet tegen de karakteristieken. Uit de matrix bleek de top 3 systemen:

1. Redis
2. Aerospike
3. NCache

De drie systemen zijn vergelijkbaar met elkaar, maar Redis heeft een grotere community en meer bedrijven maken gebruik van Redis. Zoals in paragraaf 3.3 is beschreven betekent een grotere community vaak meer en secuurdere updates. Bij een grote ingebruikname is meer kans dat de ontwikkelaar het systeem blijft ondersteunen. Op basis hiervan is Redis een betrouwbaardere keuze. Bij de in het hoofdstuk hierna beschreven testen zal Redis als distributed cache systeem worden gebruikt.

# Performance aspecten bij E-commerce omgevingen

Het aantal Nederlanders wat online aankopen doet steeg de afgelopen jaren met zo’n 3 miljoen mensen. Dit blijkt uit een rapport van het bedrijf E-commerce Foundation (Abraham, 2017), dat voor veel landen onderzoek doet naar statistieken rondom E-commerce. Het aantal mensen dat online aankopen deed telde in 2016 circa 15 miljoen mensen ten opzichte van 2013, waar dit aantal nog circa 11 miljoen was. Hieruit valt te concluderen dat een stijgende trend zit in het aantal bezoekers van webwinkels.

Deze stijgende trend is voor retailers positief, maar legt ook druk bij het continue aanbieden van een prettige en professionele online ervaring. Performanceproblemen kunnen in belangrijke mate de online ervaring negatief beïnvloeden waardoor de potentiële klant afhaakt. Dit leidt vervolgens tot inkomstenderving.

In dit hoofdstuk wordt onderzocht wat performance is en wat eventuele oorzaken van performanceproblemen zijn en wordt gekeken naar welke parameters op het gebied van E-commerce beschikbaar zijn. Dit onderzoek is voor Divide belangrijk omdat deze informatie in een vervolgonderzoek gebruikt gaat worden om performance te meten bij het uitvoeren van testen. De performance zal gemeten worden op basis van het beïnvloeden van de genoemde parameters.

## Performance en eventuele oorzaken van performanceproblemen

In dit hoofdstuk wordt uitgebreider onderzoek gedaan en bredere antwoorden gegeven dan noodzakelijk is om de lezer een compleet beeld te geven over wat performance is. Er wordt niet alleen gekeken naar de situatie van Divide maar ook naar de situaties daarbuiten.

De performance vanuit het perspectief van de provider is tweeledig:

* Het aantal aanvragen dat de website per seconde kan verwerken;
* De vertraging tussen de aanvraag en het antwoord naar de gebruiker.

Voor de gebruiker is met name het tweede punt relevant: de gebruiker wil namelijk vooral snel een pagina geladen hebben. De eerste component, het aantal gebruikers op een website, is voor een individuele gebruiker niet bekend en ook niet relevant.

In het onderzoek (He & Yang, sd) hebben Xubin He en Qing Yang een aantal praktische metingen gedaan om bovenstaande performance in een praktijksituatie te testen. De variabelen die zij gebruikte waren:

* Het type content op de pagina;
* Het aantal beschikbare servers;
* Het aantal gelijktijdige gebruikers.

Het type content is door He & Yang verdeeld in 4 gradaties:

* Een statische pagina. Hier is de inhoud van de pagina voor alle gebruikers exact hetzelfde;
* Een beperkt dynamische pagina: hierbij verschilt de pagina per gebruiker op enkele onderdelen;
* Een dynamische pagina: hierbij verschillen meerdere delen per gebruiker;
* Een volledig dynamische pagina: dit is een volledig op maat gegenereerde pagina en daarom vrijwel volledig verschillend dan de pagina’s die voor andere gebruikers gegenereerd zijn/worden.

De tweede en derde gradatie komen het dichts bij Divide in de buurt. Echter, om een compleet beeld te geven van de situatie worden de overige gradaties ook in het onderzoek opgenoemd.

In dit onderzoek wordt het onderzoek van He en Yang gebruikt. Het onderzoek van He en Yang is gepubliceerd in 2002, echter is het qua principe vandaag de dag nog steeds van toepassing. De serveromgevingen in die tijd gebruikten tragere CPU’s, RAM en harddisks. Vandaag de dag zijn alle genoemde componenten vele malen sneller, en zullen de komende jaren nog sneller worden. De te verwachten resultaten zullen met moderne hardware tientallen tot honderden keren sneller zijn, maar de parameters zullen vergelijkbaar zijn. De componenten zijn immers qua werking nog steeds hetzelfde, alleen sneller.

In een vergelijkbaar onderzoek van Wang, Balasubramanian, Krishnamurthy en Wetherall stellen zij het volgende: ‘Naarmate CPU's en netwerken evolueren, is de vraag die we stellen - hoe kunnen pagina laadtijden profiteren van het verbeteren van de netwerksnelheid of CPU-snelheid?’, zij concluderen dat de paginalaadtijd naar beneden zal gaan, maar de netwerktijd blijft hetzelfde. De resultaten suggereren dat het versnellen van zowel het netwerk als de CPU beter is voor de snelheid dan slechts een enkel hiervan. (Wang Sophia, Balasubramanian, Krishnamurthy, & Wetherall, 2013)

Volgens He en Yang is een statische pagina voor een server prettig: de inhoud is vast en de server hoeft hier niet aan te rekenen. Dit type pagina is uitstekend geschikt voor caching: de opgeslagen pagina kan hierbij onveranderd aan meerdere gebruikers beschikbaar worden gesteld. Dit in tegenstelling tot een volledig dynamische pagina waarbij caching geen nut heeft omdat elke andere pagina volledig anders is.

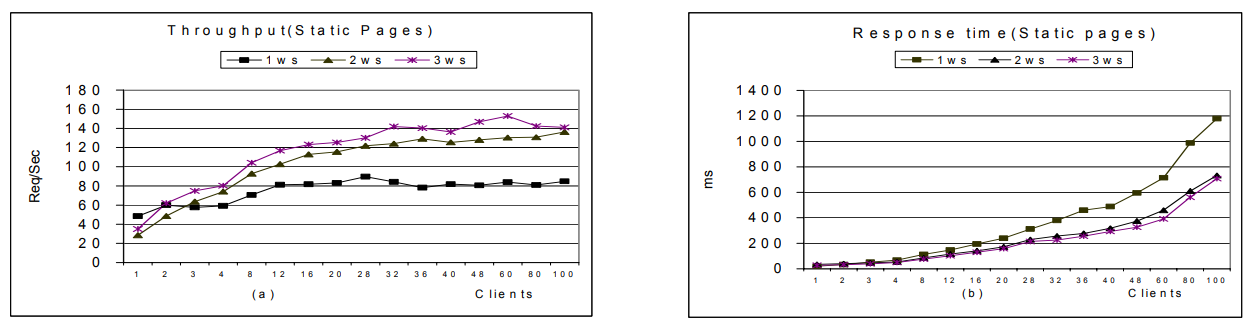
Het aantal beschikbare servers is een andere belangrijke parameter. Grote bedrijven als Google of Microsoft hebben vele datacentra met servers om verzoeken van de vele miljoenen gebruikers snel en efficiënt af te handelen. Bij het verdelen van de werklast tussen verschillende servers zal een verdelingsmechanisme gebruikt moeten worden. In de praktijk wordt dit een load balancer genoemd.

Tenslotte is het aantal gebruikers relevant. Bij meerdere gebruikers zal de server een stroom aan verzoeken krijgen en deze zo spoedig mogelijk afwerken. Net als bij een kassa in de supermarkt kunnen hier wachtrijen ontstaan. De gebruiker merkt dit door vertraging op de website.

De conclusies die He en Yang trokken zijn de volgende:

* Voor uitsluitend statische content spelen disk I/O en RAM een belangrijke rol;
* Wanneer naast de statische content ook dynamische content wordt toege­voegd, verandert het servergedrag. Vanwege de efficiënte verwerking door multi-processing en multi-threading, stijgt de doorvoer aanzienlijk. Volgens He en Yang is binnen de server architecturen nog voldoende ruimte voor verbetering;
* Bij een stijging van database activiteit, daalt de server performance vanwege zware vraag naar de CPU en disk I/Os;
* De gemiddelde doorvoer stijgt met een lagere werkdruk en daalt met een hogere werkdruk, wat suggereert dat een threshold wordt ingesteld op de binnenkomende aantal verzoeken. Die conclusie die zij trekken is dat door het toenemende verkeer congestie ontstaat, vergelijkbaar met files op een snelweg.

In de situatie van een statische pagina waren de testresultaten de volgende:



Figuur 5.1 Metingen bij statische pagina's (He & Yang)

In de Figuur 5.1 (a) wordt aangetoond dat het maximumaantal verzoeken per seconde met 1 (test)server rond de 80 ligt. Bij het gebruik van een tweede server zou idealiter de performance verdubbeld worden. In de verkregen resultaten blijkt echter de het aantal verzoeken per seconde maar te stijgen tot 140, dus ±80% verbetering. De reden hiervoor is dat in deze testopstelling de verdeling tussen de servers capaciteit vraagt dat ten koste gaat van het aantal verwerkte transacties. Een derde server leverde in de testopstelling zeer beperkte winst op. Hieruit blijkt dat het verdeelmechanisme zodanig veel capaciteit vraagt dat de resulterende performance nauwelijks stijgt.

Figuur 5.1 (b) geeft bij verschillende aantallen servers en gebruikers de responsetijd per situatie. Dit komt direct overeen met de perceptie van de gebruiker. Het is een directe representatie van de wachttijd die de gebruiker ervaart bij het laden van de pagina. Bij grotere aantallen gebruikers groeit de responsetijd exponentieel. Bij een tweede server wordt wederom verwacht dat de performance verdubbeld wordt, in realiteit blijkt dit maar ±70% verbetering te zijn. Net als bij het aantal verzoeken per seconde, lijkt een derde server bij de responsetijd ook vrijwel geen invloed te hebben. Het verschil is bij 100 clients zelfs bijna niet merkbaar.

In het onderzoek wordt ook gekeken naar belasting op de hardware in diverse aspecten. In de test die is uitgevoerd bij volledig dynamische pagina’s, blijkt dat bij een situatie met een hoge doorvoer de serverlading direct CPU-gebonden. Bij een hoger aantal servers stijgt namelijk het aantal afgehandelde verzoeken per seconde direct aan het aantal toegevoegde CPU’s. In vergelijking tot de doorvoer met één server, ±110, is de doorvoer met twee servers ±200 een forse verbetering. CPU speelt een belangrijke rol bij een goede doorvoer.

Daarnaast wordt ook gekeken naar de performance van de webservers bij een hoog aantal database verzoeken. Het aantal afgehandelde verzoeken daalt, wanneer de belasting stijgt. Bij een hoog aantal database verzoeken, wordt veel geëist van de I/O systemen. De RAM in de servers is niet groot genoeg om alle database connecties te cachen. Een groter aantal database verzoeken na dit punt zullen negatieve effecten veroorzaken, waardoor de verwerkingstijd toeneemt voor elk verzoek. Dit heeft een lagere doorvoer als gevolg.

## Parameters om performance te beïnvloeden

Zoals in de vorige paragraaf beschreven is performance op te delen in twee onderdelen, namelijk de doorvoer en de laadtijd van een pagina. Deze onderdelen zijn te beïnvloeden door diverse parameters. In deze paragraaf wordt met behulp van reeds uitgevoerde onderzoeken, performance tests en de in-house-ervaring van Divide met webshops een aantal van deze parameters benoemd en toegelicht. Een aantal van deze parameters zullen tijdens het testen met de testapplicatie gebruikt worden.

**Aantal gelijktijdige bezoekers**Een simpele, maar belangrijke parameter is het aantal gelijktijdige bezoekers op een website. Bij drukbezochte websites is het belangrijk dat de website niet bezwijkt onder het grote aantal gebruikers. Het is voor websites belangrijk dat de verzoeken efficiënt afgehandeld worden, waardoor niet te veel druk op de server komt. Het aantal gelijktijdige bezoekers is bij veel performancetests een standaard parameter en kan daarom eenvoudig getest worden. Tevens is dit voor Divide een van de belangrijkste parameters, omdat bij een groot aantal gelijktijdige bezoekers de website snel moet blijven. Wanneer de website heel traag wordt, of in het ergste geval uitvalt, valt dit niet goed bij bezoekers.

**Hoeveelheid data**De hoeveelheid data die een website in zijn database heeft bewaard, speelt een grote rol bij het doorzoeken van de database. Bij bijvoorbeeld een webwinkel met honderdduizend producten kan het ophalen van een aantal producten langer duren dan bij een webwinkel met maar duizend producten. De vertraging kan plaatsvinden omdat in een grotere set van producten gezocht moet worden om de juiste producten op te halen.

**Hoeveelheid getoonde data**  
De hoeveelheid data heeft op diverse manieren impact op de performance van de website:

* Data die over de lijn gestuurd moet worden;
* De data die de browser moet inladen.

Wanneer veel data over de lijn gestuurd moet worden zorgt dit ervoor dat de laadtijd van de pagina toeneemt. Daarnaast is het inladen van een grote hoeveelheid data ook voor de browser een zware taak en heeft dit invloed op de laadtijd. Wanneer de server klaar is met het aanleveren van de data, kan de browser nog bezig zijn met het inladen hiervan. Het is daarom belangrijk om deze hoeveelheid data beperkt te houden.

**Wijze van dataopslag**  
De wijze waarop data is opgeslagen kan op verschillende manieren variëren. Een aantal voorbeelden hiervan zijn:

* Een statische pagina, data opgehaald uit een bestand;
* Een database, bij het bezoeken van de pagina wordt de database geraadpleegd;
* Een database en cache, bij het eerste bezoek van de pagina wordt de data geraadpleegd en wordt deze data opgeslagen in de cache.

Deze parameter heeft invloed op de performance van webapplicaties, maar is wordt niet specifiek meegenomen in dit onderzoek. Dit is omdat tijdens het onderzoek verschillende types dataopslag getest worden. Het is daarom minder relevant om deze parameter mee te nemen in de lijst van parameters.

**Aantal berekeningen**Het komt zelden voor dat een gebruiker een pagina bezoekt waarmee hij een verzoek naar de database verstuurt en de database deze ‘rauwe’ data direct terugstuurt. Vaak gebeuren er namelijk nog diverse berekeningen over deze data. Dit komt bijvoorbeeld voor als een bezoeker een webwinkel bezoekt en gebruikmaakt van een zoekterm, zal de data eerst gefilterd moeten worden, in tegenstelling tot de situatie wanneer geen zoekterm gebruikt wordt. Aan de hand van de hoeveelheid data die aanwezig is, kan deze berekening lang duren.

Het aantal berekeningen is normaliter een belangrijk parameter, echter is het voor dit onderzoek niet relevant. Voorafgaand aan het testen zal de data al in de dataopslag wordt ingeladen. Dit is de reden dat geen berekeningen uitgevoerd worden en daarom is deze parameter in dit onderzoek niet relevant.

## Resultaat

In dit hoofdstuk wordt besproken wat performance is en welke parameters relevant zijn om performance te beïnvloeden in een E-commerce omgeving.

De performance van een website kan bepaald worden door de volgende twee aspecten te meten:

* Het aantal aanvragen dat de website per seconde kan verwerken;
* De vertraging tussen de aanvraag en het antwoord naar de gebruiker.

Om performance van een E-commerce omgeving te beïnvloeden is gezocht naar een aantal parameters. Op basis van de gekozen criteria zijn de volgende parameters tot stand gekomen:

* Het aantal gelijktijdige bezoekers;
* De hoeveelheid data in de database;
* De wijze waarop de data is opgeslagen;
* Hoeveelheid getoonde data.

Met het variëren van deze parameters in een testapplicatie kunnen realistische simulaties worden gedaan en daarmee testresultaten verzameld worden.

# Proces testapplicatie

In dit hoofdstuk worden de processen besproken die zijn doorlopen om de testapplicatie te realiseren. Dit gaat van het bepalen van de softwareontwikkelmethode, tot het bepalen van de methodieken en tools en tot slot op welke hardware de applicatie moet draaien wanneer het getest gaat worden.

## Softwareontwikkelmethode

Voor het ontwikkelen van de testapplicatie is een vorm van waterval gebruikt. Divide werkt sinds kort met een Scrum, maar zoals in hoofdstuk 1 besproken is daar tijdens dit project van afgeweken. Omdat Divide pas kort met Scrum werkt, worden nog niet alle processen volgens de Scrum methode gevolgd. De sprints hebben bijvoorbeeld een duur van één week in plaats van twee en er wordt nog geen formele oplevering aan het eind van elke sprint verwacht.

In samenspraak met de inhoudelijk begeleider is gekozen voor een lichtere vorm van waterval omdat niet alle stappen voor deze situatie relevant of noodzakelijk waren. Normaliter bestaat waterval namelijk uit zes stappen die achter elkaar worden uitgevoerd. Deze stappen zijn:

1. Het verzamelen van requirements
2. Het ontwerpen van de architectuur aan de hand van de requirements
3. Het bouwen van de applicatie aan de hand van het ontwerp
4. Het testen van de gebouwde applicatie
5. Het live zetten van de gebouwde applicatie
6. Het onderhouden van de live applicatie

Voor dit project zijn de laatste twee stappen niet relevant omdat de applicatie een proof of concept is. Dit betekent dat de applicatie niet live gezet noch onderhouden hoeft te worden. De andere stappen zijn daarentegen wel belangrijk om een kwalitatief goede testapplicatie te realiseren.

Er is tijdens het project ook wel eens afgeweken van het waterval traject wanneer bijvoorbeeld tijdens het bouwen van de applicatie nog een issue optrad. Er werd dan een stapje teruggezet om dit te corrigeren in het ontwerp.

## Gebruikte methodieken en tools

Tijdens de bouw van het project is gebruik gemaakt van diverse methodieken en tools. In deze paragraaf worden deze methodieken en tools toegelicht.

**Versiebeheer**

Tijdens het project wordt gebruik gemaakt van versiebeheer systeem GIT. GIT maakt het mogelijk om code online op te slaan en deze vanaf meerdere plekken te bereiken. Daarnaast kan met GIT alle wijzigingen worden ingezien en kunnen deze wijzigingen teruggedraaid worden. Met GIT kan ook in ‘branches’ gewerkt worden. Het is mogelijk een nieuwe branch maken per losse feature, zodat deze tijdens het ontwikkelen gescheiden blijven. Wanneer een feature dan af is, kan deze weer samengevoegd worden met de andere wijzigingen. Dit zorgt voor clean code en een goed overzicht. (Git, 2018)

**Hosting**

Voor het hosten van de testapplicatie zal de cloud service Azure als SaaS-oplossing gebruikt worden. (Microsoft, 2018) PaaS staat voor ‘Platform as a Service’ en houdt in dat geen rekening gehouden hoeft te worden met de inrichting van het besturingssysteem en middleware (denk bijvoorbeeld aan de database). Azure configureert ook het platform en zorgt dat alles up-to-date is. (Iaas, PaaS en Saas: wat zijn de verschillen?, 2018) De keuze voor deze oplossing zal later in dit hoofdstuk worden toegelicht.

**Dataopslag**

De testapplicatie maakt gebruik van Microsoft SQL Database 2017 (SQL-database). SQL-database is een relationele database waarin data wordt opgeslagen. Dit houdt in dat tabellen met gegevens in de vorm van kolommen en rijen opgeslagen staan, die op een later moment geraadpleegd kunnen worden.

De testapplicatie maakt gebruik van deze vorm van dataopslag omdat deze aansluit met de andere gebruikte Microsoftcomponenten en wordt vaak gebruikt in gevallen van kleine tot middelgrote databases. (Microsoft SQL server, 2018) Daarnaast maakt Divide momenteel al gebruik van SQL-database voor haar webapplicaties en daarom is de kennis hiervoor in ruime mate beschikbaar.

**Ontwikkelomgeving**

De ontwikkelomgeving waar de testapplicatie in wordt gebouwd is net als bij Divide’s huidige webshops in Visual Studio 2017 (VS2017). VS2017 is een volledig uitgeruste geïntegreerde ontwikkelomgeving (IDE) voor Android, iOS, Windows, web en cloud. (Visual Studio IDE, 2018) Dit houdt in dat met VS2017 voor al deze platformen applicaties ontwikkeld kunnen worden. Voor de testapplicatie wordt slechts het Web gedeelte gebruikt. Om de programmeerervaring beter te maken, wordt gebruik gemaakt van een Visual Studio extensie: ReSharper. ReSharper voorziet de door de gebruiker geschreven code van een kwaliteitscontrole en voorkomt veelgemaakte fouten. (ReSharper, 2018)

**Backend**

De testapplicatie wordt gebouwd in het .NET Core framework. Er is gekozen voor het .NET Core framework omdat het veel handige mogelijkheden met zich meebrengt. Zo heeft Core een ingebouwde in-memory cache en biedt het een standaard distributed cache interface aan. Daarnaast biedt Core ook ondersteuning voor onder andere depency injection wat inversion of control mogelijk maakt.

Een alternatief is het framework Web Forms, wat deel uitmaakt van het .NET Framework. Bij Web Forms moet veel nog zelfgebouwd worden. Er is geen ingebouwde ondersteuning voor in-memory cache en distributed cache. Tevens is er geen depency injection en zullen classes daarom sneller tightly coupled worden opgezet.

De programmeertaal die gebruikt wordt om de testapplicatie te schrijven is C#. Er is gekozen voor C# omdat Divide’s huidige applicaties hiermee worden geschreven.

Zoals voorheen aangegeven wordt gebruik gemaakt van MS SQL2017. Deze database kan aangeroepen worden door middel van Structured Query Language (SQL). De taal maakt het mogelijk om de database te raadplegen en hieraan bepaalde condities te verbinden. Het is in de testapplicatie nodig om alle producten uit de database op te halen om deze informatie uiteindelijk te verwerken en weer te geven.

**Frontend**

Om de producten uiteindelijk op een website te laten zien wordt gebruik gemaakt van HyperText Markup Language (HTML). HTML is de opmaaktaal voor webpagina’s, hiermee zijn webpagina’s op te maken zoals deze bijvoorbeeld ook in de tekstbewerker Word opgemaakt kunnen worden. Naast HTML wordt gebruik gemaakt van Cascading Style Sheets (CSS). In de CSS kan de stijl van de HTML worden vastgelegd. Hierin worden bijvoorbeeld een kleur en grootte van de tekst die in de HTML getoond worden vastgelegd.

## Hosting en hardware

De testapplicatie bestaat uit drie componenten, namelijk een webapplicatie, een SQL-database en een Redis cache. Het is voor de testapplicatie belangrijk dat deze componenten op een betrouwbare omgeving wordt gehost. Een betrouwbare omgeving zorgt voor integere resultaten en versterkt daarmee de uitkomst van dit onderzoek.

**Hosting**

Voor online webhosting zijn heel veel mogelijkheden beschikbaar. Om niet te ver buiten de scope van het project te treden, hebben wij deze keuze beperkt tot drie opties:

* Hosten bij servers welke Divide heeft gehuurd;
* Hosten bij een externe partij;
* Hosten bij Azure;

Alle opties zijn toepasbaar, maar hebben ieder hun voor- en nadelen. Om te beginnen met de eerste, de servers van Divide. De servers van Divide zou een goede keuze kunnen zijn voor dit onderzoek omdat de webshops van de klanten van Divide op deze servers gehost worden. Dit is een vergelijkbare situatie en heeft daarom positieve invloed op de betrouwbaarheid van de performanceresultaten. Het nadeel van deze oplossing is dat het opzetten van deze servers erg veel tijd kost en daarnaast de hosting kostbaar is. Normaliter doet een externe partij dit voor Divide, waarbij deze kosten meegerekend moeten worden. Om kosten te besparen kan de testapplicatie op een gedeelde omgeving gehost worden, maar dit heeft geen positief effect op de performanceresultaten en wordt daarom niet geprefereerd.

De tweede optie is een externe partij. Deze oplossing zou voor het onderzoek een goede keuze kunnen zijn, omdat de testapplicatie dan afzonderlijk op de afgenomen hardware gehost zal staan. Dit heeft wederom een positieve invloed op de performance, omdat er geen andere webapplicaties gebruik maken van deze hardware die de performance kunnen beïnvloeden. Het nadeel van hosting bij een externe partij is dat er veel ongewenst werk bij komt, bijvoorbeeld het opzetten van IIS, wat nodig is om .NET webapplicaties te draaien op servers. Voor dit deze werkzaamheden is tijdens het afstuderen niet voldoende tijd.

Beide besproken opties kunnen dienen als oplossingen om webapplicaties te draaien. Maar naast een webapplicatie moet ook een SQL-database en een Redis Cache worden opgezet. Om het onderzoek realistisch te maken moeten deze drie services allemaal op een aparte server worden gehost. Dit zou betekenen dat bij beide opties drie aparte servers moeten worden afgenomen of dat naar een oplossing gezocht moet worden die deze services los aanbiedt.

Een optie die dit mogelijk maakt is de derde en laatste optie, namelijk Azure. Azure biedt de mogelijkheid om een PaaS-oplossing af te nemen voor webapplicaties en daarnaast ook SaaS-oplossingen voor SQL-database en Redis Cache, die allemaal naadloos op elkaar aansluiten. Het voordeel van hosten bij Azure is dat zij de software en hardware beheren en up-to-date houden, dit haalt veel zorgen weg bij Divide als klant. Tevens is Divide silver partner van Microsoft en krijgen daarom alle ontwikkelaars van Divide een maandelijks budget van 150 dollar om te besteden aan Azure services. Ten slotte wordt het inzetten van Azure geprefereerd boven de andere opties omdat Divide interesse heeft om in de toekomst over te stappen naar Azure als hostingplatform en er bij Divide momenteel nog beperkte kennis aanwezig is.

Gebaseerd op de bovenstaande beredenering lijkt Azure zich het best voor deze situatie te lenen. De keuze is daarom Azure geworden. Azure wordt steeds vaker ingezet door grote bedrijven en is daarom een relevante ervaring bij het realiseren van de testapplicatie. Zoals genoemd is Divide geïnteresseerd om in de toekomst een overstap te maken naar Azure waardoor de testapplicatie een dienen als praktische ervaring.

**Hardware**

Nu de keuze voor de hosting is vastgelegd, kan een keuze voor hardware worden gemaakt. De keuze van hardware is belangrijk omdat dit directe invloed heeft op de performance van de testapplicatie. Azure biedt bij elke service een aantal varianten aan. Deze varianten variëren van een basispakket, dat slechts een aantal gebruikers aankan, tot een geïsoleerde omgeving die een groot aantal gebruikers aan kan. Deze varianten worden in Azure ook wel pricing tiers genoemd. Een hogere pricing tier betekent niet per se betere hardware, maar kan ook eigenschappen als een groter aantal gelijktijdig gebruikers of een grotere netwerk bandbreedte betekenen. In het volgende stuk worden de drie services besproken en de daarbij gekozen pricing tier.

*App Service*

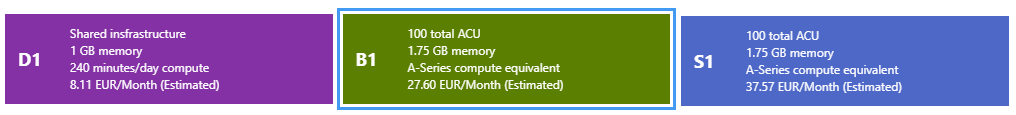
De eerste service die wordt besproken is de App Service. De App Service, ook wel de webapplicatie is waar de testapplicatie op draait. Uit de opties die Azure aanbiedt, zijn drie opties geselecteerd: D1, B1 en S1 (Figuur 6.1). De pricing tiers worden onderscheiden op verschillende eigenschappen.

De eerste eigenschap is **ACU** (Azure Compute Units), dit is te vergelijken met de rekenkracht van een vCPU (virtual CPU), dit zijn dedicated compute resources die nodig zijn om de applicatie te draaien. Optie D1 bevat 240 minuten per dag aan rekenkracht, dit houdt in dat de applicatie na 4 uur draaien geen resources meer beschikbaar heeft om te draaien. De overige opties hebben een onbeperkte rekentijd, deze werken altijd.

De hoeveelheid **GB-memory** is bij elke optie voldoende, deze varieert namelijk van 1 tot 1.75 GB. Voor het opslaan van alle producten is slechts 40 MB nodig. Eén product gebruikt in de test namelijk 1.6 kB aan geheugen, waardoor bij 25 duizend producten een totale omvang van 40 MB nodig is.

Voor het onderzoek wordt het liefst een optie gekozen die niet op een **shared infrastructure** staat, dit houdt in dat de performance met andere klanten gedeeld wordt. Bij Azure staan alle klanten op een gedeelde omgeving, maar in het geval van een shared infrastructure wordt de gegeven rekenkracht gedeeld. Bij B1 en hoger wordt deze rekenkracht exclusief voor de eigen applicatie gebruikt. Dit zou betekenen dat de performance van de testapplicatie niet altijd hetzelfde uit de test komt en dit dus een ongewenste invloed heeft op de resultaten.

Ten slotte kan pas vanaf optie B1 opgeschaald worden naar 2 of meer instanties. Dit betekent dat optie D1 afvalt. Het enige verschil tussen B1 en S1 is dat de mogelijkheid is om de App Service nog verder op te schalen en eventueel automatische back-ups te kunnen laten draaien. Optie B1 is daarom voldoende voor dit onderzoek.

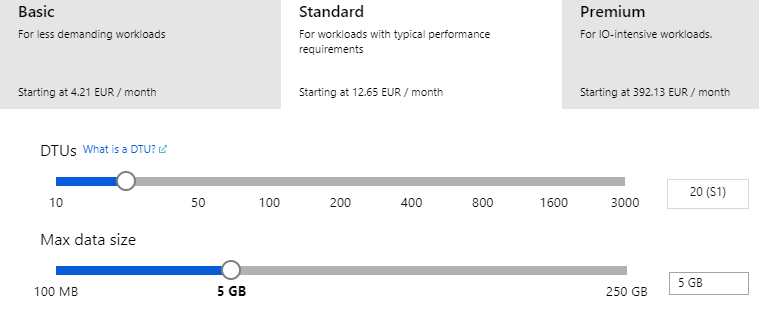


Figuur 6.1: De pricing tier van App Services

*SQL-Database*

Net als bij App Services, biedt Azure bij SQL-databases ook een variant van rekenkracht aan. Deze rekenkracht heet voor SQL-databases **DTU** (Database Transaction Unit) en bestaat uit een combinatie van CPU, memory en disk-IO. Een database voert acties uit zoals lezen, schrijven en updaten aan de hand van transacties. Deze transacties kunnen gelijktijdig worden uitgevoerd. Wanneer men bij Azure 10 DTU afneemt betekent dit dat ongeveer 10 gelijktijdige transacties mogelijk zijn.

Azure biedt SQL-databases aan vanaf 5 DTU tot 3000 DTU (Figuur 6.2). Voor de testapplicatie zou 250 DTU perfect zijn, aangezien de testen tot een maximumaantal van 250 gelijktijdige gebruikers gaan. Helaas is een DTU erg kostbaar (per 10 DTU kost het 12.65) en moet dus een lager getal gekozen worden. Tevens is 250 DTU voor een applicatie als de testapplicatie niet realistisch omdat bij grotere hoeveelheden data dit normaliter in de cache wordt gestopt zodat deze data niet telkens over het netwerk heen hoeft. Voor de testapplicatie is daarom besloten om 20 DTU af te nemen. Dit kost ongeveer 25 euro per maand en is acceptabel binnen het budget voor de testapplicatie. Met 20 DTU zou de testapplicatie nog goed moeten kunnen draaien, maar bij een groter aantal gelijktijdige gebruikers is mogelijk een hogere DTU aan te raden. Dit valt buiten dit onderzoek.



Figuur 6.2: De pricing tier van SQL-databases

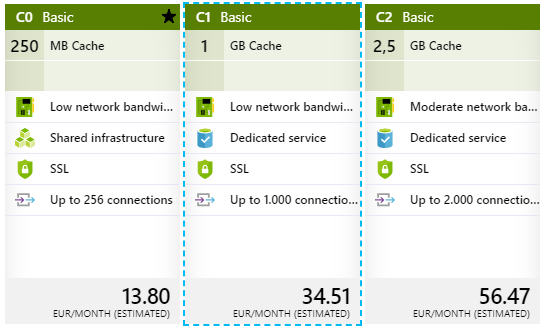
*Redis Cache*

De Redis Cache differentieert zichzelf op een aantal eigenschappen: de grootte van de cache, de beschikbare bandbreedte, het maximumaantal gelijktijdige connecties en het type infrastructuur (gedeeld of toegewijd). Wederom is gekozen voor drie opties om uit te kiezen. (Figuur 6.3)

Voor de **grootte van de cache** was zoals bij de App Service reeds berekend 40 MB voldoende. Alle opties bevatten daarom voldoende ruimte, omdat de grootte van de cache begint vanaf 250 MB. De volgende eigenschap is erg belangrijk voor de performance van het onderzoek, dat is namelijk de **beschikbare bandbreedte**. Bij een te lage beschikbare bandbreedte zal bij veel en grote verzoeken een vertraging ontstaan. Optie C0 biedt een snelheid van 100 Megabits per seconde, waar optie C1 en C2 een snelheid van 500 Megabits per seconde bieden. Dit maakt optie C1 en C2 aantrekkelijker voor de performance.

De derde eigenschap is het **maximumaantal gelijktijdige connecties** wat mogelijk is. Deze begint bij optie C0 op 256 connecties, dit is voor het onderzoek al voldoende, omdat maximaal 250 gebruikers tegelijkertijd de applicatie raadplegen. Toch zou het voor de performance beter zijn om net iets meer connecties af te nemen zodat dit geen reden voor vastlopen is. De laatste eigenschap is het **type infrastructuur**. Zoals voorheen al benoemd wordt dan de performance gedeeld met andere klanten die Redis Cache afnemen. Dit is voor het onderzoek ongunstig en wordt daarom het liefst voorkomen.

Wanneer alle eigenschappen worden vergeleken lijkt C0 een goede optie te zijn. Toch is dit niet helemaal zeker, omdat C0 bij veel eigenschappen op de grens zit ten aanzien van de performance restricties. Er is daarom gekozen voor optie C1 omdat deze in het geval van de testapplicatie geen performance restricties moet hebben. Ten slotte is niet gekozen voor optie C2 omdat de eigenschappen in dit scenario te zwaar lijken.



Figuur 6.3: De pricing tier van Redis Cache

# Ontwerp

Voor het ontwerp van de testapplicatie worden in dit hoofdstuk de requirements opgesteld, een architectuurpatroon bepaald en een analysis class diagram opgesteld van de applicatie. Ten slotte wordt toegelicht hoe de applicatie moet gaan werken in de praktijk.

## Requirements

Om ervoor te zorgen dat de testapplicatie valide resultaten produceert, moet het aan een aantal eisen voldoen (requirements). Omdat de testapplicatie een proof of concept is voor het gekozen distributed cache systeem, zal het slechts de requirements omvatten die vereist zijn om de performance te kunnen meten. De opgestelde requirements zijn onder anderen gebaseerd op het vooronderzoek.

Om ervoor te zorgen dat het systeem snel en eenvoudig performance resultaten produceert, moet aan een aantal functionaliteiten worden gedacht. Omdat in het systeem drie verschillende typen dataopslag worden getest, is het handig als het type opslag in de applicatie zelf veranderd kan worden. Dit brengt ons tot de eerste hoofdfunctionaliteit: het wisselen van het gebruikte type dataopslag tijdens runtime. Het is belangrijk dat dit tijdens runtime kan zodat alle instanties niet telkens opnieuw opgestart hoeven te worden.

Om de webshop van Divide te simuleren moet data worden opgehaald vanuit de dataopslag. Deze data wordt geserveerd in de vorm van een product en brengt ons tot de tweede functionaliteit: het weergeven van producten. Ten slotte de laatste functionaliteit: het systeem moet de performance metingen registeren. Deze functionaliteit moet de performancemeting op basis van een code in de database opslaan. Vervolgens kan met deze codes een presentatie van de resultaten gemaakt worden om hiermee een advies op te stellen.

Er zijn verder geen andere functionaliteiten nodig om de performance te meten. Om de productpagina van Divide te simuleren en hiervan de performance te meten zijn volgens de inhoudelijke begeleider de bovenstaande functionaliteiten voldoende.

Omdat het systeem zo weinig functionaliteiten omvat is besloten in samenspraak met de inhoudelijke begeleider om de overige requirements door de student op te laten stellen en hier vervolgens door de inhoudelijk begeleider een fiat op te geven. Naast de hoofdfunctionaliteiten zijn nog een aantal andere requirements nodig om de hoofdfunctionaliteiten mogelijk te maken. Deze requirements zijn als volgt:

* Het systeem moet kunnen wisselen tussen een database, in-memory cache en distributed cache tijdens runtime;
* Het systeem moet de producten in de cache op commando in kunnen laden;
* Het systeem moet een bepaalde hoeveelheid producten kunnen laten zien;
* Het systeem moet de performance afzonderlijk van het ophalen van de producten uit de dataopslag kunnen meten;
* Het systeem moet de performancemeting kunnen opslaan met een gegeven code;
* De cache moet altijd gevuld zijn voordat de testen gestart worden om een eerlijke situatie na te bootsen;
* Het systeem moet met MVC-architectuur worden gebouwd;
* Het systeem moet tijdens het testen altijd op twee instanties draaien;
* Het systeem moet tijdens het testen altijd dezelfde hardware hebben.

Naast de hoofdfunctionaliteiten zijn tevens een aantal requirements opgesteld die ervoor moeten zorgen dat de testen eerlijk verlopen, zijnde de randvoorwaarden voor het testen. Deze randvoorwaarden zijn bijvoorbeeld: het systeem moet tijdens het testen altijd op twee instanties draaien en altijd op dezelfde hardware draaien. Wanneer deze randvoorwaarden niet worden gehanteerd kunnen de resultaten verschillen, wat niet wenselijk is voor een juist advies.

## Architectuur

In deze paragraaf zal de keuze voor het architectuur patroon worden toegelicht, het ontwerp en de gemaakte architectuurkeuzes worden toegelicht op basis van de requirements en ten slotte wordt toegelicht hoe het ontwerp van de testapplicatie in de praktijk zal werken.

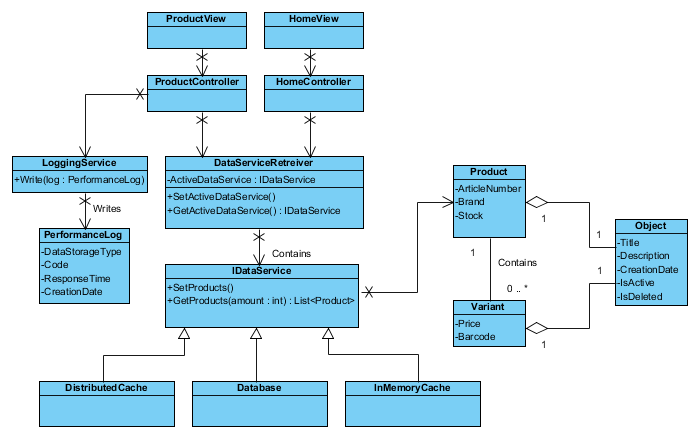
**Keuze voor architectuur patroon**

Bij het ontwerpen van de testapplicatie kon gekozen worden tussen diverse architectuur patronen. Divide’s webshops zijn gebouwd in ASP.NET Web Forms. Web Forms is een verouderde architectuur van Microsoft om webapplicaties mee te bouwen. Een nieuwe vorm om webapplicaties mee te bouwen is de MVC-architectuur van Microsoft. Deze architectuur haalt veel overhead weg en maakt applicaties overzichtelijker omdat het beter is opgezet. De applicatie baseren op MVC leidt ertoe dat er een betere separation of concerns is, inversion of control kan worden toegepast en het gereed is voor onder anderen microservices. Tevens levert dit meer kennis op over MVC, wat voor Divide en mijzelf in de toekomst een voordeel kan zijn.

De MVC-architectuur wordt in verschillende frameworks aangeboden. Er is gekozen voor MVC van Microsoft omdat deze architectuur net als Web Forms ook met het .NET Core framework samenwerkt. Het hosten van de applicaties blijft dan ook op dezelfde manier werken. Wanneer was gekozen voor een andere architectuur had dit geheel herzien moeten worden.

## Ontwerp applicatie architectuur

Om een architectuur te ontwerpen die voldoet aan alle eisen, wordt per requirement gekeken hoe deze in de architectuur verwerkt kan worden. Daarnaast worden de gemaakte architectuurkeuzes toegelicht en zou het eindproduct een ontwerp moeten zijn wat aan alle requirements voldoet.



Figuur 7.1: Het analysis class diagram van de testapplicatie

*Het gebruik van MVC*

Omdat is gekozen voor de MVC-architectuur, moet gebruik gemaakt worden van controllers. In de applicatie zal gebruik gemaakt worden van twee controllers. De eerste controller is bedoeld voor de homepagina waar ingesteld kan worden welk type dataopslag moet worden gebruikt. De tweede controller is voor het weergeven van producten die uit de dataopslag worden opgehaald. Er zijn niet meer controllers nodig omdat de applicatie geen andere eisen heeft naast deze twee functionaliteiten.

*Het wisselen van het type dataopslag in de applicatie*

Om het wisselen van het type dataopslag tijdens runtime (terwijl de applicatie draait) mogelijk te maken, moest worden gezocht naar een oplossing die hier het beste voor geschikt was. Aangezien het drie type dataopslag betreft die allemaal hetzelfde moeten doen: het ophalen en het zetten van producten, leek het verstandig om het repository design pattern toe te passen. Een repository is een class die generieke functies bevat die nodig zijn om data op te halen uit een bron. Het design pattern is geïmplementeerd door voor elke type dataopslag een interface te implementeren met Get en Set functionaliteiten. De interface wordt IDataService genoemd.

Een andere oplossing voor dit probleem had een abstracte class kunnen zijn met drie derived classes voor elk type dataopslag die de Get en Set functies overschrijven. Uiteindelijk is voor een interface gekozen omdat het toepassen van een design pattern de beste opzet is voor de testapplicatie.

Op dit momoment zijn drie verschillende repository implementaties van verschillende typen dataopslag die allemaal de interface IDataService implementeren. Nu moet een oplossing worden gekozen om deze types dataopslag individueel tijdens runtime aan te kunnen roepen. Dit kan opgelost worden door het type dataopslag hardcoded in de applicatie te zetten. Dit is echter niet flexibel, omdat dan telkens de applicatie opnieuw opgestart moet worden.

Een andere oplossing voor dit probleem is het bouwen van een bemiddelaar die op aanvraag de huidig geselecteerde repository teruggeeft. Met deze oplossing kan tijdens runtime een andere source worden gekozen en lijkt daarom de beste oplossing voor dit probleem. Deze class wordt in het ontwerp DataServiceRetreiver genoemd en is verantwoordelijk voor het aanleveren van de juiste bron (type dataopslag). De class moet naast het kunnen ophalen van de huidige geselecteerde bron deze ook opnieuw in kunnen stellen.

*De applicatie moet producten kunnen laten zien*

Zoals reeds beschreven worden twee controllers gebruikt. Een van deze controllers is de ProductController die wordt ingezet om producten te tonen wanneer de productpagina geraadpleegd wordt. De ProductController vraagt met behulp van de DataServiceRetreiver het huidige type dataopslag op en roept daar de Get functie aan om de producten op te halen. Wanneer in de querystring een amount wordt meegegeven, zal de ProductController dit getal meesturen naar de implementatie van de dataopslag waarbij dit aantal aan producten wordt opgehaald. Wanneer deze producten zijn opgehaald worden deze teruggestuurd naar de view om te tonen op de pagina.

Het product dat wordt teruggegeven is de Product class in het ontwerp. Het Product heeft een aantal properties die niet zijn opgenomen in het design. Deze properties zijn gevuld met willekeurige data om de situatie van Divide te simuleren. Aan dit product zijn een aantal varianten gekoppeld en een object met algemene informatie zoals bijvoorbeeld een titel en creatiedatum. De product class is een versimpelde versie van de class die momenteel in Divide’s webshop gebruikt wordt. De class is versimpeld omdat een generieke grootte wordt gebruikt van een product.

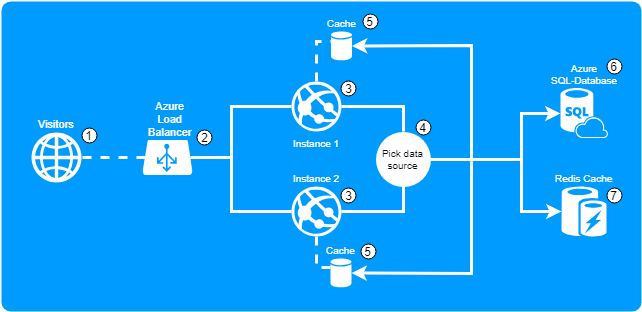
*Het meten van performance*

De testapplicatie is gebouwd voor het meten van de performance van verschillende typen dataopslag. De performance die gemeten moet worden is de tijd waarin het type dataopslag de producten ophaalt en terugstuurt. De resterende acties zijn niet relevant omdat deze voor iedere situatie anders kunnen zijn. Een voorbeeld hiervan is het inladen van de HTML.

De performance wordt gemeten in de ProductController class. Deze meting zal plaatsvinden nadat de actieve IDataService is opgehaald. De meting begint voor het aanroepen van de GetProducts functionaliteit en eindigt zodra deze antwoord heeft gegeven. Op deze manier zijn alle resterende acties uitgefilterd en bevat de gemeten tijd enkel de reactietijd van het type dataopslag. Een PerformanceLog zoals deze in het ontwerp heet, bestaat uit het type datastorage, een code, een reactietijd en een creatiedatum. De code wordt gebruikt om verschillende tests te onderscheiden. Deze logs worden weggeschreven naar de database door de LoggingService waarin een Write functie plaatsvindt.

## Applicatie in de praktijk

In bovenstaande paragraaf is omschreven hoe de architectuur van de testapplicatie technisch is opgezet. In deze paragraaf wordt beschreven hoe de opgezette architectuur in een Azure omgeving functioneert om hierbij een beeld te geven hoe de architectuur in de praktijk is ingezet.



Figuur 7.2: Werking applicatie in Azure omgeving

Bovenstaande Figuur 7.2 geeft de werking van de testapplicatie in de Azure omgeving aan. Van links naar rechts: bezoekers komen bij de applicatie binnen vanaf het internet (1) en worden door de load balancer (2) van Azure geleidt naar de op dat moment meest geschikte instance (3).

Nadat de load balancer een instance heeft aangewezen, wordt in de ProductController het ingestelde type dataopslag opgehaald van de DataServiceRetreiver (4). Hier kunnen drie verschillende implementaties uit volgen waar producten uit opgehaald kunnen worden in de instantie. Deze implementaties verwijzen naar de cache (5) (uniek per instantie), de SQL-database (6) of de Redis Cache (7). Wanneer de producten zijn opgehaald worden ze weergegeven op de pagina.

# Bouw

In dit hoofdstuk wordt alles omtrent de realisatie van de testapplicatie benoemd en toegelicht om een beter beeld te geven in hoe de testapplicatie tot stand is gekomen.

## De bouw van de applicatie

In deze paragraaf wordt beschreven hoe de belangrijkste onderdelen van de testapplicatie zijn gebouwd zonder hierin de uitdagingen mee te nemen. De testapplicatie is gebouwd aan de hand van het ontwerp dat in hoofdstuk 7.3 is toegelicht. In de volgende paragraaf worden de uitdagingen die tijdens het bouwen zijn opgetreden besproken.

**Nieuw project**

Omdat de testapplicatie losstaat van de huidige architectuur van Divide’s webshops, zal deze in een nieuw ASP.NET Core project gebouwd worden.

**De bemiddelaar van de huidige type dataopslag**

Er zijn drie verschillende typen dataopslag beschikbaar. Voor elk van deze typen dataopslag is een eigen repository implementatie gebruikt. Om de huidige type dataopslag eenvoudig op te kunnen halen en opnieuw in te kunnen stellen, is de DataServiceRetreiver bedacht.

De DataServiceRetreiver bevat een Dictionary<string, IDataService> (Figuur 8.1) waarin alle drie de implementaties zijn aangemaakt. Dit aanmaken gebeurd bij het opstarten van de applicatie, zodat deze classes niet telkens opnieuw aangemaakt hoeven te worden.

De huidig geselecteerde IDataService wordt vastgelegd door een static string property (Figuur 8.1), die dient als een key voor de dictionary. De property is static gemaakt zodat alle bezoekers dezelfde waarde terugkrijgen en deze niet voor elke gebruiker individueel gezet hoeft te worden.

Wanneer om de huidige dataopslag wordt gevraagd, zal met behulp van deze property de dictionary worden aangeroepen om het juiste type dataopslag terug te geven. Wanneer de dataopslag weer veranderd moet worden, hoeft enkel deze property worden gewijzigd naar een andere key.



Figuur 8.1: DataServiceRetreiver properties

**Opzetten van de pagina’s**

De applicatie bestaat uit twee pagina’s: de homepagina en de productoverzicht pagina. Voor dit project wordt gebruik gemaakt van MVC, dit betekent dat voor deze twee pagina’s een model, view en controller moeten worden aangemaakt.

*HomeController*

De homepagina is verantwoordelijk voor het tijdens runtime kunnen wisselen van het type dataopslag. Het ViewModel van de homepagina bevat een enum met alle beschikbare typen dataopslag. (Figuur 8.3) Deze enum wordt in de view van de homepagina weergegeven, zodat deze kan worden gebruikt om het type dataopslag te veranderen.

Wanneer op de button wordt gedrukt om het type dataopslag te wijzigen, wordt in de HomeController een POST-functie aangeroepen. Deze functie roept vervolgens met behulp van dependency injection de DataServiceRetreiver op, waarmee de huidige type dataopslag wordt veranderd naar de op dat moment geselecteerde dataopslag. Dit resulteert in een bericht waarin staat dat het gelukt is en hoeveel producten zijn ingeladen. (Figuur 8.2)

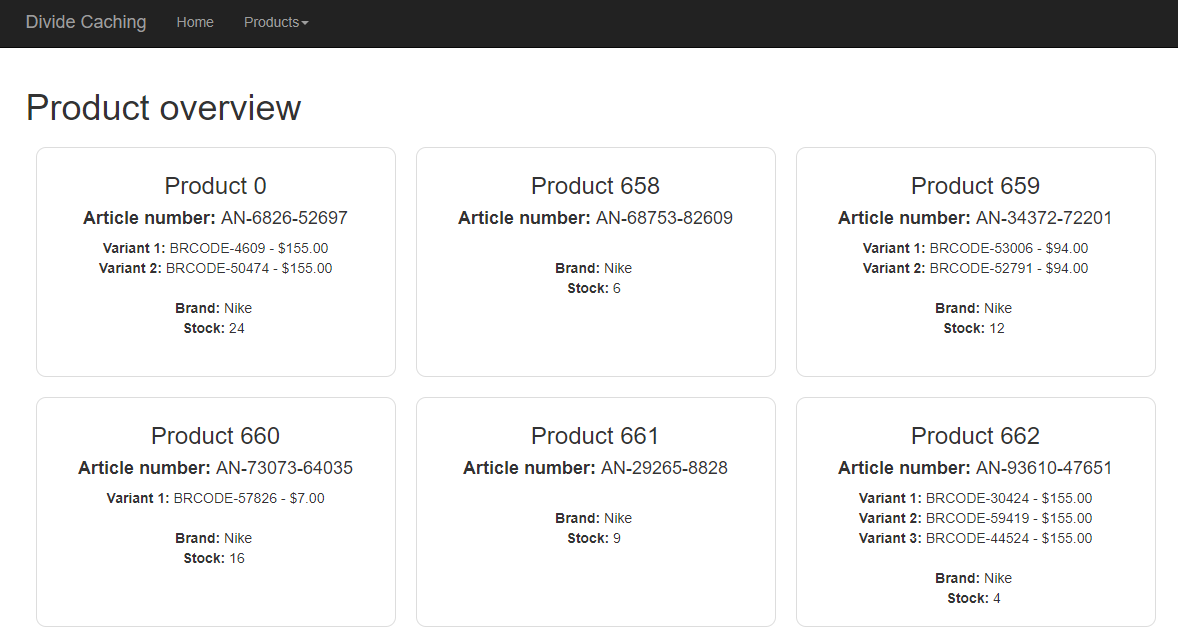
|  |  |
| --- | --- |
| Figuur 8.2: De homepagina | Figuur 8.3: Keuze uit alle typen dataopslag |

*ProductController*

De productoverzicht pagina is verantwoordelijk voor het weergeven van een vooraf bepaalde hoeveelheid producten. Deze hoeveelheid kan ingesteld worden door een aantal mee te geven in de querystring, die vervolgens wordt uitgelezen in de GET-functie in de controller. Daarnaast kan hier ook een string meegegeven worden om de geproduceerde resultaten met een unieke code te markeren. Met deze code kunnen de resultaten achteraf weer worden opgehaald.

De ProductController vraagt bij elk verzoek met behulp van depency injection de DataServiceRetreiver en LoggingService op. Met behulp van de DataServiceRetreiver wordt de juiste repository opgevraagd. Zodra de repository is opgehaald wordt een stopwatch gestart om te meten hoe lang het ophalen van de producten gaat duren. Wanneer de producten zijn opgehaald, stopt de stopwatch en wordt de tijdsduur van het verzoek met behulp van de LoggingService gelogd onder de meegegeven code.

De opgehaalde producten worden vervolgens in de ProductViewModel bewaard die wordt teruggegeven aan het einde van de functie. In de view van het product worden deze producten weergegeven als een productoverzicht pagina. (Figuur 8.4)



Figuur 8.4: Productoverzicht pagina

**Opzetten van de repositories**

Zoals het ontwerp aangeeft bevat de applicatie drie repository implementaties. De repositories implementeren de interface IDataService, die twee functionaliteiten bevat: GetProducts en SetProducts. De GetProducts- functionaliteit bevat ook nog een paramater: amount, met deze parameter kan aangegeven worden hoeveel producten opgehaald moeten worden.

Met de SetProducts-functionaliteit worden alle producten in de cache geladen. Met de GetProducts-functionaliteit worden de producten uit de cache of database opgehaald. De implementatie van deze twee functionaliteiten worden voor elke repository beschreven.

*Database repository*

Voor de database is gekozen voor een code first database, omdat het indien nodig eenvoudig is uit te breiden. Daarnaast hoeft de database niet meer met de hand aangemaakt te worden, dit wordt gedaan bij het opstarten van de applicatie.

Om de database aan te roepen is een helper class gemaakt, de AppDbContext. In deze class bevinden zich alle DbSets (tabellen). Wanneer een DbSet property wordt opgevraagd, worden deze verzoeken automatisch omgezet in queries.

Voor de database repository hoeft alleen de GetProducts-functionaliteit geïmplementeerd te worden. De SetProducts-functionaliteit heeft in deze situatie geen waarde, omdat de producten niet gecached hoeven te worden.

Bij het aanroepen van de GetProducts-functionaliteit wordt met behulp van dependency injection om de AppDbContext class gevraagd. Deze geeft vervolgens door middel van een query de gevraagde hoeveelheid producten terug uit de database.

*Memory cache repository*

Bij het opvragen van de memory cache repository werd met behulp van dependency injection de AppDbContext en de IMemoryCache implementatie opgevraagd. De IMemoryCache class is door Microsoft in het .NET Framework ingebouwd om out-of-the-box caching mogelijk te maken.

Bij het aanroepen van de SetProducts-functionaliteit worden de producten met behulp van de AppDbContext class uit de database gehaald en in de cache gestopt door de Set-functie aan te roepen van de IMemoryCache implementatie.

Bij het ophalen van de producten met de GetProducts-functionaliteit wordt gekeken of de producten al in de cache staan. Als dit nog niet zo is wordt de SetProducts-functionaliteit aangeroepen om de producten eerst in te laden. Vervolgens worden de producten alsnog teruggegeven. Dit is gedaan zodat altijd producten worden teruggeven en geen foutmelding kunnen ontstaan.

*Distributed cache repository*

Om een verbinding op te zetten met Redis wordt gebruik gemaakt van een open source library genaamd StackExchange.Redis. Deze library is gebouwd door de community van StackExchange en maakt het koppelen met Redis eenvoudiger. (StackExchange, 2018)

Voor de SetProducts-functionaliteit is wederom via dependency injection de AppDbContext opgehaald om de producten in de cache te laden. Dit inladen wordt door middel van pipelines gedaan. Dit is een feature die beschikbaar wordt gesteld door de gebruikte library. Het gebruik van pipelines houdt in dat meerdere verzoeken tegelijkertijd (asynchroon) naar Redis gestuurd kunnen worden, zonder te wachten op antwoord van elk verzoek. Op deze manier kunnen producten veel sneller worden ingeladen in de cache.

Het ophalen van producten gaat per 50 stuks, deze keuze wordt toegelicht in hoofdstuk 8.2. Bij een groter aantal producten worden de verzoeken qua inhoud snel te groot en gaat dit ten koste van de performance. Wanneer meer dan 50 producten worden opgevraagd zullen meerdere verzoeken naar Redis worden verstuurd. Met een gedistribueerd systeem moet een balans worden gevonden met het aantal verzoeken en de grootte van de verzoeken. Net als bij het zetten van de producten, verloopt het ophalen van de producten ook met behulp van een pipeline.

**Het genereren van producten**

Voor het genereren van een reeks willekeurige producten is code geschreven die dit uitvoert. De code bestaat uit een functie die een parameter accepteert met de hoeveelheid producten die aangemaakt moeten worden. De code is een simpele for-loop waarin met behulp van de AppDbContext producten worden toegevoegd aan de database.

## Uitdagingen

Bij het bouwen van de applicatie ben ik een aantal uitdagingen tegengekomen. In deze paragraaf worden deze uitdagingen besproken en hoe hiermee is omgegaan.

**Inconsistentie opvragen huidige dataopslag**

In eerste instantie werd de keuze van het type dataopslag in het geheugen (memory) opgeslagen. Tijdens het testen van de applicatie is ondervonden dat de applicatie soms producten uit twee verschillende typen dataopslag haalde. Dit werd aangetoond omdat het type dataopslag in de logs worden opgeslagen en deze soms varieerde.

Omdat de testapplicatie uit meerdere instanties bestaat, betekent dit dat bij het instellen van de databron deze alleen voor de huidige instantie ingesteld wordt. Om dit op te lossen moest de key worden opgeslagen in de distributed cache, zodat altijd alle instanties dezelfde dataopslag zullen gebruiken.

Het is echter niet wenselijk om bij elk bezoek aan de productpagina de huidige dataopslag op te halen uit de distributed cache. Al deze verzoeken stapelen op en zorgen voor vertraging bij de distributed cache. Om het aantal verzoeken te verminderen is gebruik gemaakt van een property met backing field. Bij het opvragen van deze property wordt de distributed cache eenmalig bevraagd en wordt de key vervolgens in het backing field gezet. Dit voorkomt onnodige verzoeken naar de distributed cache en houdt de applicatie snel.

**Redis Cache**

De manier waarop de data opgeslagen wordt in distributed cache systemen is erg belangrijk. Om de resultaten van de distributed cache voor dit onderzoek zo efficiënt mogelijk te maken, is gekeken naar de beste manier van dataopslag.

Het eerste scenario dat is getest was het opslaan van alle producten onder één key. Dit bleek erg inefficiënt te zijn vanwege alle overhead die meekomt bij elk verzoek. Wanneer bijvoorbeeld 50 producten ophaalt worden, krijgt het proces eerst de gehele collectie terug om daar vervolgens 50 producten uit te filteren.

Er zijn twee situaties vergeleken voor het opslaan van data. Om binnen de scope van het project te blijven is voor een tweetal situaties gekozen. Deze situaties zijn gekozen omdat deze het meest relevant zijn ten aanzien van de dataopslag.

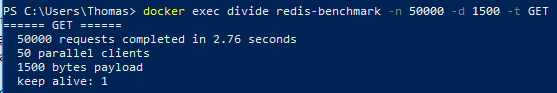
In de eerste situatie worden alle producten individueel opgeslagen onder een unieke key. Deze situatie is handig omdat alle producten hierdoor individueel te identificeren zijn. Toch blijkt dit een uitdaging, omdat bij het tonen van de producten met een groot aantal gelijktijdige bezoekers het aantal verzoeken zeer hoog opliep. Bijvoorbeeld 250 gelijktijdige gebruikers die 200 producten opvragen geeft als resultaat 50.000 verzoeken.

Deze 50.000 verzoeken zorgde voor latency problemen met de distributed cache waardoor gezocht is naar een alternatieve opzet. Deze latency problemen zorgden ervoor dat de eerste verzoeken snel werden afgehandeld maar naarmate meer verzoeken kwamen de verwerkingstijd sterk toenam.

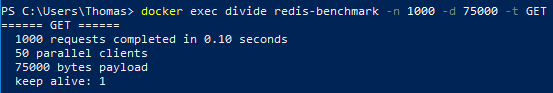
De tweede situatie die is getest bestond uit het opslaan van groeperingen van producten. Met deze situatie is getest met 50 producten onder één key. Voor het ophalen van 200 producten door 250 gelijktijdige gebruikers zijn dan slechts 1000 (250x4) verzoeken nodig. Deze aanpak resulteerde in een wenselijk resultaat en is daarom toegepast voor de testapplicatie.

Een derde situatie had alle data (producten) onder één key kunnen zijn. Dit keuze is echter niet verstandig aangezien dan telkens de gehele dataset over het netwerk verstuurd moest worden.

Deze aanpak is bevestigd door het ingebouwde benchmark systeem in Redis. Voor één product werd een omvang van 1.5 kB genomen. Om de situaties goed te simuleren worden voor de eerste situatie 50.000 verzoeken gedaan van 1.5 kB (250 gelijktijdige bezoekers x 200 producten van 1.5 kB) en voor de tweede situatie 1.000 verzoeken gedaan van 75.000 kB (250 gelijktijdige bezoekers x 4 productsets (200 producten) van 75 kB).



Figuur 8.5: Redis benchmarktest met 50.000 verzoeken van 1.5 kB



Figuur 8.6: Redis benchmarktest met 1000 verzoeken van 75 kB

De benchmarktest van de eerste situatie resulteerde in 2,76 seconden (Figuur 8.5) en de benchmarktest van de tweede situatie (Figuur 8.6) resulteerde in 0,10 seconden. Naar aanleiding van deze resultaten wordt de tweede situatie met meerdere producten per verzoek gekozen.

## Testen van de applicatie

Om ervoor te zorgen dat de testapplicatie betrouwbare resultaten genereert moet de applicatie eveneens getest worden. In deze paragraaf wordt uitgelegd op welke wijze dit is gedaan.

Wanneer je voor Visual Studio de juiste licentie hebt, kan je gebruik maken van de ingebouwde loadtest functionaliteit. Het opzetten van een loadtest in Visual Studio is een makkelijk proces. Naast het bestaande webproject moet een *Web Performance and Load Test Project* (Figuur 8.7) worden toegevoegd.

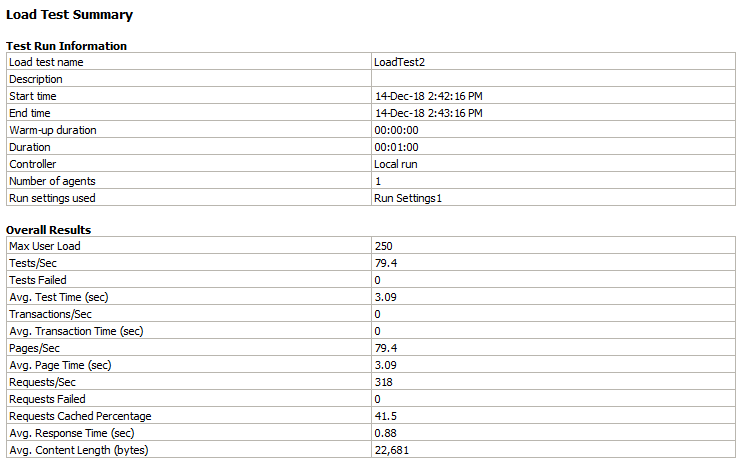
Wanneer dit project is toegevoegd kan een webtest worden aangemaakt. In deze webtest is het mogelijk een of meerdere links toe te voegen die getest moeten worden. Nadat de webtest is aangemaakt kan een loadtest worden aangemaakt. In deze loadtest zijn vele instellingen mogelijk, bijvoorbeeld het aangeven met welke browser(s) de pagina moet worden bezocht. Het belangrijkste is het aangeven welke link(s) getest moeten worden. Wanneer wordt gekozen voor meerdere links is het mogelijk een link meer bezoekers toe te wijzen dan de ander. Voor dit project is gekozen voor de Google Chrome webbrowser met alleen de link naar de productpagina.



Figuur 8.7: Loadtest project in Visual Studio

Nadat deze instellingen goed staan kan de test uitgevoerd worden. Wanneer een scenario getest werd is gekozen voor 250 gelijktijdige gebruikers. De testen draaiden op een gereserveerde lokale computer. Tevens is het kostbaar om testen uit te voeren via Azure hetgeen voor de testen niet noodzakelijk was.

Om de testresultaten uit te lezen is gebruik gemaakt van de ingebouwde log-functionaliteit. Iedere test werd een unieke code toegewezen die vervolgens uitgelezen werd ter toetsing van het resultaat. Daarnaast geeft Visual Studio aan het einde van een test een overzicht terug waarop resultaten te zien zijn (Figuur 8.8).



Figuur 8.8: Loadtest samenvatting in Visual Studio

# Testen

## Opzetten van testscenario’s

Om een zo compleet mogelijke dataset te verzamelen, is gekeken naar de verschillende parameters en de bijbehorende waarden. De waarden van deze parameters zijn gebaseerd op getallen die dicht bij de werkelijke getallen liggen van klanten bij Divide. De reden hiervoor is dat wanneer Divide voor een klant wil zien of het inzetten van distributed caching rendabel is, de daadwerkelijke getallen hiervoor aanwezig zijn. Alle gekozen waarden zullen daarom gebaseerd zijn op realistische aannames.

De parameters die tijdens het opstellen van de testen gebruikt gaan worden, komen uit hoofdstuk 5.2. Deze parameters zijn als volgt:

* **Type dataopslag**, welke situaties van dataopslag zijn mogelijk;
* **Aantal gelijktijdige bezoekers**, hoeveel bezoekers bezoeken de testpagina tegelijktijdig, ofwel hoeveel mensen vragen tegelijktijdig data op;
* **Hoeveelheid opgeslagen data**, hoeveel data in de vorm van producten wordt opgeslagen in de database en/of cache;
* **Hoeveelheid getoonde data**, hoeveel data in de vorm van producten worden getoond wanneer de bezoeker de pagina bezoekt;

Deze parameters zullen in het volgende stuk kort toegelicht worden inclusief de bijbehorende waarden die tijdens het testen gebruikt zijn.

**Type dataopslag**

Het type dataopslag kan verschillen in drie vormen van dataopslag, namelijk:

* Database (geen caching), voor vergelijkingsmateriaal;
* In-memory cache, de huidige vorm van caching;
* Distributed cache, de vorm van caching die wordt onderzocht.

**Aantal gelijktijdige bezoekers**

Het aantal bezoekers dat tegelijktijdig de pagina aan zal roepen tijdens de test is het aantal gelijktijdige bezoekers. De waarden die tijdens de testen gebruikt zullen worden zijn de volgende:

* 10
* 50
* 100
* 250

Divide heeft voor al haar klanten een Google Analytics account. Hier worden onder andere alle gelijktijdige gebruikers gemeten. Er is voor deze reeks waarden gekozen omdat het aantal gelijktijdige bezoekers van Divide’s klanten volgens Google Analytics binnen deze reeks vallen en iets daarboven voor toekomstbestendigheid. De getallen variëren van een webshop met weinig verkeer tot een webshop met veel verkeer.

**Hoeveelheid opgeslagen data**

De hoeveelheid opgeslagen data houdt in hoeveel producten zijn opgeslagen in de webshop. Ook al worden deze producten niet altijd getoond, heeft dit toch invloed op de performance van een website. Bij een grotere hoeveelheid producten moet namelijk een grotere bron geraadpleegd worden en heeft dit invloed op de performance. De volgende waarden zijn gekozen voor deze parameter:

* 1.000
* 5.000
* 12.500
* 25.000

Veel van Divide’s klanten hebben een aantal producten opgeslagen dat ligt tussen 1.000 en 25.000. . Dit is gecontroleerd om een aantal van Divide’s klanten steekproefsgewijs af te gaan en te controleren hoeveel producten in de database aanwezig waren. Hoewel 25.000 producten niet vaak voorkomt, is dit toch getest voor toekomstbestendigheid.

**Hoeveelheid getoonde data**

De standaard hoeveelheid getoonde data verschilt per klant. Dit kan liggen tussen de 20 en 100 producten. Vaak bieden webwinkels toch de mogelijkheid om op één pagina meer dan 100 producten weer te geven, zodat de klant minder van pagina hoeft te wisselen. Deze waarden zijn wederom gebaseerd door te kijken naar webshops van Divide’s klanten. De uiterste waarde (200) zal niet vaak gebruikt worden maar is voor toekomstbestendigheid toegevoegd. De gekozen reeks waarden is als volgt:

* 50
* 100
* 200

**Uit te voeren testen en vastleggen testresultaten**

De opgestelde testen bestaan uit een combinatie van de vastgelegde parameters en bijbehorende waarden. Er zijn drie typen dataopslag, voor elk type dataopslag worden 48 testresultaten verzameld:

* 1 type dataopslag;
* 4 verschillende hoeveelheden gelijktijdige gebruikers;
* 4 verschillende hoeveelheden opgeslagen data;
* 3 verschillende hoeveelheden getoonde producten.

Dit komt uit op een aantal van 1 x 4 x 4 x 3, oftewel 48 testen per type dataopslag. Alle testresultaten worden vastgelegd in een tabel. In de bijlage (Bijlage 1: De testresultaten) staan alle uit te voeren testen. De opgestelde testen geven het onderstaande resultaat:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Type dataopslag** | **Gelijktijdige bezoekers** | **Hoeveelheid data** | **Getoonde data** | **Laadtijd app** |
| 1 | Database | 10 | 1000 | 50 |  |
| 2 | Database | 10 | 1000 | 100 |  |
| 3 | Database | 10 | 1000 | 200 |  |

Tabel 9.1: Voorbeeld opgestelde testen

## Testscenario’s uitvoeren

In deze paragraaf wordt alle processen rondom het uitvoeren van de testscenario’s besproken.

**Opdelen van de testen**

De testen zijn opgedeeld in secties van de hoeveelheid opgeslagen data die werd getest. Deze parameter is namelijk het lastigste om te veranderen en werd daarom pas als laatste aangepast. Zo werden eerst alle testen met 1.000 producten in de database uitgevoerd, vervolgens de testen van 5.000, 12.500 en ten slotte 25.000 producten.

Zo ontstonden 4 secties van 36 testen, die nogmaals opgedeeld werden in kleinere secties, namelijk per type dataopslag. Dit is zo gedaan omdat dan per 12 testscenario’s even een kleine pauze genomen kon worden om tegelijkertijd ook het type dataopslag te kunnen veranderen. Dit resulteerde in 12 secties van 12 testen per sectie.

**Metingen en weergave**

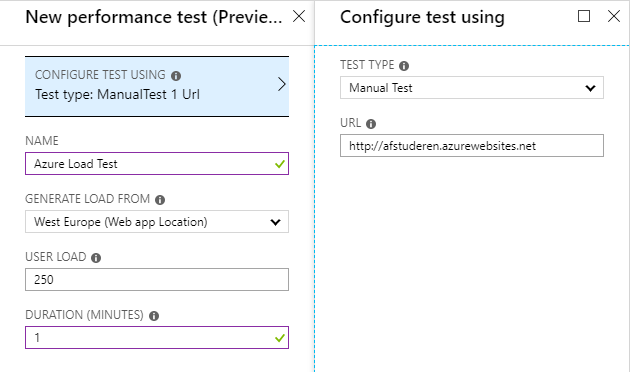
Het inladen van een pagina bestaat vaak uit meerdere elementen, bijvoorbeeld de afbeeldingen, tekstuele inhoud en eventueel externe componenten. Het element wat voor dit onderzoek belangrijk is, is de tijd dat de dataopslag nodig heeft om de data (producten) op te halen uit de bron. De andere elementen worden niet gemeten omdat deze vaak niet consistent zijn. Dit zou een negatieve invloed op de metingen hebben wat voor een minder betrouwbaar resultaat zorgt. Om deze reden zullen de performancemetingen alleen naar de performance van het ophalen van de producten kijken en zullen de overige elementen niet worden meegenomen in de resultaten.

De performance (laadtijd van de pagina) zal worden opgeslagen in de vorm van milliseconden. Er is voor deze manier van weergave gekozen omdat dit gebaseerd is op ervaringen bij Divide waarbij de laadtijd van een pagina veelal onder de 1 seconde ligt. Tevens zullen de resultaten bij het gebruik van seconden minder precies zijn dan bij het gebruik van milliseconden.

**Testmethode**

De testmethode die tijdens dit praktijkonderzoek gebruikt gaat worden is een loadtest. Een loadtest is een proces waarbij het systeem onder een bepaalde lading wordt gezet en de performance hierbij gemeten wordt. (What is Load Testing?, 2018) De loadtesten zullen worden gedaan in Azure omdat zij dit als een service aanbieden en de applicatie daar al gehost wordt.

In het dashboard van Azure kan je een nieuwe load test aanmaken voor je eigen applicatie. In deze load test geef je aan hoeveel gelijktijdige gebruikers de applicatie moeten bezoeken, voor hoeveel minuten deze gebruikers de pagina moeten bezoeken en ten slotte welke link(s) ze moeten bezoeken. In Figuur 9.1 is te zien hoe dit in Azure gepresenteerd wordt.



Figuur 9.1: Het aanmaken van een nieuwe load test in Azure

Voor dit onderzoek is gekozen voor een testduur van 1 minuut. Het gekozen aantal bezoekers bezoekt in deze minuut meerdere malen de website om zo een daadwerkelijk actieve gebruiker te simuleren. Omdat de gesimuleerde gebruiker meerdere malen de pagina bezoekt geeft dit een voldoende realistisch resultaat voor dit onderzoek. Daarnaast kost het gebruik van testminuten in Azure geld. Wanneer het aantal testminuten wordt verhoogd, verhoogt dit de kosten met een factor van het aantal gekozen minuten.

De load test wordt door Azure uitgevoerd door de bijbehorende parameters in te vullen. Deze parameters hangen af van het te testen testscenario. Ten slotte kan de afkomst van de testen bepaald worden in het dropdown menu ‘Generate Load From’ (Figuur 9.1). Voor dit project is gekozen om de testen vanuit West-Europa uit te voeren, omdat daar de webapplicatie gehost wordt. De overige parameters worden in de applicatie bepaald.

**Uitvoeren van de testen in Azure**

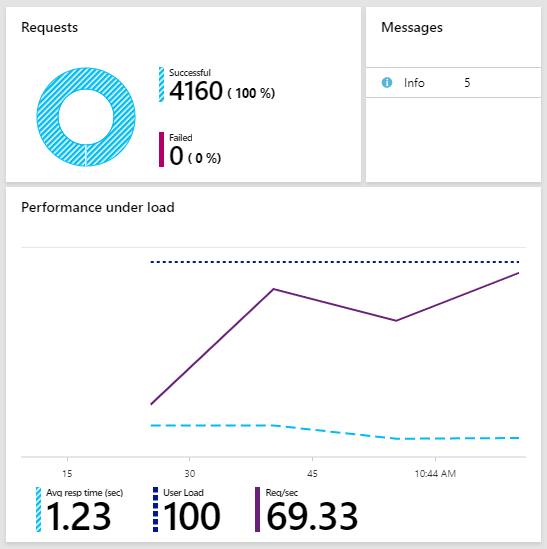
Na configureren van het testen via Azure is gekeken hoe dit proces geoptimaliseerd kon worden. Zo bleek dat bij het uitvoeren van meerdere testen deze zo snel mogelijk achter elkaar uitgevoerd moesten worden. Wanneer dit niet gedaan werd moest soms 10 tot 15 minuten gewacht worden door vertragingen aan de kant van Azure. Om dit te voorkomen werd wanneer de test begon gelijk de eerstvolgende test ook al aangezet. De volgende test kon dan alvast klaargezet worden door Azure en direct daarna uitgevoerd worden.

## Uitlezen testresultaten

In deze paragraaf wordt besproken hoe de testresultaten zijn verzameld en worden de testresultaten verwerkt en besproken.

**Uitlezen van de testresultaten**

Azure maakt het mogelijk om tijdens het uitvoeren van een test de testresultaten live mee te kijken. Wanneer een test volledig is afgerond zijn alle testresultaten inzichtelijk en worden deze weergegeven in een klein overzicht (Figuur 9.2). Het aangegeven req/sec (doorvoer) wordt meegenomen in de resultaten, maar zijn niet verwerkt in het advies.



Figuur 9.2: Testresultaten in Azure

Naast deze informatie uit Azure wordt ook de performancelog tabel uit de applicatie geraadpleegd. Het ophalen van deze resultaten gaat eenvoudig omdat de testresultaten zijn gekenmerkt met een unieke code per situatie. De gemiddelde reactietijd wordt met behulp van een query opgehaald uit de performancelogs tabel waar de unieke code van het testscenario overeenkomt. De testresultaten zijn opgenomen in de bijlage. (Bijlage 1: De testresultaten)

**Hoe generiek zijn de resultaten**

Het onderzoek is gericht op Divide. De beoogde resultaten zijn daarom alleen generiek binnen de parameters die in dit onderzoek zijn opgesteld. Toch kunnen de resultaten voor andere situaties worden ingezet, mits deze aan de volgende voorwaarden voldoen.

*Dezelfde parameters*

De resultaten zijn alleen generiek binnen de in dit onderzoek opgestelde testscenario’s. Toch kunnen deze resultaten gebruikt worden voor andere situaties mits deze aan vergelijkbare parameters voldoen als de opgestelde parameters uit dit onderzoek.

*De grootte van de opgeslagen data*

De grootte van één opgeslagen product in dit onderzoek is ongeveer 1.6 kB (Hoofdstuk 6.3). In een situatie waar ook data wordt opgeslagen van ongeveer 1.6 kB kunnen deze resultaten dus ook gebruikt worden. Echter moet wel weer gedacht worden aan de opgestelde parameters uit dit onderzoek voor verglijkbare resultaten.

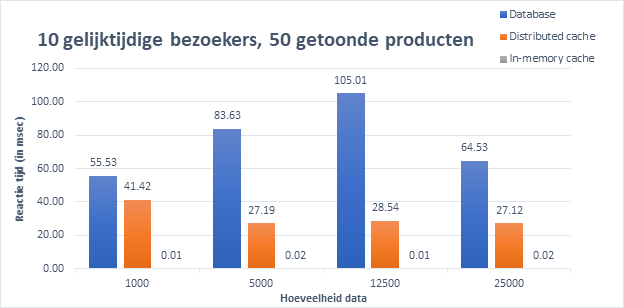
## Bespreking testresultaten

Nadat alle testen zijn uitgevoerd en de resultaten zijn verzameld, kunnen deze worden gepresenteerd in grafieken. In deze paragraaf worden de resultaten besproken.

**Leesbaar maken resultaten**

In bovenstaande paragrafen is beschreven hoe de testresultaten tot stand zijn gekomen. De testresultaten leverden een grote dataset op die niet eenvoudig in één keer te presenteren is. Om de resultaten te presenteren zijn grafieken opgesteld aan de hand van combinaties van parameters.

De eigenschap ‘hoeveelheid data’ en ‘reactietijd’ is voor iedere situatie aanwezig. Hierom is besloten om deze eigenschappen op de horizontale en verticale assen te zetten. De overige parameters: ‘gelijktijdige bezoekers’ en ‘getoonde producten’ verschillen per situatie en zijn daarom ieder in een individuele grafiek gepresenteerd. Een voorbeeld van een grafiek is Figuur 9.3. De overige grafieken zijn opgenomen in de bijlage: Bijlage 2: De testresultaten omgezet in grafieken.



Figuur 9.3: Alle testscenario’s met 10 gelijktijdige gebruikers en 50 getoonde producten

**Bespreking testresultaten**

Op basis van de testresultaten uit de bijlage (Bijlage 2: De testresultaten omgezet in grafieken) op basis waarvan conclusies getrokken kunnen worden.

*Inconsistentie testresultaten*

In een aantal situaties blijken de testresultaten niet consistent te zijn. Bijvoorbeeld in bovenstaande Figuur 9.3 is te zien dat de performance van de database bij een steeds grotere hoeveelheid producten afneemt en vervolgens bij het grootste hoeveelheid producten ineens weer omhooggaat. De oorzaak hiervan zou kunnen zijn dat de Azure niet altijd de volledige resources biedt waardoor de resultaten kunnen variëren per situatie. Het zou namelijk kunnen dat door middel van resource pooling een service van iemand anders op hetzelfde moment ook om resources vraagt en daarom minder resources over zijn. Herhaalde testen bleken dezelfde resultaten te geven waarbij uitgesloten wordt dat een eenmalige verstoring de resultaten beïnvloed heeft.

Daarnaast is het lastig om de juiste hoeveelheid hardware af te nemen, omdat geen precieze hardware specificaties zijn aangegeven. Dit komt omdat resources in Azure worden afgenomen in DTU’s en ACU’s. Zoals in hoofdstuk 6.3 is besproken is dit een verzameling van diverse gradaties in hardware. Om deze reden kunnen geen precieze hoeveelheden hardware worden afgenomen. Dit kan in het vervolg opgelost worden door de instanties allemaal op een eigen virtual machine te hosten, zodat gegarandeerd altijd de juiste resources aanwezig zijn.

*Beste type dataopslag*

Ondanks dat de resultaten niet altijd consistent zijn, geven deze wel een beeld over de performance van de diverse bronnen. In alle gevallen lijkt namelijk de in-memory dataopslag het beste te presteren, vervolgens de distributed cache en ten slotte de database.

Dit is in principe een logisch resultaat, aangezien bij de in-memory cache bijna geen latency bestaat bij het ophalen van data. De applicatie haalt de data namelijk van het geheugen waar de applicatie zelf ook op staat. In het geval van een distributed cache of database moet de data vaak worden opgehaald van een andere server, waardoor netwerkvertragingen ontstaan. De grootte van deze netwerkvertraging bepaalt of Divide het systeem in wil gaan zetten of niet.

*Externe koppelingen acceptabele netwerkvertraging*

Divide heeft sinds eind 2018 besloten dat een externe koppeling een maximale netwerkvertraging mag hebben van 150 milliseconden met een marge van 10%. De aanleiding hiervan was een intern onderzoek waaruit bleek dat de vertragingen vanuit de standaard code in de toenmalige situaties 500 milliseconden was.

*Acceptabele testresultaten*

Aan de hand van de testresultaten is een tabel (Tabel 9.2) opgesteld met de situaties die voor Divide acceptabel zijn. De tabel is verticaal in drie secties opgedeeld: database, in-memory en distributed. Ook op de verticale as staan het aantal gelijktijdige bezoekers tijdens de test. De horizontale as is opgedeeld in vier secties: 1.000, 5.000, 12.500 en 25.000 hoeveelheid data aanwezig. Ook op de horizontale as staan het aantal getoonde producten tijdens de test: 50, 100 en 200.

Alle acceptabele testresultaten met een maximale vertraging van 165 milliseconden zijn groen gemarkeerd. Alle onacceptabele testresultaten die boven de 165 milliseconden liggen zijn rood gemarkeerd.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1000 producten | | | 5000 producten | | | 12500 producten | | | 25000 producten | | |
| Database | 50 | 100 | 200 | 50 | 100 | 200 | 50 | 100 | 200 | 50 | 100 | 200 |
| 10 bezoekers | 56 | 307 | 742 | 84 | 115 | 270 | 105 | 109 | 474 | 65 | 210 | 412 |
| 50 bezoekers | 271 | 678 | 1297 | 101 | 227 | 852 | 141 | 212 | 762 | 168 | 315 | 703 |
| 100 bezoekers | 81 | 316 | 1159 | 201 | 225 | 1088 | 222 | 279 | 510 | 136 | 474 | 616 |
| 250 bezoekers | 255 | 493 | 2946 | 253 | 726 | 979 | 206 | 356 | 734 | 239 | 275 | 1739 |
| In-memory | 50 | 100 | 200 | 50 | 100 | 200 | 50 | 100 | 200 | 50 | 100 | 200 |
| 10 bezoekers | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 50 bezoekers | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 100 bezoekers | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 250 bezoekers | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Distributed | 50 | 100 | 200 | 50 | 100 | 200 | 50 | 100 | 200 | 50 | 100 | 200 |
| 10 bezoekers | 41 | 65 | 129 | 27 | 48 | 143 | 29 | 47 | 107 | 27 | 75 | 318 |
| 50 bezoekers | 63 | 210 | 348 | 68 | 139 | 376 | 126 | 131 | 401 | 80 | 154 | 372 |
| 100 bezoekers | 75 | 189 | 881 | 122 | 163 | 310 | 93 | 188 | 353 | 116 | 288 | 229 |
| 250 bezoekers | 163 | 285 | 463 | 163 | 298 | 794 | 135 | 349 | 739 | 103 | 98 | 465 |

Tabel 9.2: Alle testresultaten

# Conclusies en aanbevelingen

Om de resultaten en bevindingen van het afstuderen samen te vatten worden in dit hoofdstuk de conclusies besproken en aanbevelingen gedaan op basis van gekozen een situatie.

## Resultaten

In deze paragraaf worden de belangrijkste resultaten kort besproken zodat hier in de volgende paragraaf de conclusies op gebaseerd kunnen worden.

Uit het onderzoek in hoofdstuk 3 naar de werking van een distributed cache en de karakteristieken hiervan, bleken voor Divide 15 relevante karakteristieken. De belangrijkste karakteristieken zijn:

* Ondersteunt .NET Framework, zonder ondersteuning voor het .NET Framework kan het systeem niet ingezet worden;
* Ondersteuning cloud-oplossingen, voor het eenvoudig opzetten van een distributed cache systeem in de cloud, is het handig dat hier out of the box support voor aanwezig is;
* Licentie, open-source software heeft de voorkeur zodat het gratis uitgeprobeerd kan worden;
* Ingebruikname systeem, bij een grote ingebruikname is meer vertrouwen in het door ontwikkelen van het systeem;
* Grootte community, bij een grote community bestaat meer kans op verbeteringen voor het systeem.

Op basis van alle karakteristieken die in hoofdstuk 3.3 zijn benoemd, is in hoofdstuk 4 een marktonderzoek gedaan naar het voor Divide beste distributed cache systeem. Deze systemen zijn gekozen met behulp van een criteria in hoofdstuk 4.1. Het vijftal systemen is uitgezet tegen de karakteristieken in een matrix in hoofdstuk 4.2. Met behulp van deze matrix is een systeem gekozen, namelijk Redis.

Vervolgens is in hoofdstuk 5.1 onderzocht hoe de performance van een website gemeten kan worden. Het belangrijkste aspect is het meten van de vertraging tussen de aanvraag en de response. In ditzelfde onderzoek is in hoofdstuk 5.2 onderzocht hoe de performance van een E-commerce website beïnvloed kan worden blijkende uit de volgende parameters:

* Het aantal gelijktijdige bezoekers;
* De hoeveelheid data in de database;
* De manier waarop de data is opgeslagen;
* De hoeveelheid getoonde data.

Op basis van bovenstaande parameters zijn in hoofdstuk 9.1 testen opgesteld om realistische situaties na te bootsen. Het uitvoeren van deze testen is gedaan middels een testapplicatie. Het proces, ontwerp en de bouw van de testapplicatie worden beschreven in hoofdstuk 6, 7 en 8.

Voor de implementatie van distributed caching is onderzocht hoe de beste performance behaald kan worden. De belangrijkste punten hierbij zijn:

* Producten kunnen het beste per productoverzicht gegroepeerd opgeslagen worden om het aantal verzoeken te verminderen;
* Bij het opslaan en ophalen van data kan het beste van pipelining gebruik gemaakt worden.

Uit de testresultaten in hoofdstuk 9.4 is gebleken dat de performance van in-memory caching het beste presteert, vervolgens distributed caching en ten slotte de database. Vanwege de voordelen van distributed caching wordt dit geprefereerd boven in-memory caching, mits de duur van het ophalen van de data onder de 165 milliseconden is.

Op basis van het bovenstaande, kan het volgende gesteld worden:

* **In-memory caching**, in alle situaties is in-memory caching snel en blijft het onder de 1 milliseconde. Alle situaties zijn acceptabel.
* **Distributed caching**,distributed caching kan ingezet worden voor alle situaties waar 50 producten getoond worden. In veel situaties met 100 getoonde producten is distributed caching eveneens acceptabel. In bijna alle situaties met 200 getoonde producten is het toepassen onacceptabel.
* **Database**, de database is voor vrijwel geen van de situaties geschikt om in te zetten.

De testresultaten die zijn verzameld in dit onderzoek kunnen ook door andere bedrijven gebruikt worden, mits het bedrijf aan de volgende criteria voldoet:

* De E-commerce omgeving komt overeen met de waarden van de parameters die in dit onderzoek gebruikt zijn;
* De data die in de distributed cache wordt opgeslagen heeft een maximale grootte van 1.5 kB per object;
* De hardware van de E-commerce omgeving komt overeen of is beter is dan de hardware die tijdens dit onderzoek is gebruikt;
* Distributed caching is op dezelfde manier geïmplementeerd als in dit onderzoek is gedaan.

## Conclusies

In dit afstudeerdossier is een onderzoek verricht naar distributed caching en de performance hiervan. Dit onderzoek is uitgevoerd om te ontdekken hoe distributed caching ingezet kan worden om performanceproblemen in Divide’s E-commerce omgevingen tegen te kunnen gaan. De hoofdvraag luidt dan ook: ‘Op welke wijze kan Divide distributed caching inzetten om performance problemen binnen hun E-commerce omgevingen tegen te gaan?’

In deze paragraaf wordt de hoofdvraag beantwoord op basis van de resultaten die in hoofdstuk 10.1 zijn genoemd.

**Inzetten van distributed caching**

Distributed caching mag ingezet worden bij alle situaties waar een maximale netwerkvertraging van 165 milliseconden is gemeten. Alles hierboven is onacceptabel omdat de webshop dan te traag wordt wat voor de gebruiker onwenselijk is. Van de 48 verschillende situaties zijn er 28 acceptabel bij gebruik van het distributed caching systeem Redis.

**Minder time-outs**

Wanneer een webshop op meerdere servers gehost wordt, zal door het inzetten van distributed caching de webshop minder time-outs ervaren. Alle servers maken namelijk gebruik van dezelfde gedistribueerde cache en hoeven daarom niet telkens individueel een cache op te bouwen. Dit haalt veel druk weg van de database en de server zelf, waardoor minder time-outs plaatsvinden.

**Data altijd up-to-date**

Bij het inzetten van distributed caching hebben de webshop en het CMS geen gescheiden caches meer. Dit zorgt ervoor dat alle wijzigingen die in het CMS worden doorgevoerd, direct zichtbaar zijn op de webshop. Het voordeel hiervan is dat wanneer een klant een wijziging doorvoert die ervoor zorgt dat de webshop crasht, de klant dit direct kan zien en terugdraaien. De klant heeft hierom meer controle over zijn/haar acties.

**Opschalen eenvoudiger**

Opschalen zal in de toekomst met het gebruik van distributed caching eenvoudiger zijn. Als momenteel de te cachen data toeneemt, moet op alle servers waar de webshop op draait meer geheugen worden toegevoegd. Met distributed caching wordt alles nog maar één keer gecached en is in het algemeen minder geheugen nodig. Met distributed caching kan tevens gebruik gemaakt worden van partitioneren waarbij alle data verdeeld wordt over meerdere servers. Dit maakt veel grotere databases mogelijk, omdat data kan worden verspreid over meerdere servers.

**Back-ups**

Wanneer op dit moment de webshop crasht, moet de cache opnieuw gevuld worden. Het vullen van een cache is vaak een intensief proces voor de processor en harde schijf. Dit zorgt voor downtime van het systeem en is zeer onwenselijk. Met behulp van data persistence kan een distributed cache back-ups maken van de data. Wanneer een distributed cache crasht hoeft niet alle data opnieuw berekend te worden en kan eenvoudig een back-up worden teruggezet.

## Advies en aanbevelingen

In deze paragraaf wordt het volgende besproken:

* Een aanbeveling over welke situaties zich het beste lenen voor distributed caching;
* Adviezen over het behalen van de beste performance met distributed caching.

**Welke situaties lenen zich het beste voor distributed caching**

Op basis van de uit dit onderzoek behaalde testresultaten kan de onderstaande tabel (Tabel 10.1) opgesteld worden. De testresultaten zijn alleen geldig in een situatie waar dezelfde of betere hardware wordt gebruikt en de parameters vergelijkbaar zijn als in dit onderzoek. Bij een situatie waar minder dan 165 milliseconden vertraging is gemeten, is het acceptabel om distributed caching in te zetten. Voor deze situaties (in het groen gemarkeerd) wordt aanbevolen om distributed caching in te zetten.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1000 producten | | | 5000 producten | | | 12500 producten | | | 25000 producten | | |
|  | 50 | 100 | 200 | 50 | 100 | 200 | 50 | 100 | 200 | 50 | 100 | 200 |
| 10 bezoekers | 41 | 65 | 129 | 27 | 48 | 143 | 29 | 47 | 107 | 27 | 75 | 318 |
| 50 bezoekers | 63 | 210 | 348 | 68 | 139 | 376 | 126 | 131 | 401 | 80 | 154 | 372 |
| 100 bezoekers | 75 | 189 | 881 | 122 | 163 | 310 | 93 | 188 | 353 | 116 | 288 | 229 |
| 250 bezoekers | 163 | 285 | 463 | 163 | 298 | 794 | 135 | 349 | 739 | 103 | 98 | 465 |

Tabel 10.1: Testresultaten van de distributed cache

**Behalen van de beste performance met distributed caching**

Voor het behalen van de beste performance bij het inzetten van distributed caching worden een aantal adviezen gegeven over de implementatie van distributed caching bij Divide.

*Grootte producten*

De grootte van de producten is bij het gebruik van een distributed cache erg belangrijk. Redis is over de jaren heen geoptimaliseerd om zo snel mogelijk data op te halen en op te slaan. Toch is het belangrijk en moet rekening worden gehouden met de grootte van het object wat wordt gecached. Hoe kleiner het object is, hoe meer verzoeken per seconden afgehandeld kunnen worden. Er wordt geadviseerd om de producten zo klein mogelijk te houden.

*Manier van dataopslag*

Uit dit onderzoek is gebleken dat het ophalen van een set van producten beter is voor de performance dan de producten individueel op te slaan. Het individueel ophalen van producten resulteerde namelijk in het ophopen van verzoeken, waardoor vertraging ontstond. Er wordt geadviseerd om producten in een unieke dataset per productoverzicht op te slaan.

*Snelheid van het netwerk*

Alle data uit die uit de distributed cache komt moet over het netwerk verstuurd worden. Dit houdt in dat bij een groot aantal bezoekers veel data over het netwerk wordt verstuurd. Een lage netwerkbandbreedte kan daarom voor veel vertraging zorgen. Volgens Redis wordt de cache eerder gelimiteerd door het netwerk dan door de CPU. Er wordt geadviseerd om gebruik te maken van een netwerk interface kaart van 10 Gbit/s of meerdere gecombineerde netwerk interfacekaarten van 1 Gbit/s.

*Ophalen van de producten*

Om het ophalen en opslaan van data in de distributed cache sneller te laten verlopen wordt geadviseerd om gebruik te maken van pipelining. Met gebruik van pipelines is het mogelijk om meerdere verzoeken tegelijkertijd te versturen zonder telkens te wachten op antwoord van het vorige verzoek. Dit verhoogt het aantal verzoeken per seconden, waardoor de performance beter wordt.

# Evaluatie

In dit hoofdstuk wordt geëvalueerd hoe het afstuderen is gegaan. Dit is opgedeeld in drie onderdelen: product, proces en beroepstaken.

## Product

Tijdens het afstuderen zijn een aantal tussenproducten geproduceerd, die samen het afstudeerdossier vormen. De tussenproducten in het afstudeerdossier worden individueel besproken.

**Onderzoek performance (**Performance aspecten bij E-commerce omgevingen**).** Het stuk over performance aspecten bij E-commerce omgevingen was lastig om te schrijven, omdat ik moeite had met het vinden van bronnen hiervan. Er waren veel oude, maar weinig nieuwe bronnen. Het bracht uiteindelijk toch het verwachte resultaat. Het stuk zou in eerste instantie ook een onderdeel over impact caching op hardware bevatten. Dit onderdeel is uiteindelijk geschrapt omdat het niet in het eindresultaat gebruikt is.

**Onderzoek distributed caching (**Onderzoek naar distributed caching**).** Het onderzoek van over distributed caching resulteerde in een stuk waarin distributed caching wordt uitgelegd en karakteristieken van distributed cache systemen worden toegelicht. Ik ben tevreden met het onderzoek wat het geworden is.

**Marktonderzoek (**Marktonderzoek distributed cache systemen**).** Methet marktonderzoek kon er een goede keuze gemaakt worden voor een distributed cache systeem. Er is een kleine selectie gemaakt van distributed cache systeem op basis van een aantal criteria. In het vervolg zou ik de selectie nog iets kwalitatiever willen uitvoeren door er meer tijd aan te besteden. Desalniettemin blijkt het gekozen systeem goed te werken waar ook Divide blij mee is.

**Praktijktesten (**Testen**).** Het opstellen van de praktijktesten verliep soepel. Op basis van de opgestelde parameters uit het onderzoek naar performance konden alle testen worden opgesteld. In het vervolg zou ik minder testen willen opstellen en meer focussen op alleen de écht relevante situaties, waarbij ik van tevoren uitdenk welke situaties distributed echt nodig zou zijn.

**Ontwerp testapplicatie (**Ontwerp**).** Ik benover het algemeen erg tevreden met het ontwerp van de testapplicatie. Het ontwerp heeft achteraf nog een aantal kleine wijzigingen gehad, maar dat had geen impact op de duur. De testapplicatie doet voor de rest wat het moet doen, wat voor dit onderzoek voldoende is.

**Testapplicatie (**Bouw**).** De bouw van de testapplicatie verliep goed. Toch bleek bij de implementatie van de distributed cache nog wat performance winst te behalen. Dit lag aan de manier waarop het geïmplementeerd was. In het vervolg moet ik nog beter onderzoeken hoe het beste een systeem te implementeren in een bepaalde situatie, dit ligt namelijk niet altijd voor de hand. De testresultaten konden verder eenvoudig worden uitgelezen uit de applicatie.

**Testresultaten (**Bespreking testresultaten**)**. Omdat er veel testscenario’s waren duurde het verzamelen van de testresultaten wat langer dan verwacht, dit had ik beter moeten afbakenen. De testresultaten bleken niet consistent te zijn, dit is besproken in het onderzoek. Gezien de tijd kon hier helaas niks meer mee gedaan worden. Ondanks de inconsistentie kon toch een conclusie uit de resultaten getrokken worden.

**Advies (**Advies en aanbevelingen**).** Het schrijven van het advies duurde iets langer dan verwacht. Ik wilde een goed doordacht advies neerzetten. Uiteindelijk is het toch een mooi stuk geworden. Wellicht in het vervolg iets beknopter.

**Afstudeerdossier**. Het schrijven van dit dossier verliep redelijk goed, alhoewel ik sommige dingen toch opnieuw heb moeten doen. Ik schreef namelijk soms dingen dubbel zonder dit door te hebben. Daarnaast kan de tekst vaak beknopter geschreven worden en mag over sommige algemene beslissingen meer tijd genomen worden, zodat ik hier achteraf niet meer tijd aan hoef te besteden.

## Proces

Ik ben over het algemeen tevreden met hoe het proces is gegaan. Aan het begin van het afstuderen ging het af en toe wat moeizamer omdat de planning nog niet goed genoeg bijgehouden werd. De aanpak is voor Divide relatief nieuw en kwam daarom niet altijd goed van de grond. Nadat de aanpak voor dit project is veranderd verliep het beter en kon de voortgang beter in de gaten gehouden worden.

Het uitvoeren van de onderzoeken verliep goed, maar niet altijd vlekkeloos. Ik moest namelijk weer even inkomen in het uitvoeren van onderzoeken. Ik prefereer praktische opdrachten boven theoretische, daarom had ik soms een beetje moeite met onderzoeken. Uiteindelijk is dit toch goed gekomen en zijn kwalitatieve resultaten geproduceerd.

Wat ik tijdens het uitvoeren van de onderzoeken en het bouwen van de applicatie heb gemerkt, is dat ik soms te snel vooruit wil gaan. Ik maak daarom soms foutieve beslissingen, waardoor ik bepaalde dingen opnieuw moet doen. Ik heb hiervan geleerd dat ik af en toe even een stapje terug moet doen, om het vervolgens wel goed uit te kunnen voeren.

Daarnaast ben ik tevreden met de overige zaken en heb ik veel van het afstuderen kunnen leren. Dit zijn onder anderen het leren over nieuwe systemen, het inrichten van Azure, goed kunnen communiceren over bepaalde zaken met collega’s en ten slotte heb ik nu een beter beeld hoe caching in elkaar zit.

## Beroepstaken

Voorafgaand aan het afstuderen zijn een aantal beroepstaken gekozen. In het volgende stuk worden deze individueel beknopt toegelicht en de mate waarin het tijdens het afstuderen is geraakt.

**A1 – Analyseren probleemdomein & opstellen probleemstelling.** In hoofdstuk 2 wordt de huidige situatie van Divide besproken en de problemen die zich momenteel voordoen.

**Gc – Kritisch, onderzoekend en methodisch werken.** Tijdens het afstuderen is periodiek gevraagd om feedback van de inhoudelijk begeleider. Daarnaast is tijdens het schrijven van het afstudeerdossier frequent op een kritische manier gekeken naar de geschreven teksten.

**Gf – Leren leren: voorbereiden op volgende studiefase en beroep.** Tijdens het afstuderen heb ik veel kunnen leren. Een van de belangrijkste punten is het zelfstandig uitvoeren van onderzoeken. Ik werk in een omgeving met veel collega’s aan wie ik snel dingen vraag als ik iets niet begrijp. Met het afstuderen kon dit niet en heb ik veel dingen zelf uit moeten zoeken.

Dit gaat mij in het vervolg tijdens het uitvoeren van mijn beroep zeker helpen, aangezien niet altijd hulp in mijn directe omgeving aanwezig zal zijn. Dit zijn activiteiten als het zelfstandig opzetten van software, het bepalen van welke design patterns het beste werken op een bepaalde situatie en ten slotte op een juiste manier koppelen met API’s om de beste performance te behalen.

Tijdens het afstuderen werd gewerkt met software waar ik nog nooit eerder mee gewerkt had (Redis). Om te leren hoe met Redis om te gaan, moest ik dingen uitzoeken, uitproberen en bouwen. Dit leerproces gaat mij in de toekomst helpen wanneer ik met andere externe software moet koppelen. Ik weet dan sneller hoe ik bepaalde processen aan moet vliegen, waardoor ik dit in de toekomst makkelijker kan doen.

**B5 – Adviseren over inrichting van ICT gerelateerde oplossingen en processen.** In hoofdstuk 10.3 is een advies gevorm op basis van de informatie die tijdens het afstuderen is verzameld.

**C6 – Ontwerpen software.** Voorafgaand aan het realiseren van de software, is een ontwerp gemaakt. Het ontwerp is een analysis class diagram dat gebouwd is in Visual Paradigm, wat is te vinden in hoofdstuk 7.3.

**D14 – Realiseren van software.** Voor het afstuderen is een proof of concept ontwikkeld waarmee performance van diverse typen dataopslag gemeten kan worden. De realisatie hiervan wordt besproken in hoofdstuk 8.

**D15 - Testen: inrichten en uitvoeren van testen en testprocedures op basis van een praktijksituatie met omgevingsparameters.** In hoofdstuk 9 wordt alles omtrent de testen toegelicht. On anderen hoe de testresultaten verzameld zijn met behulp van de testapplicatie en welke parameters hiervoor zijn gebruikt met bijbehorende waarden.

# Literatuurlijst

Abraham, J. (2017). *The Netherlands B2C Ecommerce Country Report.* Ecommerce Wiki.

Aerospike. (sd). *GitHub*. Opgehaald van Aerospike Server: https://github.com/aerospike/aerospike-server

André. (2016, November 16). *Wat is een harde schijf (HDD)?* Opgehaald van Computerkiezen: https://computerkiezen.nl/computer-onderdelen/harde-schijf-hdd/

Apache Ignite. (sd). *Partitioning and Replication*. Opgehaald van Apachge Ignite: https://apacheignite.readme.io/docs/cache-modes

Betts, A. (2015, December 22). *Are RAM Drives Faster Than SSDs? 5 Things You Must Know*. Opgehaald van Makeuseofus: https://www.makeuseof.com/tag/ram-drives-faster-ssds-5-things-must-know/

*Cache (computing)*. (2018, Augustus 28). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Cache\_(computing)

DB-Engines. (sd). *DB-Engines*. Opgehaald van NCache System Properties: https://db-engines.com/en/system/NCache

*Distributed cache*. (2018, Juli 31). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Distributed\_cache

Docker. (sd). *Docker docs*. Opgehaald van Docker: https://docs.docker.com/samples/library/redis/#start-a-redis-instance

EhCache. (sd). *Cache Consistency Options*. Opgehaald van http://www.ehcache.org/documentation/2.8/get-started/consistency-options.html

G2Crowd. (sd). *Redis Reviews*. Opgehaald van G2Crowd: https://www.g2crowd.com/products/redis/reviews

Git. (2018). *Git*. Opgehaald van About - Git: https://git-scm.com/about

Han, X., Liu, K., Zhao, P., Lu, E., Yang, J., & Ke, T. (2013, December 31). *Distributed Cache System*. Opgehaald van Google Patents: https://patentimages.storage.googleapis.com/e0/9c/e6/f8be2306f0af5a/US20150189033A1.pdf

He, X., & Yang, Q. (sd). *Performance Evaluation of Distributed Web Server Architectures under*. Opgehaald van http://www.people.vcu.edu/~xhe2/publications/Reports/URI-TR-06252002.pdf

*Iaas, PaaS en Saas: wat zijn de verschillen?* (2018). Opgehaald van https://www.byte.nl/blog/iaas-paas-en-saas-wat-zijn-de-verschillen

*Intel 1103*. (2018, Februari 6). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Intel\_1103

Intogadgets. (2015, Oktober 10). *Advies: Wat is RAM?* Opgehaald van Intogadgets: https://www.intogadgets.nl/tips-en-adviezen/wat-is-ram-geheugen/

Intogadgets. (2016, Januari 26). *Wat is CPU?* Opgehaald van Intogadgets: https://www.intogadgets.nl/tips-en-adviezen/wat-is-cpu/

*Memcached*. (2018). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Memcached

Memcached. (sd). *Memcached*. Opgehaald van GitHub: https://github.com/memcached/memcached

Microsoft. (2017, Juli 27). *Azure Redis Cache FAQ*. Opgehaald van https://docs.microsoft.com/nl-nl/azure/redis-cache/cache-faq

Microsoft. (2018). Opgehaald van Wat is Azure?: https://azure.microsoft.com/nl-nl/overview/what-is-azure/

Microsoft. (2018, Oktober 22). *DTU-based service tiers*. Opgehaald van https://docs.microsoft.com/en-us/azure/sql-database/sql-database-service-tiers-dtu

*Microsoft SQL server*. (2018). Opgehaald van https://nl.wikipedia.org/wiki/Microsoft\_SQL\_Server

NCache. (sd). *NCache*. Opgehaald van GitHub: https://github.com/Alachisoft/NCache

Oracle. (sd). *Data Concurrency and Consistency*. Opgehaald van https://docs.oracle.com/cd/B19306\_01/server.102/b14220/consist.htm

Pfeil, M. (2010, Oktober 22). *What is persistence and why does it matter?* Opgehaald van Datastax: https://www.datastax.com/dev/blog/what-persistence-and-why-does-it-matter

*Random-access memory*. (2018, Augustus 2). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Random-access\_memory

Redis. (2018, December 23). *How fast is Redis?* Opgehaald van https://redis.io/topics/benchmarks

Redis. (2018, December 5). *Who's using Redis?* Opgehaald van https://redis.io/topics/whos-using-redis

Redis. (sd). *Redis*. Opgehaald van GitHub: https://github.com/antirez/redis

Redis. (sd). *Who's using Redis?* Opgehaald van Redis: https://redis.io/topics/whos-using-redis

*ReSharper*. (2018). Opgehaald van https://www.jetbrains.com/resharper/

Smith, S. (2018, Januari 8). *Overview of ASP.NET Core MVC*. Opgehaald van https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.1

solid IT. (sd). Opgehaald van https://db-engines.com/en/ranking/key-value+store

Stack Exchange. (sd). *GitHub*. Opgehaald van https://stackexchange.github.io/StackExchange.Redis/

StackExchange. (2018, December 13). *GitHub StackExchange.Redis*. Opgehaald van https://github.com/StackExchange/StackExchange.Redis

Techopedia. (sd). *Concurrency*. Opgehaald van Techopedia: https://www.techopedia.com/definition/27385/concurrency-databases

TrustRadius. (sd). *Redis Reviews*. Opgehaald van TrustRadius: https://www.trustradius.com/products/redis/reviews

*Visual Studio IDE*. (2018). Opgehaald van https://visualstudio.microsoft.com/vs/

Wang Sophia, X., Balasubramanian, A., Krishnamurthy, A., & Wetherall, D. (2013). Demystifying Page Load Performance with WProf.

*Wat is Scrum?* (2018). Opgehaald van Scrum: https://www.scrum.nl/wat-is-scrum-agile-scrum/

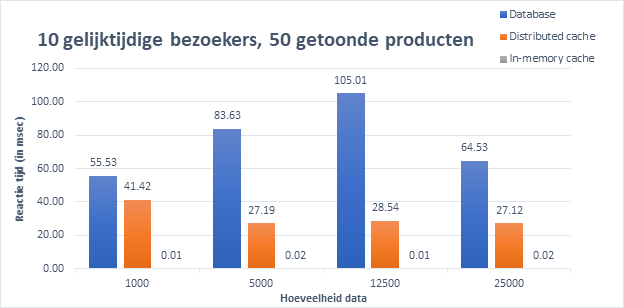
*What is Load Testing?* (2018). Opgehaald van https://smartbear.com/learn/performance-testing/what-is-load-testing/

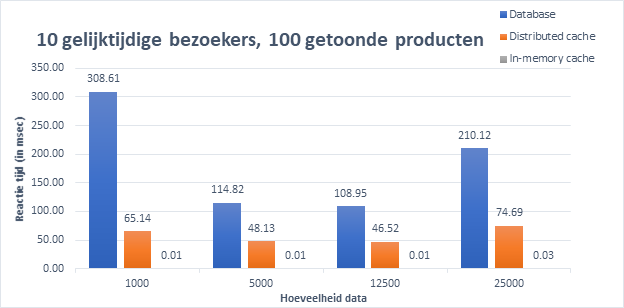
Xu, G., Mitchell, N., Arnold, M., Rountev, A., & Sevitsky, G. (2010). *Software Bloat Analysis*. Opgehaald van http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.639.9204&rep=rep1&type=pdf

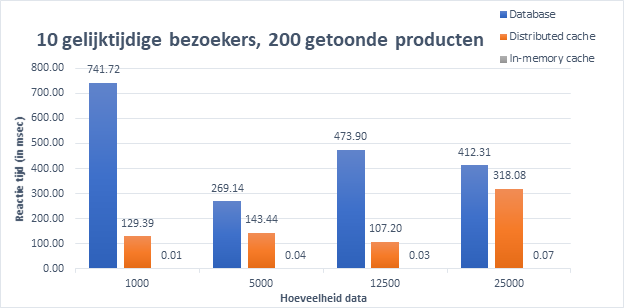
# Bijlage 1: De testresultaten

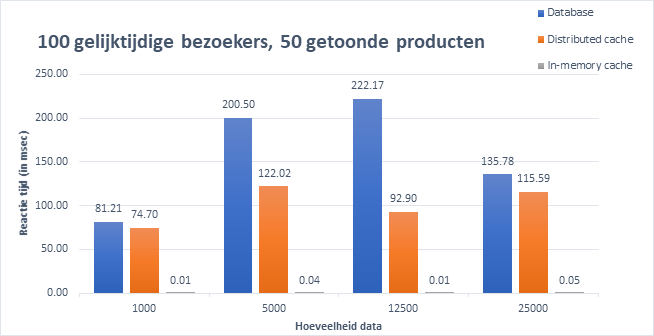
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Type dataopslag** | **Gelijktijdige bezoekers** | **Hoeveelheid data** | **Getoonde data** | **Laadtijd app** |
| 1 | Database | 10 | 1000 | 50 | 55.53 |
| 2 | Database | 10 | 1000 | 100 | 308.61 |
| 3 | Database | 10 | 1000 | 200 | 741.72 |
| 4 | Database | 10 | 5000 | 50 | 83.63 |
| 5 | Database | 10 | 5000 | 100 | 114.82 |
| 6 | Database | 10 | 5000 | 200 | 269.14 |
| 7 | Database | 10 | 12500 | 50 | 105.01 |
| 8 | Database | 10 | 12500 | 100 | 108.95 |
| 9 | Database | 10 | 12500 | 200 | 473.90 |
| 10 | Database | 10 | 25000 | 50 | 64.53 |
| 11 | Database | 10 | 25000 | 100 | 210.12 |
| 12 | Database | 10 | 25000 | 200 | 412.31 |
| 13 | Database | 50 | 1000 | 50 | 270.57 |
| 14 | Database | 50 | 1000 | 100 | 678.12 |
| 15 | Database | 50 | 1000 | 200 | 1297.13 |
| 16 | Database | 50 | 5000 | 50 | 100.53 |
| 17 | Database | 50 | 5000 | 100 | 227.08 |
| 18 | Database | 50 | 5000 | 200 | 851.87 |
| 19 | Database | 50 | 12500 | 50 | 141.14 |
| 20 | Database | 50 | 12500 | 100 | 211.69 |
| 21 | Database | 50 | 12500 | 200 | 762.21 |
| 22 | Database | 50 | 25000 | 50 | 168.10 |
| 23 | Database | 50 | 25000 | 100 | 314.55 |
| 24 | Database | 50 | 25000 | 200 | 703.41 |
| 25 | Database | 100 | 1000 | 50 | 81.21 |
| 26 | Database | 100 | 1000 | 100 | 315.85 |
| 27 | Database | 100 | 1000 | 200 | 1158.56 |
| 28 | Database | 100 | 5000 | 50 | 200.50 |
| 29 | Database | 100 | 5000 | 100 | 224.87 |
| 30 | Database | 100 | 5000 | 200 | 1087.82 |
| 31 | Database | 100 | 12500 | 50 | 222.17 |
| 32 | Database | 100 | 12500 | 100 | 278.68 |
| 33 | Database | 100 | 12500 | 200 | 509.71 |
| 34 | Database | 100 | 25000 | 50 | 135.78 |
| 35 | Database | 100 | 25000 | 100 | 473.77 |
| 36 | Database | 100 | 25000 | 200 | 616.38 |
| 37 | Database | 250 | 1000 | 50 | 254.70 |
| 38 | Database | 250 | 1000 | 100 | 492.84 |
| 39 | Database | 250 | 1000 | 200 | 2946.05 |
| 40 | Database | 250 | 5000 | 50 | 252.64 |
| 41 | Database | 250 | 5000 | 100 | 725.58 |
| 42 | Database | 250 | 5000 | 200 | 979.94 |
| 43 | Database | 250 | 12500 | 50 | 206.22 |
| 44 | Database | 250 | 12500 | 100 | 356.32 |
| 45 | Database | 250 | 12500 | 200 | 734.70 |
| 46 | Database | 250 | 25000 | 50 | 239.22 |
| 47 | Database | 250 | 25000 | 100 | 274.50 |
| 48 | Database | 250 | 25000 | 200 | 1739.94 |
| 49 | In-memory cache | 10 | 1000 | 50 | 0.01 |
| 50 | In-memory cache | 10 | 1000 | 100 | 0.01 |
| 51 | In-memory cache | 10 | 1000 | 200 | 0.01 |
| 52 | In-memory cache | 10 | 5000 | 50 | 0.02 |
| 53 | In-memory cache | 10 | 5000 | 100 | 0.01 |
| 54 | In-memory cache | 10 | 5000 | 200 | 0.04 |
| 55 | In-memory cache | 10 | 12500 | 50 | 0.01 |
| 56 | In-memory cache | 10 | 12500 | 100 | 0.01 |
| 57 | In-memory cache | 10 | 12500 | 200 | 0.03 |
| 58 | In-memory cache | 10 | 25000 | 50 | 0.02 |
| 59 | In-memory cache | 10 | 25000 | 100 | 0.03 |
| 60 | In-memory cache | 10 | 25000 | 200 | 0.07 |
| 61 | In-memory cache | 50 | 1000 | 50 | 0.01 |
| 62 | In-memory cache | 50 | 1000 | 100 | 0.02 |
| 63 | In-memory cache | 50 | 1000 | 200 | 0.02 |
| 64 | In-memory cache | 50 | 5000 | 50 | 0.06 |
| 65 | In-memory cache | 50 | 5000 | 100 | 0.02 |
| 66 | In-memory cache | 50 | 5000 | 200 | 0.02 |
| 67 | In-memory cache | 50 | 12500 | 50 | 0.02 |
| 68 | In-memory cache | 50 | 12500 | 100 | 0.03 |
| 69 | In-memory cache | 50 | 12500 | 200 | 0.03 |
| 70 | In-memory cache | 50 | 25000 | 50 | 0.03 |
| 71 | In-memory cache | 50 | 25000 | 100 | 0.06 |
| 72 | In-memory cache | 50 | 25000 | 200 | 0.02 |
| 73 | In-memory cache | 100 | 1000 | 50 | 0.01 |
| 74 | In-memory cache | 100 | 1000 | 100 | 0.01 |
| 75 | In-memory cache | 100 | 1000 | 200 | 0.02 |
| 76 | In-memory cache | 100 | 5000 | 50 | 0.04 |
| 77 | In-memory cache | 100 | 5000 | 100 | 0.02 |
| 78 | In-memory cache | 100 | 5000 | 200 | 0.03 |
| 79 | In-memory cache | 100 | 12500 | 50 | 0.01 |
| 80 | In-memory cache | 100 | 12500 | 100 | 0.02 |
| 81 | In-memory cache | 100 | 12500 | 200 | 0.02 |
| 82 | In-memory cache | 100 | 25000 | 50 | 0.05 |
| 83 | In-memory cache | 100 | 25000 | 100 | 0.01 |
| 84 | In-memory cache | 100 | 25000 | 200 | 0.04 |
| 85 | In-memory cache | 250 | 1000 | 50 | 0.01 |
| 86 | In-memory cache | 250 | 1000 | 100 | 0.01 |
| 87 | In-memory cache | 250 | 1000 | 200 | 0.04 |
| 88 | In-memory cache | 250 | 5000 | 50 | 0.02 |
| 89 | In-memory cache | 250 | 5000 | 100 | 0.02 |
| 90 | In-memory cache | 250 | 5000 | 200 | 0.07 |
| 91 | In-memory cache | 250 | 12500 | 50 | 0.05 |
| 92 | In-memory cache | 250 | 12500 | 100 | 0.02 |
| 93 | In-memory cache | 250 | 12500 | 200 | 0.07 |
| 94 | In-memory cache | 250 | 25000 | 50 | 0.01 |
| 95 | In-memory cache | 250 | 25000 | 100 | 0.03 |
| 96 | In-memory cache | 250 | 25000 | 200 | 0.10 |
| 97 | Distributed cache | 10 | 1000 | 50 | 41.42 |
| 98 | Distributed cache | 10 | 1000 | 100 | 65.14 |
| 99 | Distributed cache | 10 | 1000 | 200 | 129.39 |
| 100 | Distributed cache | 10 | 5000 | 50 | 27.19 |
| 101 | Distributed cache | 10 | 5000 | 100 | 48.13 |
| 102 | Distributed cache | 10 | 5000 | 200 | 143.44 |
| 103 | Distributed cache | 10 | 12500 | 50 | 28.54 |
| 104 | Distributed cache | 10 | 12500 | 100 | 46.52 |
| 105 | Distributed cache | 10 | 12500 | 200 | 107.20 |
| 106 | Distributed cache | 10 | 25000 | 50 | 27.12 |
| 107 | Distributed cache | 10 | 25000 | 100 | 74.69 |
| 108 | Distributed cache | 10 | 25000 | 200 | 318.08 |
| 109 | Distributed cache | 50 | 1000 | 50 | 63.04 |
| 110 | Distributed cache | 50 | 1000 | 100 | 209.67 |
| 111 | Distributed cache | 50 | 1000 | 200 | 348.29 |
| 112 | Distributed cache | 50 | 5000 | 50 | 68.30 |
| 113 | Distributed cache | 50 | 5000 | 100 | 138.72 |
| 114 | Distributed cache | 50 | 5000 | 200 | 375.61 |
| 115 | Distributed cache | 50 | 12500 | 50 | 125.80 |
| 116 | Distributed cache | 50 | 12500 | 100 | 131.10 |
| 117 | Distributed cache | 50 | 12500 | 200 | 401.48 |
| 118 | Distributed cache | 50 | 25000 | 50 | 79.96 |
| 119 | Distributed cache | 50 | 25000 | 100 | 153.68 |
| 120 | Distributed cache | 50 | 25000 | 200 | 371.76 |
| 121 | Distributed cache | 100 | 1000 | 50 | 74.70 |
| 122 | Distributed cache | 100 | 1000 | 100 | 189.37 |
| 123 | Distributed cache | 100 | 1000 | 200 | 880.84 |
| 124 | Distributed cache | 100 | 5000 | 50 | 122.02 |
| 125 | Distributed cache | 100 | 5000 | 100 | 163.01 |
| 126 | Distributed cache | 100 | 5000 | 200 | 309.96 |
| 127 | Distributed cache | 100 | 12500 | 50 | 214.37 |
| 128 | Distributed cache | 100 | 12500 | 100 | 187.54 |
| 129 | Distributed cache | 100 | 12500 | 200 | 352.65 |
| 130 | Distributed cache | 100 | 25000 | 50 | 115.59 |
| 131 | Distributed cache | 100 | 25000 | 100 | 287.59 |
| 132 | Distributed cache | 100 | 25000 | 200 | 228.69 |
| 133 | Distributed cache | 250 | 1000 | 50 | 190.65 |
| 134 | Distributed cache | 250 | 1000 | 100 | 285.28 |
| 135 | Distributed cache | 250 | 1000 | 200 | 463.87 |
| 136 | Distributed cache | 250 | 5000 | 50 | 189.40 |
| 137 | Distributed cache | 250 | 5000 | 100 | 298.23 |
| 138 | Distributed cache | 250 | 5000 | 200 | 794.31 |
| 139 | Distributed cache | 250 | 12500 | 50 | 134.99 |
| 140 | Distributed cache | 250 | 12500 | 100 | 348.79 |
| 141 | Distributed cache | 250 | 12500 | 200 | 739.43 |
| 142 | Distributed cache | 250 | 25000 | 50 | 103.22 |
| 143 | Distributed cache | 250 | 25000 | 100 | 97.90 |
| 144 | Distributed cache | 250 | 25000 | 200 | 464.74 |

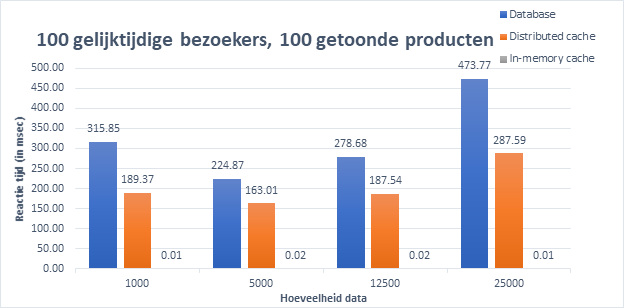
# Bijlage 2: De testresultaten omgezet in grafieken

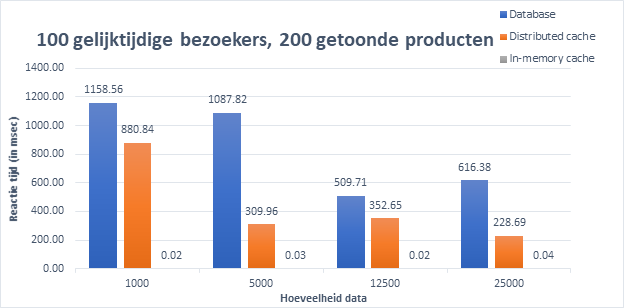


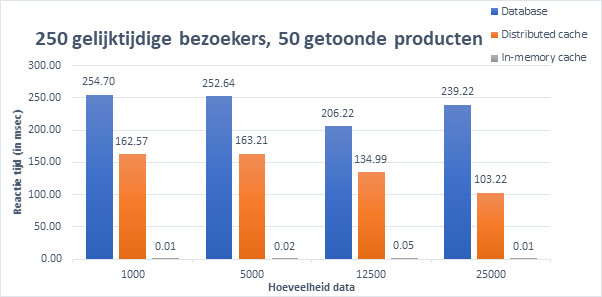


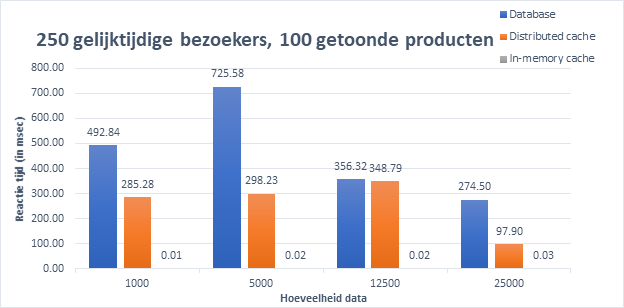


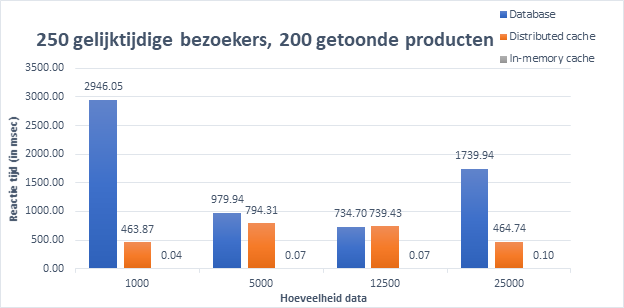












# Bijlage 3: Afstudeeropdracht

**Informatie afstudeerder en gastbedrijf (***structuur niet wijzigen***)**

**Afstudeerblok**: 2018-2.1 (start uiterlijk 27 augustus 2018)

**Startdatum uitvoering afstudeeropdracht**: 16 juli 2018

**Inleverdatum afstudeerdossier volgens jaarrooster**: 7 januari 2019

**Studentnummer**: 15088405

**Achternaam**: dhr Vieveen

**Voorletters**: A.M.

**Roepnaam**: Thomas

**Adres**: Molenweg 23

**Postcode**: 2631AA

**Woonplaats**: Nootdorp

**Telefoonnummer**: 015 310 93 50

**Mobiel nummer**: 0636095996

**Privé emailadres**: thomascv321@gmail.com

**Opleiding**: SE HBO ICT

**Locatie**: Den Haag

**Variant**: deeltijd

**Naam studieloopbaanbegeleider**: Rianne Bechet - Tjoonk

**Naam begeleidend examinator**: Lousberg – Orbons, A.M.J.J.

**Naam expert examinator**: Toet, A.J.

**Naam bedrijf**: Divide

**Afdeling bedrijf**: Back-end

**Bezoekadres bedrijf**: Koraalrood 33

**Postcode bezoekadres**: 2718SB

**Postbusnummer**:

**Postcode postbusnummer**:

**Plaats**: Zoetermeer

**Telefoon bedrijf**: 079 316 90 78

**Telefax bedrijf**:

**Internetsite bedrijf**: <https://www.divide.nl/>

**Achternaam opdrachtgever**: dhr Bliek

**Voorletters opdrachtgever**: S.C.

**Titulatuur opdrachtgever**:

**Functie opdrachtgever**: Directeur

**Doorkiesnummer opdrachtgever**:

**Email opdrachtgever**: bliek@divide.nl

**Achternaam bedrijfsmentor**: dhr Bina

**Voorletters bedrijfsmentor**: T.V.

**Titulatuur bedrijfsmentor**:

**Functie bedrijfsmentor**: Senior Developer

**Doorkiesnummer bedrijfsmentor**:

**Email bedrijfsmentor**: tom.bina@divide.nl

**Doorkiesnummer afstudeerder**:

**Functie afstudeerder (deeltijd/duaal)**: Backend Developer

**Titel afstudeeropdracht**

Adviseren en inzetten van distributed caching bij Divide.

**1. Bedrijf**

Divide bestaat sinds 2000 en is opgericht door drie jeugdvrienden. Anno 2018 is Divide de marktleider op het gebied van webshops in de kledingindustrie. Bekende ketens zoals Only for Men, Schuurman Schoenen en Christine le Duc zijn klanten van Divide.

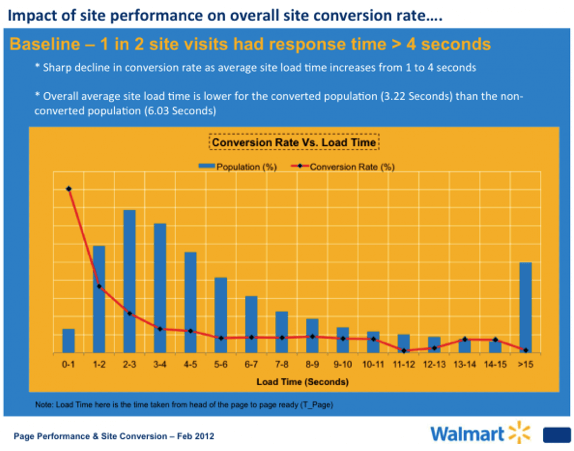
De oprichters van Divide hebben een aantal jaar geleden een centraal artikelensysteem ontwikkeld. Enerzijds is dit een database van producten met afbeeldingen voor diverse bekende merken (bijvoorbeeld Puma). Een tweede functie van dit systeem is de voorraad over verschillende locaties te bundelen tot één groot virtueel voorraadsysteem. Een E-commerce klant zal het niet erg vinden dat een paar zwarte Puma’s maat 43 geleverd wordt uit een winkel in Kerkrade, in plaats van de dichtstbijzijnde winkel in Zoetermeer. Dit systeem wordt als Stockbase in de markt gezet.

De afgelopen jaren heeft E-commerce een enorme vlucht genomen, met name voor kleding. Voor veel ketens is de webshop een belangrijk kanaal geworden. Daarnaast is het aantal producten enorm toegenomen en worden collecties in een steeds hoger tempo vernieuwd. Daarbij zijn de consumenten verwend geraakt: zij willen een snelle, interactieve website met mooie foto’s, veel informatie, adviezen, veel functies (wensenlijstjes, vergelijkingen, snel zoeken, adviezen voor alternatieven) en een snelle navigatie. Divide heeft dit traject de laatste jaren intensief gevolgd en geïnvesteerd in een eigen Content Management Systeem (CMS) met als doel de wensen uit de markt te volgen en voorop te lopen in alle ontwikkelingen. Daarnaast is een goede performance van de E-commerce omgevingen van groot belang.

**2. Probleemstelling**

Bij grote webshops met veel gelijktijdige bezoekers treedt vandaag de dag een merkbare vertraging op. Dit kan het winkelplezier nadelig beïnvloeden. Daarnaast neemt de vertraging bij een bepaald volume verkeer exponentieel toe voor alle gebruikers. Vergelijk dit met een file op de weg, waarbij deze vrijwel altijd optreedt boven een bepaald volume, licht afhankelijk van de lokale situatie.

Wallmart heeft hiernaar in 2012 onderzoek naar gedaan door de conversion rate (aantal verkopen) uit te zetten tegen de vertraging op de website (bron: https://www.pinterest.com/pin/256494141259075386/). Uit de onderstaande grafiek blijkt dat conversion rate sterk terugloopt bij een groeiende vertraging:



Hoewel Divide deze conversion rates niet rechtstreeks van klanten ontvangt is er voor grote websites een merkbare vertraging bij grotere aantallen bezoekers. De opdrachtgevers klagen hier periodiek over, zij willen de vertraging zo laag mogelijk zien.

De probleemstelling hierbij is: op welke wijze kan Divide distributed caching inzetten om performance problemen binnen hun E-commerce omgevingen tegen te gaan.

Distributed caching is een uitbreiding van originele caching concept. Een distributed cache kan meerdere servers omvatten, zodat deze eenvoudig kan groeien in omvang en capaciteit. Het wordt vooral gebruikt om data uit de database op te slaan, zodat deze efficiënter is te raadplegen.

**3. Doelstelling van de afstudeeropdracht**

Het doel van deze afstudeeropdracht is het opstellen van een adviesrapport een distributed cache ingericht kan worden, welke randvoorwaarden en kosten daarbij van toepassing zijn en welke reductie van vertragingen daarbij gerealiseerd kunnen worden. Daarbij wordt een prototype ontwikkeld om het advies te onderbouwen.

Het onderzoek moet Divide een onderbouwing geven in:

* Verschillende technische mogelijkheden van distributed cache systemen;
* Verschillen in inrichtingen van distributed cache systemen
* Beperkingen aan het centrale E-commerce platform bij het toepassen van een distributed cache systeem;
* Mogelijke parameters bij het inrichten van een distributed cache systeem;
* Praktische resultaten in reductie van vertragingen op basis van praktijktesten, op basis van de verschillende variaties in implementatie en randvoorwaarden;

Het advies is gebaseerd op literatuuronderzoek en praktijktesten op basis van een prototype. Met behulp van het prototype wordt geanalyseerd in welke mate de performance problemen gereduceerd worden in diverse relevante scenario’s.

Tenslotte wordt middels een business case bepaald in welke situaties een distributed cache een verantwoorde investering is voor Divide en de opdrachtgevers.

**4. Resultaat**

Na het afronden van de opdracht zal er een onderbouwd advies beschikbaar zijn. Het advies vertelt Divide hoe zij distributed caching kunnen toepassen ter voorkoming van vertragingen bij hun E-commerce omgevingen. Bij dit onderzoek wordt er getest met een prototype. De resultaten die uit dit prototype komen zullen gebruikt worden om het advies te onderbouwen. Door dit inzicht, aangevuld met een business case, kan Divide een gericht implementatietraject opzetten en de opdrachtgevers de beoogde versnelling van de E-commerce omgevingen realiseren tegen vooraf bekende kosten en randvoorwaarden. Hiermee wordt tevens voorkomen dat in situaties waar een distributed cache geen toegevoegde waarde biedt, wordt geïmplementeerd, waardoor onnodige kosten en werkzaamheden worden voorkomen of verbeteringen worden verwacht die niet realistisch zijn. Tenslotte is dit onderzoek een basis voor een doorontwikkeling van het toekomstige Divide E-commerce platform waarbij het verwerken van grotere volumes data bij een acceptabele (lage) vertraging een onderdeel van het basisontwerp is.

**5. Uit te voeren werkzaamheden, inclusief globale fasering, mijlpalen en bijbehorende activiteiten**

Om dit project in goede banen te leiden zal een globale projectplanning gemaakt worden. Deze planning moet bestaan uit het stuk onderzoek en het schrijven van het adviesraport. Voor het bouwen van het prototype zal conform de Scrum methodiek worden gewerkt. Scrum is een agile aanpak voor het incrementeel ontwikkelen van nieuwe producten. Het is een waarde gedreven aanpak die ervoor zorgt dat de opdrachtgever en eindgebruiker het meest waardevolle producten ontvangen binnen de beperkingen van tijd en budget.

Divide werkt momenteel al met Scrum. Aan het einde van de week worden er user stories van de backlog naar de sprint verplaatst. De user stories van het afstuderen kunnen hier eenvoudig tussen worden geplaatst.

Het ontwerp van het prototype zal in UML-diagrammen weergegeven worden. Bijvoorbeeld een analyse class en context diagram. Het prototype zelf zal worden gemaakt in de IDE Visual Studio 2017, met de taal C# in een console- of WPF-applicatie. Eventuele data zal worden opgeslagen in een MSSQL-server. Alle code zal opgeslagen worden in GIT. GIT is een versiebeheer tool waarbij je alle versies van je code kan bewaren.

De globale planning hierbij is:

|  |  |
| --- | --- |
| Plan van aanpak schrijven | 3 werkdagen |
| Literatuuronderzoek vertragingen E-commerce omgevingen | 4 werkdagen |
| Literatuuronderzoek distributed cache systemen en praktijkervaringen | 4 werkdagen |
| Vastleggen resultaten literatuuronderzoeken | 4 werkdagen |
| Vaststellen randvoorwaarden en kenmerken E-commerce omgevingen Divide | 3 werkdagen |
| Opstellen praktijktesten op basis van gevonden systemen, randvoorwaarden en kenmerken | 4 werkdagen |
| Opstellen onderzoekshypotheses praktijktesten | 4 werkdagen |
| Opbouwen E-commerce testomgevingen op basis van kenmerken | 4 werkdagen |
| Opbouwen distributed cache systemen op basis van opgestelde praktijktesten | 5 werkdagen |
| Ontwerpen van prototype | 5 werkdagen |
| Bouwen van het prototype aan de hand van opgestelde voorwaarden | 7 werkdagen |
| Uitvoeren testen door combinatie van E-commerce omgevingen en distributed cache systemen onder toepassing van de opgestelde praktijktesten | 5 werkdagen |
| Vastleggen testresultaten en onderbouwingen van resultaten | 4 werkdagen |
| Zo mogelijk vergelijkende test met 1 of 2 commercieel verkrijgbare systemen, toegepast op de opgestelde praktijktesten | 5 werkdagen |
| Vastleggen testresultaten en onderbouwingen van resultaten | 5 werkdagen |
| Schrijven onderbouwd implementatieadvies | 4 werkdagen |

De testomgeving zal een beperkte omvang hebben om praktisch haalbaar te zijn. Voor de inrichting wordt zoveel mogelijk de praktijk gevolgd. De testomgeving wordt zorgvuldig gedocumenteerd om reproduceerbare resultaten te verkrijgen. Daarbij is het van belang dat de verwachte resultaten in een operationele situatie realistisch voorspeld kunnen worden.

**6. Op te leveren (tussen)producten**

* Plan van aanpak
* Twee literatuuronderzoeken met verslaglegging
* Document randvoorwaarden en kenmerken E-commerce omgevingen Divide
* Document praktijktesten
* Document onderzoek/test hypotheses
* Document testresultaten en onderbouwing/verantwoording resultaten
* Document testresultaten commercieel verkrijgbare systemen en onderbouwing/verantwoording resultaten
* Onderbouwd adviesdocument op basis van testresultaten
* Testplan voor prototype
* Ontwerp prototype
* Applicatie prototype
* Resultaten van het testen met het prototype

**7. Te demonstreren competenties en wijze waarop**

|  |  |  |
| --- | --- | --- |
| **Nummer** | **Beschrijving** | **Niveau** |
| A1 | Analyseren probleemdomein & opstellen probleemstelling | 3 |
| Gc | Kritisch onderzoekend en methodisch werken | 3 |
| Gf | Leren leren: voorbereiden op volgende studiefase en beroep | 3 |
| B5 | Adviseren over inrichting van ICT gerelateerde oplossingen en processen | 3 |
| D15 | Testen: inrichten en uitvoeren van testen en testprocedures op basis van een praktijksituatie met omgevingsparameters | 3 |
| C6 | Ontwerpen software | 3 |
| D14 | Realiseren van software | 3 |

Competentie A1 zal aangetoond worden door het beschrijven van de huidige situatie en de daarbij behorende probleemstelling. Deze worden ondersteund door relevant literatuuronderzoek en bestaande analyses op dit vlak. Zoals ook in de planning is aangegeven.

Competentie Gc wordt aangetoond door het gestructureerd opzetten van het onderzoek en het uitvoeren van de beoogde projectfases.

Het aantonen van competentie Gf blijkt uit het zelfstandig opstellen van de literatuuronderzoeken, het rapporteren daarover, het inkaderen van de probleemstelling en testomgeving en het praktisch uitvoeren daarvan alsmede het onderbouwd rapporteren en adviseren hierover. Het bijblijven op het vakgebied, onderwerpen onderzoeken die op school niet aangeboden zijn.

Competentie B5 moet blijken uit het opleveren van een onderbouwd advies op basis van verschillende omgevingsparameters en testscenario’s blijkt hoe de gestelde probleemstelling in de praktijk gerealiseerd kan worden.

Het gehele project bevat diverse test-scenario’s en het uitvoeren van testen alsmede het verklaren van de resultaten hiervan. Dit resulteert in competentie D15.

Competentie C6 en D14 zullen aangetoond worden door het bouwen van een prototype. Met het prototype zullen de testresultaten verzameld worden.

# Bijlage 4: Plan van aanpak

|  |
| --- |
|  |
| Plan van aanpak |
| Versie 1.3 |

**Thomas Vieveen**15088405

**Afstudeerbedrijf**Divide B.V.  
Koraalrood 27  
2718 SB, Zoetermeer

**Haagse Hogeschool**HBO ICT - Software Engineering

**Afstudeerbegeleiders**Alwine Lousberg  
Arie Toet

**Afstudeerperiode**16-07-2018 – 07-01-2019   
(Juli t/m December)

Versie geschiedenis

|  |  |  |
| --- | --- | --- |
| Versie | Datum | Reden voor verandering |
| 1.0 | 01-07-2018 | Initiële document opgezet |
| 1.1 | 07-09-2018 | Aanpassen van diverse deelvragen |
| 1.2 | 01-10-2018 | Aanpassen deelvragen, bijwerken onderzoeksmethoden en planning |
| 1.3 | 26-11-2018 | Planning voorzien van meer tussenresultaten |
|  |  |  |
|  |  |  |
|  |  |  |

Inhoudsopgave

[1 Inleiding en achtergrond 1](#_Toc534571377)

[1.1 Inleiding 1](#_Toc534571378)

[1.2 De organisatie – Divide 1](#_Toc534571379)

[2 Projectdefinitie 4](#_Toc534571380)

[2.1 Probleemstelling 4](#_Toc534571381)

[2.2 Aanvulling op probleemstelling 4](#_Toc534571382)

[2.3 Doelstelling 6](#_Toc534571383)

[2.4 Onderzoeksvragen 6](#_Toc534571384)

[2.5 Onderzoeksmethoden 7](#_Toc534571385)

[2.6 Afbakening project 8](#_Toc534571386)

[2.7 Gewenste resultaat 8](#_Toc534571387)

[2.8 Planning 9](#_Toc534571388)

[2.9 Risico’s en maatregelen 10](#_Toc534571389)

[3 Onderzoek naar distributed caching 11](#_Toc534571390)

[3.1 Wat is caching? 11](#_Toc534571391)

[3.2 De werking van een distributed cache 11](#_Toc534571392)

[3.3 Karakterisering van een distributed cache systeem 12](#_Toc534571393)

[4 Marktonderzoek distributed cache systemen 17](#_Toc534571394)

[4.1 Selectie distributed cache systemen 17](#_Toc534571395)

[4.2 Vergelijking van de distributed cache systemen 19](#_Toc534571396)

[4.3 Resultaat 20](#_Toc534571397)

[5 Performance aspecten bij E-commerce omgevingen 21](#_Toc534571398)

[5.1 Performance en eventuele oorzaken van performanceproblemen 21](#_Toc534571399)

[5.2 Parameters om performance te beïnvloeden 23](#_Toc534571400)

[5.3 Resultaat 25](#_Toc534571401)

[6 Proces testapplicatie 26](#_Toc534571402)

[6.1 Softwareontwikkelmethode 26](#_Toc534571403)

[6.2 Gebruikte methodieken en tools 26](#_Toc534571404)

[6.3 Hosting en hardware 28](#_Toc534571405)

[7 Ontwerp 32](#_Toc534571406)

[7.1 Requirements 32](#_Toc534571407)

[7.2 Architectuur 33](#_Toc534571408)

[7.3 Ontwerp applicatie architectuur 33](#_Toc534571409)

[7.4 Applicatie in de praktijk 35](#_Toc534571410)

[8 Bouw 37](#_Toc534571411)

[8.1 De bouw van de applicatie 37](#_Toc534571412)

[8.2 Uitdagingen 40](#_Toc534571413)

[8.3 Testen van de applicatie 42](#_Toc534571414)

[9 Testen 44](#_Toc534571415)

[9.1 Opzetten van testscenario’s 44](#_Toc534571416)

[9.2 Testscenario’s uitvoeren 45](#_Toc534571417)

[9.3 Uitlezen testresultaten 47](#_Toc534571418)

[9.4 Bespreking testresultaten 48](#_Toc534571419)

[10 Conclusies en aanbevelingen 51](#_Toc534571420)

[10.1 Resultaten 51](#_Toc534571421)

[10.2 Conclusies 52](#_Toc534571422)

[10.3 Advies en aanbevelingen 53](#_Toc534571423)

[11 Evaluatie 55](#_Toc534571424)

[11.1 Product 55](#_Toc534571425)

[11.2 Proces 56](#_Toc534571426)

[11.3 Beroepstaken 56](#_Toc534571427)

[12 Literatuurlijst 58](#_Toc534571428)

[Bijlage 1: De testresultaten 61](#_Toc534571429)

[Bijlage 2: De testresultaten omgezet in grafieken 65](#_Toc534571430)

[Bijlage 3: Afstudeeropdracht 69](#_Toc534571431)

[Bijlage 4: Plan van aanpak 75](#_Toc534571432)

Introductie

**Algemene introductie**De afgelopen jaren heeft E-commerce een enorme vlucht genomen. Steeds meer consumenten verleggen hun bestedingspatroon naar online of oriënteren zich uitgebreid online. Er is al jaren een sterke groei in online bestedingen. Consumenten raken verwend en eisen steeds meer van de webshops die zij bezoeken. Denk bijvoorbeeld aan een aantrekkelijke presentatie, hoge snelheid van de website, een ruim en overzichtelijk aanbod, goede mogelijkheden om te vergelijken en een responsive website met de mogelijkheid om op alle apparaten te kunnen gebruiken. E-commerce op mobiele telefoons en tablets realiseert de sterkste groei in de laatste jaren.

E-commerce is vandaag de dag een supply chain. Dit betekent dat er een keten is vanaf de producent via eventuele tussenhandel naar de consument. Het optimaliseren van deze supply chain is steeds belangrijker: om kosten te reduceren moeten voorraden zo laag mogelijk gehouden worden terwijl de consument een snelle levering van alle producten wenst. Vandaag de dag is het voor een succesvolle E-commerce website absoluut noodzakelijk de supply chain te integreren. Grote bekende ketens zoals Amazon, Coolblue en BOL hebben hierin geïnvesteerd en een eigen Content Management Systeem (CMS) ontwikkeld. Coolblue heeft daarbij een eigen warehouse gerealiseerd en biedt bijvoorbeeld leveringen op zondag of dezelfde dag. Om te zorgen dat deze dienstverlening betaalbaar blijft is het beheer van de supply chain een voorwaarde. De klanten van Divide zijn niet groot genoeg om een eigen CMS te realiseren, maar willen wel dezelfde mogelijkheden als de grote partijen die dat wel kunnen. Divide heeft een generiek CMS ontwikkeld waarmee haar klanten in staat zijn te concurreren met de grotere concurrenten.

Zoals eerdergenoemd is de snelheid van een webshop van groot belang, onafhankelijk van de grootte van de webshop en het aantal gelijktijdige gebruikers. Dit onderzoek gaat in op distributed caching als instrument om de snelheid van de webshop te verbeteren. Distributed caching is een uitbreiding van het traditionele concept van caching. In tegenstelling tot de traditionele cache, kan een distributed cache meerdere servers omvatten, zodat deze eenvoudig kan groeien in omvang en in transactionele capaciteit. In dit onderzoek kijken we naar de verschillende technische achtergronden en onderzoeken we hoe de gewenste snelheid daarbij gerealiseerd kan worden bij webshops met verschillende karakteristieken (bijvoorbeeld omvang, aantal gelijktijdige gebruikers, grafische aspecten etcetera).

**Introductie organisatie**Divide bestaat sinds 2000 en is opgericht door drie jeugdvrienden. Anno 2018 is Divide de marktleider op het gebied van webshops in de kledingindustrie. Bekende ketens zoals Only for Men, The Stone en Christine le Duc zijn klanten van Divide.

E-commerce is sterk gegroeid in het afgelopen decennium. Voor de online verkoop van kleding is de groei nog veel groter dan andere sectoren. Daarnaast is het aantal producten enorm toegenomen en worden collecties in een steeds hoger tempo vernieuwd. Daarbij zijn de consumenten verwend geraakt: zij willen een snelle, interactieve website met mooie vormgeving, veel informatie, advies, veel functies (wensenlijstjes, vergelijkingen, snel zoeken, adviezen voor alternatieven) en een snelle navigatie. Divide heeft deze ontwikkelingen intensief gevolgd en heeft geïnvesteerd in een eigen Content Management Systeem (CMS) met als doel deze wensen uit de markt beschikbaar te maken voor haar opdrachtgevers en hiermee voorop te lopen.

Divide is sinds 2018 onderdeel van TSS. TSS biedt met haar contacten een opstap naar het buitenland. Zo worden de eerste webshops nog in 2018 in Duitsland gelanceerd worden. Dit is een opstap naar verdere groei voor de komende jaren.

Aanleiding van de opdracht

Divide maakt het voor mogelijk voor kleding- en schoenenwinkels hun producten online aan te bieden. Een aantal van Divide haar klanten bieden tientallen duizenden producten aan. Deze producten worden dagelijks bekeken door duizenden consumenten en moeten voor deze consumenten snel weergegeven worden. Om dit proces snel te laten verlopen worden deze producten momenteel allemaal opgeslagen in een traditioneel cache systeem. De cache zorgt ervoor dat het systeem niet telkens de database hoeft te raadplegen en dat de producten dus snel weergegeven kunnen worden.

Veel van Divide haar klanten kunnen met deze traditionele cache uit de voeten, maar voor een aantal grotere klanten begint het een bottleneck te worden. Het cache systeem is namelijk niet schaalbaar genoeg. Het systeem is afhankelijk van de CPU en memory van de server waar het op draait. Deze twee componenten kan je maar tot zekere hoogte verbeteren. Om dit traditionele caching systeem te vervangen, is Divide een onderzoek naar distributed caching gestart. Dit is namelijk een oplossing die wel schaalbaar is. Met dit onderzoek probeert Divide een distributed caching systeem te vinden wat voor haar klanten het meest geschikt is.

Dit onderzoek richt zich op de volgende aspecten:

* Op welke wijze kan distributed caching de performance verbeteren?
* Welke parameters en variabelen zijn relevant en welke effecten hebben zij?

Probleemstelling

“Op welke wijze kan Divide distributed caching inzetten om performance problemen binnen hun E-commerce omgevingen tegen te gaan?”

Deelvragen

* Wat wordt er verstaan onder performance op het gebied van E-commerce en welke parameters hebben invloed hierop?
* Wat wordt er verstaan onder een distributed cache systeem caching en hoe is dit te karakteriseren?
* Welke distributed cache systemen zijn er momenteel beschikbaar en welke karakteristieken hebben deze?
* Op welke wijze kan een prototype, gebaseerd op de performance parameters, inzicht geven op de effecten bij het toepassen van een distributed cache systeem?
* Welke adviezen kunnen gegeven worden voor praktische toepassing binnen Divide op basis van de resultaten van het prototype?

Korte omschrijving deelvragen

1. Wat wordt er verstaan onder performance op het gebied van E-commerce en welke parameters hebben invloed hierop?

*Een literatuuronderzoek naar de oorzaken van performanceproblemen bij grote (E-commerce) websites. Websites kunnen verschillen in karakteristiek, bijvoorbeeld omvang, aantal gebruikers, aantal grafische plaatjes, het gebruik van scripts, etc. Voor deze deelvraag wordt er een relevante set van karakteristieken bepaald. Met behulp van het prototype willen we bepalen wat het effect van de verschillende karakteristieken hebben op de werking van de cache.*

1. Hoe is een distributed cache systeem te karakteriseren en welk systeem is voor Divide het meest optimaal?

*Wat is een distributed cache volgens de literatuur. In deze deelvraag wordt er uitgelegd wat een distributed cache is en hoe het werkt. Bijv. wat voor soort algoritmes er zoal worden gebruikt. Er worden een aantal voor Divide relevante karakteristieken uitgekozen.*

1. Welke distributed cache systemen zijn er momenteel beschikbaar en welke karakteristieken hebben deze?

*Er wordt een keuze gemaakt uit een aantal distributed cache systemen op basis van een aantal punten. De geselecteerde systemen worden in een matrix uitgezet tegen de relevante karakteristieken. Met behulp van deze matrix worden de systemen met elkaar vergeleken. Uiteindelijk wordt er één systeem gekozen dat wordt gebruikt bij het bouwen van het prototype.*

1. Op welke wijze kan een prototype, gebaseerd op de performance parameters, inzicht geven op de effecten bij het toepassen van een distributed cache systeem?

*Er worden een aantal testen opgesteld aan de hand van vooraf bepaalde parameters. Deze parameters worden tijdens het uitvoeren van deze testen beïnvloed om te kijken wat deze voor invloed hebben op de performance. Denk aan parameters als: aantal gebruikers, hoeveelheid data, etc.*

1. Welke adviezen kunnen gegeven worden voor praktische toepassing binnen Divide op basis van de resultaten van het prototype?

*Op basis van het prototype en de verschillende resultaten op basis van variaties in karakteristieken en parameters kunnen we bepalen wat de beste strategie is om een distributed cache in te richten.*

Onderzoeksmethoden

|  |  |  |
| --- | --- | --- |
| Deelvraag | Onderzoeksmethode | Motivatie |
| 1: Wat wordt er verstaan onder een distributed cache systeem caching en hoe is dit te karakteriseren? | Beschrijvend onderzoek | Het in kaart brengen van de werking van distributed caching en hoe deze opgedeeld kan worden in karakteristieken. |
| 2: Welke distributed cache systemen zijn er momenteel beschikbaar en welke karakteristieken hebben deze? | Marktonderzoek | Een aantal relevante systemen vinden en deze vergelijken op de gevonden karakteristieken. |
| 3: Wat wordt er verstaan onder performance op het gebied van E-commerce en welke parameters hebben invloed hierop? | Deskresearch en exploratief onderzoek | Data verzameling uit reeds uitgevoerde onderzoeken. Wat verstaan anderen onder performance en met welke parameters kan deze performance beïnvloed worden? Parameters uit onderzoek en persoonlijke ervaringen. |
| 4: Op welke wijze kan een prototype, gebaseerd op de performance parameters, inzicht geven op de effecten bij het toepassen van een distributed cache systeem? | Beschrijvend onderzoek | Alle informatie omtrent de testapplicatie verzamelen. Hoe wordt de applicatie ontworpen, hoe is het gebouwd, welke testen worden er uitgevoerd, etc. |
| 5: Welke adviezen kunnen gegeven worden voor praktische toepassing binnen Divide op basis van de resultaten van het prototype? | Pilotstudie | Een onderzoek om voor Divide de meest relevante strategie op te stellen, gebaseerd op een prototype en de informatie uit voorgaande deelvragen. |

Doelgroep

Doelgroepen zijn Divide als leverancier van de software en klanten van Divide met groot aantal producten en bezoekers.

Randvoorwaarden

Periode: 16-07-2018 t/m 07-01-2019  
Inhoudelijke begeleiding:

* Tom Bina: senior software developer

Verwacht eindresultaat

Een onderbouwd advies met voor het toepassen van distributed caching voor E-commerce websites in verschillende omgevingen.

1. Een literatuuronderzoek naar performance en hoe deze is te beïnvloeden op basis van parameters;
2. Een literatuuronderzoek naar de werking van distributed caching hoe deze is te karakteriseren;
3. Een vergelijkend onderzoek naar diverse distributed caching systemen die momenteel op de markt zijn en de keuze uit een systeem;
4. Een praktijkonderzoek met behulp van een prototype naar de maten van snelheidsverbetering in E-commerce websites bij het toepassen van een geselecteerd distributed cache systeem bij E-commerce websites met verschillende karakteristieken.
5. Adviesrapport over het toepassen van distributed caching voor E-commerce websites op basis van verschillende parameters.

Planning

|  |  |  |
| --- | --- | --- |
| Resultaat / Actie | Aantal uren | Deadline |
| Voorbereidend onderzoek, afstudeeropdracht, plan van aanpak, oriëntatie op werk | 96 uur | 07-09-18 |
| Literatuuronderzoek naar de oorzaken van performanceproblemen bij E-commerce websites | 80 uur | 30-09-18 |
| Literatuuronderzoek naar de achtergronden en werking van distributed caching | 80 uur | 16-10-18 |
| Een marktonderzoek naar diverse distributed caching systemen | 64 uur | 30-10-18 |
| Testplan opzetten en uitwerken voor prototype | 24 uur | 04-11-18 |
| Onderzoeken waar de testapplicatie gehost moet worden en deze omgeving opzetten | 48 uur | 11-11-18 |
| Uitzoeken welke type ontwerpen voor het prototype te maken en deze maken | 48 uur | 18-11-18 |
| Met behulp van de gemaakte ontwerpen, het prototype bouwen | 48 uur | 25-11-18 |
| Met behulp van het prototype tests uitvoeren | 80 uur | 13-12-18 |
| Scriptie schrijven, alle details uitwerken en één geheel van maken | 112 uur | 07-01-18 |
|  |  |  |