Computest

Performance voorspelling

De Haagse Hogeschool

Eindverslag

Versie 1.2

29 mei 2020
Laura de Koning (15076806)

Bedrijfsbegeleider: Nico Welmer
Afstudeerbegeleider: Ruud Vermeij

Voorwoord

Voor u ligt het eindrapport “Performance voorspelling”, hierin wordt onderzocht op welke manier de performance van een applicatie voorspeld kan worden. Dit rapport is geschreven in het kader van mijn afstuderen aan de opleiding Toegepaste Wiskunde aan de Haagse Hogeschool te Delft en in opdracht van het stagebedrijf Computest. In de periode februari 2020 tot en met mei 2020 ben ik bezig geweest met het onderzoek. In februari ben ik enthousiast bij Computest op kantoor begonnen, maar door de coronacrisis heb ik de laatste elf van de zestien weken mijn onderzoek thuis mogen afronden.

Ik wil graag Computest bedanken dat zij mij, ondanks dat ik pas een jaar na de geplande datum kon beginnen met afstuderen, nog steeds wilden hebben als afstudeerstagiaire.

Ik wil graag mijn begeleider, Nico Welmer, heel erg bedanken voor alle hulp en positieve feedback. Daarnaast wil ik ook mijn begeleider vanuit de opleiding, Ruud Vermeij, bedanken voor zijn snelle en tevens uitgebreide feedback op de tussenrapporten.

Tot slot wil ik mijn ouders en zus bedanken voor alle steun en reviews.

Laura de Koning
Voorhout, 29 mei 2020

Summary

Computest is a company where performance, functional and security tests are run on software commissioned by other companies. Performance tests are used to determine the speed of an application with a certain user load. Companies want to know what the impact on the performance of their application would be with an extra server or processor. Computest cannot answer these kind of questions if the additional hardware is not available. That is why the following research question has been asked:

How can the performance of an application be predicted?

To be able to answer this question, a literature review has been conducted into the subject of application performance, especially into the factors that influence it. Research has shown that everything connected to an application has influence on its performance, hardware as well as software, as network connections. Performance indicators such as availability, response times, throughput and utilization can be used for measurement.

The methods which can be used to predict the performance of an application have also been researched. The literature review showed there are four different methods that can predict the performance. The first one is neural networks. It can predict the performance of an application by taking a lot of input, measured throughputs and response times, and using that to form a prediction model. However, it can only predict the performance with the setup of the given input. So, this method cannot be used to answer the questions asked by the other companies.
Using the same input, a model can be formed by using the relations between the user load, response time and throughput. This method is called the historical method. The relations used in the historical method can also be used in the hybrid method, this one uses layered queueing models to generate the input for the relations. The layered queueing method, however, can also be used on its own as a prediction method. It uses the setup of the application and some measured values as input.

A prediction model for a certain application has been made using the layered queueing network model. To accomplish this, a tool developed by the Carleton University was used, the LQNS-tool. It has been developed to solve layered queueing networks. The output of the model was compared with results from multiple load tests to determine the accuracy of the model. To measure the accuracy, an error percentage has been used. This percentage needs to be as close as possible to zero for the model to be accurate. The average error percentages of the model are: 84,3% for the response time and 82% for the throughput with the original configuration, and 94,4% for the response time and 88,8% for the throughput with a new configuration. The model made for the application is not accurate.

Samenvatting

Computest is een bedrijf dat performance, functionele en security testen uitvoert in opdracht van andere bedrijven. Dit wordt gedaan om de applicaties zo optimaal mogelijk werkend te krijgen. Met performance tests wordt onder andere gemeten hoe snel een applicatie is bij een bepaalde hoeveelheid gebruikers. Veel bedrijven willen echter ook weten wat de performance van de applicatie is als er een extra server of processor toegevoegd zou worden. Als deze hardware niet beschikbaar is, kan Computest daar niet zomaar antwoord op geven. Daarom is de volgende onderzoeksvraag opgesteld:

Op welke manier kan de performance van een applicatie voorspeld worden?

Om antwoord te kunnen geven op deze vraag is er literatuuronderzoek uitgevoerd naar de factoren die invloed hebben op de performance van een applicatie. Zowel onderdelen van de hardware, als de software en netwerkverbindingen hebben invloed op de performance van een applicatie. Performance kan worden omschreven met behulp van een aantal indicatoren: beschikbaarheid, reactietijd, *throughput* en *utilization*.

Er is ook literatuuronderzoek uitgevoerd om uit te zoeken welke methoden gebruikt kunnen worden om de performance te voorspellen. Performance kan op een aantal manieren voorspeld worden. Zo kan er met behulp van veel input in de vorm van reactietijden en *throughputs* met neurale netwerken een voorspellingsmodel worden opgezet. Dit model kan echter niet voor een andere configuratie voorspellen, enkel voor de bestaande opzet. Met dezelfde input kan ook een model worden opgezet door middel van de verbanden tussen het aantal gebruikers, de reactietijd en de *throughput*, ook wel het analysemodel genoemd. Dit model kan ook worden opgesteld met behulp van data gegenereerd door gelaagde wachtrijmodellen, deze combinatie wordt het hybride model genoemd. Als laatste kunnen gelaagde wachtrijmodellen op zichzelf ook gebruikt worden om een voorspellingsmodel op te stellen. Deze methode heeft als input de opstelling van de applicatie nodig, samen met bijbehorende waardes.

Met gelaagde wachtrijmodellen is er een voorspellingsmodel opgesteld voor een applicatie. Dit is gedaan met behulp van de LQNS-tool, ontwikkeld door een team van de Carleton University om gelaagde wachtrijmodellen op te kunnen lossen. De uitkomsten van dit model zijn met de resultaten van load tests vergeleken door middel van een error-percentage. Bij een goed model is dit percentage dichtbij de 0%. De gemiddelde error-percentages van het opgestelde model zijn: 84,3% voor de reactietijd en 82% voor de *throughput* bij de oorspronkelijke configuratie, en 94,4% voor de reactietijd en 88,8% voor de *throughput* bij een nieuwe configuratie. Het opgestelde model is dus niet accuraat.

Figuren- en tabellenlijst

Begrippenlijst

Begrip	Uitleg
Availability	De hoeveelheid tijd dat de applicatie beschikbaar is voor de gebruiker.
Cloud	Met Cloud worden servers bedoeld die via het internet toegankelijk zijn. Deze servers staan in datacenters over de hele wereld.
Hardware	Hardware is een verzamelnaam voor alle fysieke onderdelen in en rond de computer.
Reactietijd	Reactietijd is de hoeveelheid tijd die de applicatie nodig heeft om te reageren op de gebruiker.
Schaalbaarheid	Schaalbaarheid is de mogelijkheid om een applicatie eenvoudig groter te maken.
Software	Software is een verzamelnaam van alle code geïnstalleerd op de hardware.
Systeembronnen	Systeembronnen is een verzamelnaam voor alle soorten software en hardware die in de computer zitten, zowel de CPU, harddrive en geïnstalleerde programma's. Echter, meestal wordt er met systeembronnen de hoeveelheid geheugen die beschikbaar is bedoeld.
Throughput	Throughput is de hoeveelheid data die kan worden afgehandeld binnen een bepaalde tijd.
Utilization	Utilization is het percentage van de totale capaciteit van een bepaalde systeembron die in gebruik is.
Webapplicatie	Een webapplicatie is een applicatie die via een webbrowser te benaderen is.

Inhoudsopgave

1. Inleiding

1.1. Aanleiding

Computest test al ongeveer vijftien jaar lang applicaties voor andere bedrijven om ervoor te zorgen dat het allemaal optimaal werkt: zo snel, betrouwbaar en veilig als mogelijk is. Het testen is onderverdeeld in drie groepen: functioneel testen, performance testen en security testen (Ruijs, z.d.). Bij performance testen wordt onder andere getest hoeveel bezoekers een applicatie aankan, of het systeem de belasting aankan als er een overstap naar de Cloud gemaakt wordt en waar de problemen in een applicatie kunnen zitten in het geval dat er plotseling meer personen gebruik van gaan maken. Eén van de tests die uitgevoerd wordt binnen het performance testen is de load test. Bij een load test wordt gekeken wat de reactietijd van een applicatie is bij een bepaald aantal gebruikers (Computest, z.d.). Maar de consultants van Computest krijgen ook wel eens vragen die ze niet kunnen beantwoorden, zoals hoeveel extra geheugen er bij een applicatie geplaatst moet worden om een bepaalde groei aan te kunnen. Daarom wordt er in dit verslag onderzoek gedaan naar de mogelijkheid van een voorspellingsmodel van de performance die zulke vragen met een bepaalde nauwkeurigheid zou kunnen beantwoorden.

1.2. Praktische relevantie

Als kan worden aangetoond dat het model correcte voorspellingen kan maken, met een bepaalde nauwkeurigheid, dan kan het model gebruikt worden door Computest in opdracht van verscheidene andere bedrijven. Er bestaan al tools die de performance van een applicatie kunnen voorspellen, maar deze zijn erg prijzig en laten niet zien wat voor methode er gebruikt wordt.

1.3. Probleemstelling

Er worden door klanten performance-gerichte vragen gesteld die de consultants van Computest niet (direct) kunnen beantwoorden. Computest wil dat natuurlijk minimaliseren. Een aantal van deze vragen zouden opgelost kunnen worden met behulp van een voorspelling van de performance van de bijbehorende applicatie.

1.4. Doel

Het doel van dit onderzoek is om te achterhalen of het mogelijk is om een voorspellingsmodel te maken voor de performance van applicaties. Als er gevonden wordt dat het mogelijk is, wordt een dergelijk model opgesteld en getest.
Om dit doel te behalen wordt er antwoord gegeven op de volgende hoofdvraag:

Op welke manier kan de performance van een applicatie voorspeld worden?

Om deze hoofdvraag te kunnen beantwoorden zijn de volgende deelvragen opgesteld:

1. *Waar is de performance van een applicatie op gebaseerd?*
   Om antwoord te kunnen geven op deze deelvraag wordt literatuuronderzoek uitgevoerd naar onderdelen van applicaties en andere factoren die invloed kunnen hebben op de performance van een applicatie.

2. *Welke methode kan gebruikt worden voor het voorspellen van de performance van een applicatie?*

Voor deze deelvraag wordt literatuuronderzoek gedaan naar verschillende technieken die gebruikt kunnen worden om de performance van een applicatie te voorspellen.

3. *Welke data is nodig voor het opstellen van het voorspellingsmodel?*
   Bij deze deelvraag wordt onderzocht welke data beschikbaar is en hoe deze gebruikt kan worden voor het opstellen van een voorspellingsmodel.

4. *Hoe kan de betrouwbaarheid van het voorspellingsmodel vastgesteld worden?*
   Bij deze deelvraag wordt er ten eerste onderzocht hoe het voorspellingsmodel vergeleken kan worden met de werkelijkheid. Ten tweede gaat er bepaald worden wat de betrouwbaarheid van het model is.

1.5. Afbakening

Dit onderzoek wordt uitgevoerd in de periode van 10 februari 2020 tot en met 29 mei 2020.
Het onderzoek richt zich enkel op webapplicaties, omdat 95% van de applicaties die getest worden door Computest webapplicaties zijn. Daarnaast wordt ter illustratie voor het opstellen van het voorspellingsmodel enkel gebruik gemaakt van één applicatie van een bedrijf waar Computest al een contract mee heeft. Hiervan is informatie beschikbaar die aangeleverd wordt door de bedrijfsbegeleider. Daarnaast is er de mogelijkheid om zelf tests uit te voeren.

1.6. Leeswijzer

In hoofdstuk 2 wordt het theoretisch kader van dit onderzoek besproken. Vervolgens worden in hoofdstuk 3 verschillende gevonden methoden beschreven. De beschikbare data wordt onderzocht in hoofdstuk 4. De resultaten van de gekozen methoden zijn zichtbaar in hoofdstuk 5. In hoofdstuk 6 wordt de conclusie van dit onderzoek besproken en tot slot worden in hoofdstuk 7 en 8 enkele discussiepunten en de aanbevelingen beschreven.

2. Theoretisch kader

Performance is een breed concept en is over het algemeen dan ook afhankelijk van veel verschillende onderdelen. In dit hoofdstuk wordt ten eerste uitgewerkt wat er precies bedoeld wordt met de performance van een applicatie. De performance van een applicatie kan afhankelijk zijn van elementen binnen de hardware, software of communicatie binnen een computer of tussen computers onderling. Deze onderwerpen worden in de rest van dit hoofdstuk besproken.

"When trying to pin down the top factors impacting application performance, the right answer is that there is no right answer … the source of a performance problem could be almost anywhere! Almost anything that touches an application either improves or degrades performance; determining whether that is infrastructure, code, data, the network, the application architecture, the endpoint or another application is the name of the game – and the big reason why APM solutions are so valuable."

- Julie Craig, Research Director, Application Management, Enterprise Management Associates (EMA) (APMdigest, 2013)

2.1. Performance

De performance van een applicatie is goed als de gebruiker een taak kan uitvoeren zonder overdreven merkbare vertraging of irritatie (Molyneaux, 2009).

Performance is onder andere afhankelijk van deze factoren:

- Hoeveel systeembronnen zijn er beschikbaar?
- Hoeveel personen hebben de systeembronnen nodig?
- Hoe lang moeten deze personen wachten op de systeembronnen?
- Hoe lang maken ze gebruik van de systeembronnen (Oracle, z.d.)?

Er zijn enkele indicatoren die de performance kunnen omschrijven. Deze kunnen onderverdeeld worden tussen service-georiënteerde en efficiency-georiënteerde indicatoren. Service-georiënteerde indicatoren zijn reactietijd en beschikbaarheid; efficiency-georiënteerde indicatoren zijn *throughput* en *utilization*.

- Beschikbaarheid
  - De hoeveelheid tijd dat de applicatie beschikbaar is voor de gebruiker.
- Reactietijd
  - De hoeveelheid tijd die de applicatie nodig heeft om te reageren op een actie van de gebruiker. Meestal wordt de tijd tussen een actie van de gebruiker en het moment dat de reactie daarop helemaal geladen is gemeten.
- *Throughput*
  - Een maatstaaf voor de verwerkingssnelheid van een systeem; de hoeveelheid data die kan worden afgehandeld binnen een bepaalde tijd.
- *Utilization*
  - Percentage van de totale capaciteit van een systeembron die in gebruik is (Molyneaux, 2009).

De belangrijkste problemen die voorkomen bij de performance hebben grotendeels te maken met snelheid.

- *Lange laadtijd*

Laadtijd is over het algemeen de tijd die het kost om een applicatie op te starten. Het beste is als deze tijd maximaal een paar seconden is, maar het kan voorkomen dat dit niet mogelijk is.

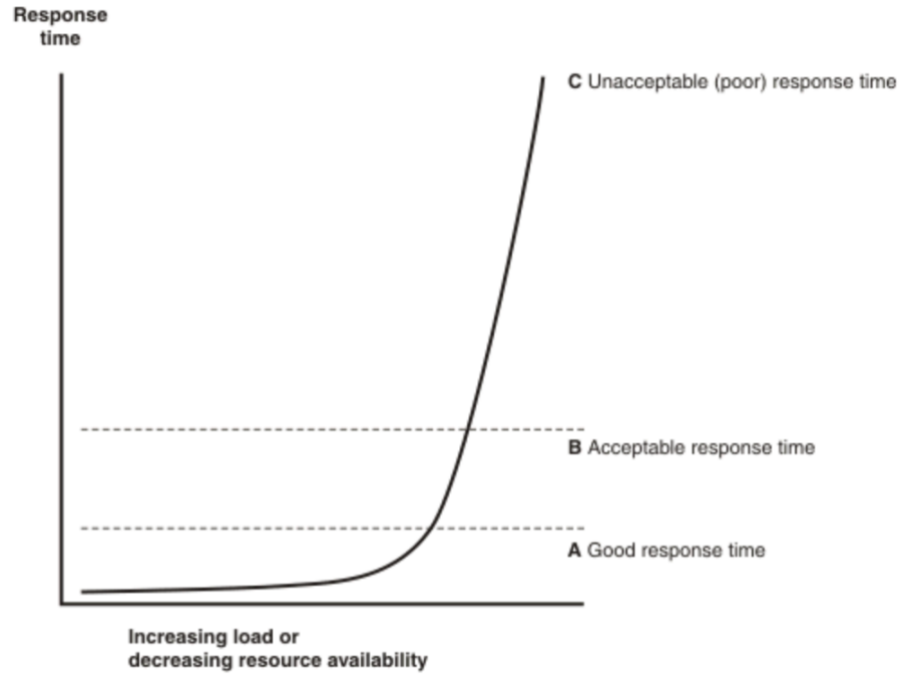
- *Slechte reactietijd*
  Bij de meeste applicaties wordt verwacht dat de reactietijd heel snel is. Als het te lang duurt, verliest de gebruiker zijn of haar interesse.
- *Slechte schaalbaarheid*
  Slechte schaalbaarheid komt voor als de applicatie niet het verwachte aantal gebruikers aankan, zelfs als er extra systeembronnen worden toegevoegd.
- *Bottlenecks*
  Bottlenecks zijn ophopingen in een systeem die voor een slechte performance van de gehele applicatie zorgen. Deze ophopingen worden veroorzaakt door slechte code of hardware problemen.
    - CPU gebruik
    - Geheugen gebruik
    - Netwerk gebruik
    - Beperkingen van het besturingssysteem
    - Disk gebruik

(Guru99, z.d.)

2.1.1 Reactietijd

Reactietijd op een computer is over het algemeen gedefinieerd als het aantal seconden tussen het moment dat een gebruiker een activiteit start en het moment dat de computer resultaat daarvan laat zien op het beeldscherm of iets naar de printer stuurt (Hoxmeier & DiCesare, 2000).
De reactietijd wordt beïnvloed door meerdere factoren: onder andere gebruikersvereisten, beschikbare capaciteit, de betrouwbaarheid van het systeem en het ontwerp van de applicatie (IBM, 2020).
Reactietijd is onder te verdelen in wachttijd en bedieningstijd. Wachttijd is hierbij de tijd tussen het moment dat de gebruiker een activiteit start en het moment dat de computer begint met het maken van het resultaat. Bedieningstijd staat voor de tijd tussen de start van het maken van het resultaat en het moment dat het resultaat hiervan zichtbaar is voor de gebruiker. Er zijn twee manieren om de reactietijd te verbeteren, namelijk door de wachttijd of de bedieningstijd te verkorten (Oracle, z.d.).

Wat de verwachte reactietijd bij een applicatie is, verschilt per applicatie. Bij een dataverwerkingsprogramma is een reactietijd van minder dan een duizendste seconde normaal. Bij productiesystemen wordt tussen de vijf tot tien milliseconden verwacht. Bij meer wetenschappelijke berekenprogramma's of printers kan een goede reactietijd ook één tot twee minuten bedragen (IBM, 2020).



Figuur 1: Grafiek die de reactietijd uitzet tegenover een steeds groter wordende lading (IBM, 2020)

2.1.2 Performance testen

Een performance test is een test die de *responsiveness*, *throughput*, betrouwbaarheid en de uitbreidbaarheid van een applicatie probeert vast te stellen met de verwachte belasting. Performance tests bestaan over het algemeen uit de volgende onderdelen:

1. Identificeren van de testomgeving
2. Identificeren van de acceptatiecriteria
3. Plannen en ontwerpen van de tests
4. Test omgeving in orde maken
5. Implementeren van het testontwerp
6. Test uitvoeren
7. Resultaten analyseren, rapporteren en hertesten (Meier, Farre, Bansode, Barber, & Rea, 2007).

Er bestaan meerdere tools die de performance van een applicatie kunnen testen. Hieronder zijn enkele tools kort beschreven (Guru99, z.d.).

- *Gatling*

Gatling is een tool speciaal voor webapplicaties. Het is gecodeerd in de codetaal Scala, wat naast normaal coderen ook functioneel coderen ondersteunt.

- *Neotys Neoload*

Neoload is een performance test platform dat veel gebruikt wordt voor DevOps. Hiermee kan snel een load test uitgevoerd worden op een hele applicatie of een component daarvan.

- *Apache JMeter*

JMeter is een van Java afkomstige tool voor het testen van web- en applicatieservers.

- *HP Loadrunner*

Loadrunner is een populaire performance test tool die honderdduizenden gebruikers kan simuleren.

2.2. Hardware

Hardware is een verzamelnaam voor alle fysieke onderdelen in en rond de computer (Computer Hope, 2019). Voorbeelden van hardware zijn: computermuis, toetsenbord, harddisk, geheugen en de CPU.

2.2.1 CPU

De CPU (Central Processing Unit) is het hoofdbestanddeel van de computer dat instructies verwerkt. Onder andere het besturingssysteem en enkele applicaties worden met behulp van de CPU gerund. Input van een gebruiker of actieve software wordt opgenomen en wordt verwerkt tot output. De CPU bevat minstens één processor (Christensson, 2014).

2.2.2 Werkgeheugen

De algemene regel van geheugen is: hoe sneller het geheugen, hoe minder het op kan slaan. Met geheugen wordt meestal RAM (*random access memory)* bedoeld, maar werkgeheugen bestaat uit meerdere soorten opslagplekken. Bij het opstarten van een applicatie wordt eerst de informatie van de applicatie van de harddrive naar het geheugen verplaatst, zodat de processor sneller bij informatie kan komen dan als het nog op de harddrive had gestaan (Dell, 2018).
Verschillende soorten opslagplekken zijn onder andere (met oplopende snelheid): secundaire opslag (magneetband, cd, dvd), RAM, cache en registergeheugen (ELPROCUS, z.d.).

2.2.3 Drives

Er bestaan meerdere soorten drives. Twee bekende zijn de harddrive en de solid-state drive (SSD). De harddrive was jarenlang de standaard bij computers, maar de SSD begint de markt over te nemen. Een drive is het onderdeel van de computer waar onder andere het besturingssysteem op staat. Als de drive erg langzaam is, zal de computer minder snel opstarten en zullen applicaties ook lang moeten laden (Raymond.cc, 2019).
De harddisk is een onderdeel van de harddrive. De harddrive verplaatst data naar de disk, of haalt het ervan af. Zo verplaatst de harddrive data van de CPU naar de harddisk of andersom. Een harddisk bestaat uit magnetische disks die informatie op kunnen slaan. In tegenstelling tot bij het RAM-geheugen blijft de opgeslagen data bestaan als de computer uitgezet wordt (Christensson, 2006).
De SSD maakt gebruik van een geheugenchip. Deze chip heeft geen bewegende onderdelen en is een stuk sneller dan de harddrive. Daarmee is het vergelijkbaar met het RAM-geheugen, al blijft bij de SSD, net zoals bij de harddrive, de data intact als de computer uitstaat (Villinger, 2020).

2.2.4 Servers

Een server is een computer of systeem die systeembronnen, data of programma's levert aan andere computers. Om als een server te kunnen functioneren moet een apparaat opdrachten van meerdere gebruikers of computers kunnen ontvangen via een netwerkconnectie (Paessler, z.d.). Er bestaan meerdere soorten servers, waarvan er drie bij dit onderzoek naar voren komen: de applicatieserver, de databaseserver en de webserver.

2.2.4.1 Applicatieserver

Applicatieservers worden vaak gebruikt voor applicaties die veel systeembronnen gebruiken en voor meerdere personen tegelijkertijd toegankelijk moeten zijn. Op deze manier hoeft een applicatie maar één keer geïnstalleerd en maar op één plek geüpdatet te worden (Paessler, z.d.).

2.2.4.2 Databaseserver

Veel data gebruikt door bedrijven staat in databases opgeslagen. Deze data neemt veel ruimte in beslag en moet toegankelijk zijn voor veel verschillende personen tegelijkertijd. Daarom is het handig als de databases op een server staan (Paessler, z.d.).

2.2.4.3 Webserver

Een webserver is een speciaal soort applicatieserver die data en programma's bevat die beschikbaar zijn voor mensen op het internet (Paessler, z.d.).

2.3. Software

Software is een verzameling van code geïnstalleerd op de hardware (Computer Hope, 2019).

2.3.1 Besturingssysteem

Een besturingssysteem (Engels: Operating System, ofwel OS) is software die de interface vormt tussen de hardware en de gebruiker. Als een proces toegang nodig heeft tot het geheugen, de CPU of een printer of iets dergelijks, dan zorgt het besturingssysteem ervoor dat het proces toegang krijgt. Daarnaast zorgt het ook voor de beveiliging. Zo kan het ervoor zorgen dat een gebruiker niet bij een programma of data kan komen waar diegene geen toegang toe heeft. Voorbeelden van bekende besturingssystemen zijn: Windows, Linux en Apple OS (Tutorialspoint, z.d.).

2.3.2 Programmeertaal

Welke programmeertaal gebruikt is voor het schrijven van bepaalde software heeft invloed op de performance van een applicatie. De performance van een programmeertaal hangt onder andere af van de compiler, de *garbage collector* en beschikbare *libraries*. Bij een onderzoek kwamen *C, C++* en *Rust* naar voren als de snelste programmeertalen en als het meest efficiënt met energieverbruik (Pereira et al., 2017).

2.3.3 Code optimalisatie

De efficiëntie waarmee code geschreven is heeft ook invloed op de performance van een applicatie. Indien de code inefficiënt is, kan deze geoptimaliseerd worden. Code optimalisatie is een verzamelnaam voor elke methode die de kwaliteit van code kan verbeteren. Zo kan code korter gemaakt worden, minder geheugen gebruiken en sneller runnen (PVS-Studio, z.d.).

2.4. Communicatie

Binnen een computer en tussen computers onderling vindt veel communicatie plaats. De snelheid van deze communicatie is dan ook een belangrijk onderdeel van de performance van een applicatie.

2.4.1 Bandbreedte

Bandbreedte is de hoeveelheid data die binnen een netwerk verplaatst kan worden binnen een bepaalde tijd. Over het algemeen wordt bandbreedte gemeten in bits per seconde (Paessler, z.d.).

Data moet meestal door meerdere netwerkconnecties, waardoor de connectie met de laagste bandbreedte de bottleneck is. Bij webbrowsers is de bottleneck gewoonlijk de verbinding met de internetprovider, aangezien de connecties tussen de servers en tussen verbonden computernetwerken een hoge bandbreedte hebben (Christensson, 2012).

2.4.2 Internetverbinding

Er zijn drie soorten internetverbindingen:

1. *Koper/kabel/DSL*

Een koperen netwerk wordt al sinds de ontwikkeling van de telefoon gebruikt om een verbinding aan te leggen. Het was oorspronkelijk dus enkel bedoeld voor een geluidsignaal. Een nadeel van koper is dat het een lage bandbreedte heeft.

2. *Glasvezel*
   Onder glasvezel wordt technologie verstaan die data overbrengt door een dunne doorzichtige kabel gemaakt van glas of plastic. De overdracht van data is sneller dan bij koper. Ook gaat er minder van het signaal verloren bij de overdracht.
3. *Draadloos*
   Draadloos internet heeft, zoals de naam al zegt, geen bedrading nodig. Daardoor is het ook een stuk goedkoper dan glasvezel of een koperen netwerk. Een nadeel van draadloos internet is dat het signaal verslechtert naarmate de gebruiker verder weg gaat van de bron, in tegenstelling tot glasvezel. Glasvezel en draadloos internet kunnen elkaar echter wel complementeren (Lieber, 2015).

3. Methoden

In dit hoofdstuk worden de methoden die naar voren zijn gekomen bij het literatuuronderzoek beschreven.

3.1. Voorspellingsmodel

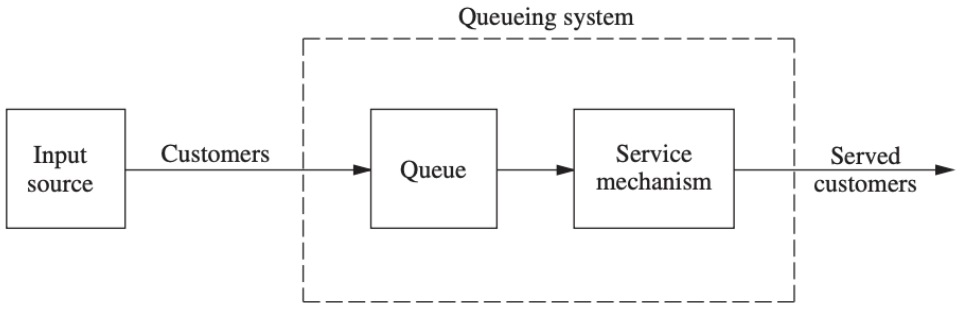
Voor het maken van een voorspellingsmodel van de performance van een computerapplicatie zijn in de literatuur vier soorten methoden gevonden, namelijk wachtrijmodellen, neurale netwerken en twee analysemodellen, waarvan één gebaseerd is op historische data en de ander op data gegenereerd door een wachtrijmodel (hybride methode) (Bacigalupo et al., 2005). In de rest van deze paragraaf worden deze vier methoden uitgewerkt.

De criteria die gebruikt gaan worden om de beste methode uit te kunnen kiezen zijn:

- Is de benodigde data beschikbaar?
- Is het model toe te passen op meerdere soorten applicaties?
- Kan het aantal CPU/servers aangepast worden en kunnen daar nieuwe voorspellingen op gebaseerd worden?

3.1.1 Wachtrijmodellen

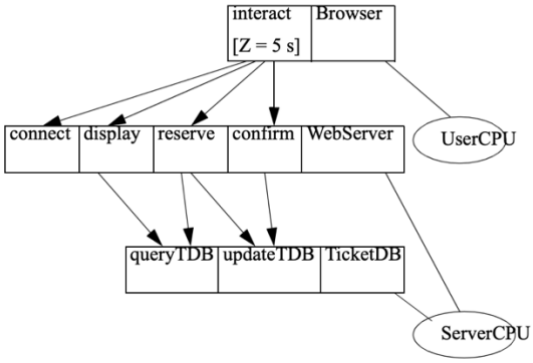
Wachtrijmodellen worden gebruikt voor het simuleren van systemen die wachtrijen bevatten. Met behulp van deze modellen kan onder andere berekend worden wat de gemiddelde wachttijd is, of hoe de wachttijd verandert als er een extra server/balie/machine wordt toegevoegd.
Wachtrijmodellen zijn dus handig voor het bepalen wat de meest optimale status van een systeem kan zijn (Hillier & Lieberman, 2015).
Een basis wachtrijmodel bestaat uit een bron waar de klanten vandaan komen, een rij waarin de klanten wachten om geholpen te worden, en een server waar de klanten om de beurt geholpen worden. De klanten komen binnen volgens een vastgestelde kansverdeling. De volgorde waarin klanten geholpen worden heeft ook meerdere opties. De snelheid van de server heeft ook een kansverdeling. Met behulp van deze kansverdelingen kunnen er meerdere berekeningen worden gemaakt, om bijvoorbeeld de gemiddelde lengte van de wachtrij te berekenen.



Figuur 2: Basis wachtrijmodel (Hillier & Lieberman, 2015)

Om een applicatie te kunnen modelleren met behulp van een wachtrijmodel zijn er meerdere lagen nodig. Daarom wordt hierna dieper ingegaan op gelaagde wachtrijmodellen.

Gelaagde wachtrijmodellen (Engels: Layered Queueing Networks Model (LQN)) zijn een uitbreiding van algemene wachtrijmodellen. De server is nu niet meer enkel de server, maar ook een klant voor een server die daarachter staat. Op die manier kunnen er meerdere lagen ontstaan in een wachtrijmodel. Een voorbeeld van een gelaagd wachtrijmodel is het repareren van een machine. De machine maakt normaal gesproken een product, op dat moment is de machine zelf een server. Maar als de machine kapot zou gaan, kan die in de wachtrij komen te staan voor een reparateur. Dan wordt de machine een klant en is de reparateur de server (Dorsman, 2015).



Figuur 3: Voorbeeld van een gelaagd wachtrijmodel (Woodside, 2002)

Een gelaagd wachtrijmodel bestaat uit verschillende onderdelen die software of hardware nabootsen. Onderdelen van een gelaagd wachtrijmodel zijn *processors, tasks* en *entries*. De *processors* zijn de onderdelen die enkel als server dienen; over het algemeen stelt een *processor* in het model een processor in het werkelijke systeem voor. *Tasks* kunnen onder andere bestaan uit taken, gebruikers en hardware-onderdelen. Een *task* kan *requests* ontvangen in een wachtrij. Verschillende soorten services die geleverd kunnen worden door een *task* zijn gespecificeerd met *entries*. Een *entry* kan nog worden gespecificeerd met *activities* of *phases*. Een voorbeeld van een gelaagd wachtrijmodel is zichtbaar in Figuur 3, hierbij zijn de *processors* zichtbaar als ellips, de *tasks* zijn de meest rechter rechthoeken van elke rij en de *entries* staan links naast de *tasks*.

Een gelaagd wachtrijmodel kan meerdere soorten applicaties nabootsen. Zo kunnen er meerdere kenmerken van applicatie gebruikt worden in het maken van model, zoals (Bacigalupo et al., 2005):

- Open, gesloten en gemengde wachtrijnetwerken
    - Open wachtrijnetwerk
      Bij een open netwerk komen de klanten van buiten het systeem, en verlaten het systeem ook na afloop. Het aantal klanten binnen het systeem verschilt dan ook per moment.
    - Gesloten wachtrijnetwerk
      Bij een gesloten netwerk blijven de klanten juist in het systeem, de output is verbonden met de input. Hierdoor blijft het aantal klanten in het systeem altijd gelijk.
    - Gemengde wachtrijnetwerk
      Bij een gemixt netwerk zijn er twee soorten klanten, klanten die van buitenaf komen en ook weer vertrekken, en klanten die constant in het systeem blijven.
- FIFO en prioriteit als wachtrij discipline
    - FIFO
      First In, First Out. Bij deze wachtrijdiscipline wordt de klant die als eerste in de wachtrij is gekomen als eerste geholpen.
    - Prioriteit
      Bij deze wachtrijdiscipline hebben de klanten verschillende prioriteiten. De klanten met de hoogste prioriteit worden het eerst geholpen. Als er meerdere klanten met dezelfde prioriteit zijn, worden die met behulp van de FIFO-discipline geholpen (Hillier & Lieberman, 2015).
- Synchronous, asynchronous en forwarding calls (Wainer, 2017)
    - *Synchronous call*
      Na een vraag om informatie blijft de server geblokkeerd tot er een antwoord is gegeven.
    - *Asynchronous call*

Een *asynchronous call* is een vraag om informatie waarvan het antwoord niet direct wordt terugverwacht, waardoor andere acties in de tussentijd door kunnen gaan.
    - *Forwarding call*
      Een *synchronous call* die door de server doorgestuurd wordt naar een andere server. De nieuwe server stuurt daarna het antwoord direct terug, zonder langs de eerste server te gaan.
- Een server die twee fases heeft binnen de bediening van klanten: eerst alle klanten als groep, daarna ook nog elke klant individueel.

Er zijn twee verschillende manieren om een gelaagd wachtrijmodel te gebruiken voor het voorspellen van de performance, namelijk door middel van simulatie van het model of met een analytische methode. De simulatiemethode is gedetailleerder en geeft de beste resultaten, maar het is erg complex en tijdsintensief. Bij de analytische methode zijn er goede resultaten, maar deze zijn wel een stuk sneller te behalen (Nikolov, 2014).

Mean Value Analysis (MVA) is een algoritme waarmee de gemiddelde waardes van de wachtrijlengte, de reactietijd en de *throughput* van een wachtrijmodel berekend kunnen worden. Voor een gelaagd wachtrijmodel is er de *method of layers* (MOL). Deze methode maakt gebruik van MVA.
Het gelaagde wachtrijmodel wordt verdeeld in normale wachtrijmodellen, waarna op elk apart wachtrijmodel een analyse (MVA) uitgevoerd wordt. Uitkomsten van een wachtrijmodel kunnen weer gebruikt worden als input bij een ander wachtrijmodel. Ten slotte worden alle modellen weer bij elkaar gevoegd tot een geheel, waarna er gezocht moet worden naar een punt waar de voorspelde reactietijden en *throughput* overeenkomen (Van der Mei, Harihan, & Reeser, 2001).

3.1.1.1 LQNS-tool

Een tool genaamd LQNS (layered queueing network solver) is ontwikkeld door een onderzoeksgroep binnen de Carleton Universiteit van Canada. Deze tool neemt als input een gelaagd wachtrijmodel en kan dan met behulp van een analytische methode of simulatie een benadering geven van performance-indicatoren.

Het algoritme dat gebruikt wordt voor de analytische methode staat hieronder in Tabel 1 beschreven.

LQNS algoritme
1: Lees de input, en creëer een LQNS model
2: Creëer S submodellen gebaseerd op de lagen van het model
Do
For s=1 **to** S
Los submodel s op met behulp van MVA
Stel de wachttijden in van submodel s
Stel de denktijden in van submodel s+1
Next
Until convergentie of iteratielimiet
Sla de resultaten op

Tabel 1: LQNS-algoritme (Franks & Li, 2012)

Het creëren van de submodellen kan gedaan worden met verschillende strategieën:

Standaard wordt *batched-back* gebruikt, dit is een uitbreiding van de strategie *batched*.
De *batched*-strategie gaat binnen het gelaagde wachtrijmodel van boven naar beneden en maakt submodellen met zoveel mogelijk servers erin. *Batched-back* begint op dezelfde manier als de

batched-strategie, maar gaat naderhand ook nog van beneden naar boven om de oplossingssnelheid te verbeteren. De andere opties zijn:

- *Hardware/software lagen*, hierbij wordt het model opgedeeld in twee submodellen, een hardware en een software submodel. Het software submodel bestaat enkel uit de *tasks,* hierbij worden alle non-reference *tasks* als server beschouwd. Het hardware submodel bestaat uit *tasks* die de *processors* aanroepen.
- Bij *Method of layers* (MOL) wordt het model ook in twee submodellen die uit hardware of uit software bestaan, hierbij zijn de twee submodellen complementair aan elkaar. Het submodel die de hardware voorstelt is hetzelfde als bij de hardware/software lagen strategie. Het submodel die de software voorstelt is wel anders vergeleken met de hardware/software lagen strategie. Bij MOL worden namelijk alle *tasks* beschouwd als server.
- *MOL-back*, de Method of Layers strategie die eerst van boven naar beneden in het model gaat, en daarna nog van beneden naar boven.
- *Squashed,* bij deze strategie worden alle *tasks* en *processors* in één submodel geplaatst. Bij deze strategie kan de oplossingssnelheid wel sterk toenemen.
- *SRVN*, hierbij worden submodellen gecreëerd met maar één server (Franks & Li, 2012).

Binnen de *for-loop* in Tabel 1 staat dat elk submodel opgelost gaat worden met behulp van *mean value analysis* (MVA). Hier zijn echter verschillende opties voor:

- *Exacte MVA*
  Bij exacte MVA wordt er recursief één-staps MVA uitgevoerd voor alle gebruikers, beginnend bij nul, waarbij gebruik gemaakt wordt van de laatste gevonden oplossing waarbij er één gebruiker minder was dan in de huidige iteratie.
- *Bard-Schweitzer*
  Bard-Schweitzer maakt niet gebruik van de recursie die bij exacte MVA gebruikt wordt. In plaats daarvan wordt het model opgelost met de alle gebruikers in één keer.
- *Linearizer*
  Het Linearizer algoritme is een verbetering op Bard-Schweitzer door een schaalfactor toe te voegen (Franks & Li, 2012).

Bij de Bard-Schweitzer en de Linearizer opties is er ook nog de mogelijkheid om maar één stap van het algoritme uit te voeren voor elke iteratie van het submodel. Van al deze algoritmes is het Linearizer algoritme als standaard ingesteld (Franks et al., 2013).

3.1.1.2 Formules

Om een gelaagd wachtrijmodel op te kunnen stellen die als input dient voor de LQNS-tool moet de service demand van de systeembronnen gevonden worden. Dit kan met behulp van belangrijke formules die bij een gelaagd wachtrijmodel bruikbaar zijn, namelijk de *Utilisation law*, de *Forced flow law* en de *Service demand law*.

$\boldsymbol{c}$	Request type
$\boldsymbol{i}$	Een systeembron
$\boldsymbol{V_{c,i}}$	Gemiddeld aantal bezoeken door request type c bij systeembron i
$\boldsymbol{S_{c,i}}$	Gemiddelde service tijd per gebruiker door request type c bij systeembron i
$\boldsymbol{U_{c,i}}$	Gebruik van systeembron i door request type c
$\boldsymbol{X_{c,i}}$	Throughput van systeembron i door request type c
$\boldsymbol{X_c}$	Throughput van het systeem door request type c
$\boldsymbol{D_{c,i}}$	Service demand van systeembron i door request type c

Tabel 2: Variabelen van de wachtrijmodel formules (Shoaib & Das, 2011)

Utilisation law $$U_i = X_{c,i} * S_{c,i} \quad (1)$$

Forced flow law $$X_{c,i} = V_{c,i} * X_c \quad (2)$$

Service demand law $$D_i = V_{c,i} * S_{c,i} = \frac{U_{c,i}}{X_c} \quad (3)$$

Om de service demand te kunnen bepalen van de systeembronnen moeten ten eerste de bestaande *requests* opgedeeld worden in *request* types op basis van welke systeembronnen aangeroepen worden en de hoeveelheid data die geassocieerd wordt met de *requests* (Bacigalupo, 2005). Dit wordt gedaan omdat het aantal soorten *requests* bij een systeem erg groot kan worden. In het geval van een kleiner systeem kan ook enkel gekeken worden naar de *requests* met een hoge reactietijd. Met deze *request* types wordt een korte loadtest gedraaid met één gebruiker zonder denktijd. Tijdens deze loadtest moeten de systeembronnen gemonitord worden. Op deze manier worden bepaalde waardes van de variabelen van de formules hierboven gevonden, waardoor de service demand bepaald kan worden (Shoaib & Das, 2011).

3.1.1.3 Replicatie en multiplicatie

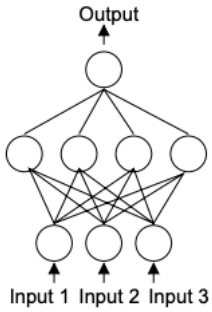
Eén techniek om een eenvoudig gelaagd wachtrijmodel of een applicatie uit te kunnen breiden is door kopieën van servers toe te voegen. Dit kan in de LQNS-tool op twee manieren: replicatie en multiplicatie. Multiplicatie is de simpelste van de twee, er is één wachtrij die door alle kopieën van servers wordt bediend. Bij replicatie zijn de wachtrijen ook gekopieerd. Multiplicatie bij een *processor* stelt meestal een multi-core processor of een symmetrische multiprocessor voor, de multipliciteit is dan het aantal cores of processors. Replicatie met een *processor* is een identieke maar andere *processor* die voor gerepliceerde *tasks* wordt ingezet. Multipliciteit bij *tasks* is over het algemeen de hoeveelheid threads van een software taak. Een replica van een *task* is een identieke maar aparte taak met dezelfde parameters, calls en processor (of een replica van dezelfde processor) (Franks et al., 2013).

3.1.2 Neurale netwerken

Neurale netwerken zijn gebaseerd op de werking van de hersenen. Het is een onderdeel van de *machine learning*. *Machine learning* is een overkoepelende term voor algoritmes die niet expliciet geprogrammeerd hoeven te worden, maar zelf van data kunnen leren. Binnen de *machine learning* zijn de belangrijkste drie categorieën: *supervised, unsupervised* en *semi-supervised machine learning*.

- *Supervised machine learning*
  Bij *supervised machine learning* bestaat de data uit input en output. Eerst wordt een model getraind met behulp van een gedeelte van de input en output, waarna het model getest kan worden op de rest van de input. De berekende output kan dan vergeleken worden met de achtergehouden output. Er zijn twee categorieën binnen *supervised machine learning*: regressie en classificatie modellen. Het verschil hiertussen is dat de output bij een regressiemodel een waarde is, en bij classificatie wordt de input over twee of meer groepen verdeeld.
- *Unsupervised machine learning*
  Bij *unsupervised machine learning* wordt er gebruik gemaakt van data zonder output. Er wordt enkel gezocht naar onderliggende verbanden in de data. De twee belangrijkste modellen bij *unsupervised machine learning* zijn clusteranalyse en principale componenten analyse. Bij clusteranalyse wordt de input in meerdere groepen geclusterd, bij principale componenten analyse worden de variabelen gegroepeerd.
- *Semi-supervised machine learning*
  Semi-supervised machine learning ligt tussen *supervised* en *unsupervised* in. Binnen de data is er zowel input met bijbehorende output, als input zonder output. Hiermee kan een model getraind worden die de output kan voorspellen. Door de extra data zonder output kan de *accuracy* van het model verbeterd worden, omdat de onderliggende verbanden duidelijker worden met een grotere hoeveelheid data (Expert System, 2017).

Neurale netwerken kunnen data clusteren, maar ook classificeren of een regressieanalyse uitvoeren, dus het kan zowel onder *supervised* als onder *unsupervised machine learning* vallen (Nicholson, z.d.).
Een neuraal netwerk bestaat uit meerdere lagen; een input laag, een output laag en één of meerdere verborgen lagen ertussenin. Binnen een laag zitten neuronen. De neuronen van verschillende lagen zijn met elkaar verbonden. Aan deze verbindingen zitten gewichten. Binnen een neuron vinden berekeningen plaats die ervoor zorgen dat de input uiteindelijk getransformeerd wordt naar de juiste output. Het leren van het model gebeurt door de gewichten op de verbindingen tussen de neuronen bij te stellen. Hiervoor worden de gemaakte output en de verwachte output met elkaar vergeleken; als de output hoger of lager is dan verwacht worden de gewichten aangepast (Wiering, z.d.).



Figuur 4: Standaard feedforward neuraal netwerk met één verborgen laag (Ipek, De Supinski, Schulz, & McKee, 2005)

Neurale netwerken zijn onder te verdelen in twee categorieën: *feedforward networks* en *recurrent/feedback networks*. Bij *feedforward* neurale netwerken gaat de input via de verborgen lagen naar de output laag. Bij de *feedback* neurale netwerken gaat de data er echter in een loop doorheen. Hierbij onthoudt het model ook de data die het al gezien heeft, terwijl de *feedforward* netwerken enkel het stukje data kent waar die op dat moment mee bezig is (Nicholson, z.d.).

In het artikel van Ipek et al. (2005) wordt een *feedforward* neuraal netwerk met zestien verborgen neuronen gebruikt voor het opstellen van een voorspellingsmodel van de performance van een grootschalige applicatie. Er zijn ongeveer dertienduizend datapunten gebruikt, tienduizend voor het opstellen van het model en de resterende drieduizend voor het maken van de voorspellingen.
Ook *feedback* neurale netwerken kunnen gebruikt worden voor het opstellen van een voorspellingsmodel, zo hebben Mendis, Renda, Amarasinghe, & Carbin (2019) een model gemaakt die de *throughput* van processoren kan voorspellen met behulp van een *feedback* netwerk.

Neurale netwerken kunnen dus voorspellingen maken van de performance van een applicatie. Echter, er kunnen enkel voorspellingen gemaakt worden voor de applicatie met de huidige configuratie. Er kan dus geen voorspelling gemaakt worden wat van de performance zal zijn met een extra server of processor. Dit is een groot nadeel voor dit onderzoek.

3.1.3 Analysemodel

Deze methode bestaat uit het verzamelen van waardes van performance indicatoren (reactietijd/*throughput*) en het analyseren hoe deze waardes in verband staan met variabelen, zoals de *workload* en de configuratie van de applicatie.
In het artikel van Bacigalupo et al. (2005) zijn drie verbanden gevonden die onder andere voor de servers van *business to business* applicaties gebruikt kunnen worden:

$\boldsymbol{mrt}$	Gemiddelde reactietijd
$\boldsymbol{no\_of\_clients}$	Aantal gebruikers
$\boldsymbol{c_L, c_U, \lambda_L}$ **en** $\boldsymbol{\lambda_U}$, $\Delta(\boldsymbol{c_L}), \boldsymbol{C}(\boldsymbol{c_L}), \Delta(\boldsymbol{\lambda_L})$ **en** $\boldsymbol{C}(\boldsymbol{\lambda_L})$	Parameters die gevonden moeten worden met behulp van historische data
$\boldsymbol{m}$	Hellingsgraad
$\boldsymbol{mx\_throughput}$	De maximale throughput
$\boldsymbol{N}$	Nieuwe server
$\boldsymbol{E}$	Bestaande server

Tabel 3: Parameters van de formules van het analysemodel

1. *Aantal gebruikers ~ reactietijd*
   Dit verband is lineair voor het maximale *throughput* moment en exponentieel na het maximale *throughput* moment.

$$mrt = c_L e^{\lambda_L * no\_of\_clients} \quad (4)$$

$$mrt = \lambda_U * no\_of\_clients + c_U \quad (5)$$

   De juiste keuze voor de parameters van vergelijking 4 en 5 kan worden gemaakt met het aantal gebruikers op het moment dat de maximale *throughput* bereikt is. Dit kan worden bepaald met behulp van de relatie tussen het aantal gebruikers en de *throughput*.

$$throughput = m * no\_of\_clients \quad (6)$$

   Deze relatie is lineair totdat de maximale *throughput* voor de server bereikt is, daarna is hij constant. De hellingsgraad van deze lineaire relatie, m, kan bepaald worden met behulp van historische data.

2. *Effect van maximale throughput van een server op het eerste verband*
   Dit verband bestaat uit twee functies die aangeven hoe de maximale *throughput* van een server in verband staat met de parameters van verband 1.

$$c_L = \Delta(c_L) * mx\_throughput + C(c_L) \quad (7)$$

$$\lambda_L = C(\lambda_L) * mx\_throughput^{\Delta(\lambda_L)} \quad (8)$$

   Vergelijkingen 7 en 8 zijn verbonden met de parameters van vergelijking 4. Voor vergelijking 5 kan op dezelfde manier vergelijkingen opgesteld worden.

3. *Percentage requests ~ maximale throughput van de server*
   Er is gebleken dat de maximale *throughput* bij een bepaald percentage *requests* bij een nieuwe server in verband staat met de maximale *throughput* bij hetzelfde percentage *requests* bij een al bekende server. Hiervoor moet wel bij beide servers de maximale *throughput* bij 0% requests bekend zijn.

$$mx_{throughput_N}(b) = \frac{mx_{throughput_E}(b)}{mx_{throughput_E}(0)} * mx_{throughput_N}(0) \quad (9)$$

   Hierbij is $mx_{throughput_N}(b)$ de maximale *throughput* van de nieuwe server bij b% requests en $mx_{throughput_E}(0)$ is de maximale *throughput* bij 0% requests.

De parameters van deze verbanden worden gevonden door trendlijnen aan te passen aan de data. Bij dit voorbeeld werd een *accuracy* gevonden van 89.1% bij de servers waarvan de data gebruikt is, en een *accuracy* van 83% bij een nieuwe server (Bacigalupo et al., 2005).

3.1.4 Hybride methode

Deze methode maakt gebruik van gelaagde wachtrijmodellen en het analysemodel, beide zijn hierboven uitgelegd. In dit geval wordt er met behulp van gelaagde wachtrijmodellen historische data gegenereerd die als basis dient voor het analysemodel. Op deze manier hoeven er dus niet meerdere datapunten gevonden te worden, maar dient de configuratie van de applicatie als input. Bij het analysemodel wordt het derde verband dan niet gebruikt, omdat alle servers al bekend zijn. Bij het voorbeeld van Bacigalupo et al. (2010) is gevonden dat dit model een vergelijkbare *accuracy* heeft als het gelaagde wachtrijmodel, namelijk 67.1% voor de bekende server en 74.9% voor de

nieuwe server (bij het gelaagde wachtrijmodel: 68.8% voor de bekende server en 73.4% voor de nieuwe server).

3.1.5 Conclusie

Zowel voor neurale netwerken als voor het analysemodel is historische data nodig om een voorspelling te kunnen maken. Dit kan wel geproduceerd worden, maar het is gewenst om de input van het model zo simpel mogelijk te houden, zodat het op meerdere applicaties eenvoudig toegepast kan worden. Bij neurale netwerken kan enkel voor de huidige configuratie worden voorspeld, wat niet wenselijk is voor dit onderzoek.

	Gelaagde wachtrijmodel	Neurale netwerken	Analysemodel	Hybride methode
Is de benodigde data beschikbaar?	✓	✗	✗	✓
Is het model toe te passen op meerdere soorten applicaties?	✓	✓	✓	✓
Kan het aantal CPU/servers aangepast worden en kunnen daar nieuwe voorspellingen op gebaseerd worden?	✓	✗	✓	✓

Tabel 4: Methoden afwegen

Dan blijven alleen het gelaagde wachtrijmodel en de hybride methode over. De hybride methode bestaat uit het gelaagde wachtrijmodel gecombineerd met het analysemodel. De *accuracy* van beide modellen is vergelijkbaar, dus wordt er gekozen voor het gelaagde wachtrijmodel. Deze keuze is gemaakt omdat een uitbreiding maken op het gelaagde wachtrijmodel wat geen verbetering oplevert ten koste gaat van tijd die beter in andere onderdelen van dit onderzoek gestopt kan worden.

3.2. Testen model

Om bij het gelaagde wachtrijmodel na te gaan hoe accuraat de voorspellingen zijn, moeten de uitkomsten van het model vergeleken worden met de gemeten waardes. Dit wordt gedaan met behulp van waardes van de *throughput* en de reactietijd. Dit wordt eerst gedaan met behulp van waardes van de configuratie waarmee ook de service demand waardes van het model zijn opgesteld. Daarnaast wordt ter validatie van het model ook nog de waardes vergeleken bij een andere configuratie. Deze vergelijking van voorspelde waardes en gemeten waardes wordt weergegeven met behulp van een error-percentage. Hoe dichter dit percentage bij de 0% zit, hoe accurater het model is (Bacigalupo, 2010).

$$error = \frac{voorspelde\ waarde - gemeten\ waarde}{gemeten\ waarde} * 100\% \qquad (10)$$

4. Data

In dit hoofdstuk wordt uitgelegd welke applicatie ter illustratie gebruikt wordt, welke testen er gedraaid zijn om het model op te stellen en met welke data de error-percentages berekend gaan worden.

4.1. De applicatie

De applicatie die ter illustratie gebruikt gaat worden is van een bedrijf dat leermiddelen levert aan basisscholen, het voortgezet onderwijs en het MBO. Omdat de gegevens die gebruikt worden vertrouwelijk zijn, wordt de naam van het bedrijf en de naam van de applicatie weggelaten en wordt de applicatie simpelweg “applicatie” genoemd.
Bij de applicatie die gebruikt wordt kan een docent een toets aanmaken en een door een student gemaakte toets nakijken, een student kan een toets maken en de resultaten ervan bekijken. Deze functionaliteiten worden hierna “scenario’s” genoemd.
De applicatie bestaat uit een paar servers: een databaseserver, een user-managementserver en de applicatieserver:

- De database server wordt in het model niet meegenomen omdat deze zo opgeschaald is dat deze geen probleem kan vormen, dit is aangegeven door Computest.
- De user-managementserver houdt zich bezig met de toegang van gebruikers tot de applicatie.
- De applicatieserver bevat de functionaliteiten van de applicatie.

De applicatie server en de user-managementserver kunnen opgeschaald of verkleind worden met behulp van replica’s. Dit komt erop neer dat het aantal CPU’s meer of minder kan worden, waardoor er meer of minder gebruikers de applicatie kunnen gebruiken zonder dat er wat misgaat. Bij dit onderzoek is er bij twee verschillende hoeveelheden replica’s getest, namelijk twee replica’s van de user management server samen met twee replica’s van de applicatieserver, maar ook met zes replica’s van de user-managementserver in combinatie met vier replica’s van de applicatieserver. Naast de replica’s bestaat elke CPU uit vier cores.

4.2. Load testen

Om data te verzamelen voor het bepalen van de service demands en om de error-percentages te kunnen gaan berekenen, zijn er meerdere loadtests uitgevoerd. Hiervoor is de test applicatie Gatling gebruikt. De testcode die gebruikt is bestond al; deze is al eerder gebruikt voor de applicatie. Hierbij hoefde enkel een aantal waardes aangepast te worden of er moest gedeelten van de code weggelaten worden om de benodigde tests te laten runnen.
Voorbeeld van een stukje testcode:

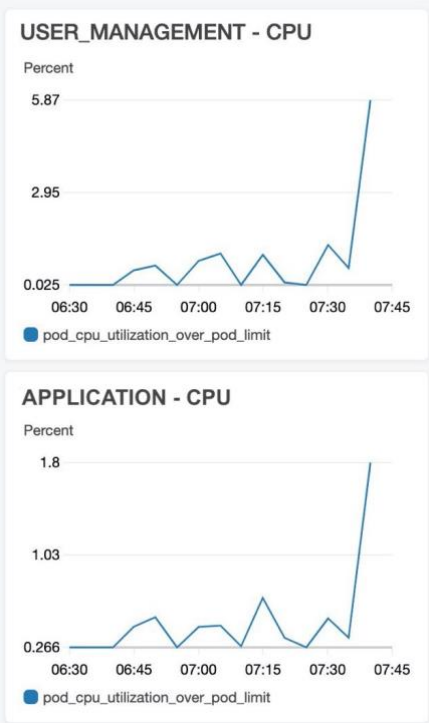
```
//Login Student
      .group("TC2_01_login_student") {
      exec(http("login")
        .get("/oauth-stub/stub/login?studentId=${uid}")
        .disableUrlEncoding
        .headers(header3)
        .basicAuth("…")
        .check(status.in(200 to 305))
          )
}
```

4.2.1 Service demand

Om de service demand van de verschillende CPU's te kunnen gaan berekenen is er per scenario een load test gedraaid met één gebruiker zonder denktijd voor vijf minuten. Naast de *throughput* zijn ook gegevens over het CPU gebruik tijdens deze tests nodig, zoals aangegeven in 3.1.1. Wat nu beschikbaar is zijn de grafieken hiernaast (Figuur 5). Hierin staat het CPU gebruik tijdens de loadtests aangegeven in procenten. In de grafieken zijn de vier load tests van de scenario's goed zichtbaar. De stijgende lijn aan het eind mag genegeerd worden, deze is van een andere load test. Om het CPU gebruik per test te vinden is het gemiddelde genomen van de grafiekstukjes die de loadtests voorstellen. De waardes zijn zichtbaar in de tabel hieronder (Tabel 5).

			CPU gebruik (%)	
Scenario	Gem. reactietijd (ms)	Throughput (aantal/s)	User management	Applicatie
1	65	15,26	0,57	0,47
2	68	14,59	0,89	0,44
3	61	16,31	0,54	0,51
4	65	15,21	1,35	0,48

Tabel 5: CPU gebruik



Figuur 5: CPU gebruik

4.2.2 Error-percentages

Om uiteindelijk de error-percentages van het model te kunnen berekenen zijn twee waardes nodig: de gemeten waarde en de berekende waarde:

- De berekende waarde wordt gevonden met behulp van de LQNS-tool.
- De gemeten waarde is met behulp van een load test gevonden.

Deze test werd gedurende vijftien minuten gedraaid, om een steady-state te bereiken. Als er maar een paar seconden gedraaid zou worden, dan zouden er te veel uitschieters kunnen zijn die het gemiddelde beïnvloeden. Omdat de reactietijd en *throughput* waarschijnlijk niet een lineair verband hebben met de userload, zijn er meerdere tests gedraaid met verschillende hoeveelheden gebruikers. Hiervoor is, behalve bij vier gebruikers, de verhouding twintig studenten : één docent aangehouden.

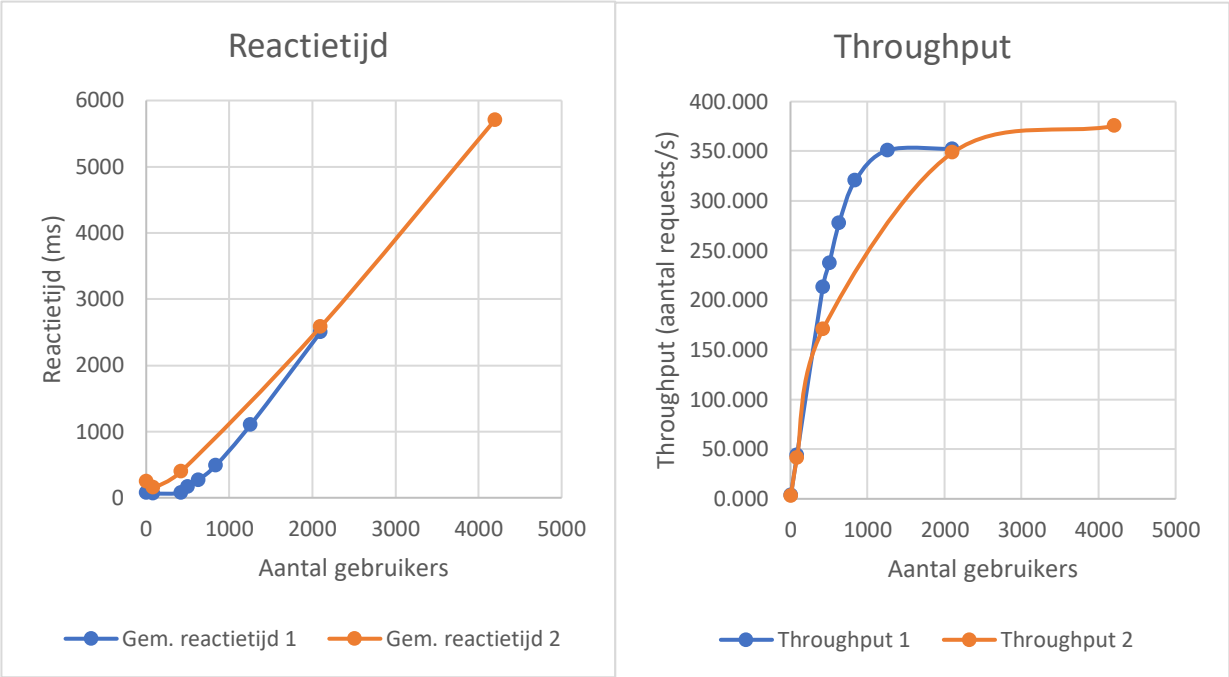
Daarnaast zijn de tests gedraaid met de twee verschillende hoeveelheden aan gerepliceerde servers. De configuratie met zes replica's van de user-managementserver met vier replica's van de applicatieserver is ook gebruikt voor het bepalen van de service demand. De andere configuratie gaat gebruikt worden om te zien hoe goed het model de performance bij een verschillende hoeveelheid servers kan voorspellen. De uitkomsten van de loadtests zijn te zien in Tabel 6 en 7. De rijen waarbij het aantal users in het rood staat waren er foutmeldingen tijdens de test, wat betekent dat niet alle *requests* van de test goed gingen.

Aantal users	Gem. reactietijd 1	Throughput 1
4	76	3,19
84	64	43,77
420	72	213,26
504	168	237,31
630	270	277,15
840	492	320,39
1260	1109	350,97
2100	2503	352,08

Tabel 6: Performance bij 6 & 4 replica's

Aantal users	Gem. reactietijd 2	Throughput 2
4	246	2,75
84	157	40,81
420	398	170,41
2100	2583	348,37
4200	5710	375,27

Tabel 7: Performance bij 2 & 2 replica's



Figuur 6: Reactietijd en throughput waardes van beide configuraties

In de grafieken hierboven zijn de reactietijden en *throughputs* van beide tests naast elkaar gezet. Net zoals in tabel 6 en 7 staat reactietijd of *throughput* 1 voor de configuratie met 6 en 4 replica's, en reactietijd of *throughput* 2 staat voor de configuratie met 2 en 2 replica's. Bij de reactietijden valt op dat de blauwe lijn onder de oranje lijn blijft. Dit was te verwachten, aangezien met meer replica's van de bestaande servers de applicatie ook meer gebruikers gemakkelijker aankan, wat overeenkomt met een lagere reactietijd.
Bij de *throughput* grafiek liggen de waardes in het begin wat dichter bij elkaar, al blijft de blauwe lijn bij de meetpunten altijd boven de oranje lijn. Dit betekent dus dat de configuratie met meer replica's ook meer requests per seconde aankan. Dit is dus ook volgens verwachting. Wel lijkt het alsof na de 2100 gebruikers de oranje lijn boven de blauwe uit gaat komen, maar dit is niet met zekerheid te zeggen.

5. Resultaten

De methode die besproken is in 3.1.1 is gebruikt voor het opstellen van het voorspellingsmodel. In dit hoofdstuk wordt het model besproken, en worden de uitkomsten vergeleken met de gemeten waardes. Voor het opstellen van het model is de LQN-editor gebruikt, en om de resultaten van het model te krijgen is gebruik gemaakt van de LQNS-tool. Beide zijn ontwikkeld door een team van de Carleton University in Canada.

5.1. Voorspellingsmodel

5.1.1 Service demand

Om de service demand te vinden met behulp van de gevonden waardes van het CPU gebruik (zie 4.2.1), is de *Utilization Law* gebruikt:

$$Service\ demand\ van\ een\ systeembron = \frac{Utilization\ van\ de\ systeembron}{Throughput\ van\ het\ systeem} \qquad (11)$$

Hiervoor is het afgelezen gemiddelde CPU gebruik gedeeld door de *throughput* van die test. De uitkomsten van deze breuk zijn hieronder zichtbaar in Tabel 8. Deze waardes worden in het model ingevuld bij de *entries* van de user management en application t*asks*.

			Service demand	
Scenario	**Gem. reactietijd (ms)**	**Throughput (aantal/s)**	**User management**	**Application**
1	65	15,26	0,0376	0,0310
2	68	14,59	0,0583	0,0291
3	61	16,31	0,0354	0,0333
4	65	15,21	0,0888	0,0312

Tabel 8: Service demand

5.1.2 LQN-editor

Met behulp van de LQN-editor, bijgeleverd bij de LQN-solver, kan een gelaagd wachtrijmodel opgesteld en aangepast worden. In Bijlage 1 zijn screenshots van de editor te vinden. Om een wachtrijmodel in de editor te zetten, is er een bepaalde volgorde die aangehouden moet worden:

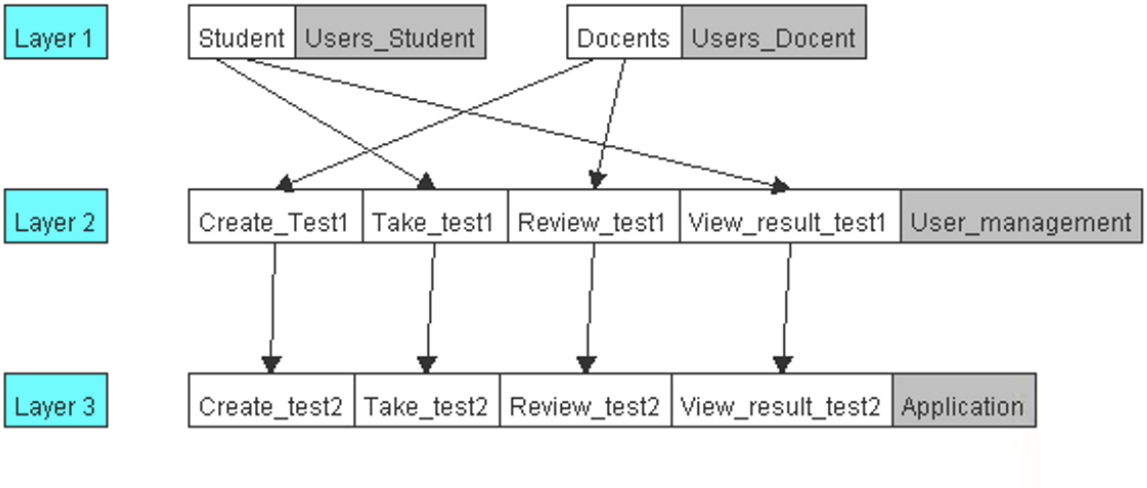
- Processors aanmaken;
- Tasks aanmaken en die aan de bijbehorende processor koppelen;
- Entries binnen de tasks aanmaken;
- Calls tussen de entries aanmaken.

Met de informatie die beschikbaar is over de applicatie kan een gelaagd wachtrijmodel opgesteld worden. De onderdelen die gebruikt zijn om de applicatie te weerspiegelen zijn:

Processors
CPU (voor de docenten en de studenten)
User_management – CPU
Application - CPU
Tasks met bijbehorende entries
Users_students
Students
Users_docents
Docents
User_management
Create_test1
Take_test1
Review_test1
View_result_test1
Application
Create_test2
Take_test2
Review_test2
View_result_test2

Tabel 9: Onderdelen gelaagd wachtrijmodel

De calls in het model zijn geneste synchrone calls. Dit betekent dat een gebruiker een call maakt naar de user_management laag, deze stuurt een antwoord terug maar maakt ook een call naar de application laag.
Een visualisatie van het model is hieronder zichtbaar in Figuur 7. Hierin zijn alleen de processors niet zichtbaar.

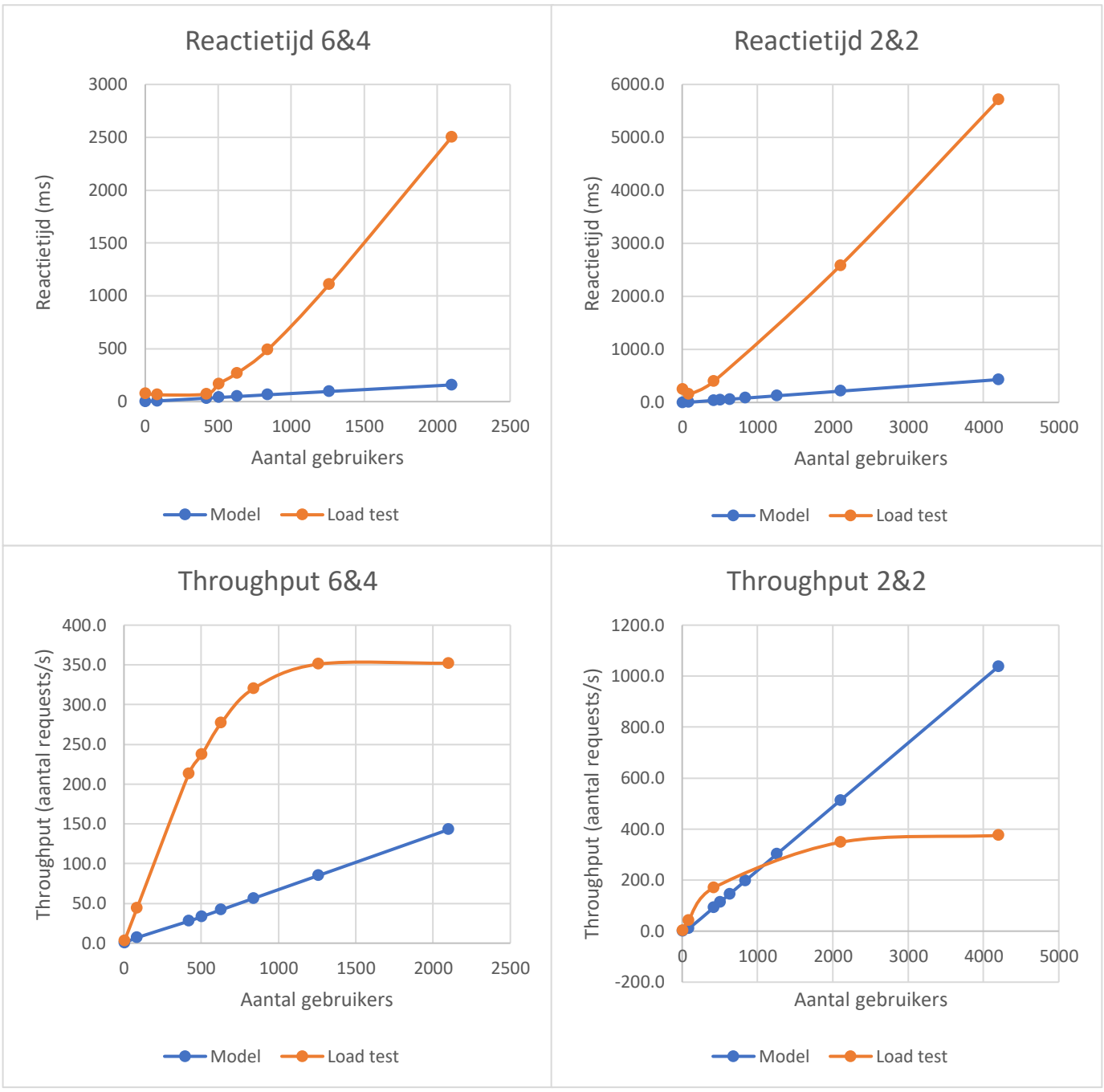


Figuur 7: Gelaagd wachtrijmodel van de applicatie

5.2. Testen model

Om performance indicatoren uit het model te krijgen, wordt via de command line de tool in combinatie met het model aangeroepen. Er wordt gebruik gemaakt van de analytische methode, en niet van de simulatie, ondanks dat deze betere resultaten op kan leveren. Dit komt omdat de simulatie geen replicatie aankan in het model.
Voordat de error-percentages berekend zijn, zijn eerst de gevonden waardes samen met de gemeten waardes van de load tests in een grafiek gezet. Dit is gedaan voor zowel de 6&4 configuratie als de 2&2 configuratie, voor de reactietijden en voor de *throughputs* (zie Figuur 8). Wat gelijk opvalt is dat alle uitkomsten van het model lineair zijn in tegenstelling tot de waardes uit de load tests. Volgens onder andere Bacigalupo (2010) is de relatie tussen de reactietijd en het aantal gebruikers tot het maximale *throughput* moment lineair, daarna exponentieel. Bij de *throughput* is de relatie eerst lineair en daarna constant. Dit is te zien bij de uitkomsten van de load tests, maar niet bij de uitkomsten van het model. Dit betekent dus dat het model niet goed de performance van de applicatie kan voorspellen. Hier kunnen verschillende redenen achter zitten, zoals:

- Geen goede input voor de service demand.
- Model is te simpel waardoor de bottlenecks niet gevonden worden. Hierdoor kan de maximale *throughput* niet bereikt worden.



Figuur 8: Reactietijden en throughputs, vergelijking tussen load test en model

Ondanks dat al uit de grafieken afgelezen kan worden dat de uitkomsten niet accuraat zijn, worden toch de error-percentages berekend, zodat de grootte van de fout precies bekend is. De uitkomsten hiervan zijn zichtbaar in Tabel 10 en 11.

Aantal users	Reactietijd error	Throughput error
4	98,0%	92,8%
84	89,9%	84,1%
420	55,2%	87,0%
504	77,0%	85,9%
630	82,1%	84,8%
840	86,9%	82,4%
1260	91,4%	75,7%
2100	93,7%	59,4%
Gemiddelde	**84,3%**	**82%**

Tabel 10 Error-percentages bij de configuratie met 6&4 replica's

Aantal users	Reactietijd error	Throughput error
4	99,9%	97,6%
84	97,5%	77,1%
420	90,3%	45,4%
2100	91,7%	47,3%
4200	92,4%	176,6%
Gemiddelde	**94,4%**	**88,8%**

Tabel 11: Error-percentages bij de configuratie met 2&2 replica's

6. Conclusie

In dit hoofdstuk worden de hoofd- en deelvragen beantwoord aan de hand van de uitkomsten van het onderzoek.

In dit onderzoek is gezocht naar een antwoord op de vraag: "*Op welke manier kan de performance van een applicatie voorspeld worden?*". Om antwoord te geven op deze vraag is er literatuuronderzoek uitgevoerd naar de factoren die invloed hebben op de performance van een applicatie, maar ook naar de methoden die toegepast kunnen worden om de performance te voorspellen. Daarna is aan de hand van informatie over een bestaande applicatie getracht een voorspellingsmodel op te stellen met behulp van de gekozen methode: gelaagde wachtrijmodellen.

Wat goed naar voren kwam in de literatuur is dat de performance van een applicatie overal van afhankelijk is: zowel hardware, software, als netwerkverbindingen hebben invloed op de performance. Performance kan worden omschreven met behulp van deze termen: beschikbaarheid, reactietijd, *throughput* en *utilization*.

Performance kan op een aantal manieren voorspeld worden. Zo kan er met behulp van veel input in de vorm van reactietijden en *throughputs* met behulp van neurale netwerken een voorspellingsmodel worden opgezet. Dit model kan alleen niet voor een andere configuratie voorspellen, enkel voor de bestaande opzet. Met dezelfde input kan ook een model worden opgezet door middel van de verbanden tussen het aantal gebruikers, de reactietijd en de *throughput*, ook wel het analysemodel genoemd. Dit model kan ook worden opgesteld met behulp van data gegenereerd door gelaagde wachtrijmodellen, dit wordt het hybride model genoemd. Als laatste kunnen gelaagde wachtrijmodellen op zichzelf ook gebruikt worden om een voorspellingsmodel op te stellen. Deze methode heeft als input de opstelling van de applicatie nodig, samen met bijbehorende waardes. Er is gekozen voor gelaagde wachtrijmodellen omdat de benodigde input beschikbaar was, maar ook omdat bij dit model eenvoudig het aantal servers of processors is aan te passen. Er wordt gebruik gemaakt van de LQNS-tool, ontwikkeld door een team van de Carleton University om gelaagde wachtrijmodellen op te kunnen lossen.

Om het model op te kunnen stellen is er informatie nodig over de onderdelen van de applicatie. Alle servers en informatie over de bijbehorende CPU's moeten in ieder geval bekend zijn. De service demand van elk los onderdeel is één van de vereisten. Het gemiddelde aantal keer dat een call gemaakt wordt moet ook worden ingevuld. Hoe meer informatie over het interne proces van de applicatie er beschikbaar is, hoe accurater het model wordt.

De betrouwbaarheid van een voorspellingsmodel kan vastgesteld worden door de uitkomsten te vergelijken met de werkelijkheid. Om het vergelijkingsmateriaal te verkrijgen kan er met behulp van load tests de reactietijd en *throughput* bij verschillende hoeveelheden gebruikers worden bepaald. Deze waardes kunnen dan met bijvoorbeeld een error-percentage worden vergeleken met de uitkomsten van het model.

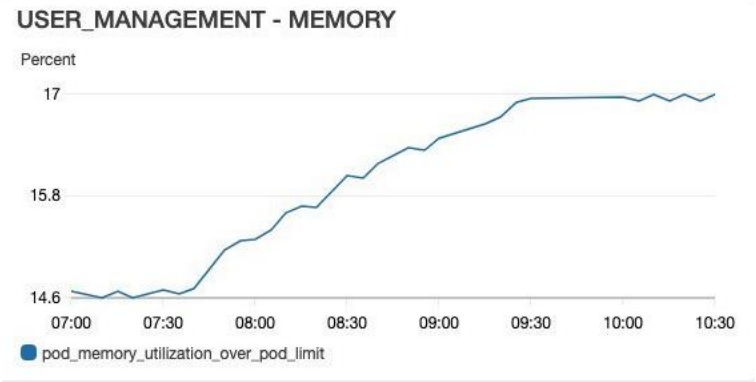
Om de gekozen methode te illustreren is er informatie van een applicatie ter beschikking van het onderzoek gesteld. Deze applicatie heeft vier scenario's, welke zijn gemodelleerd met de tool. De uitkomsten van het model waren echter niet nauwkeurig genoeg om lage error-percentages te krijgen. Dit kan komen omdat het opgestelde model te simpel was, er had dieper op de scenario's ingezoomd moeten worden.

7. Discussie

In dit hoofdstuk worden de onderdelen van dit onderzoek besproken die de betrouwbaarheid van dit onderzoek in twijfel zouden kunnen trekken.

Bij het indelen van de *request* types is er in dit onderzoek gebruik gemaakt van de bestaande scenario's. Dit is in tegenstelling tot wat er beschreven staat in hoofdstuk 3.
De aangeraden methode bestaat namelijk uit informatie verzamelen van elke *request*, zoals de gebruikte systeembronnen en de hoeveelheden data die gebruikt worden. Als deze informatie beschikbaar was geweest, dan was deze methode toegepast. Een mogelijke manier om deze informatie te achterhalen is door per losse *request* een test te draaien met minstens tien minuten tussen elke test, dit zouden minstens twintig tests worden bij de applicatie gebruikt in dit onderzoek. Hierna zou dan uit de grafieken met het CPU-gebruik afgelezen worden welke *request* wel of niet gebruik maakt van de user management, applicatie of database CPU. Zodra dit bepaald is, kan op basis van het gebruik van de systeembronnen de *request* types opgesteld worden.

Naast dat de *request* types niet op de aangegeven wijze bepaald zijn, was er ook geen informatie over disk- of netwerkgebruik. Voor het geheugengebruik waren er wel grafieken aanwezig (zie Figuur 9 hiernaast), maar uit de grafieken werd niet duidelijk hoeveel van het gebruik elke individuele test toebehoorde. Dit komt omdat er informatie in de cache van het geheugen blijft staan, waardoor het percentage in de grafiek cumulatief oploopt.



Figuur 9: Geheugengebruik

Bij de pieken in de grafieken met het CPU gebruik die bij de gemaakte load tests horen, zijn bij dit onderzoek de gemiddelden genomen. Dit is gedaan omdat er gezocht werd naar het gebruik van de hele load test. Echter, er is een andere mogelijkheid om de grafiek af te lezen, namelijk door het maximum van de piek te nemen. Dit kan ook voorstellen hoeveel procent van de CPU in gebruik is bij de bepaalde *request* type. Daarnaast is er een onderdeel van de grafiek niet meegenomen: voordat de load test begint zijn de percentages van het CPU gebruik niet nul. Dus dan is de vraag of die ondergrens wel of niet meegeteld zou moeten worden. Bij dit onderzoek is dit wel gedaan. Dit is gedaan omdat in de meeste literatuur er niet met grafieken wordt gewerkt waarin het CPU-gebruik staat, maar die kunnen het CPU-gebruik monitoren met een andere tool die enkel een percentage geeft. Op die manier kan ook geen rekening gehouden worden dat de CPU nog voor een klein percentage met iets anders bezig is.

Een van de onderdelen die in het model ingevoerd moest worden was het gemiddelde aantal calls. Deze waarde moest voor elke specifieke call ingevuld worden. Er was echter geen informatie over deze waardes bij deze applicatie beschikbaar. Daarom is ervoor gekozen per scenario de waarde één in te vullen, zodat tijdens een scenario er standaard een call gaat van één user naar de user management, en één van de user management naar de applicatie. Om dit te verbeteren kan er, net zoals bij de *request* types, gekeken worden naar CPU gebruik bij load tests met maar één *request*.

8. Aanbevelingen

In het literatuuronderzoek is naar voren gekomen dat gelaagde wachtrijmodellen een goede methode is om de performance van een applicatie te voorspellen. In de uitvoering kwamen echter minder goede resultaten naar voren. Advies is om het model eerst zo te specificeren dat de error-percentages dichter bij 0% komen. Daarna kan het modelleren eventueel bij meerdere applicaties worden toegepast.

Als iemand zich hier verder in wil gaan verdiepen, zonder voorkennis van de LQNS-tool, dan luidt het advies: maak gebruik van de tutorial (Woodside, 2002) en de user manual (Franks et al, 2013). Bruikbare voorbeelden om te volgen bevinden zich in artikelen van Bacigalupo et al. (2005) en van Shoaib & Das (2011). De benodigde software is beschikbaar op http://www.sce.carleton.ca/rads/lqns/. Voor het installeren vanaf deze locatie zijn wel inloggegevens nodig, deze zijn verkrijgbaar door een mail te sturen naar Murray Woodside (cmw@sce.carleton.ca).

Als het gelukt is om de tool met een lage error-percentage werkend te krijgen voor één applicatie, dan is het advies om een uitgebreide tutorial te maken over het opstellen van het model, inclusief informatie over de benodigde tests die gedraaid moeten worden en de monitoring van de applicatie voor input van het model. Indien performance testers dan nog twijfels hebben over de tool, of geen accurate uitkomsten krijgen bij hun applicatie, dan luidt het advies om degene die de tutorial gemaakt heeft naar de tester toe te sturen. Diegene kan er dan voor zorgen dat er een werkend model komt die accurate uitkomsten geeft over de applicatie. Op deze manier zullen langzamerhand alle performance testers deze tool gaan gebruiken als extra hulpmiddel.

Indien er niet verder naar de gelaagde wachtrijmodellen gekeken wordt, dan luidt het advies om met behulp van meerdere load tests het analysemodel op te stellen en te onderzoeken hoe accuraat dat model is. Indien dit accuraat is, dan kunnen deze formules in een Excel-bestand opgenomen worden, zodat deze methode gebruikt kan worden door de performance testers van Computest.

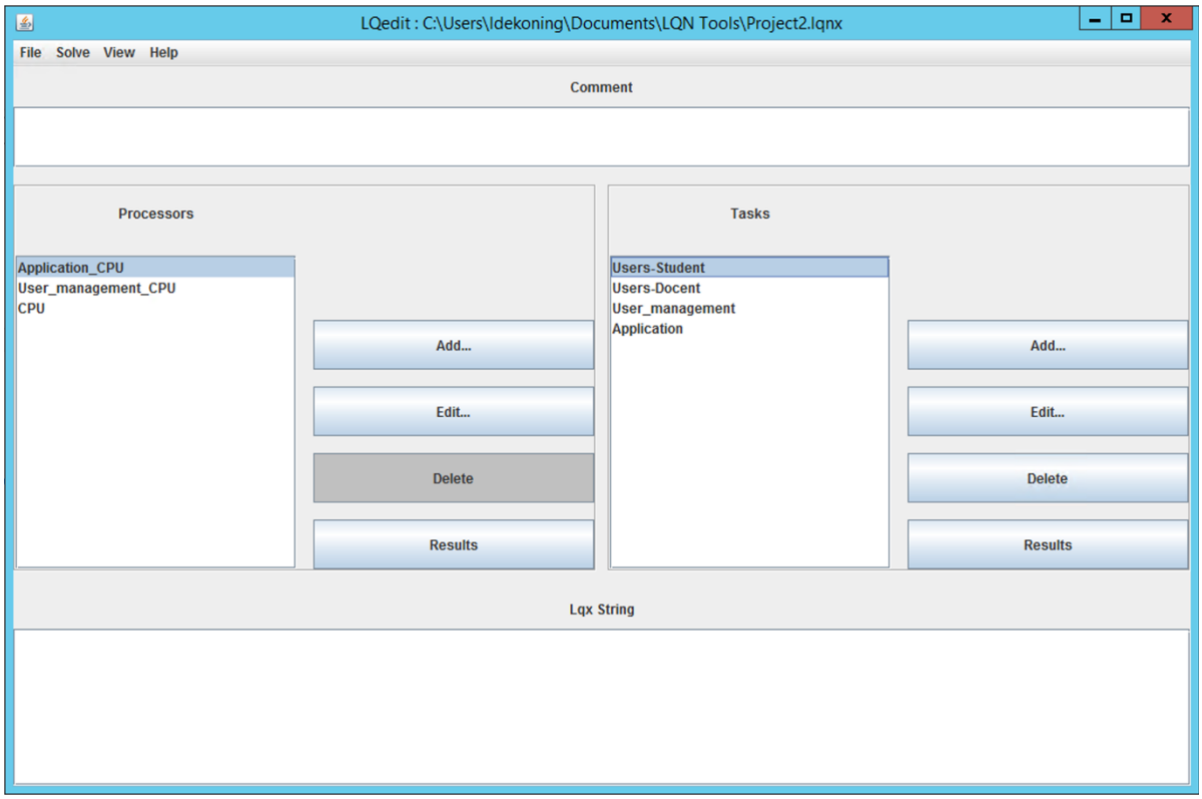
Literatuurlijst

- APMdigest (2013, 17 april). *15 top factors that impact application performance*. Geraadpleegd op 9 maart 2020, van https://www.apmdigest.com/15-top-factors-that-impact-application-performance
- AWS (z.d.). *Amazon EC2 instance types*. Geraadpleegd op 8 april 2020, van https://aws.amazon.com/ec2/instance-types/
- Bacigalupo, D. A., Jarvis, S. A., He, L., Spooner, D. P., Dillenberger, D. N., & Nudd, G. R. (2005). An Investigation into the Application of Different Performance Prediction Methods to Distributed Enterprise Applications. *The Journal of Supercomputing*, 34. 93-111. DOI: 111. 10.1007/s11227-005-2335-z.
- Bacigalupo, D. A., Van Hemert, J., Usmani, A., Dillenberger, D. N., Wills, G. B., & Jarvis, S. A. (2010). *Resource management of enterprise cloud systems using layered queuing and historical performance models*. DOI: 10.1109/IPDPSW.2010.5470782
- Christensson, P. (2012, 16 mei). *Bandwidth definition*. Geraadpleegd op 25 maart 2020, van https://techterms.com/definition/bandwidth
- Christensson, P. (2014, 11 juli). *CPU definition*. Geraadpleegd op 13 maart 2020, van https://techterms.com/definition/cpu
- Christensson, P. (2006). *Hard disk definition*. Geraadpleegd op 14 april 2020, van https://techterms.com/definition/harddisk
- Christensson, P. (2006). *Memory definition*. Geraadpleegd op 14 april 2020, van https://techterms.com/definition/memory
- Computer Hope (2019, 10 juli). *What are the differences between hardware and software?* Geraadpleegd op 20 maart 2020, van https://www.computerhope.com/issues/ch000039.htm
- Computest (z.d.). *Performance*. Geraadpleegd op 10 februari 2020, van https://www.computest.nl/nl/diensten/performance/
- Dell (2018, 8 juni). *How Random Access Memory (RAM) affects performance*. Geraadpleegd op 16 april 2020, van https://www.dell.com/support/article/nl-nl/sln179266/how-random-access-memory-ram-affects-performance?lang=en
- Dorsman, J. L. (2015). *Layered queueing networks : Performance modelling, analysis and optimisation* (Proefschrift). Geraadpleegd op 26 maart 2020, van https://doi.org/10.6100/IR784295
- ELPROCUS (z.d.). *Memory hierarchy in computer architecture*. Geraadpleegd op 14 april 2020, van https://www.elprocus.com/memory-hierarchy-in-computer-architecture/
- Expert System (2017). *What is machine learning? A definition*. Geraadpleegd op 2 april 2020, van https://expertsystem.com/machine-learning-definition/
- Guru99 (z.d.). *Performance testing tutorial: What is, types, metrics & example*. Geraadpleegd op 10 maart 2020, van https://www.guru99.com/performance-testing.html
- Franks, G., & Li, L. (2012, april). Efficiency Improvements for Solving Layered Queueing Networks. *ICPE '12: Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering* (pp. 279-282). DOI: 10.1145/2188286.2188338
- Franks, G., Maly, P., Woodside, M., Petriu, D. C., Hubbard, A., & Mroz, M. (2013, 30 januari). *Layered queueing network solver and simulator user manual*. Geraadpleegd van http://www.sce.carleton.ca/rads/lqns/LQNSUserMan-jan13.pdf
- Hillier, F. S., & Lieberman, G. J. (2015). *Introduction to Operations Research* (10e druk). New York: McGraw-Hill

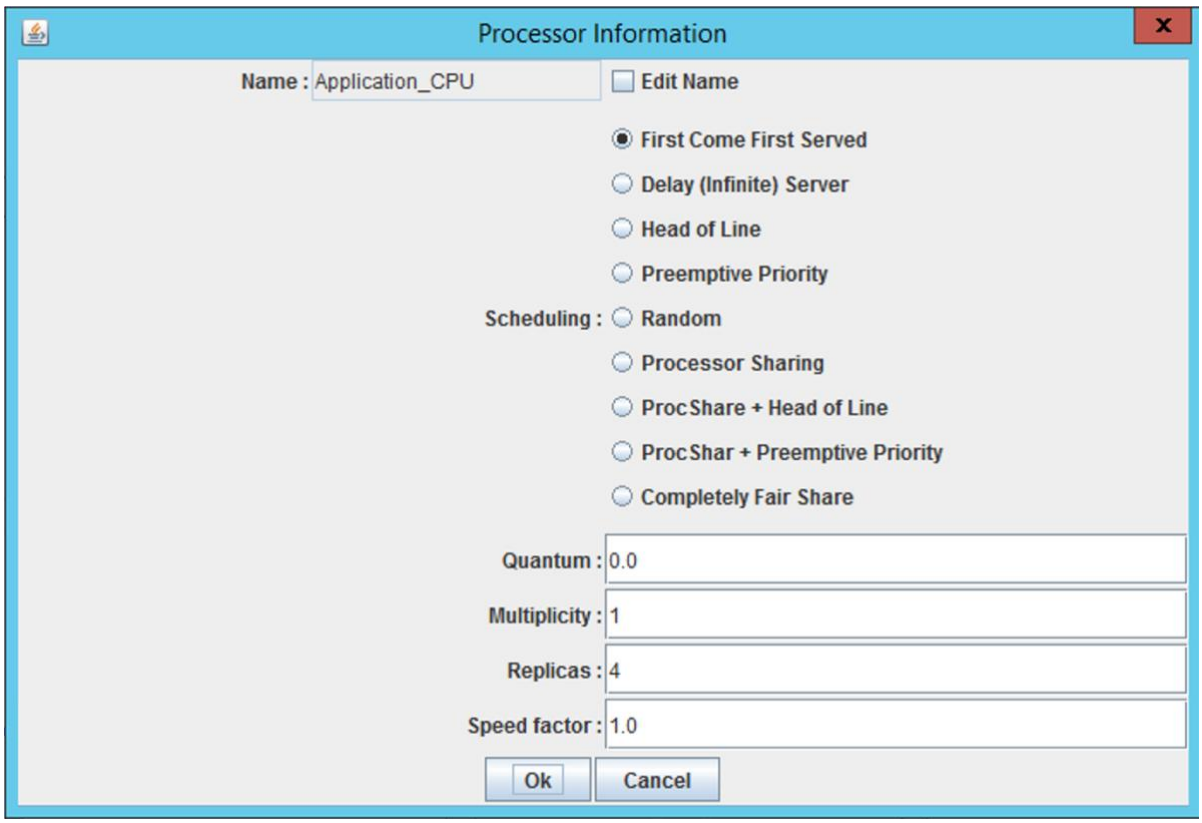
- Hoxmeier, J., & DiCesare, C. (2000). System Response Time and User Satisfaction: An Experimental Study of Browser-based Applications. *Proceedings of the Association of Information Systems Americas Conference*. Geraadpleegd op 12 februari 2020, van http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.99.2770&rep=rep1&type=pdf
- IBM (2020, 30 januari). *Observing response time*. Geraadpleegd op 12 februari 2020, van https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.5.0/tuning/dfht3ka.html
- Ipek, E., De Supinski, B. R., Schulz, M., & McKee, S. A. (2005). An Approach to Performance Prediction for Parallel Applications. *Euro-Par'05: Proceedings of the 11th international Euro-Par conference on Parallel Processing* (pp. 196-205). DOI: 10.1007/11549468_24.
- Lieber, E. (2015). *Fiber, Copper, or Wireless: Which Connection Is Best for Your Company?*. Geraadpleegd op 18 maart 2020, van https://smallbiztrends.com/2015/08/fiber-optic-copper-wireless-internet-transmission-methods.html
- Meier, J. D., Farre, C., Bansode, P., Barber, S., & Rea, D. (september 2007). *Performance testing guidance for web applications*. Geraadpleegd op 26 februari 2020, van https://docs.microsoft.com/en-us/previous-versions/msp-n-p/bb924356(v=pandp.10)?redirectedfrom=MSDN
- Mendis, C., Renda, A., Amarasinghe, S., & Carbin, M. (2019). Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks. *Proceedings of Machine Learning Research - Volume 97 (ICML 2019)*. Geraadpleegd op 6 april 2020, van https://arxiv.org/pdf/1808.07412.pdf
- Molyneaux, I. (2009). *The Art of Application Performance Testing - Help for Programmers and Quality Assurance*. Geraadpleegd op 13 februari, van https://books.google.nl/books?id=DccaCe9WzeoC&printsec=frontcover&hl=nl#v=onepage&q&f=false
- Nicholson, C. (z.d.). *A beginner's guide to LSTMs and recurrent neural networks*. Geraadpleegd op 2 april 2020, van https://pathmind.com/wiki/lstm
- Nicholson, C. (z.d.). *A beginner's guide to neural networks and deep learning*. Geraadpleegd op 2 april 2020, van https://pathmind.com/wiki/neural-network
- Nikolov, A. V. (2014). Analytical Model for Performance Evaluation of an Asynchronous Web Server. *Asian Journal of Information Technology, 13: 126-130*. DOI: 10.36478/ajit.2014.126.130
- Oracle (z.d.). *Oracle9i application server Oracle HTTP server powered by Apache performance guide*. Geraadpleegd op 13 februari, van https://docs.oracle.com/cd/A95428_01/a86059/concpts.htm
- Paessler (z.d.). *IT-explained*. Geraadpleegd op 24 februari 2020, van https://www.paessler.com/it-explained
- Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernando, J. P., & Saraiva, J. (2017). Energy Efficiency across Programming Languages. *Proceedings of SLE' 17*. 256-267. DOI: 10.1145/3136014.3136031
- PVS-Studio (z.d.). *Code optimization*. Geraadpleegd op 25 maart 2020, van https://www.viva64.com/en/t/0084/
- Raymond.cc (2019, september). *10 free tools to measure hard drive and SSD performance*. Geraadpleegd op 16 april 2020, van https://www.raymond.cc/blog/measure-actual-hard-disk-perfomance-under-windows/
- Ruijs, H. (z.d.). *Over ons*. Geraadpleegd op 10 februari 2020, van https://www.computest.nl/nl/over-computest/over-ons/
- Shoaib, Y., & Das, O. (2011). Web Application Performance Modeling Using Layered Queueing Networks. *Electronic Notes in Theoretical Computer Science 275*. 123-142. DOI: 10.1016/j.entcs.2011.09.009
- Tutorialspoint (z.d.). *Operating system - overview*. Geraadpleegd op 23 maart, van https://www.tutorialspoint.com/operating_system/os_overview.htm

- Van der Mei, R., Hariharan, R., & Reeser, P. (2001). Web Server Performance Modeling. *Telecommunication Systems* 16, 361–378. DOI: 10.1023/A:1016667027983
- Villinger, S. (2020, 16 maart). *What is a solid-state drive (SSD)?* Geraadpleegd op 15 april 2020, van https://www.avast.com/c-what-is-ssd
- Wainer, G. A. (2017). *Discrete-Event Modeling and Simulation: a Practitioner's Approach*. Geraadpleegd op 9 maart, van https://books.google.nl/books?id=E0T5dmTrG7EC&printsec=frontcover&hl=nl&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false
- Wiering, M. (z.d.). *Transparanten bij het vak inleiding adaptieve systemen: Neurale netwerken*. Geraadpleegd op 2 april 2020, van http://www.cs.uu.nl/docs/vakken/ias/HANDOUTS/07._(24)_NEURAL_NETWORKS_HANDOUTS.pdf
- Woodside, M. (2002, 2 mei). *Tutorial introduction to layered modeling of software performance*. Geraadpleegd van http://www.sce.carleton.ca/rads/lqns/lqn-documentation/tutorialg.pdf
- Zheng, T., & Woodside, M. (2003). Heuristic Optimization of Scheduling and Allocation for Distributed Systems with Soft Deadlines. *Lecture notes in Computer Science*, 2794, 169:181. DOI: 10.1007/s11227-005-2335-z

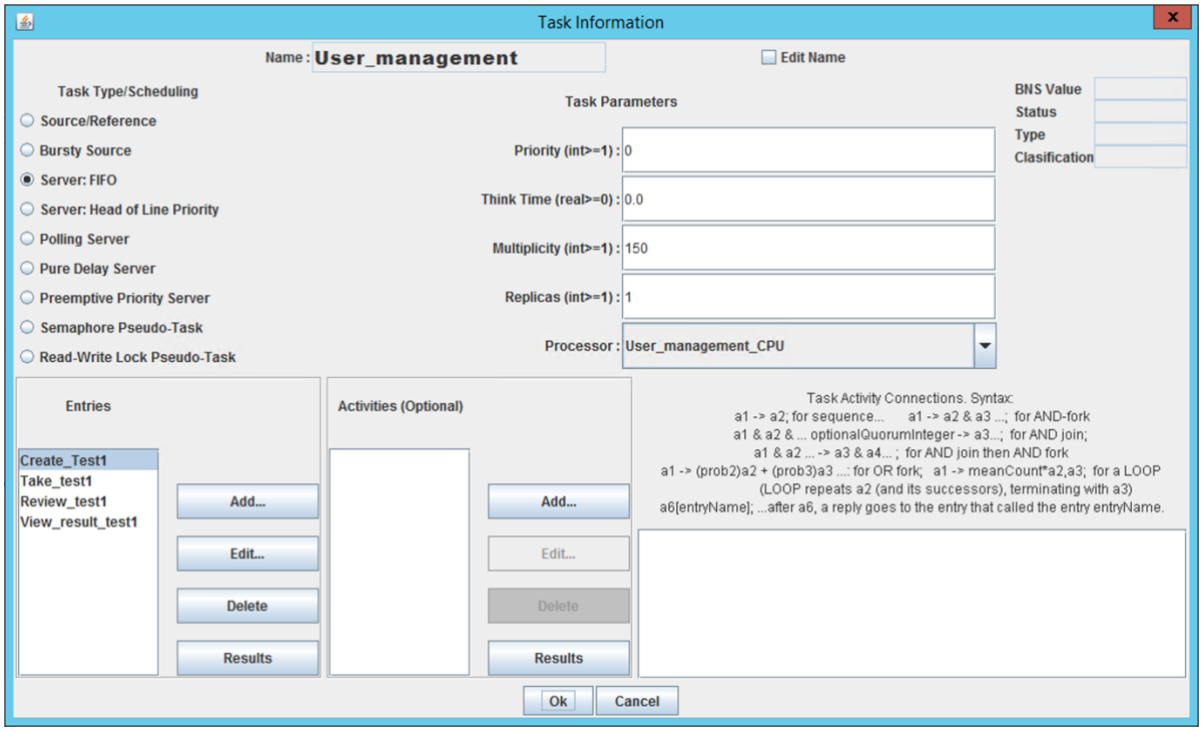
Bijlage 1: Screenshots LQN-Editor



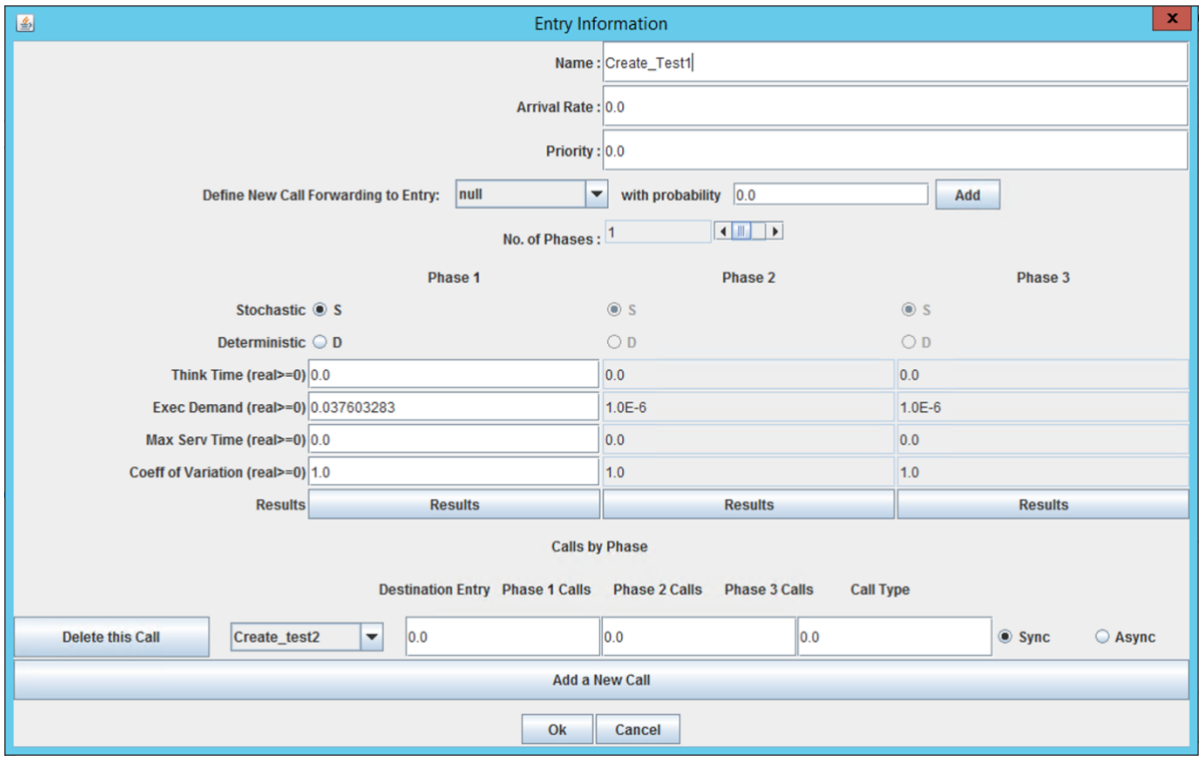
Figuur 11: Editor hoofdblad



Figuur 10: Editor processor



Figuur 12: Editor task



Figuur 13: Editor entry