Radboud Universiteit Nijmegen

Bachelorscriptie

Een keuzemodel voor webserver architecturen afhankelijk van het dynamisch karakter van een website

Auteur:
Xander Damen
0213845
x.damen@student.science.ru.nl

Begeleider:
Prof dr. M.C.J.D. van Eekelen
Institute for Computing and Information Sciences
Radboud Universiteit Nijmegen

Januari 2012

Abstract

De architectuur van een webserver is van grote invloed op de prestaties van een webserver. Deze architectuur bestaat uit verschillende onderdelen die allen op hun eigen manier bedragen aan de prestaties. Aan de hand van de combinatie van onderdelen kan er een voorspelling worden gedaan hoe deze webserver presteert. Deze prestaties wisselen echter per karakter van de website. Door de architectuur te relateren aan het karakter van een website kan worden voorspeld welke architectuur de beste prestatie levert voor een website. Deze bevindingen zijn door middel van metingen te valideren.

Inhoudsopgave

1 Inleiding

Steeds meer mensen krijgen toegang tot het internet, en de verbindingssnelheden waarover gebruikers de beschikking hebben worden hoger. Het aantal verzoeken aan webservers stijgt hierdoor flink en gebruikers verwachten vanwege hun snelle verbinding dat de opgevraagde website snel op hun scherm wordt getoond. De vraag naar snelle webserver software, die vele verzoeken tegelijkertijd af kan handelen, groeit hierdoor. Maandelijks voert NetCraft een onderzoek uit over het gebruik van webserver software. Uit het onderzoek van september 2011 [37] blijkt dat de *Apache* webserver de meestgebruikte software is op webservers. Echter, het gebruik van Apache lijkt af te nemen ten faveure van *Nginx* (wat wordt uitgesproken als *Engine X*).

Waarom is dit het geval? Apache staat al vele jaren bekend als een stabiele en veilige webserver. En de communicatiemethode tussen server en cliënt is al jaren HTTP [32]. Wat zorgt er voor dat systeembeheerders toch de overstap maken naar bijvoorbeeld Nginx? Nginx blijkt een asynchrone webserver [38] te zijn, terwijl Apache traditioneel gebaseerd is op processen. Brengt dit verschil in onderliggende architectuur zoveel voordelen met zich mee? En geldt dit altijd of zijn er situaties denkbaar dat een webserver gebaseerd op processen alsnog de voorkeur geniet?

Deze scriptie gaat in op deze webserver architectuur, en specifiek de procesarchictectuur. Aan de hand van de onderzoeksvraag wordt er onderzoek gedaan naar de toepassing deze procesarchitecturen bij websites.

1.1 Onderzoeksvraag

De volgende vraag staat centraal in dit onderzoek:

> Voor welk dynamisch karakter van een website presteert welke webserver procesarchitectuur het best?

Om deze vraag te kunnen beantwoorden, dienen er een aantal onderwerpen uitgediept te worden. Om te beginnen moet het dynamisch karakter van websites beschreven worden, omdat dit grote invloed heeft op de prestaties van de webserver [1]. Daanaast dient de procesarchitecturen geïdentificeerd te worden, omdat ook dit de prestaties erg kan beïnvloeden [2,39]. Vervolgens dient bepaald te worden aan welke voorwaarden een architectuur moet voldoen om het best geschikt te zijn voor een bepaald dynamisch karakter van een website. Aan de hand van deze theorie wordt vervolgens een model opgesteld. Dit model dient vervolgens gevalideerd te worden en wordt in een grotere context geplaatst. Ten slotte wordt er een blik in de toekomst geworpen.

Dit creeërt de volgende vragen:

- Welke soorten dynamisch karakter van websites kunnen er onderscheiden worden?
- Wat zijn de verschillende webserver procesarchitecturen en hoe onderscheiden deze zich?
- Wanneer is een webserverprocesarchitectuur beter geschikt dan een andere?

- Welke variabelen bepalen de prestaties van een webserver?
- Waaruit moet een prestatiemodel voor dynamische karakters en procesarchitecturen bestaan?
- Klopt het model in de praktijk?

Deze vragen vormen de basis voor de hoofdstukken van deze scriptie. Enige basiskennis van netwerken, processen, threads en programmeren is vereist voor een volledig begrip van de beschreven onderwerpen.

1.2 Methode

Er wordt een kwalitatief onderzoek gedaan naar de verschillende karakters en architecturen en hun prestaties. Uit deze bevindingen wordt een model opgesteld. Deze bevindingen worden door middel van kwantitatieve metingen in een simulatie-omgeving gevalideerd.

In deze scriptie wordt vanwege financiële redenen alleen gebruik gemaakt van vrij beschikbare software. Daarnaast maakt dit gebruik door andere onderzoekers eenvoudiger [3]. Commerciële alternatieven worden, indien relevant, kort beschreven op basis van vrij beschikbare informatie of informatie beschikbaar via de bibliotheek van de Radboud Universiteit.

2 Dynamisch karakter van websites

Zoals in de introductie al genoemd, heeft de dynamiek van een webpagina invloed op de prestaties. Niet alleen aan de kant van de gebruiker, maar ook aan de kant van de webserver. Dit hoofdstuk heeft tot doel de verschillende vormen van dynamiek van webpagina's in kaart te brengen.

Ten eerste zijn webservers in staat twee verschillende typen data te leveren: statische data uit bestanden opgeslagen op de server en dynamische data verkrijgen uit programma's op het moment van het opvragen van de webpagina [1, 4, 33]. Daarnaast is het mogelijk om dynamiek aan de kant van de cliënt te creeëren [4] of een combinatie technieken genaamd AJAX [5].

Deze dynamische websites vinden hun oorsprong in CGI (*Common Gateway Interface*) programma's die aan de hand van een aantal variabelen dynamische HTML pagina's genereerden. Dit vond (en vindt) volledig plaats aan de kant van de server. In 1995 deed JavaScript zijn intreden in de Netscape webbrowser en gaf ontwikkelaars de mogelijkheid tot manipulatie van HTML aan de browserkant: *client-side* dynamische websites. Omdat zowel de CGI-programma's als JavaScript slechts weinig informatie gebruiken voor de manipulatie of generatie van een enkele pagina worden deze programma's vaak *scripts* genoemnd [6]. Hoewel vandaag de dag de dynamiek stukken groter is en de programma's zijn uitgegroeid volwassen applicaties met grotere, complexere structuren wordt hier de naamgeving scripts aangehouden om verwarren met de webserver applicatie zelf te voorkomen.

2.1 Statische websites

Statische websites zijn websites die door de webserver precies op dezelfde manier worden aangeboden als dat deze is opgeslagen: de webserver haalt het bestand op van het opslagmedium (bijvoorbeeld harde schijf of geheugen) en verstuurt de inhoud hiervan naar de opvrager [4]. Er is verder geen enkele berekening nodig door de server. Ook websites die grotere bestanden ter download aanbieden, zoals DVD-bestanden, zijn statische websites.

Statische websites worden over het algemeen sneller dan dynamische naar de gebruiker teruggestuurd; er wordt immers geen enkele rekenkracht gevraagd van de server.

2.2 Client-side scripting

Client-side scripting is een techniek waarin een stuk programmacode, een *script*, het uiterlijk van een webpagina aanpast. Alle moderne browsers bieden hier ondersteuning voor. De programmataal wordt JavaScript of JScript genoemd. Deze code heeft toegang tot het *Document Object Model* (DOM) van de webpagina en kan deze aanpassen. Hiermee wordt (de weergave van) de website aangepast en wordt een vorm van dynamiek gecreeërd.

De code van het script wordt binnen het document zelf gedownload (*embedded*) of als los bestand (extern script). De gehele uitvoering van deze scripts ligt bij de browser.

2.3 Server-side scripting

Ook aan de kant van de webserver vindt *scripting* plaats. Hierbij wordt een aanvraag voor een webpagina niet afgehandeld door het ophalen van een bestand, maar is er enige rekenkracht nodig. In plaats het bestand op te halen wordt een programma aangeroepen dat wordt uitgevoerd op een bestand. Dit programma genereert de HTML-code voor de webpagina die vervolgens naar de opvrager wordt gestuurd. Het grote voordeel van server-side scripting is de mogelijkheid tot het ophalen en opslaan van gegevens op een centrale plaats. Daarom vindt dit ook vaak plaats in combinatie met een database. Nadeel is echter dat deze generatie veel meer processorkracht, geheugen en tijd nodig heeft. Dit kan bij veel gelijktijdige verzoeken tot prestatieproblemen leiden.

Enkele voorbeelden talen die gebruikt worden bij server-side scripting zijn Perl, PHP, JSP en ASP(.NET).

2.4 AJAX

Zowel *client-side* als *server-side scripting* zijn een handige manier voor het creëren van dynamische webpagina's. Echter, het gebruik van slechts één van deze technieken heeft nadelen voor de gebruiker. Deze ziet bijvoorveeld verouderde informatie (in het geval van client-side scripting), of heeft een verminderde gebruikerservaring in het geval van server-side scripting. De techniek genaamd AJAX (Asynchronous Javascript And XML) combineert deze twee en elimineert de start-stop-start-stop interactie die gebruikelijk is op het web [5].

De webbrowser kan op de achtergrond verbinden met de webserver en zo de laatste informatie ophalen, zonder dat de gebruiker dit ziet. Het client-side script wacht op antwoord van de server, ontvangt de gegevens en plaatst deze in de DOM. Het antwoord van de server kan statisch zijn, maar is veelal gegenereerd door een server-side.

Een bijkomend voordeel is dat webserver slecht bij het eerste verzoek een volledige pagina dient te genereren. De overige, asynchrone, verzoeken kunnen worden voldaan met een klein gedeelte van de pagina, de browser zorgt voor de verdere invulling van de pagina. Hoewel de naamgeving van de techniek doet denken dat het antwoord altijd in XML wordt gegeven, is het ook mogelijk dat er JSON of HTML terugkomt van de server.

3 Procesarchitectuur

De prestaties (en betrouwbaarheid) van een webserver worden zwaar beïnvloed door de onderliggende software architectuur [2]. Maar welke verschillende architecturen zijn er beschikbaar?

De architectuur van webservers kan op verschillende niveau's bekeken worden. Het procesmodel beschrijft de proces- en threadingmethode die gebruikt wordt [2]. Ontwikkelaars kunnen verschillende I/O (Input/Output) strategieën kiezen [7, 39]. Deze strategie kan er voor zorgen dat een proces of thread één of juist meerdere aanvragen tegelijkertijd kan behandelen. Ook is de hoeveelheid processen en threads (genaamd pool-size) van invloed op de prestaties van een webserver [2]. Dit is echter geen onderdeel van alle architecturen. Pool-size wordt dan ook maar gedeeltelijk meegenomen in het gehele verhaal maar wordt kort bediscussieerd waar dit relevant is.

Het gebruik van programmeer- of scriptingtalen op webservers is sinds een aantal jaren ook erg gebruikelijk. Deze talen kunnen op verschillende manieren in een webserver geladen worden. Dit kan gezien worden als een andere architecteur of architecteurkeuze. De meest gebruikte en relevante opties worden meegenomen in dit onderzoek omdat deze mogelijk invloed hebben op de prestaties.

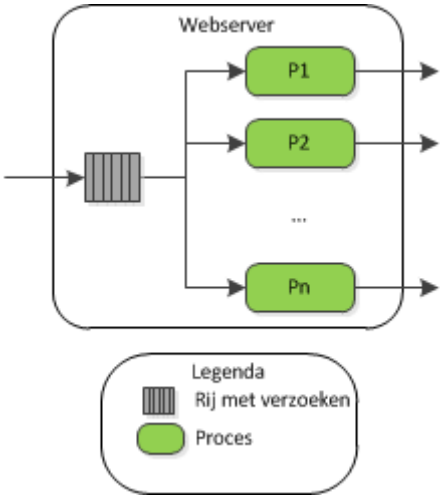
3.1 Procesmodel

Kijkend naar de procesmodellen van webserver software, dan kunnen er drie typen geïdentificeerd worden. Dit zijn een proces gebaseerd model, een model gebaseerd op threads en een hybride tussen deze twee [2, 8].

3.1.1 Proces gebaseerd model

Dit model maakt gebruik van meerdere processen met één enkele thread. De populaire Apache webserver maakt gebruik van dit model in versie 1.3 (in de module 'prefork') en in versie 2.x (in de 'multiprocessing prefork' module). Het gebruik van meerdere processen bevordert de stabiliteit van de software. Crasht één van de processen, dan heeft dit geen effect op de andere processen en blijft de software gewoon verder draaien.

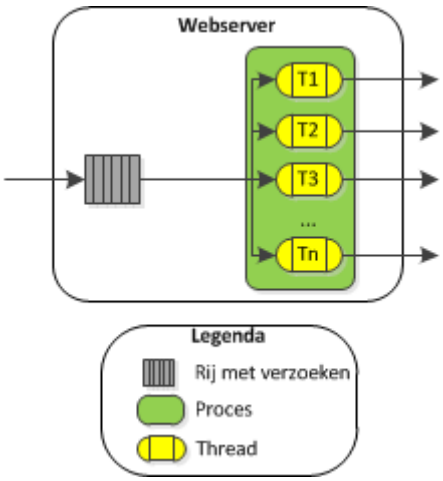
Bij standaard gebruik van I/O (synchrone blokkerende I/O, zie 3.2.1) heeft iedere verbinding met de webserver een proces nodig. Omdat ieder proces zijn eigen adresruimte nodig heeft in het geheugen, heeft dit tot gevolg dat de geheugeneisen aan een webserver van dit type erg hoog zijn. In het ergste geval lijdt dit tot een *overload* op de server door de vele geheugenoperaties die nodig zijn [2].



Figuur 3.1: Proces gebaseerd model [2]

3.1.2 Thread gebaseerd model

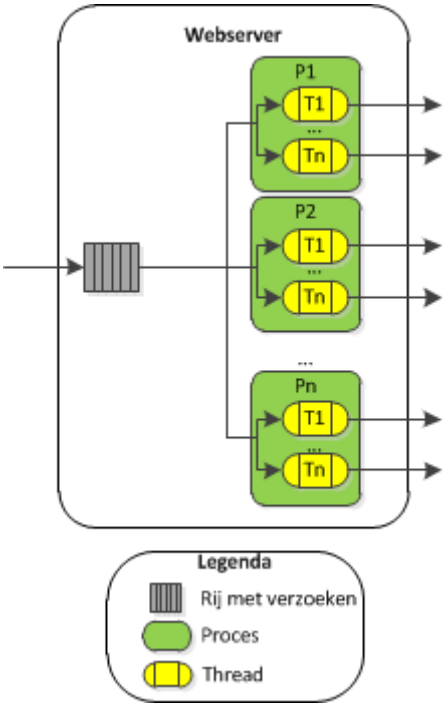
Gelijk aan het proces gebaseerde model, draagt in dit model één thread zorg voor één aanvraag tegelijk. Echter in dit model delen de threads van één proces de geheugenruimte. Een voordeel hiervan is dat bijvoorbeeld cachestructuren gedeeld kunnen worden, maar daarentegen kan het falen van een enkele thread de gehele software laten crashen. Daarnaast is het starten van een nieuwe thread veel efficiënter dan het *forken* van een proces omdat het besturingssysteem geen additionele geheugenoperaties hoeft uit te voeren [2].



Figuur 3.2: Thread gebaseerd model [2]

3.1.3 Hybride model

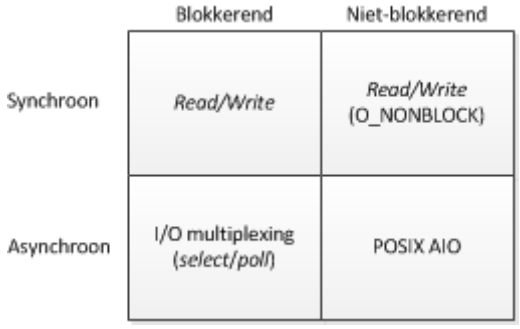
Het hybride model combineert het proces en thread gebaseerde model Dit resulteert in software met meerdere multithreaded processen, waarbij iedere thread een aanvraag per keer kan afhandelen [2]. De 'Worker MPM module' van Apache 2.X is hier een voorbeeld van. Het hybride model combineert de voordelen van beide modellen en verminderd hun nadelen. Binnen een proces delen threads geheugenruimte, maar een crashende thread haalt niet de volledige webserver naar beneden. Dit omdat op dat moment slechts één van de processen crasht. Vanwege de gedeelde adresruimte in het geheugen, kan een webserver gebruik makend van het hybride model met minder geheugen, hetzelfde aantal aanvragen afhandelen als een webserver van het proces gebaseerde model.



Figuur 3.3: Hybride model [2]

3.2 Input/Output

Zoals hierboven gezegd, kan een proces of thread traditioneel één connectie per keer afhandelen [39]. Dit door het gebruik van zogenaamde blokkerende (*blocking*) of synchrone (*synchronous*) aanroepen. Om zonder het verlies van prestaties meerdere connecties per proces of thread mogelijk te maken, dient een andere aanpak gekozen te worden. Deze wordt gebeurtenisgestuurd (*event-driven*) genoemd. Hierbij wordt gebruik gemaakt van niet blokkerende (*nonblocking*) of asynchrone (*asynchronous*) I/O [9]. In figuur 3.4 staan de verschillende I/O mogelijkheden binnen het Linux besturingssysteem [40] (het meest gebruikte besturingssysteem voor webservers [41, 42]) schematisch weergegeven. Deze modellen worden vervolgens kort besproken.



Figuur 3.4: I/O modellen binnen Linux [40]

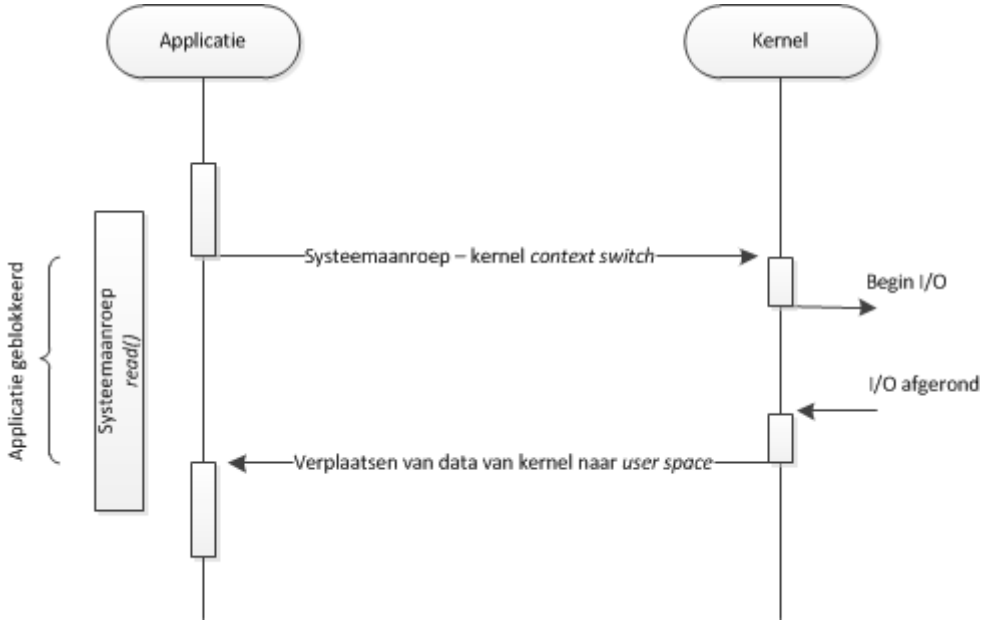
3.2.1 Synchrone blokkerende I/O

Een van de meest gebruikte modellen die wordt toegepast binnen applicaties is het synchrone blokkerende I/O model. De applicatie voert een systeemaanroep uit (*system call*) welke resulteert in een geblokkeerde applicatie. Dit betekent dat de applicatie wacht tot de systeemaanroep is afgerond. Dit is het geval als alle gegevens zijn overgestuurd of als er iets fout is gegaan. Op het moment dat een applicatie geblokkeerd wordt,

bevindt deze zich in een toestand waarin het geen rekenkracht van de processor gebruikt en slechts wacht op een antwoord. Vanuit de processorkant is dit dus een efficiënte manier van werken. Daarom heeft scheduler van de Linux 2.6 kernel een voorkeur voor processen die deze methode gebruiken. Dit omdat ander werk efficiënt verweven kan worden in het proces gedurende de blokkades [40].

Synchrone blokkerende I/O werkt als volgt: de applicatie voert een systeemaanroep uit (`read()` of `write()`), de applicatie blokkeert en er vindt een zogenaamde *context-switch* plaats naar de kernel. De aanroep wordt dan daadwerkelijk uitgevoerd en zodra het antwoord van het apparaat is gearriveerd, wordt de data verplaatst naar de *user-space buffer*. De applicatie wordt dan gedeblokkeerd en kan verder worden uitgevoerd waar deze gebleven was.

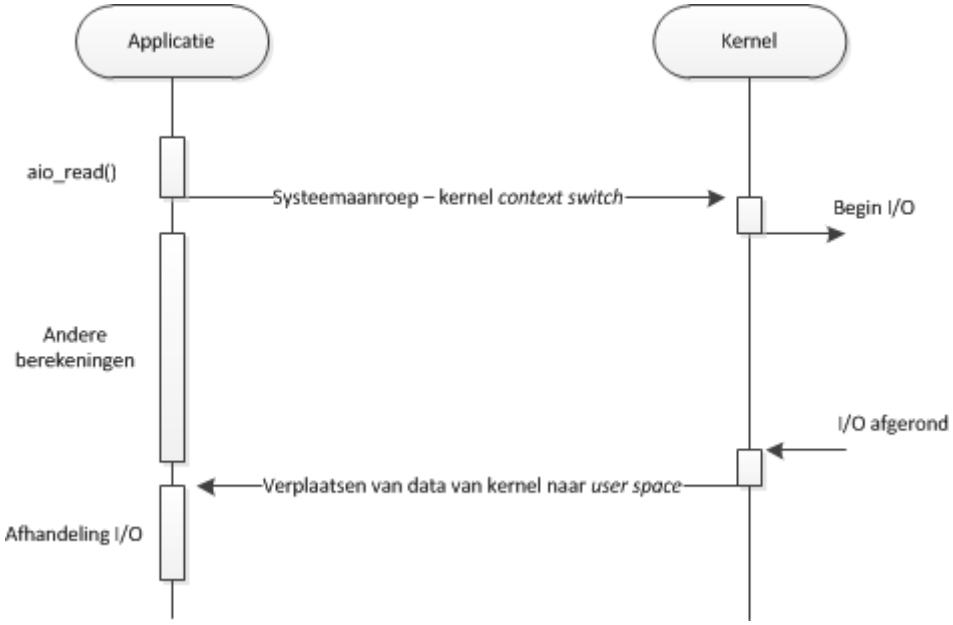
Dit ziet er schematisch als volgt uit:



Figuur 3.5: Synchrone blokkerende I/O [40]

3.2.2 Synchrone niet-blokkerende I/O

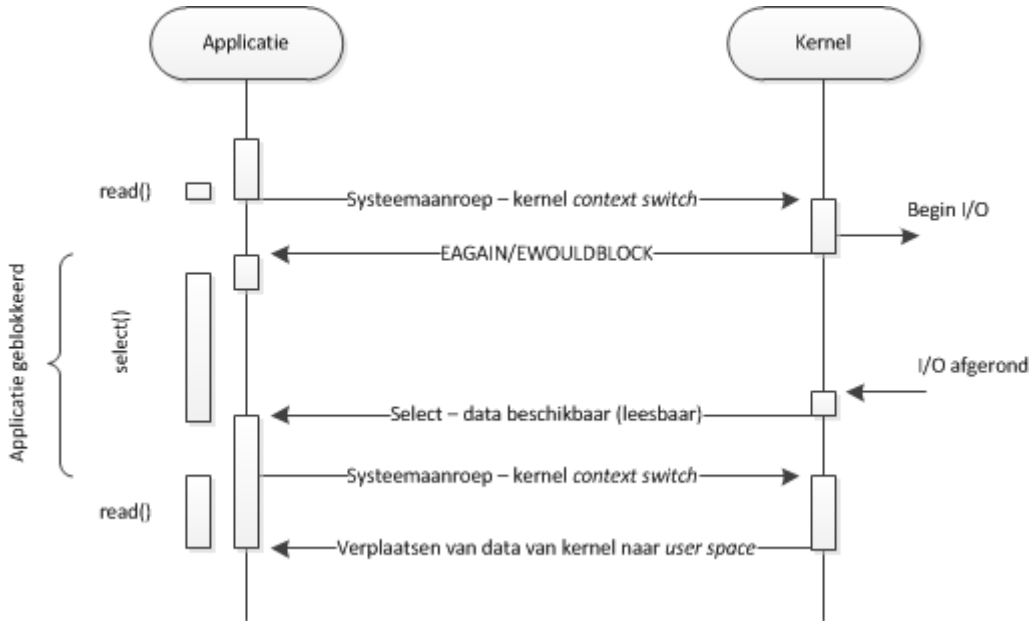
Een minder efficiënte variant van synchrone I/O is synchrone niet-blokkerende I/O. In dit model wordt een apparaat geopend als niet blokkerend. In plaats van een foutmelding geeft de kernel een melding terug dat de aanroep niet onmiddellijk afgerond kon worden (EAGAIN of EWOULDBLOCK). Door het niet blokkeren van de applicatie en het feit dat de I/O aanroep niet meteen kan worden afgerond, moet de applicatie aan de kernel blijven vragen of de aanroep is afgerond. Dit is erg inefficiënt, omdat de applicatie constant moet vragen of de aanroep is afgerond: het zogenaamde *busy waiting*. Verder zorgt deze methode voor enige vertraging in de I/O, omdat er een gat ontstaat tussen het beschikbaar worden van de data en een nieuwe systeemaanroep in de applicatie. Dit hele model is schematisch weergegeven in figuur 3.6.



Figuur 3.6: Synchrone niet-blokkerende I/O [40]

3.2.3 Asynchrone blokkerende I/O

Een derde model is asynchrone blokkerende I/O. Dit model maakt gebruik van blokkeernotificaties. De systeemaanroepen maken gebruik van niet-blokkerende I/O en de blokkerende select()-aanroep wordt gebruikt om te bepalen of de systeemaanroep afgerond is. Interessant aan de select()-methode is dat er veel informatie over een *descriptor* te verkrijgen is: mogelijkheid tot schrijven, beschikbaarheid van gelezen data en foutmeldingen. Deze blokkerende aanroep blijkt echter niet erg efficiënt te werken en wordt afgeraden voor I/O waarvan hoge prestaties vereist worden [40].

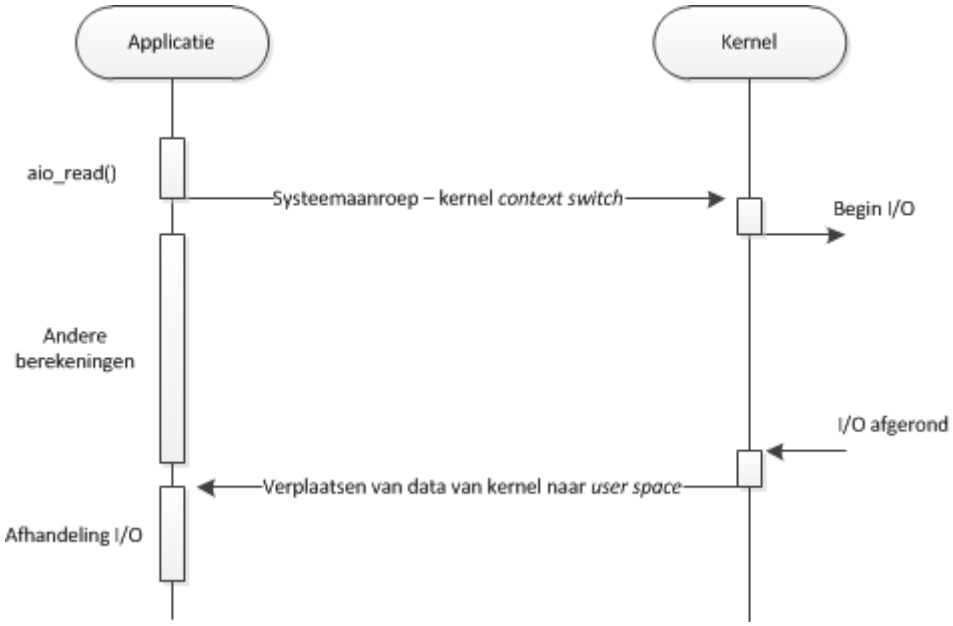


Figuur 3.7: Asynchrone blokkerende I/O [40]

3.2.4 Asynchrone niet-blokkerende I/O

Asynchrone niet-blokkerende I/O is een model waarin I/O en andere rekenoperaties elkaar overlappen. Een aanroep van `read()` keert onmiddellijk terug naar de applicatie, waarbij wordt aangegeven dat de leesoperatie succesvol is gestart. De applicatie kan op de achtergrond andere berekeningen uitvoeren terwijl de leesoperatie wordt uitgevoerd. Zodra het antwoord van de systeemaanroep arriveert kan met een signaal of *callback* de leesopdracht afgerond worden.

Deze manier van I/O is pas beschikbaar sinds Linux kernel 2.6 en wordt naar verwezen als AIO (Asynchronous non-blocking Input Output). Door de overlap tussen berekeningen en I/O voor een enkel proces (of thread), wordt het gat tussen reken- en I/O-snelheid gedicht. Terwijl er 1 of meerdere langzame I/O-aanvragen open staan, kan de processor voor andere doeleinden gebruikt worden. Dit kunnen ook eenvoudig zojuist afgeronde I/O-aanroepen zijn [40].



Figuur 3.8: Asynchrone niet-blokkerende I/O [40]

3.3 Server-side scripting

Een ander aspect wat een invloed heeft op de prestaties van een webserver zijn de dynamisch gegeneerde pagina's [1], zoals genoemd in 2.3 en 2.4. Ondersteuning voor deze toepassingen kan op verschillende manieren worden ingeladen in de webserver applicatie: via een module of via FastCGI. Deze manieren hebben invloed op de werking van de procesarchitectuur.

3.3.1 Module

Vele dynamische talen, zoals bijvoorbeeld PHP en Perl, kunnen door middel van een module in een webserver worden geladen. Of dit mogelijk is hangt van de taal en webserver af, omdat ondersteuning hiervoor vanuit beide kanten nodig is. De meeste moderne webservers bieden echter ondersteuning voor modules en veelgebruikte talen voegen deze ondersteuning snel toe. Een module wordt ingevoegd in de executie van de webserver en heeft dan toegang tot alle bronnen en middelen van de webserver [34]. Dit heeft echter wel invloed op het geheugengebruik, omdat elk proces de ondersteuning met zich meedraagt.

3.3.2 FastCGI

FastCGI is een methode die veel gebruikt wordt in veel verschillende webservers om de functionaliteit van een dynamische taal aan te bieden [35]. Het vormt een *interface* tussen een webapplicatie en de webserver. Een voordeel, en voor een stabiele webserver vereiste, is dat FastCGI er voor zorgt dat de taal wordt uitgevoerd in haar eigen geheugenruimte en dus niet het gehele proces kan crashen bij problemen. Ook biedt het de mogelijkheid tot een gedistribueerde omgeving, waarbij de applicatie op een andere machine wordt uitgevoerd dan de webserver zelf.

4 Model

Om de prestaties van de verschillende architecturen te kunnen bepalen, moet er een maat voor deze prestaties gevonden worden. Er zijn vele mogelijkheden. In de literatuur zijn talrijke voorstellen gedaan voor metingen aan de prestaties van webserver [1, 3, 10–13]. Dit hoofdstuk geeft een overzicht van de voorstellen uit de literatuur. Gecombineerd met eigen inzichten wordt een definitie van de prestatie van een webserver gegeven die gebruikt gaat worden. Er wordt hier onderscheid gemaakt tussen prestaties gemeten aan de kant van de server en aan de kant van de cliënt.

4.1 Server

Aan de kant van de server kunnen vele variabelen gebruikt worden om de prestaties van de server uit te drukken. Vanuit de literatuur wordt vaak het responsiepad hiervoor gebruikt. Dit is het pad dat de server doorloopt vanaf het ontvangen van het verzoek tot het uitgeven hiervan. TODO cite. Een methode die ook vaak naar voren komt is verwerkingscapaciteit van de server, de *throughput* [1]. Echter, in dit onderzoek worden het geheugen- en processorgebruik als maat genomen voor de prestaties aan de serverkant, aangezien gebleken is dat dit een grote bottleneck is bij het afhandelen van vele verzoeken door een webserver [3]. Immers, een processor kan slechts tot 100 procent benutting werken en er zal een wachtrij ontstaan als er meer taken dan rekenkracht beschikbaar is. als er geen

4.2 Cliënt

Een van de belangrijkste variabelen die genoemd wordt is de *request throughput*. Dit kan worden gemeten door HTTP verzoeken naar de server te sturen en te meten met welke snelheid de antwoorden aankomen [13]. Ook is het mogelijk het gehele afhandelen van het HTTP verzoek op te splitsen in delen en al deze delen te meten [14].

4.3 Model

Vanuit de literatuur wordt een prestatiemodel veelal opgesteld vanuit een wiskundige invalshoek. Kansrekening, stochastische processen en wachtrijtheorie [15,30]. Echter, dit onderzoek heeft als doel een beslismodel, of beslismatrix, op te stellen om systeemontwerpers handvatten te bieden bij de keuze voor een webserver.

Alvorens deze matrix op te stellen, wordt er eerst een keuze gemaakt uit de beschikbare combinaties van de procesarchitectuur. Zonder deze keuzes wordt de beslismatrix te groot en worden er keuzes aangeboden die in de praktijk niet bestaan.

4.3.1 Combinaties

Om een keuze te kunnen maken tussen de verschillende combinaties, dient er gekeken te worden welke combinaties in de praktijk bestaan. 3 procesmodellen, 4 I/O-modellen en 2 modellen voor de dynamische taal maakt een totaal van $3 * 4 * 2 = 24$ mogelijkheden.

Procesmodel	I/O-methode	Server-side scripting	Beschikbare software
Proces	Sync. blok. I/O	module	Apache MPM prefork [33, 43]
Proces	Sync. blok. I/O	FastCGI	Apache MPM prefork [33, 43]
Proces	Sync. niet-blok. I/O	module	
Proces	Sync. niet-blok. I/O	FastCGI	
Proces	Async. blok. I/O	module	
Proces	Async. blok. I/O	FastCGI	Nginx [38], [2]
Proces	Async. niet-blok. I/O	module	
Proces	Asyn. niet-blok. I/O	FastCGI	Lighttpd [44], Nginx [38],[1] [2]
Thread	Sync. blok. I/O	module	[4]
Thread	Sync. blok. I/O	FastCGI	Apache MPM worker [33, 45][3]
Thread	Sync. niet-blok. I/O	module	
Thread	Sync. niet-blok. I/O	FastCGI	
Thread	Async. blok. I/O	module	
Thread	Async. blok. I/O	FastCGI	[5]
Thread	Async. niet-blok. I/O	module	
Thread	Asyn. niet-blok. I/O	FastCGI	[5]
Hybride	Sync. blok. I/O	module	[4]
Hybride	Sync. blok. I/O	FastCGI	Apache MPM worker [33, 45]
Hybride	Sync. niet-blok. I/O	module	
Hybride	Sync. niet-blok. I/O	FastCGI	
Hybride	Async. blok. I/O	module	
Hybride	Async. blok. I/O	FastCGI	
Hybride	Async. niet-blok. I/O	module	
Hybride	Asyn. niet-blok. I/O	FastCGI	

[1] Mits AIO is ingeschakeld in de configuratie

[2] μserver [46] is ook een webserver gebaseerd op deze architectuur. Het is echter ontwikkelt als een simpele server voor prestatie experimenten gerelateerd aan webservers, besturingssystemen en implementatie [47]

[3] Alleen mogelijk met configuratie *StartServers* en *ServerLimit* op 1

[4] Het gebruik van threads in combinatie met een module wordt afgeraden [48]. Apache MPM worker [45] in combinatie met een module is dan op vele besturingssystemen ook niet te installeren.

[5] De KNOT webserver bevat deze architectuur. Ook deze webserver is echter puur op onderzoek gericht. Maakt voor asynchrone niet-blokkerende I/O gebruik van de Capriccio threading library die geen 1:1 thread-connectie model meer vereist [36]

Tabel 4.1: Architecturele combinaties en beschikbare software

Wat opvalt is dat er geen webservers bestaan die synchrone niet-blokkerende I/O geïmplementeerd hebben. Dit is te verklaren doordat dit model zorgt voor het zogenaamde *busy waiting*, zie 3.2.2. Ook combinaties tussen modellen die gebruik maken van threads (thread en hybride model) en asynchrone I/O komen niet voor. Ook dit is eenvoudig te verklaren: door het gebruik van asynchrone I/O is het nut van de thread, het afhandelen van een "extra" connectie, verdwenen.

Overige webservers

Een aantal webservers hebben gekozen voor een andere architectuur, namelijk embedding in de kernel van het besturingssysteem. TUX [16] en kHTTPd [17, 49] zijn hier voorbeelden van. Beide ondersteunen slechts statische webpagina's. Daarnaast hebben beide een andere structuur, omdat ze grotendeels in *kernel space* opereren.

Daarnaast zijn er natuurlijk nog vele andere webservers beschikbaar [50, 51], maar hun documentatie, gebruik en beschrijving in de literatuur is dusdanig summier dat deze verder niet genoemd worden.

4.3.2 Prestatiemodel per dynamisch karakter

In deze sectie wordt per geïdentificeerd dynamisch karakter van een website (zie 2) aan de hand van beschikbare modellen en literatuur bepaald welke combinatie van architectuurkeuzes uit 4.3.1 de beste prestaties biedt, op basis van de variabelen zoals hierboven bepaald in secties 4.1 en 4.2. Alleen geïmplementeerde opties worden in deze sectie besproken, aangezien de techniek achter webservers de afgelopen jaren een dusdanige ontwikkeling heeft doorgemaakt dat e

Statische websites en client-side scripting

Hoewel het voor de gebruiker heel anders over kan komen, voor een webserver zijn statische websites en websites die gebruik maken van client-side scripting gelijk. Immers, het uitvoeren van de code ligt geheel aan de kant van de browser.

Uit onderzoek is gebleken dat synchrone (blokkerende) I/O beter presteert voor kleine bestanden tot 50 Kb, terwijl asynchrone I/O betere prestaties levert bij grotere bestanden [12]. 50Kb is echter dusdanig klein in de huidige tijd waarin websites vaak bestaan uit grote scripts en *full-color* afbeeldingen, dat asynchrone I/O de voorkeur geniet. Asynchrone niet-blokkerende I/O zal sneller functioneren dan asynchrone blokkerende I/O vanwege het ontbreken van context-switches. Voor het afhandelen van vele connecties lijkt een hybride model het meest ideaal, door het gedeelde geheugen tussen de vele threads binnen een proces en de spreiding over meerdere processen. Echter, in de praktijk blijkt deze combinatie niet als webserver te bestaan (zie 4.3.1). De tweede optie is een variant met threads, omdat deze de grootste voordelen van het hybride model ook ondersteunt. Echter, ook deze webserver is in de praktijk niet beschikbaar. Slechts de experimentele, voor onderzoeksdoeleinden opgezette KNOT webserver heeft dit model geïmplenteerd. Er blijft nog slechts één enkel model over: het procesmodel. Dit is echter eenvoudig te verklaren, immers is er door het gebruik van asynchrone niet-blokkerende I/O de benodigdheid voor meerdere threads of processen verdwenen omdat één enkel proces de connecties nu zelf kan afhandelen [40].

Hoewel niet geheel relevant voor statische websites is het voor de volledigheid interessant om even te kijken naar het inladen van dynamische talen. In het procesmodel kunnen zowel de module als De enige optie voor dynamische talen is dus een FastCGI optie, die in de praktijk dus nooit aangeroepen zal worden.

Deze uitkomst, het procesmodel met asynchrone niet-blokkerende I/O en FastCGI komt overeen met bevindingen uit vergelijkbare onderzoeken, waarin Lighttpd bij veel gelijktijdige verbindingen de meeste verzoeken kon verwerken [35], zonder dat er gebruik gemaakt werd van extra eigenschappen van het HTTP protocol zoals `Keep-Alive`.

Server-side scripting

Websites gebaseerd op server-side scripting bestaan uit pagina's die volledig door de dynamische taal gegenereerd worden. Dit kan op een tweetal manieren: via een module of door middel van FastCGI. In het geval van FastCGI wordt er een extra proces gestart waar de webserver op moet wachten. In het eerste geval genereert de webserver zelf de pagina, waardoor het I/O model minder van belang is. Dit heeft dan ook de voorkeur. Er is slechts één combinatie mogelijk waarin de module fungeert, namelijk het procesmodel in combinatie met synchrone blokkerende I/O. Uit eerder onderzoek is gebleken dat dit inderdaad de meeste verzoeken af kan handelen [35].

AJAX

Een website gebaseerd op AJAX krijgt te maken met verschillende soorten verzoeken. Het eerste verzoek van een sessie vraagt een grote dynamische pagina en de bijbehorende statische bestanden op (afbeeldingen, stijldefinities en Javascripts). De volgende verzoeken bestaan voornamelijk uit verzoeken voor een simpele dynamische pagina die een kleine hoeveelheid nieuwe pagina-inhoud aan de webbrowser levert. Omdat de

webserver toch voornamelijk met dynamische pagina's te maken krijgt, lijkt een zelfde oplossing als voor server-side scripting hier het meest voor de hand te liggen.

Overzicht

Uit bovenstaande bevindingen kan eenvoudig een simpel beslismodel opgesteld worden:

Karakter	**Procesmodel**	**I/O methode**	**Server-side scripting**
Statisch	Proces	Asynchrone niet-blokkerende I/O	FastCGI
Dynamisch	Proces	Synchrone blokkerende I/O	Module
AJAX	Proces	Synchrone blokkerende I/O	Module

Tabel 4.2: Keuzemodel per dynamisch karakter

Wat hier opvalt is dat het oudste procesmodel, gebaseerd op meerdere processen hier op alle vlakken voorkomt. Ook de module geniet in het geval dat er dynamische inhoud gegenereerd dient te worden de voorkeur.

5 Validatie

Om in de praktijk te kunnen testen hoe een webserver presteert moet er een realistisch en reproduceerbaar aantal HTTP verzoeken naar de server worden gestuurd [13].

Dit hoofdstuk verkent deze opties om verzoeken te genereren en geeft een overzicht over hoe de variabelen uit het model gemeten kunnen worden. Deze opties worden tegen elkaar afgezet en er wordt een keuze gemaakt.

5.1 Verzoeken genereren

Op het web zijn verschillende applicaties te vinden die gebruikt kunnen worden om HTTP verzoeken te genereren. In onderzoeken met webservers wordt veelal gebruik gemaakt van httperf [13], Surge of Specweb99/Specweb2005 [18]. Er zijn echter veel meer opties [34]. Een groot aantal van deze opties wordt hier kort besproken en de belangrijkste eigenschappen en mogelijkheden van de applicaties worden aangestipt.

5.1.1 ab

ab [52] is een applicatie die bij de Apache webserver wordt geleverd om deze te benchmarken. Het geeft de systeembeheerder een beeld van de prestaties van de webserver, kijkend naar het aantal verzoeken per seconde dat de server af kan handelen. Via een bestand kan er HTTP POST data meegestuurd worden met de verzoeken. De op te vragen URL wordt als parameter meegegeven, wat er toe leidt dat de tool slechts 1 pagina kan opvragen tijdens het benchmarken.

5.1.2 hammerhead

Hammerhead [53] wordt omschreven als een “web site coverage, HTTP load generator, HTTP benchmarking, and stress testing tool". Het is ontworpen om meerdere gebruikers vanaf meerdere ipadressen te emuleren op maximale snelheid. Het is volledig instelbaar door middel van een configuratiebestand. Verder ondersteunt de applicatie verzoeken gebaseerd op scenario’s.

5.1.3 httperf

httperf [13] is een applicatie gemaakt voor het meten van de prestaties van een webserver. Het kan verschillende zogenaamde *workloads* genereren. De drie belangrijkste eigenschappen zijn het genereren en in stand houden van een server *overload*, HTTP/1.1 en SSL protocollen en uitbreidbaarheid met *workload* generatoren en prestatiemetingen. De applicatie ondersteunt scenario’s en POST data.

5.1.4 weighttp

Weighttp [54] (uitspraak: weighty) is een kleine, “lichtgewicht” benchmark applicatie voor webservers. Om klein en simpel in gebruik te zijn, ondersteunt het slechts een klein gedeelte van het HTTP protocol. Het ondersteunt meerdere threads om efficiënt gebruik te maken van moderne processoren met meerdere kernen

(*cores*) en asynchrone I/O voor meerdere verzoeken per thread. Het maakt gebruik van dezelfde argumenten als `ab`, maar ondersteunt er minder en biedt niet de mogelijkheid tot het meesturen van HTTP POST gegevens.

5.1.5 JMeter

JMeter is begonnen als een applicatie om web applicaties mee te testen, maar heeft inmiddels veel meer functionaliteiten, zoals FTP en database connectiviteit. JMeter kan overweg met statische en dynamische inhoud en kan hoge serverload simuleren. Het is in staat een grafische analyse van de prestaties te maken.

5.1.6 Sclient

Sclient [10], wat staat voor *Scalable Client*, maakt een groot aantal TCP-aanvragen in combinatie met lage TCP time-out tijden.

5.1.7 SPECweb2005

Specweb2005 [18, 19] is een software benchmark ontwikkeld door SPEC. Het is ontworpen om de mogelijkheden van een systeem met betrekking tot statische en dynamische webpagina's te testen. Het ondersteunt drie verschillende workloads: *banking*, *e-commerce* en *support*. Deze worden door SPEC gezien als de drie meest voorkomende scenario's.

5.1.8 Surge

Surge (Scalable URL Reference Generator) [20] is een applicatie die een lijst van statische URL aanvragen genereert, gebaseerd op de een aantal variabelen van de server.

5.1.9 Wagon

WAGON [34],Web trAffic GeneratOr and beNchmark, kan protocollen en webservers testen. Het gebruikt een model waarin de gebruikersessie zo nauwkeurig mogelijk wordt gegenereerd, door rekening te houden met wachttijden (*think time*). Ook maakt het gebruik van een navigatiemodel dat de populariteit van documenten in ogenschouw neemt.

5.1.10 Wasp

WASP [34] is ontworpen om een zogenaamd Wide Area Network (WAN) te emuleren door middel van *packet loss* en het vertragen van pakketten. Het was ontworpen om realistisch verkeer te genereren. Het is gebaseerd op Surge, maar houdt geen rekening met de denktijd van de gebruiker.

5.1.11 Webstone

Webstone [34] simuleert meerdere gebruikers. Dit gebeurt vanaf één of meerdere machines, om verzoeken naar de webserver te sturen. Het is een oud pakket en de standaard opstelling voldoet niet meer aan de huidige standaarden

5.1.12 Keuze

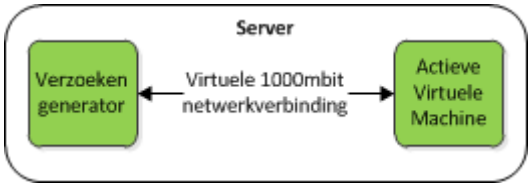
Hoewel het wel degelijk mogelijk is om de prestaties van een webserver te meten aan een enkel verzoek door het gehele verzoekpad te onderzoeken, geniet het de voorkeur om het gedrag van een gewone webbrowser en -gebruiker zo goed mogelijk te benaderen. Typisch wordt er bij het opvragen van een website een verzoek gedaan dat leidt tot meerdere verzoeken naar bestanden als afbeeldingen, tekstbestanden met daarin *javascripts* en *cascading style sheets*

5.2 Omgeving

De omgeving bestaat uit één machine die fungeert als cliënt en server. De server draait als virtuele machine met de webserver. De machine bestaan uit een AMD Phenom(tm) II X4 965 Processor 3.40 GHz, 6 GB RAM, 7200 rpm disk en 1000 mbit ethernetkaart. Het gekozen besturingssysteem is Ubuntu (11.10, Oneiric Ocelot, 64-bit), de desktop editie. Dit werd, na updates, geleverd met Linux kernel 3.0.0-15.

Er is voor gekozen om de machine ook als cliënt te laten fungeren om de netwerkverbinding geen enkele bottleneck te laten zijn. Omdat de machine over meerdere cores (4 in totaal) beschikt, en ruim voldoende geheugen bevat om zowel het gastbesturingssysteem als de virtuele machine te draaien, is dit geen enkel probleem. Daarnaast beschikt de processor van de machine over AMD-V, de virtualisatietechniek voor AMD-processoren.

Schematisch ziet dit er als volgt uit:



Figuur 5.1: Validatie-omgeving

5.2.1 Virtuele machine

Omdat het complete model gevalideerd dient te worden, en deze vele combinaties bevat, is er gekozen voor een opzet met een virtuele omgeving. Door het gebruik van een virtuele omgeving kunnen de omgevingen grotendeels gelijk gehouden worden en is het eenvoudig de testen te herhalen waar nodig. Binnen de literatuur is verschillende opties besproken voor virtualisatie. In deze sectie worden deze en andere opties kort verkend.

COTSon

COTSon [21] is een infrastructuur die volledige systeem emulatie uit kan voeren. Het is ontwikkeld door HP labs om volledige computersystemen, van multicore machines tot complete clusters, na te bootsen. De functionaliteit is gebaseerd op SimNow van AMD en beschikbaar voor 32 (`x86`) en 64 bits (`x86_64`) platformen.

gem5

Gem5 [55] is een modulair platform voor computerarchitectuur onderzoek. Het wordt met name gebruikt in het onderzoek en de applicatie is in staat een volledig systeem te emuleren.

Simics

Simics [22] is een platform voor volledige systeemsimulatie. Het zoekt een balans tussen prestaties en nauwkeurigheid, dit door een balans te vinden tussen abstractie en functionaliteit. Het kan commerciële *workloads* aan en ook de timing modeleerd het hardware model voldoende. Aan Simics zijn enkele voorwaarden verbonden voor het (gratis) gebruik.

SimOS

SimOS [23] is een systeemsimulator. Het stond aan de basis van het hierboven beschreven Simics. Het draait op MIPS en Alpha processoren.

Qemu

Qemu [24] kan verschillende processoren (x86, PowerPC, ARM and Sparc) emuleren op verschillende gastprocessoren x86, PowerPC, ARM, Sparc, Alpha and MIPS). Het ondersteunt volledige systeememulatie waarin een compleet en onaangepast besturingssysteem kan draaien als een virtuele machine. Daarnaast kan het ook programma's die gecompileerd zijn voor een bepaald type processor, uitvoeren op een andere processor.

VirtualBox

Virtualbox [56] is een virtualisatieprogramma. Dit maakt het mogelijk een besturingssystemen te draaien binnen een (ander) besturingssysteem. Het is erg uitgebreid, bevat verschillende controle-applicaties en verschillende versies met bijbehorende licenties.

VMWare

VMware [25, 26] is een virtueel machine platform dat een abstractie van x86 hardware levert, zodat een besturingssysteem binnen een ander besturingssysteem kan draaien.

5.3 Metingen

Ook op de server dienen metingen verricht te worden. Enkele beschikbare applicaties worden hier genoemd:

5.3.1 ps

`ps` geeft informatie weer over de actieve processen van een systeem [31]. Deze informatie omvat onder andere status, gebruiker, processortijd en geheugengebruik.

5.3.2 process accounting

De linuxkernel is in staat om informatie over de processen op te slaan. Het gaat hier over I/O tijden en CPU tijden van uitgevoerde programma's. Na afsluiting van het programma worden deze tijden toegevoegd aan het logbestand. Accounting wordt in de literatuur ook genoemd als éń van de mogelijkheden om server prestaties te meten [31]

5.3.3 free

Het programma `free` kan gebruikt worden om de hoeveelheid vrij en gebruikt geheugen van het systeem te tonen.

5.3.4 sysstat

Het linuxpakket `sysstat` levert onder andere het programma `vmstat` wat het geheugengebruik in kaart brengt [31]. Door middel van parameters is het mogelijk *vmstat* op een bepaald interval te laten rapporteren over het virtuele geheugen. Door middel van mpstat kan er informatie over het processorgebruik getoond worden [31]. Het kan onder andere systeemtijd, gebruikerstijd en I/O wachttijd laten zien.

5.4 Uitvoering

Er is gekozen voor een Virtualbox omgeving, waarbij op de virtuele machine gebruik wordt gemaakt van `sysstats` `vmstat` voor het geheugengebruik. Dit wordt elke seconde gerapporteerd in een bestand dat na afloop van de test naar de cliënt wordt gehaald voor analyse. Het processorgebruik van de processen wordt door middel van Linux process accounting bijgehouden. Na afronding van de test en stoppen van de webserver wordt deze uitgelezen en opgeslagen. Ook dit wordt voor analyse naar de cliënt gekopieerd. Virtualbox is eenvoudig te installeren middels apt-get, maakt gebruik van de virtualisatie-techniek van de processor en biedt uitgebreide configuratie van de virtuele machine middels de grafische interface. Daarnaast biedt het meegeleverde `VBoxManage` de mogelijkheid om via de *shell* virtuele machines te starten. Dit bevordert automatisch en herhaald testen. Ook deze virtuele machine is voorzien van een 64-bit Ubuntu variant, de server editie van 11.10 Oneiric Ocelot. Deze draait kernel 3.0.0-12-server en is voorzien van een Ext4-filesystem. De virtuele machine heeft 2 GB geheugen toegewezen gekregen, alsmede 1 processor. Als dynamische taal is gekozen voor PHP, veruit de meest gebruikte voor dynamische websites. Nginx en Lighttpd maakten gebruik van PHP-FPM omdat zij in deze configuratie de beste prestaties moeten kunnen leveren, Apache maakt gebruik van PHP-CGI, de standaardmanier om PHP in Apache te gebruiken in combinatie met FastCGI. Voor architectuurcombinaties waar meerdere beschikbare software is genoemd in tabel 4.3.1 is de eerstgenoemde gekozen.

Er moet wel aangetekend worden dat deze omgeving haaks staat tegenover vele opzetten uit de literatuur. Hierin wordt vaak gesproken over realistische situaties waarin ook *delay* en *packet loss* worden nagebootst [27]. Het is echter de opzet om de piekbelasting van de servers te bekijken, zonder rekening te houden met netwerkverlies. Verder wordt er vaak gekozen voor een simulatie-omgeving die uitgebreide statistieken kan weergeven over het systeem, zoals het L2-cache.

De gekozen benchmark software is hammerhead. Deze software is in staat om binnen een gestelde tijd zo vaak mogelijk een scenario af te werken. Deze scenario's zijn volledig configureerbaar en hebben de mogelijkheid HTTP POST-gegevens te bevatten. Ook biedt het uitgebreide statistieken na afronding van de test. Hammerhead is geconfigureerd (zie B.1) om 256 gelijktijdige connecties te starten. Dit omdat dit het maximale aantal gelijktijdige connecties is dat Apache aan kan in de standaard configuratie. Hammerhead voert gedurende 60 tot 300 seconden, de optie die via een parameter wordt meegegeven (zie A.1.4 zo veel mogelijk verzoeken uit op de webserver.

Voor elke gekozen webserver (Zie 4.3.1) is een virtuele machine aangemaakt als kopie van een template. De *clone* functionaliteit van Virtualbox is hiervoor toegepast zodat alle virtuele machines identiek zijn. De gekozen webservers zijn middels de standaard pakketten van Ubuntu geïnstalleerd en de meegeleverde configuratie is slechts aangepast om dynamische websites middels FastCGI of module te maken. Deze eenvoudige aanpassingen aan de configuraties zijn uitgebreid beschreven op het Internet. Deze wijzigingen hebben verder geen invloed op het gedrag van de webserver.

5.4.1 Dynamisch karakter

Voor elk dynamisch karakter is er gekozen voor kleine website die voldoet aan de eigenschappen zoals beschreven in hoofdstuk 2.

De website met een statisch karakter wordt gesimuleerd door een 6-tal bestanden: 1.html, 2.html, 1.css, 1.js, 1.png en 2.png. Deze bestanden bevatten allen volledig willekeurige gegevens (zie A.3). Hoewel een

browser deze bestanden niet kan interpreteren, is het voor een benchmark applicatie als hammerhead geen enkele probleem. Overigens zijn deze bestanden eenmaal gegenereerd en vervolgens op de virtuele machines geplaats; alle webservers bieden dus exact dezelfde bestanden aan.

DokuWiki [57] is de basis voor de website met dynamisch karakter. Er is voor dit pakket gekozen omdat het redelijk uitgebreide code omvat, maar geen database zoals MySQL verlangt. Een simpele Hello World pagina is aangemaakt op de startpagina. De geïnstalleerde versie is laatste stabiele versie: 2011-05-25a genaamd "Rincewind".

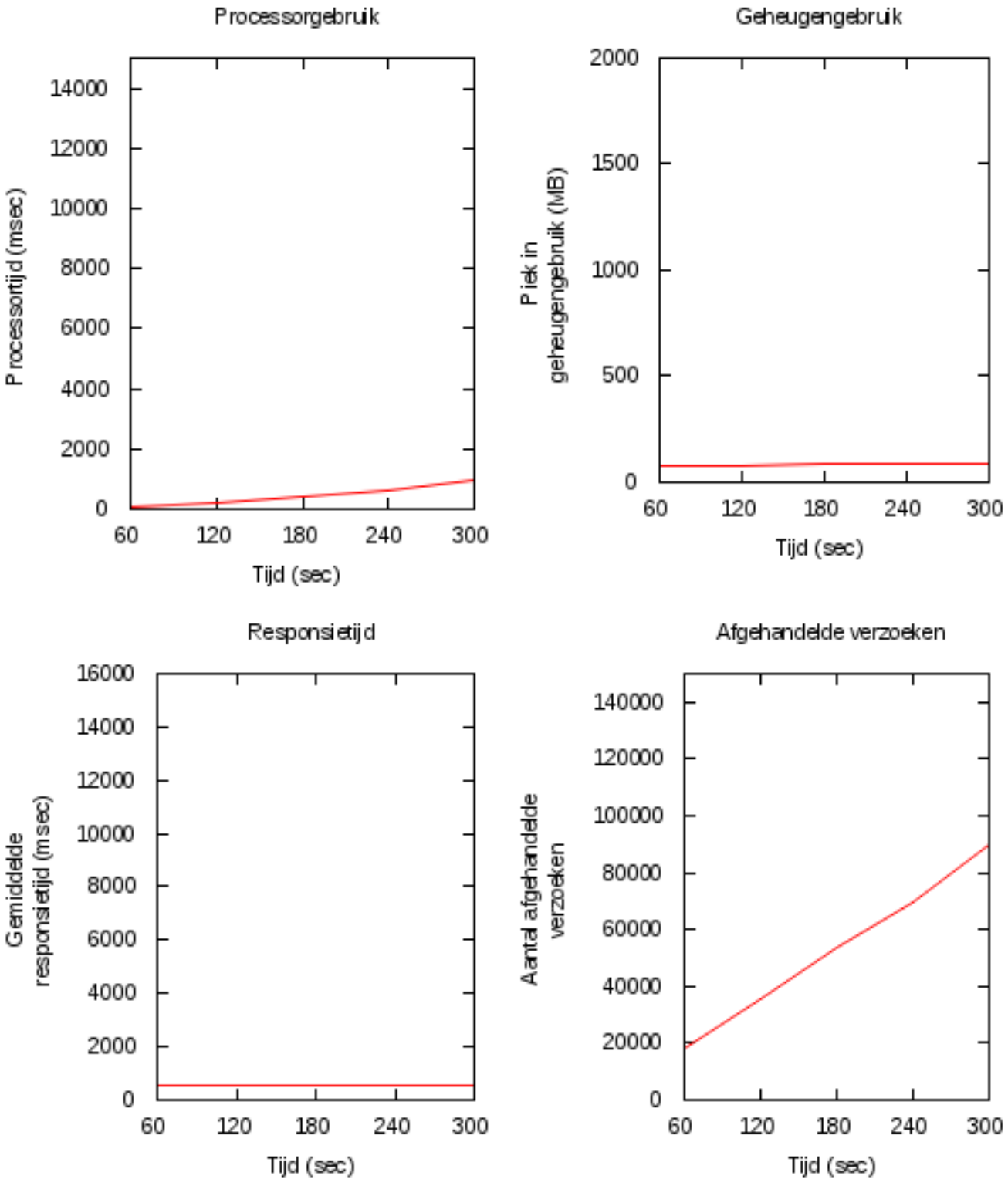
Ook voor een website met dynamisch karakter op basis van Ajax is DokuWiki de basis. Er is hier echter 1 bestand aan toegevoegd, ajax.php. Dit simpele bestand (zie A.4.1) geeft aan de hand van wat HTTP POST variabelen een JSON string terug.

Er is er voor gekozen om een vijftal testen uit te voeren per combinatie dynamisch karakter en webserver. Hammerhead voert, zoals al gezegd, 1 tot 5 minuten zo veel mogelijk verzoeken uit. Elke test wordt drie maal uitgevoerd en van deze resultaten wordt een gemiddelde berekend. De code van het geautomatiseerd uitvoeren van de testen is te vinden in Appendix A.1.4

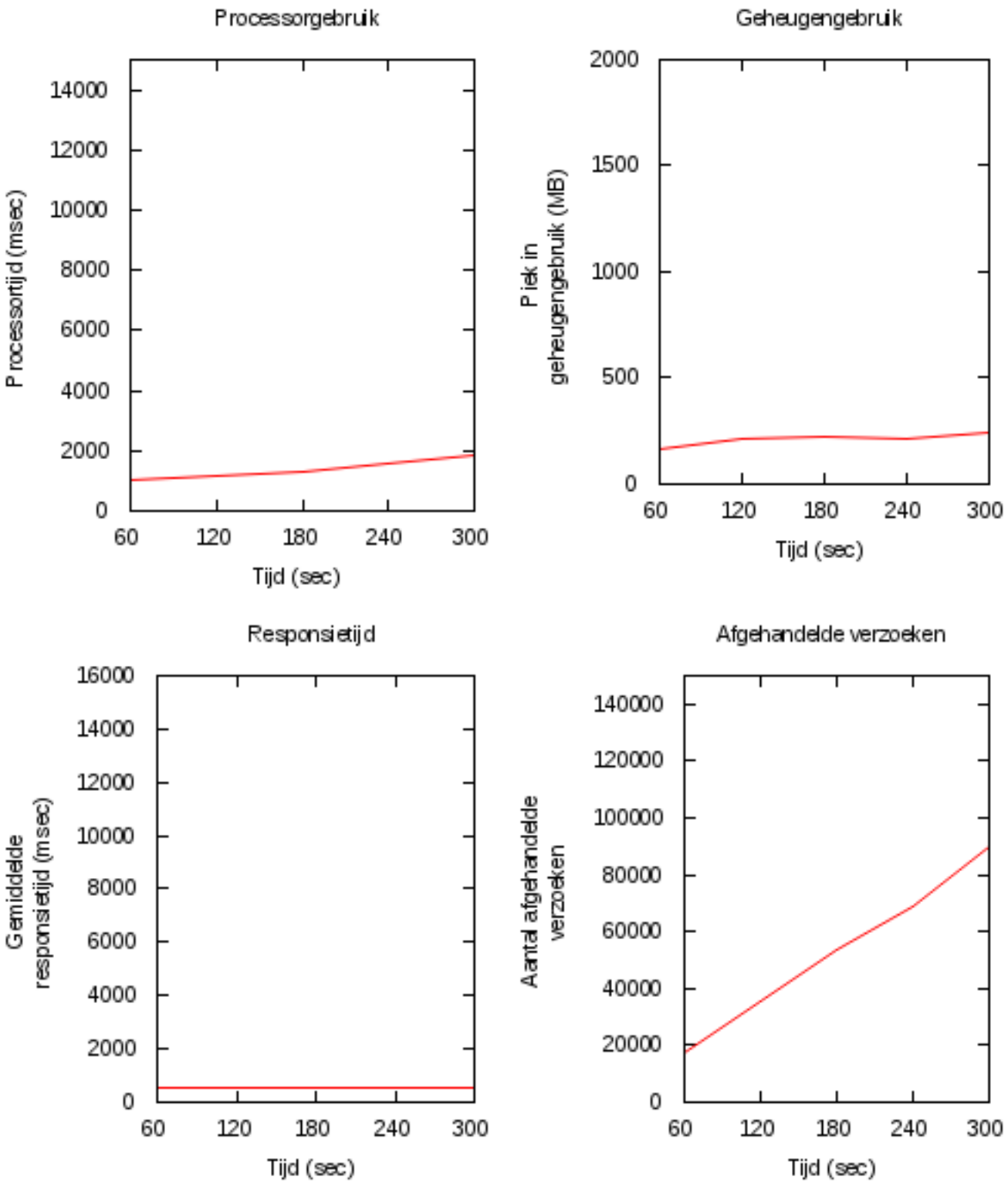
5.5 Resultaten

Door middel van *gnuplot* zijn de resultaten in grafieken geplaatst. Uit de grafieken zijn de verschillende prestaties eenvoudig af te lezen. De verschillende webserver worden per dynamisch karakter apart en in samengevoegde grafieken getoond.

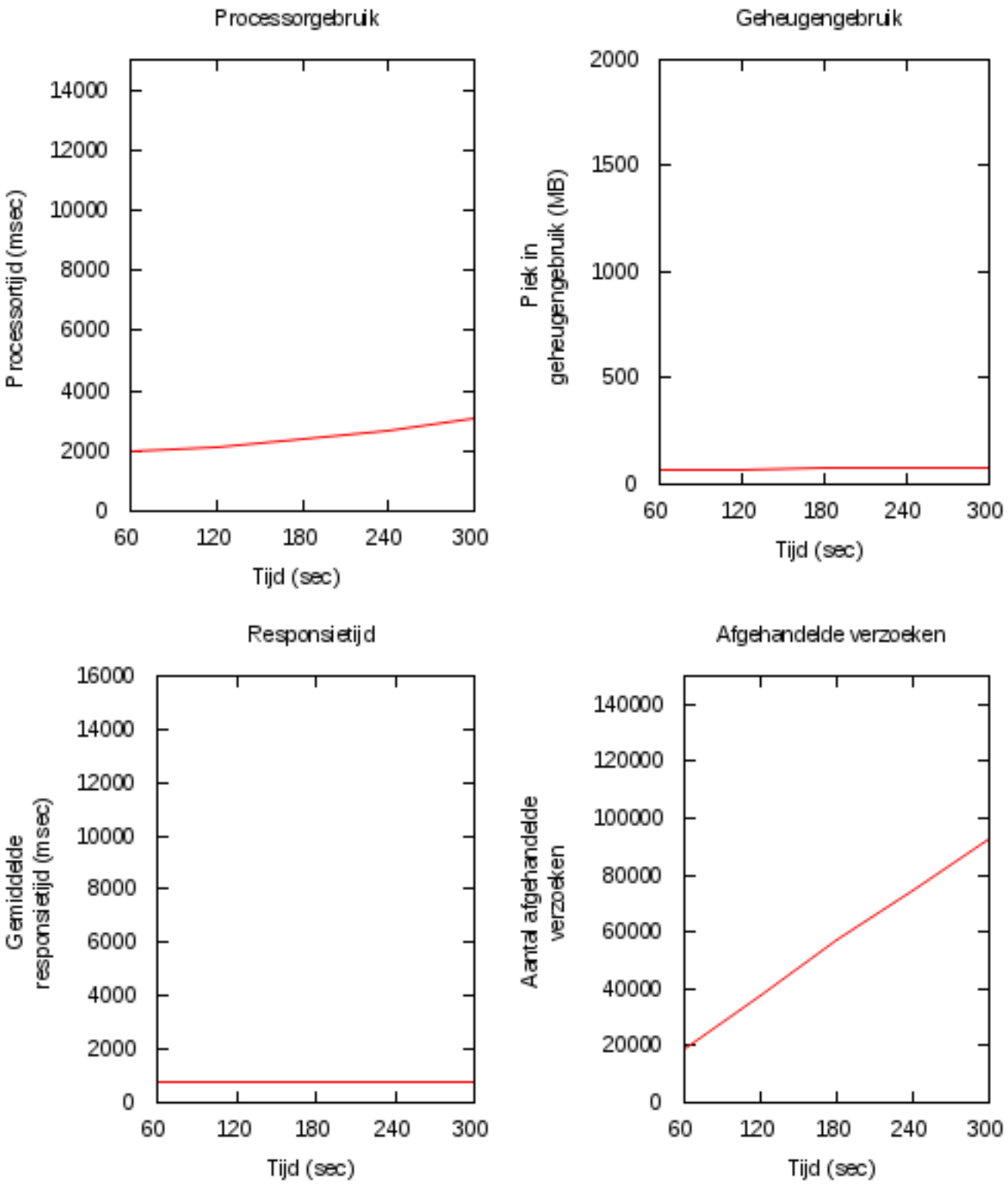
5.5.1 Statisch karakter



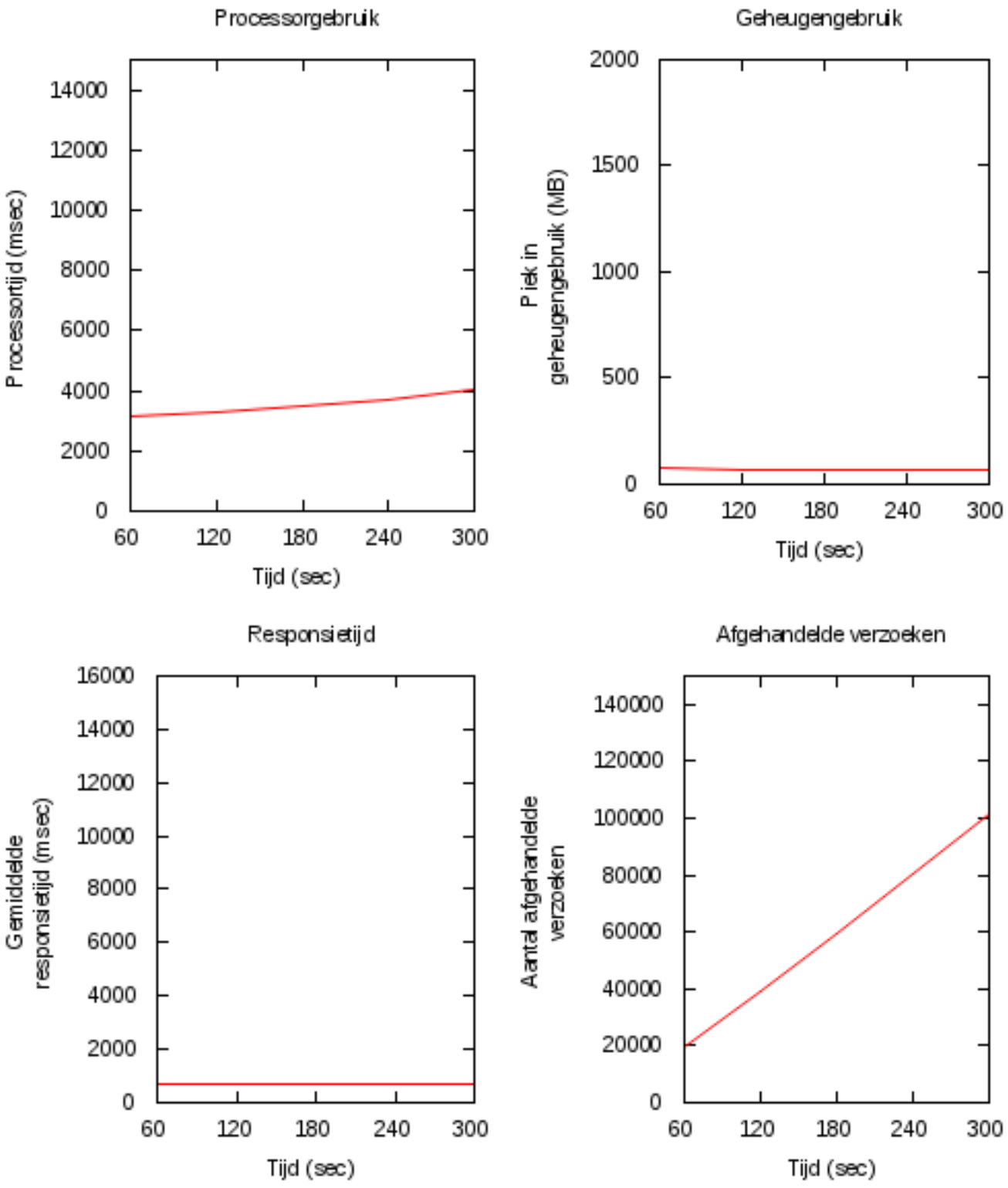
Figuur 5.2: Apache prefork met FastCGI



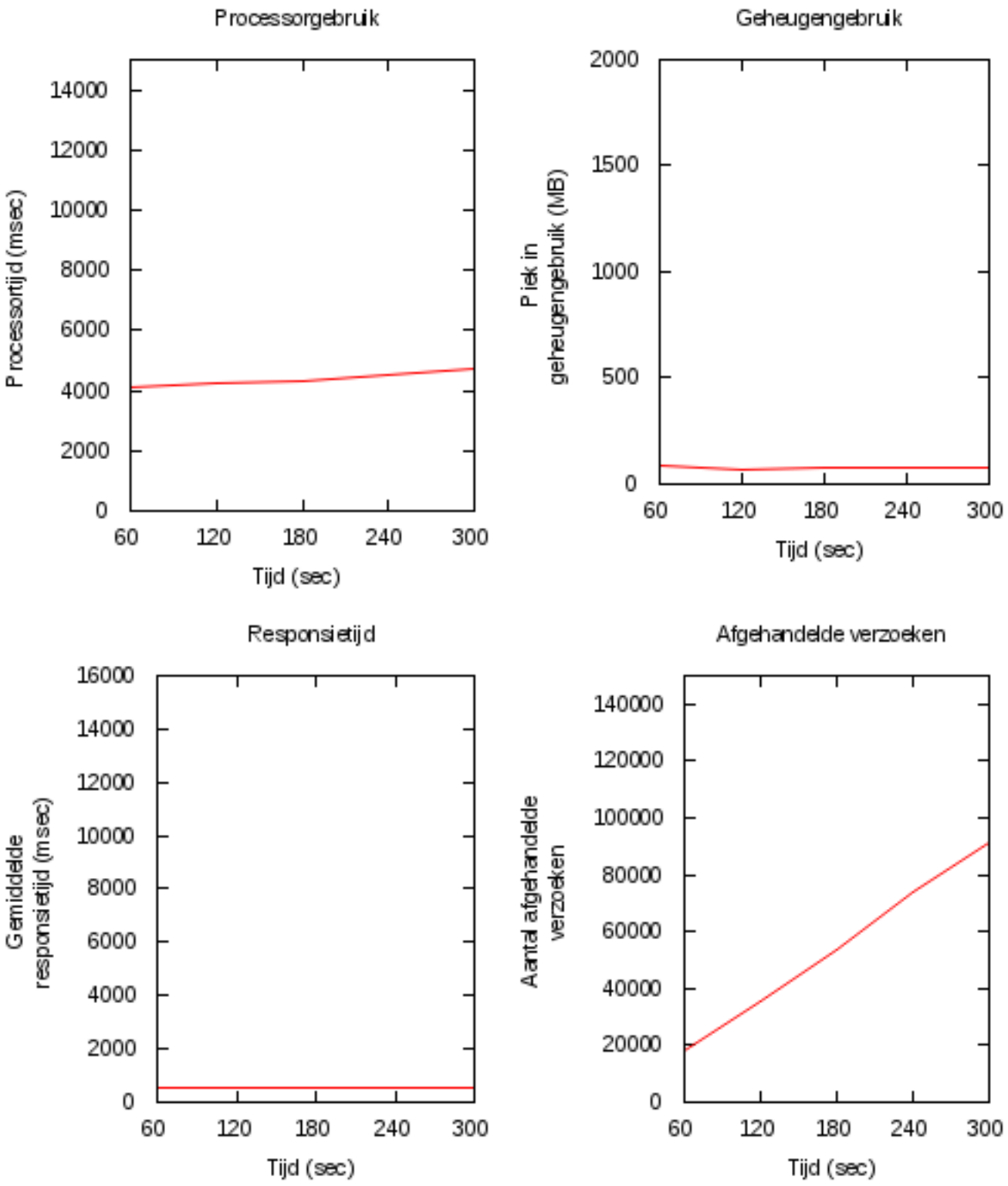
Figuur 5.3: Apache prefork met module



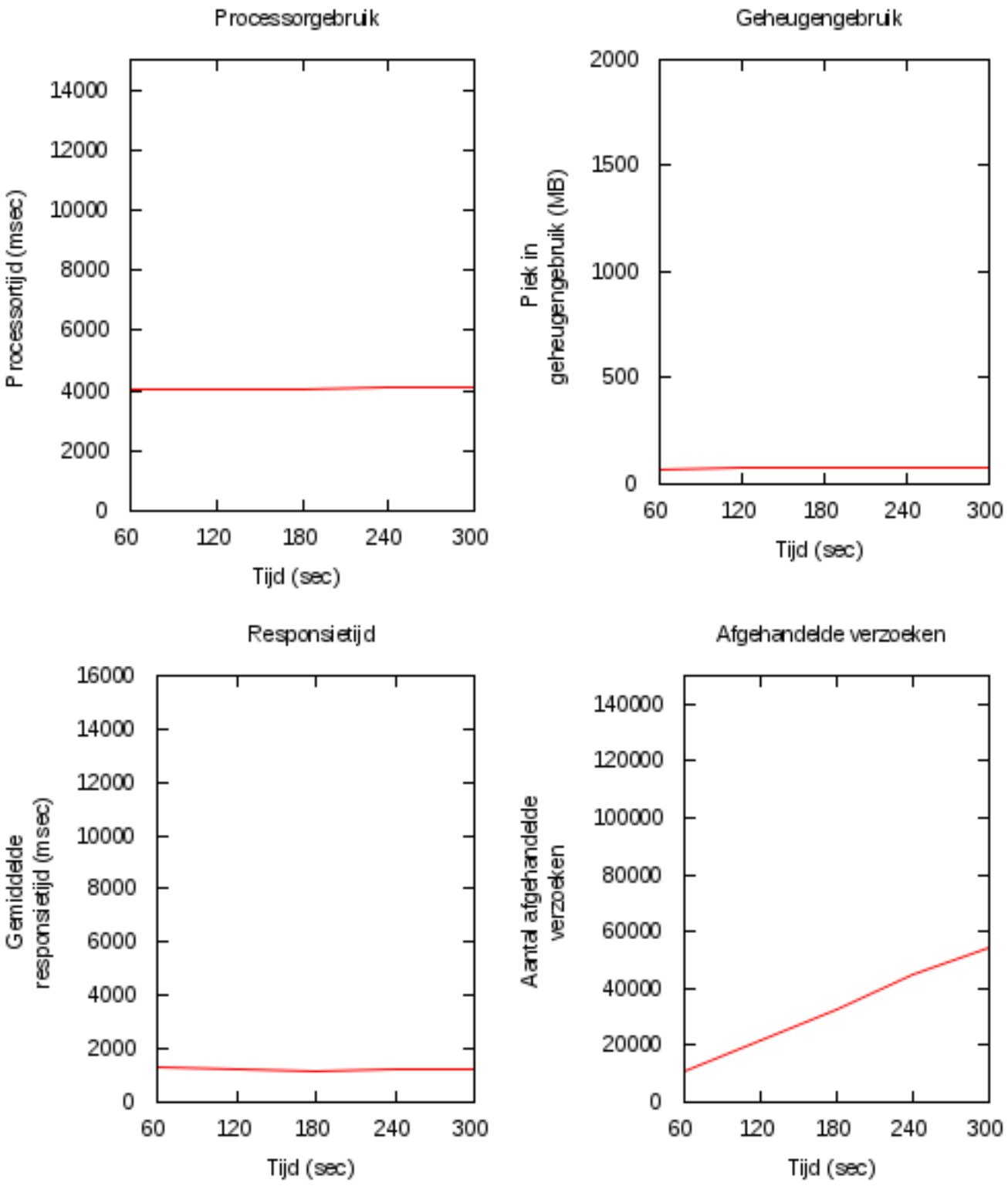
Figuur 5.4: Nginx



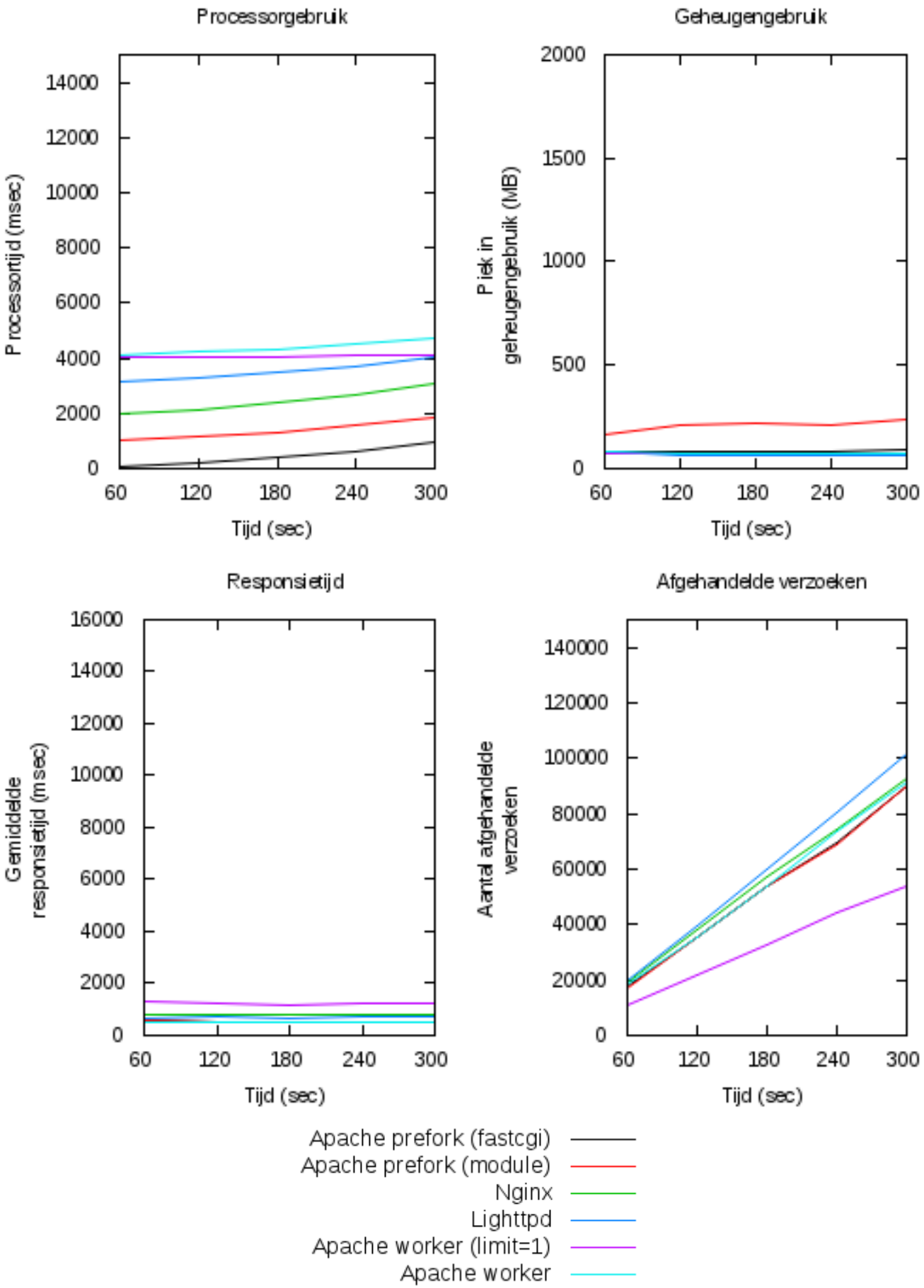
Figuur 5.5: Lighttpd



Figuur 5.6: Apache worker

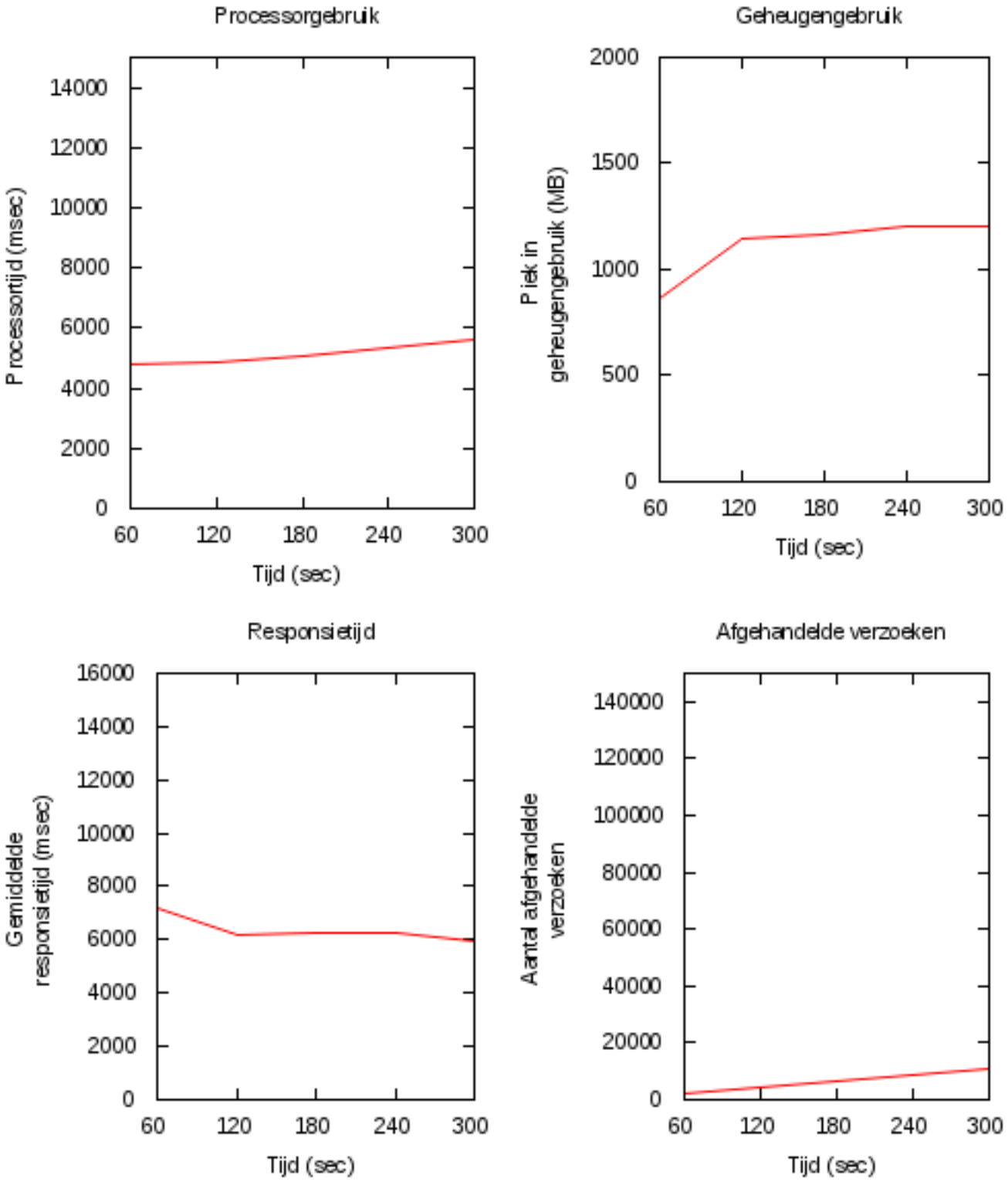


Figuur 5.7: Apache worker met limit=1

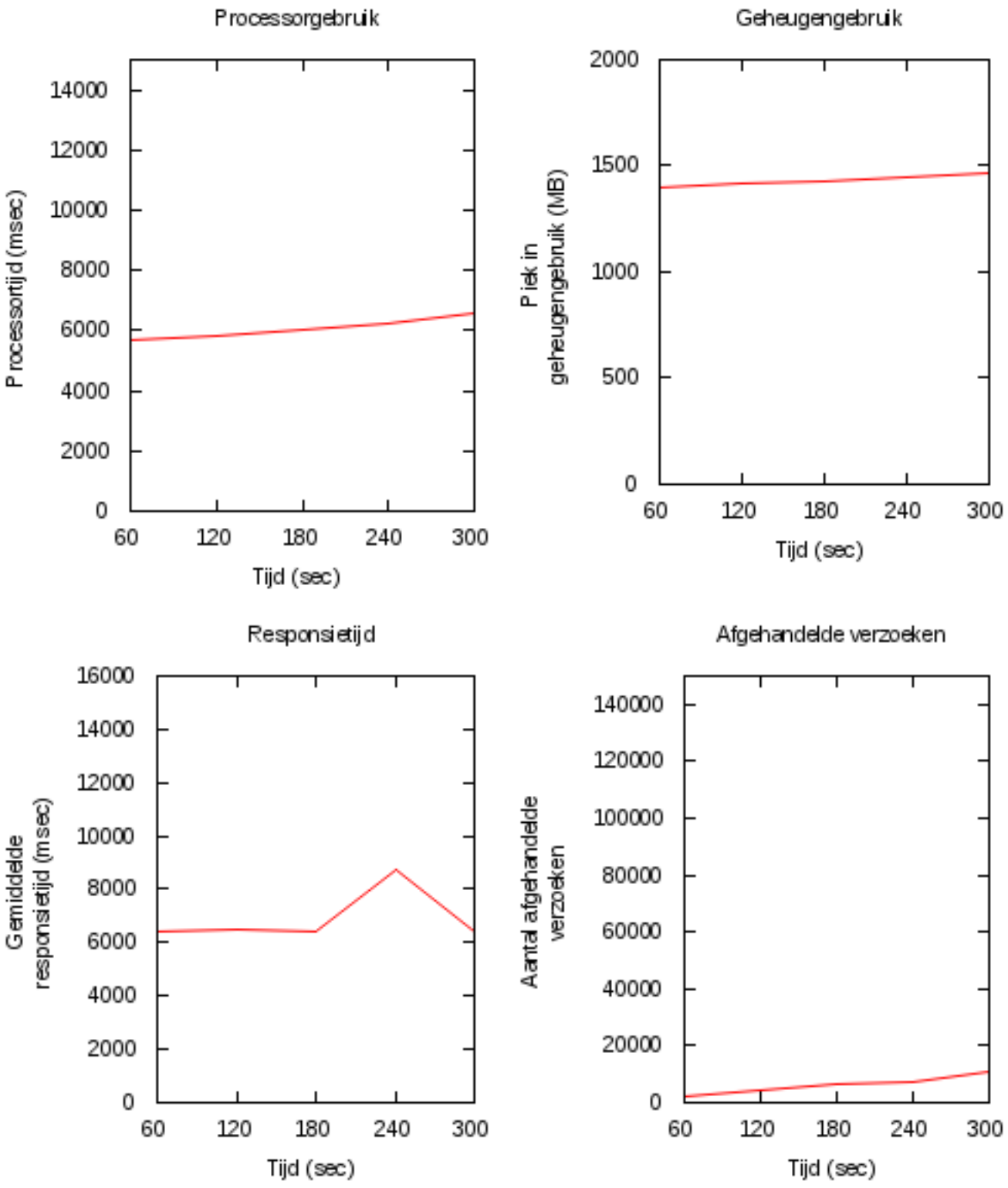


Figuur 5.8: Totaaloverzicht

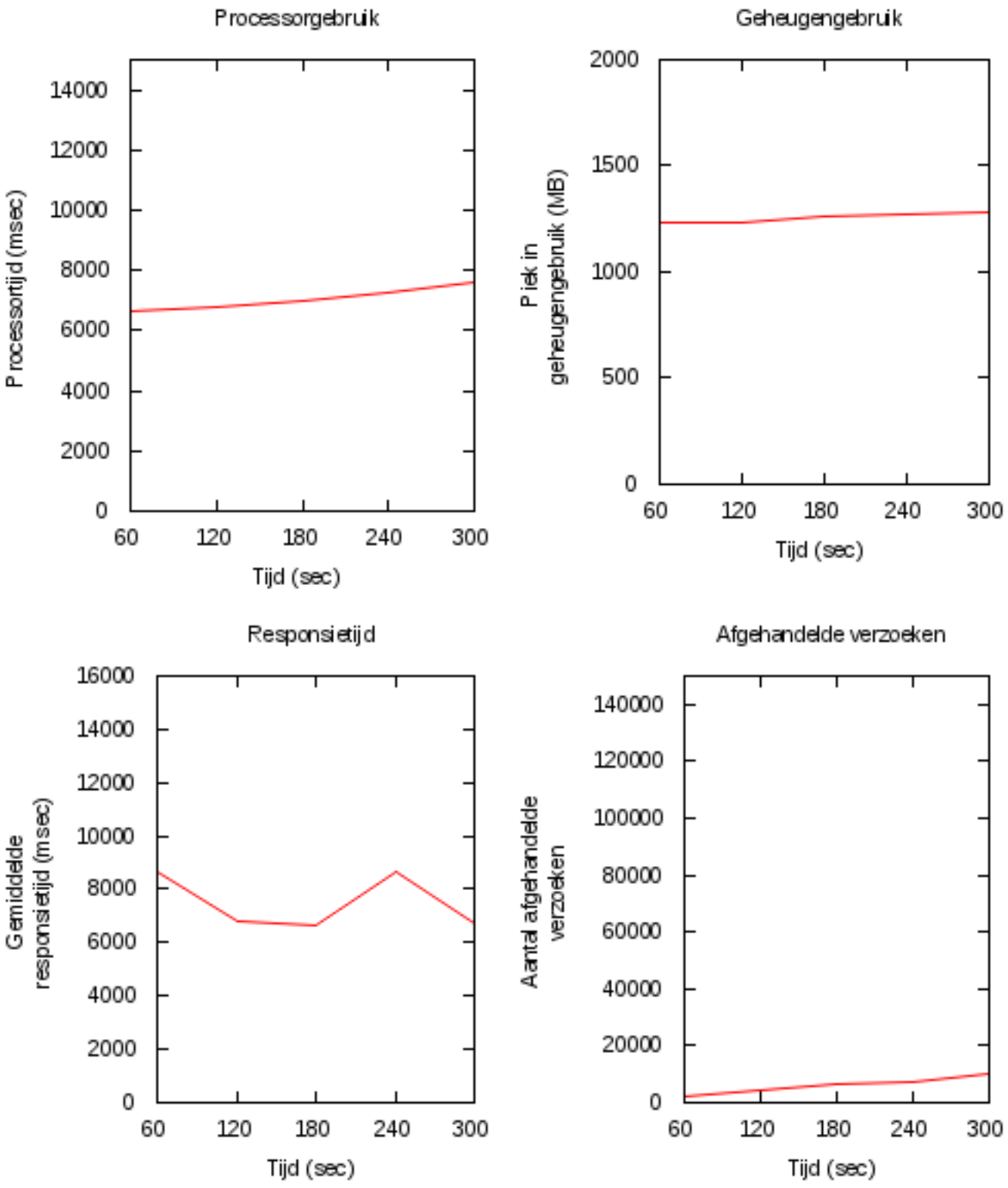
5.5.2 Dynamisch karakter



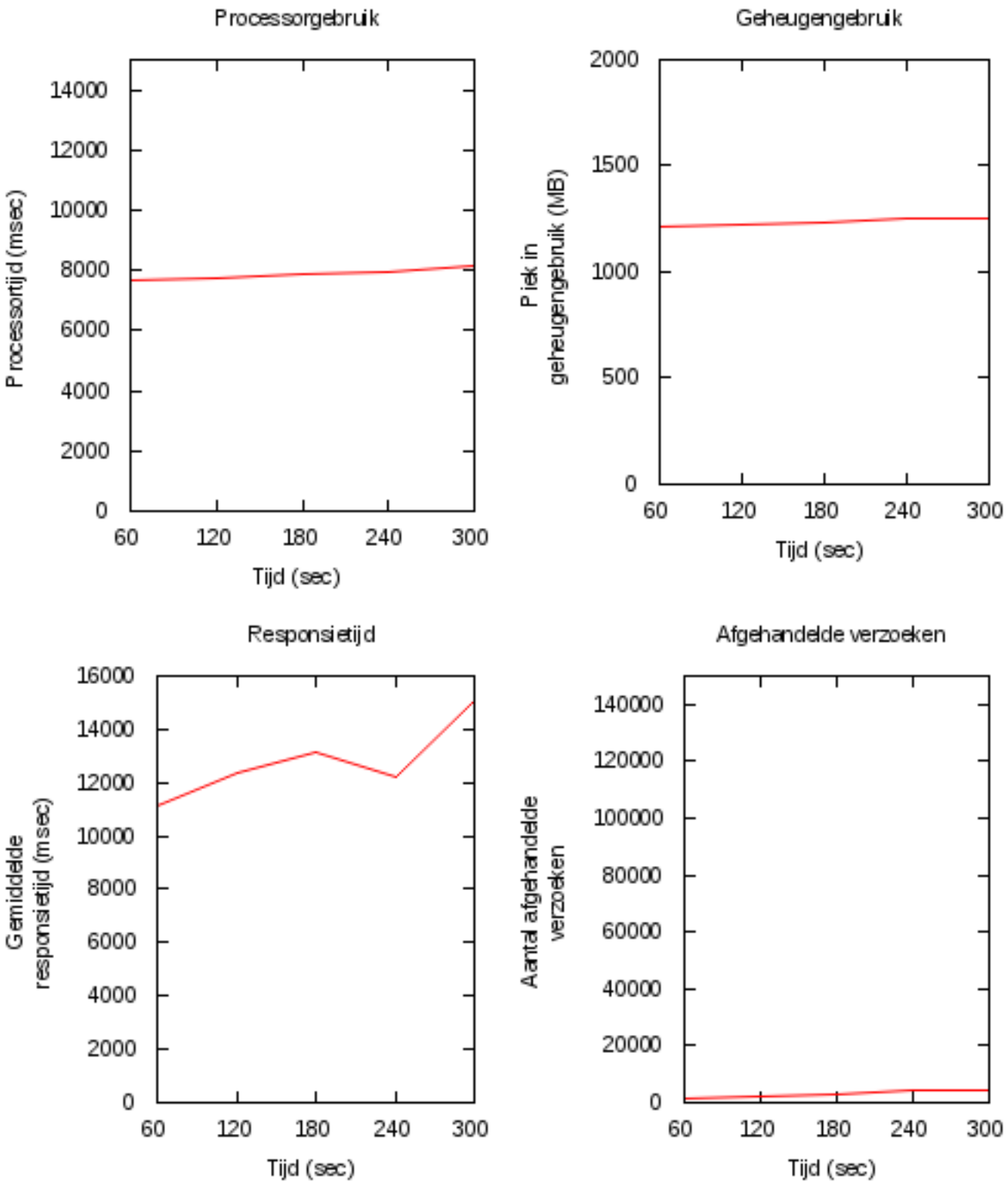
Figuur 5.9: Apache prefork met FastCGI



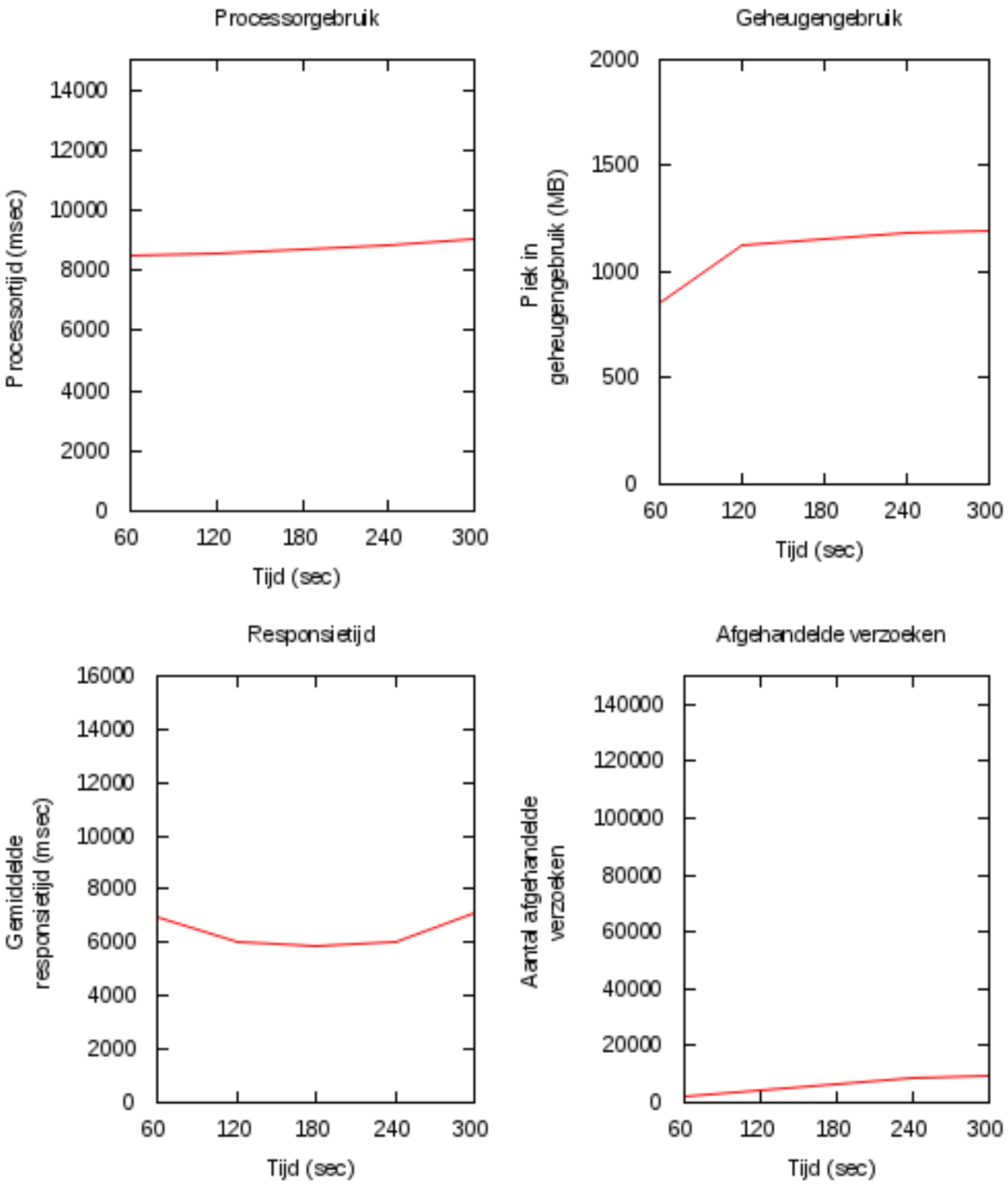
Figuur 5.10: Apache prefork met module



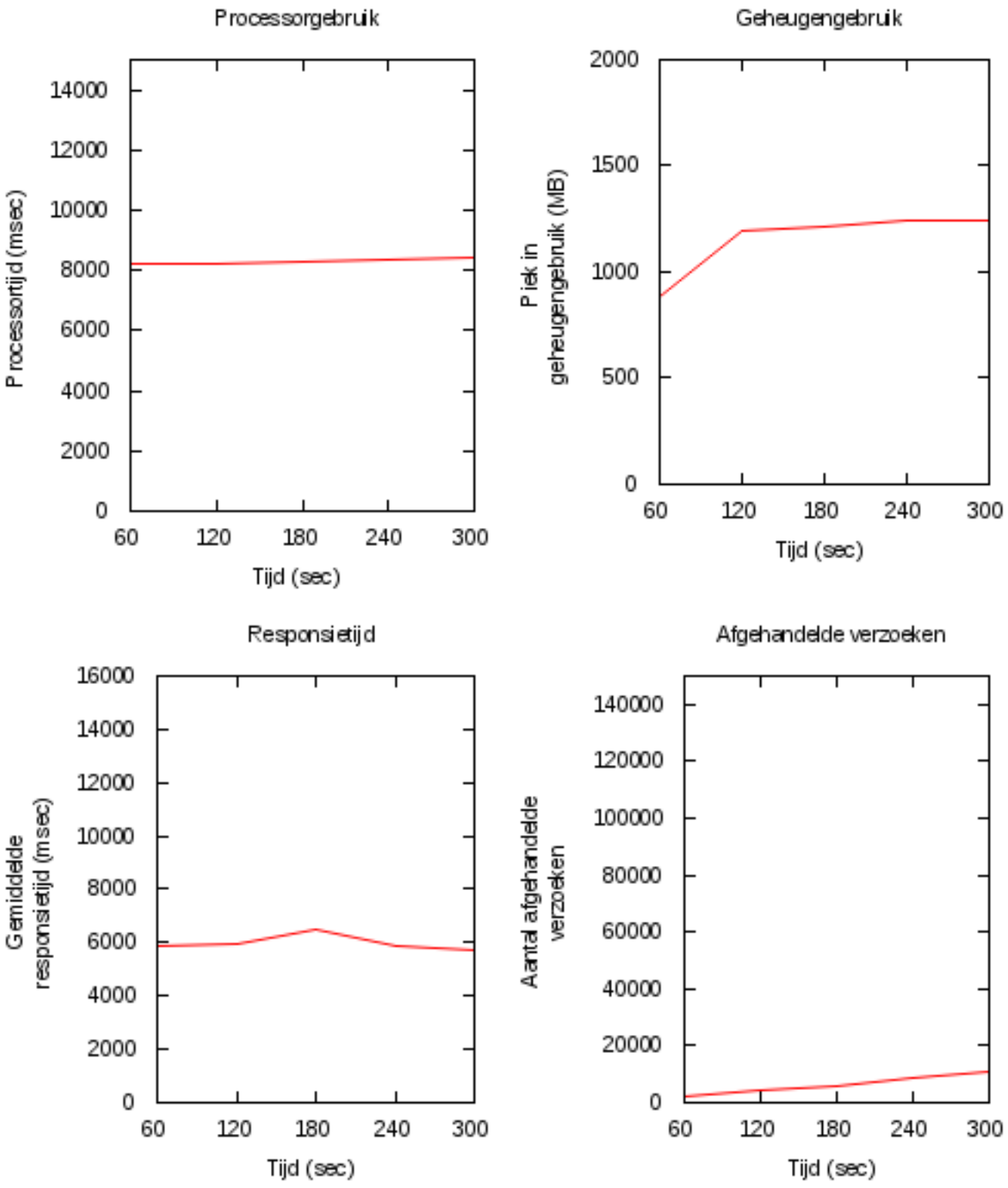
Figuur 5.11: Nginx



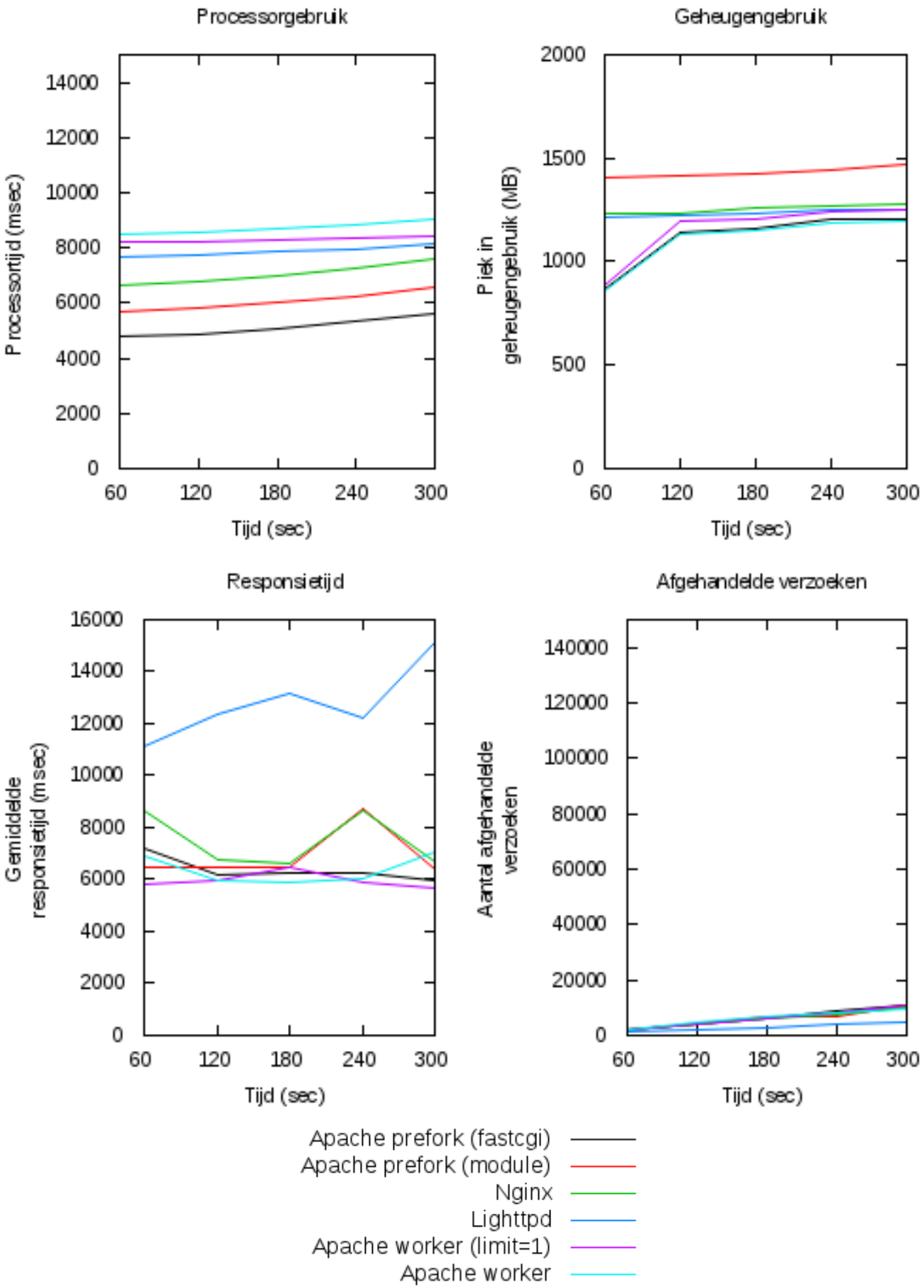
Figuur 5.12: Lighttpd



Figuur 5.13: Apache worker

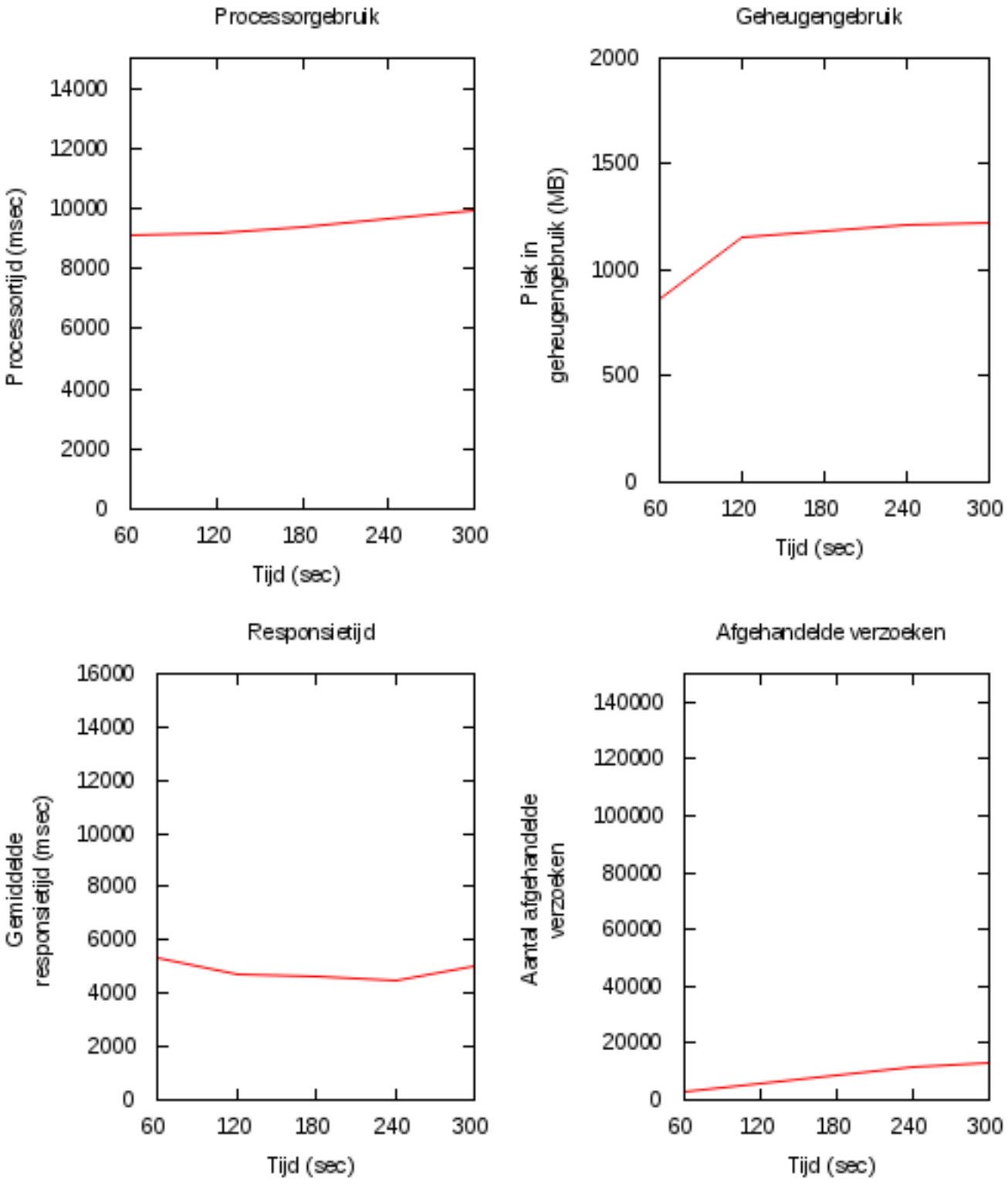


Figuur 5.14: Apache worker met limit=1

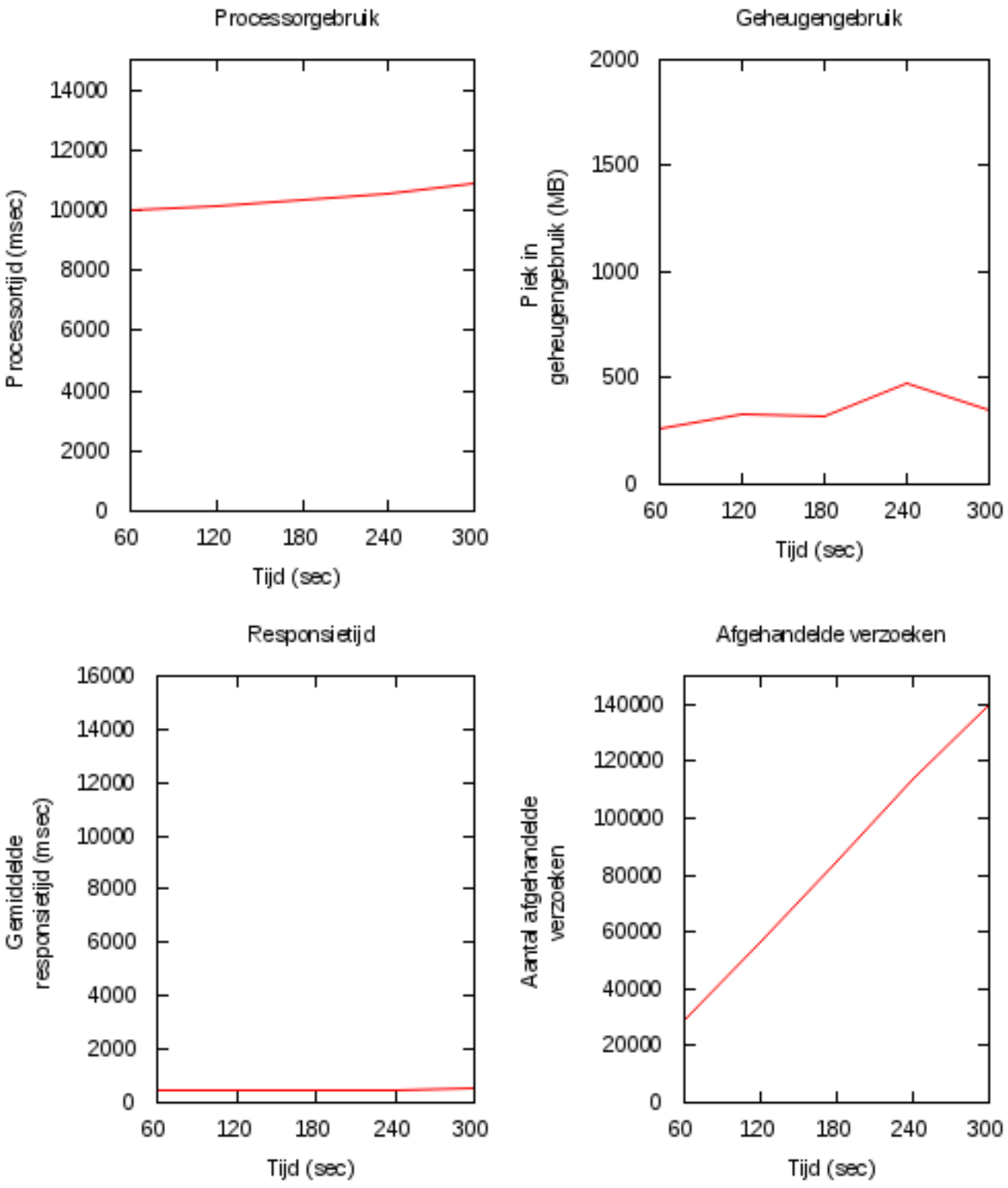


Figuur 5.15: Totaaloverzicht

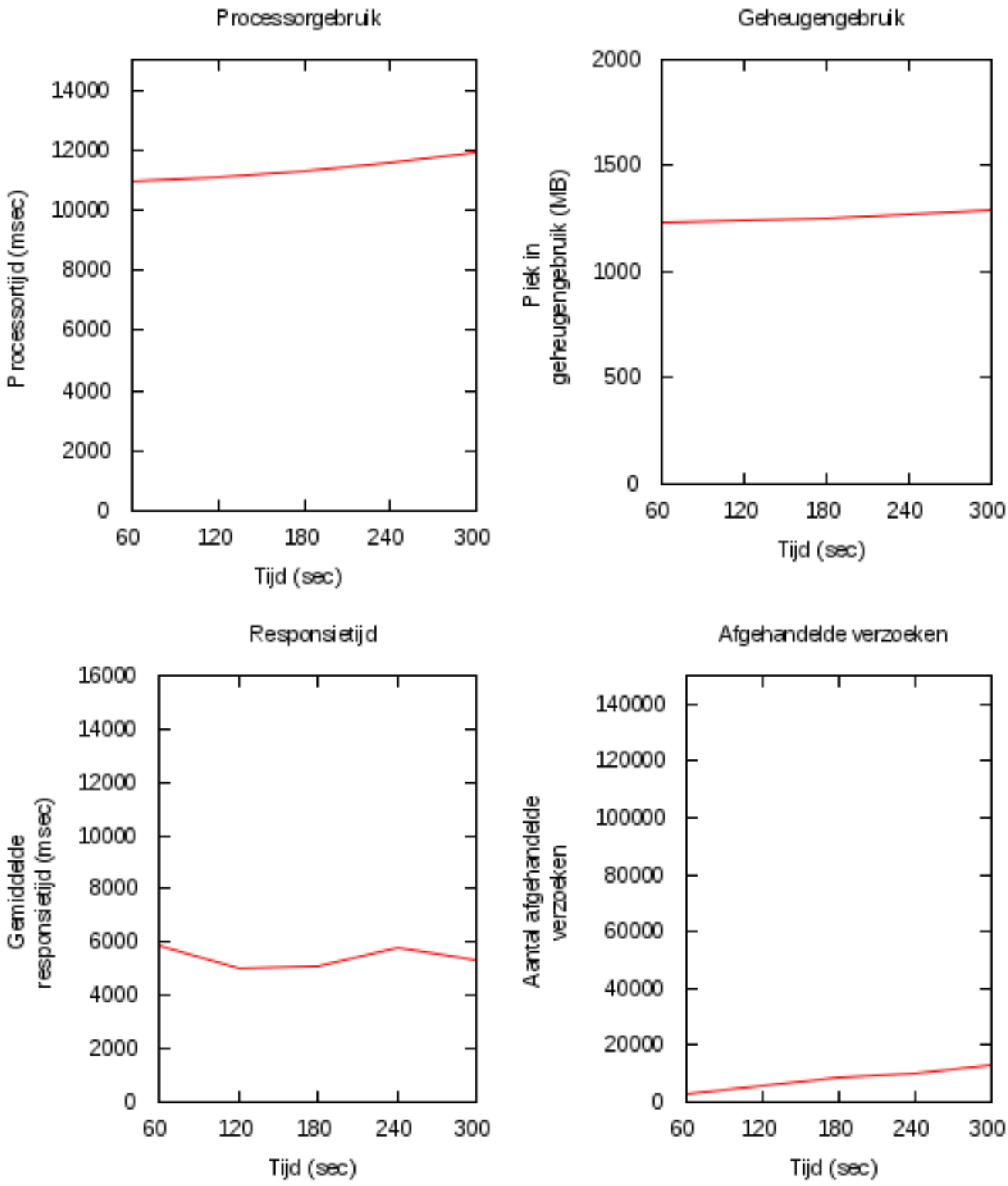
5.5.3 AJAX



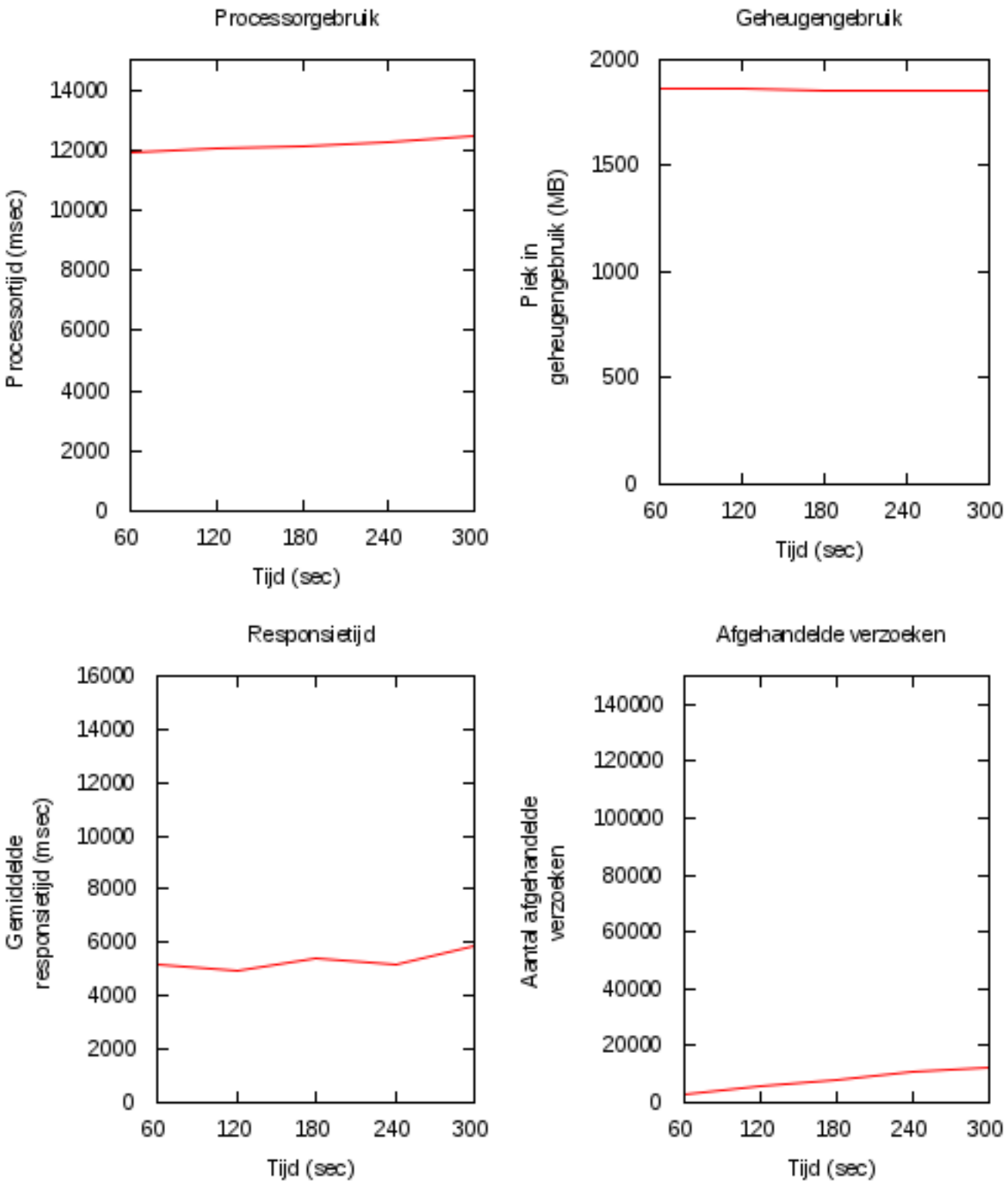
Figuur 5.16: Apache prefork met FastCGI



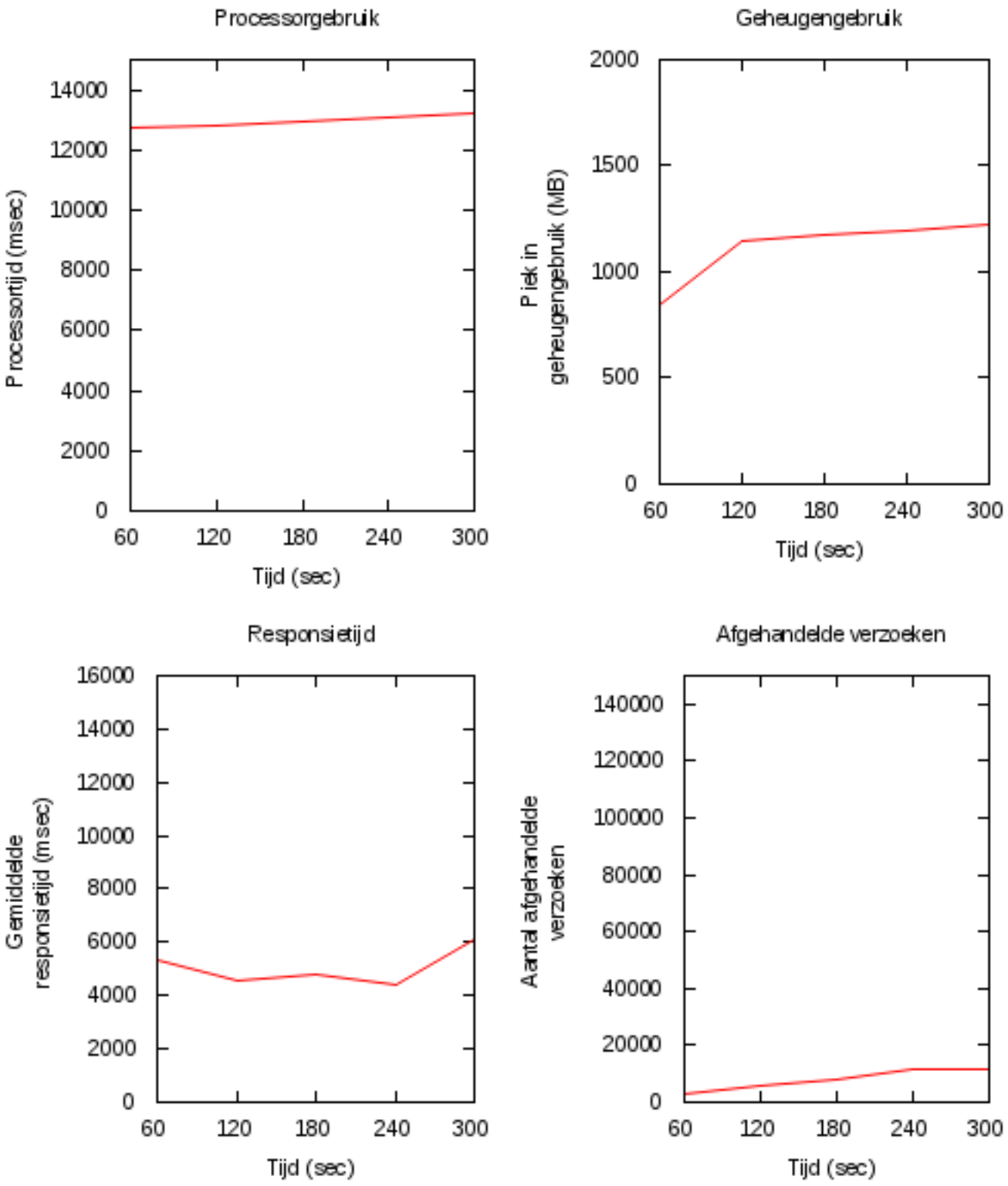
Figuur 5.17: Apache prefork met module



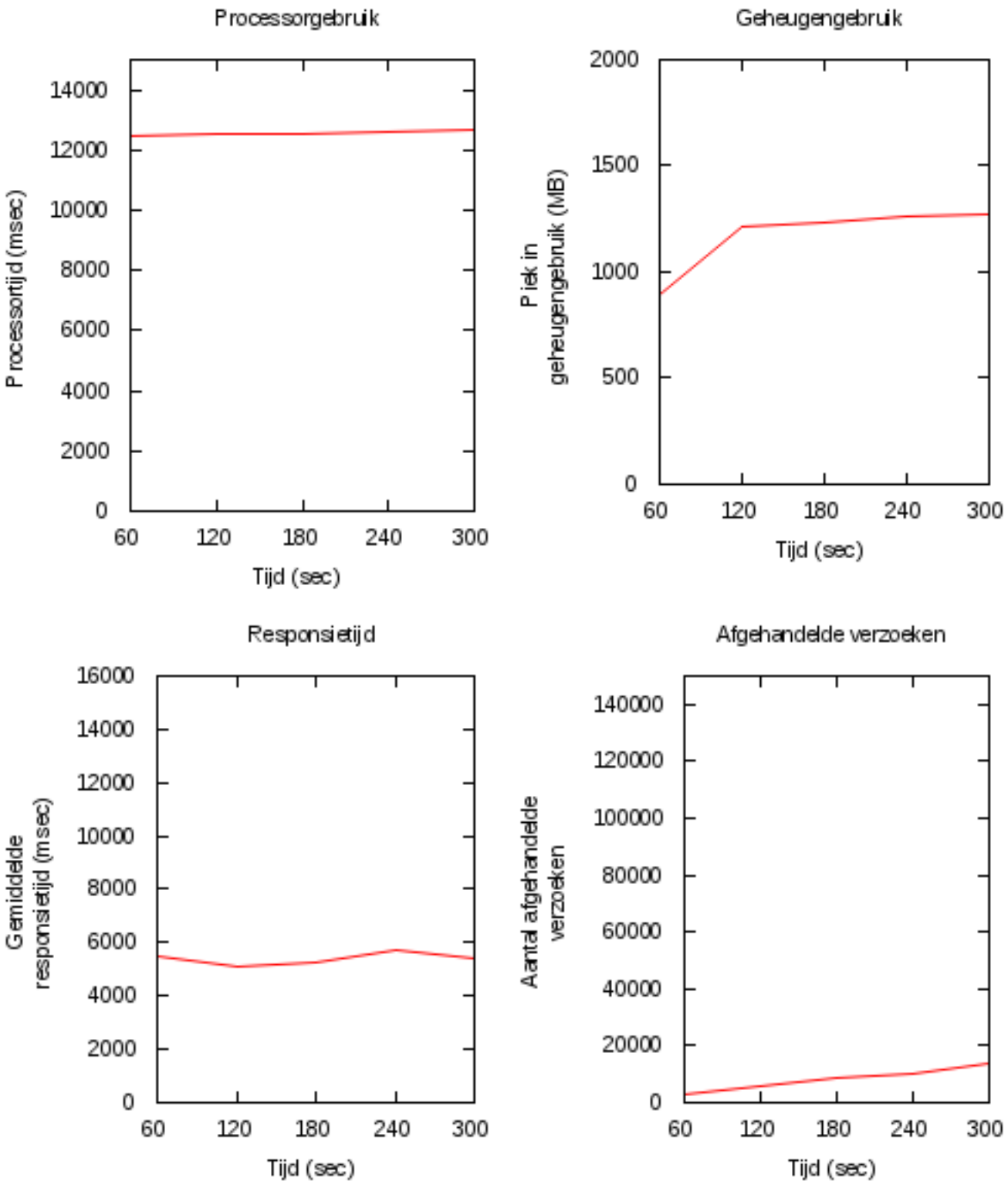
Figuur 5.18: Nginx



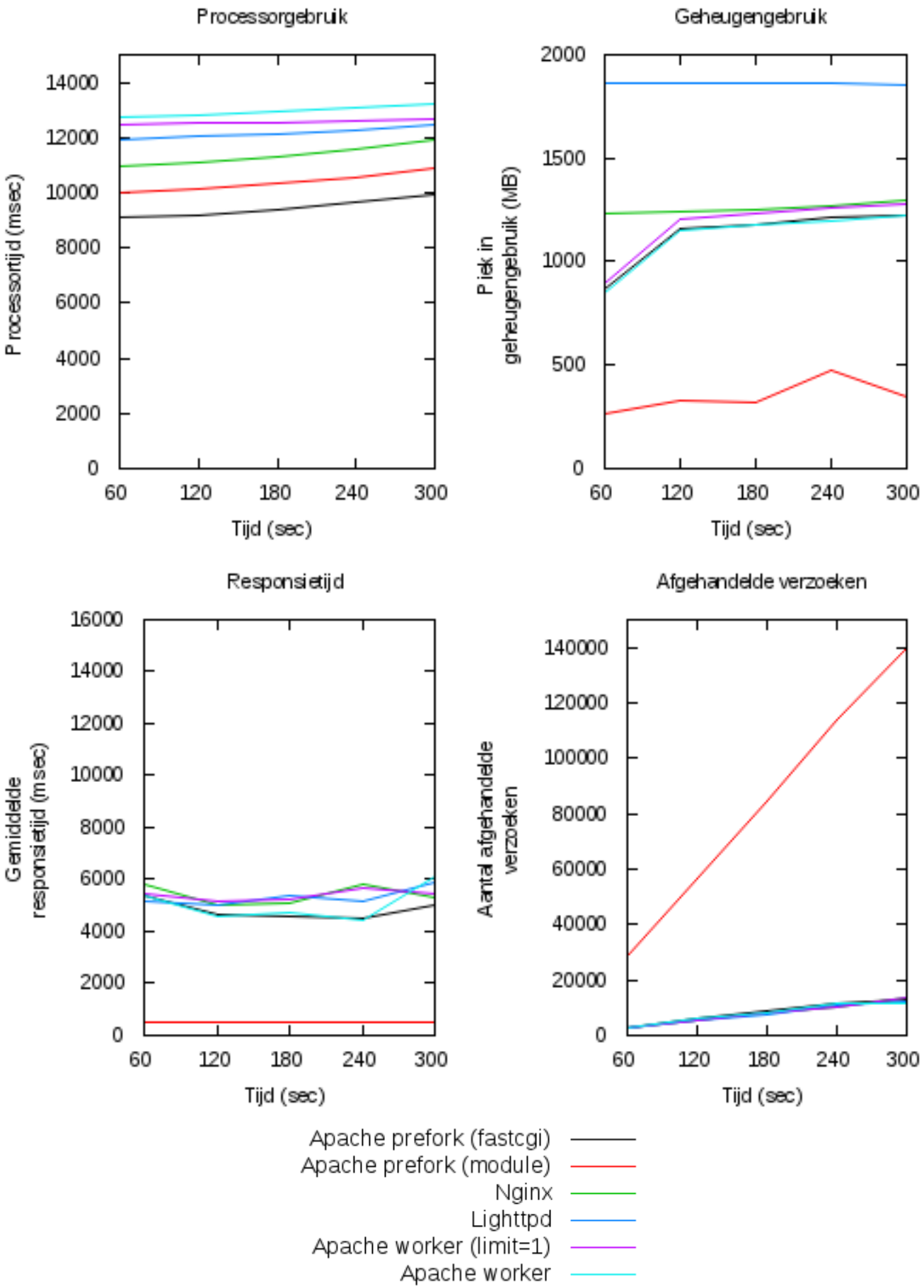
Figuur 5.19: Lighttpd



Figuur 5.20: Apache worker



Figuur 5.21: Apache worker met limit=1



Figuur 5.22: Totaaloverzicht

5.6 Bevindingen

Bij de statische websites is Lighttpd in staat de meeste verzoeken af te handelen. Dit is conform de verwachtingen uit het model, hoewel de verschillen niet erg groot zijn.

Ook bij de dynamische websites liggen de resultaten erg dicht bij elkaar. De Apache variant met de prefork MPM presteert het beste, maar wordt op de voet gevolgd door Nginx en Apache worker. Alleen Lighttpd blijft hier achter. De prefork variant met de module en met FastCGI presteren nagenoeg identiek, hoewel de variant met de module natuurlijk meer processorkracht en geheugen nodig heeft.

Bij websites op basis van AJAX zijn de verschillen stukken groter: hier is de Apache prefork MPM met de module in staat veel meer verzoeken af te handelen dan de FastCGI-variant. De FastCGI-variant presteert ongeveer gelijkwaardig aan alle andere webservers. Opvallend is, is dat het geheugengebruik op het systeem hier juist het laagste is bij de module. Dit is te verklaren door de vele PHP-processen die opgestart moeten worden om de verzoeken af te handelen. De scheduler van het besturingssysteem gooit hier naar alle waarschijnlijkheid roet in het eten.

6 Evaluatie

Hoewel het opgestelde model lijkt te voldoen aan de praktijk, zijn er wel enkele kanttekeningen te plaatsen. Ten eerste is *hammerhead* dusdanig geconfigureerd dat het compatibel was met de standaard instellingen van de Apache webserver. De verschillen bij een dynamisch karakter AJAX lijken echter dusdanig dat ook meerdere gelijktijdige sessies geen probleem zouden moeten zijn. Het is echter niet geheel ondenkbaar dat andere architecturen beter gaan presteren bij een hoger aantal gelijktijdige connecties.

Verder moet worden aangetekend dat in de praktijk de modernste browsers volledig gebruik maken van HTTP 1/1. Een van de belangrijkste eigenschappen van HTTP 1/1 is de mogelijkheid tot blijvende verbindingen [28]. Het effect hiervan wordt echter niet meegenomen in de validatie. Dit omdat hier bij het opstellen van het model ook geen rekening is gehouden en door het gebruik van een `HTTP Keep-Alive` bepaalde modellen mogelijk een extra voordeel krijgen [35].

Ook het bestandssysteem kan van invloed zijn op de prestaties van een webserver [29] en per I/O-model een andere prestatie leveren. Mogelijk levert een ander bestandssysteem dan Ext4 voor bepaalde architecturen winst op.

In de configuratie van de virtuele machine is gekozen voor slechts één core, omdat het model ook geen rekening hield met de mogelijkheid van meerdere processoren. De FastCGI architecturen, zeker in combinatie met asynchrone niet-blokkerende I/O hebben baat bij een multicore machine. Zodoende kan één core de webserver voor zijn rekening nemen en de ander de FastCGI applicatie. Dit voorkomt veel *context switches* die zorgen voor de tegenvallende prestaties.

Daarnaast zijn er natuurlijk vele instellingen aan alle webservers mogelijk die invloed hebben op de prestaties. Enkele instellingen die terugkomen in de literatuur zijn `EnableSendfile` en `EnableMmap` [29]. Deze hebben invloed op respectievelijk de I/O en het geheugengebruik. Ook zijn er modules die het aantal verzoeken per seconde kunnen verhogen, zoals mod_cache [29].

7 Toepassing

Nu het model gevalideerd en geëvalueerd is, is het zinvol om het in een grotere context te plaatsen en de toepassing ervan te verkennen. Het model dient geplaatst te worden in de ontwerpruimte van het systeem. Deze ontwerpruimte wordt door de ontwerpers vaak erg beperkt, terwijl een uitgebreide ruimte vele vraagstukken al kan oplossen voor de implementatie van het systeem. Denk hierbij aan visuele mogelijkheden, logistiek, interactie, gegevens en zo verder. Dit model voegt een extra dimensie toe aan deze ruimte, door rekening te houden met de dynamiek van het systeem. Daarnaast voegt het een eenvoudige beslismogelijkheid toe voor de ontwerper en verschaft het bij de implementatie duidelijkheid over het systeem. Het model dient dan ook gekoppeld te worden aan het functionele ontwerp van het systeem. Doordat voorafgaand aan het ontwikkel- en implementatieproces de keuze voor de webserver architectuur al bekend is, wordt onnodige vertraging voorkomen. Deze vertraging ontstaat vaak omdat er keuze tussen software gemaakt moeten worden. Deze keuze worden vaak gemaakt op basis van beschikbare kennis of door alternatieven te vergelijken in een simulatie-omgeving. Dit laatste is veelal tijdsintensief. Het model zorgt voor duidelijkheid in het proces en draagt bij aan een heldere tijdsplanning vanwege het wegnemen van de keuzes.

8 Blik in de toekomst

Hoewel het opgestelde model een mooie handreiking biedt voor systeemarchitecten, is het model slechts een start. Een van de belangrijkste opties die verkend dient te worden is het gebruik van HTTP `Keep-Alive`. De literatuur verwijst hier vaak naar en de invloed hiervan lijkt groot [35]. Het viel echter buiten het bereik van dit onderzoek om deze gevallen mee te nemen in het model.

Ook de prestaties onder een beveiligde verbinding door middel van SSL zijn interessant om mee te nemen, aangezien steeds meer websites hier gebruik van maken. De technologie hierachter is echter universeel en op Linux wordt er in dat geval gebruik gemaakt van `openssl` dus mogelijk is hier weinig verschil te merken.

Daarnaast zijn er vele configuratie-opties die uitgeprobeerd kunnen worden in de verschillende webservers, zodat er bijvoorbeeld ook gevarieerd kan worden met het aantal gelijktijdige sessies.

Zoals in de evaluatie (hoofdstuk 6 al werd genoemd, moet het effect van meerdere cores of processoren niet onderschat worden. FastCGI lijkt hier erg bij gebaat en meerdere cores kan wel degelijk van grote invloed zijn op het systeem.

9 Conclusie

In dit onderzoek is er gekeken welke procesarchitectuur het best presteert voor een dynamisch karakter van een website. Op basis van een aantal subvragen zijn eerst de verschillende architecturen en karakters in kaart gebracht. Deze subvragen vormen ook de basis van deze conclusie:

- Welke soorten dynamisch karakter van websites kunnen er onderscheiden worden?
- Wat zijn de verschillende webserver procesarchitecturen en hoe onderscheiden deze zich?
- Wanneer is een webserverprocesarchitectuur beter geschikt dan een andere?
- Welke variabelen bepalen de prestaties van een webserver?
- Waaruit moet een prestatiemodel voor dynamische karakters en procesarchitecturen bestaan?
- Klopt het model in de praktijk?

Er worden een viertal karakters onderscheiden: statisch, dynamisch aan de cliënt kant, dynamisch aan de server kant en AJAX. In de praktijk blijkt voor een webserver echter dat een statisch karakter en een dynamisch karakter aan de kant van de cliënt gelijk zijn.

De procesarchitectuur van een webserver bestaat uit een aantal onderdelen: het procesmodel, het I/O-model en de manier waarop ondersteuning dynamische talen wordt ingeladen. Van het procesmodel bestaat een drietal varianten, er bestaat 4 I/O-modellen en er zijn twee manieren waarop er ondersteuning voor dynamische talen ingeladen kan worden. Dit komt op een totaal van 24 architecturen. In de praktijk blijken er echter slechts een aantal geïmplementeerd te zijn.

De prestaties van een webserver zijn met name af te lezen aan het aantal verzoeken dat deze per seconde kan afhandelen: bij afhandeling van meer verzoeken per seconde levert de server een betere prestatie. Ook de responsietijd per verzoek (de tijd van van het verzenden van het verzoek tot het ontvangen van een antwoord) is een maat voor de prestaties.

Een prestatiemodel is eenvoudig af te leiden uit de literatuur, kijkend naar de verschillende onderdelen van de architectuur en de invloed die deze hebben op de prestaties van de webserver. Deze bevindingen komen ook grotendeels overeen met metingen uit de praktijk.

Bibliografie

— Artikelen —

[1] A. Iyengar and J. Challenger. Improving web server performance by caching dynamic data. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, pages 5–5. Usenix Association, 1997.

[2] Daniel A. Menascé. Web server software architectures. *IEEE Internet Computing*, 7(6):78–81, 2003.

[3] C. Amza, A. Chanda, A.L. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite. Specification and implementation of dynamic web site benchmarks. In *Workload Characterization, 2002. WWC-5. 2002 IEEE International Workshop on*, pages 3–13. IEEE, 2002.

[4] A. Scarsbrook R. Perriss, R. Graham. Understanding the internet, website design and intranet development: a primer for radiologists. *Clinical Radiology*, 61(5):377–389, 2006.

[5] J.J. Garrett et al. Ajax: A new approach to web applications. *Adaptive path*, 18, 2005.

[6] P. Graunke, S. Krishnamurthi, S. Van Der Hoeven, and M. Felleisen. Programming the web with high-level programming languages. *Programming Languages and Systems*, pages 122–136, 2001.

[7] David Pariag, Tim Brecht, Ashif S. Harji, Peter A. Buhr, Amol Shukla, and David R. Cheriton. Comparing the performance of web server architectures. In Paulo Ferreira, Thomas R. Gross, and Luís Veiga, editors, *EuroSys*, pages 231–243. ACM, 2007.

[8] Swapna S. Gokhale, Paul J. Vandal, and Jijun Lu. Performance and reliability analysis of web server software architectures. In *PRDC*, pages 351–358. IEEE Computer Society, 2006.

[9] Vivek S. Pai, Peter Druschel, and Willy Zwaenepoel. Flash: An efficient and portable web server. In *USENIX Annual Technical Conference, General Track*, pages 199–212. USENIX, 1999.

[10] G. Banga and P. Druschel. Measuring the capacity of a web server. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, pages 6–6. Usenix Association, 1997.

[11] Y. Hu, A. Nanda, and Q. Yang. Measurement, analysis and performance improvement of the apache web server. In *Performance, Computing and Communications Conference, 1999. IPCCC'99. IEEE International*, pages 261–267. IEEE, 1999.

[12] J.C. Hu, I. Pyarali, and D.C. Schmidt. Measuring the impact of event dispatching and concurrency models on web server performance over high-speed networks. In *Global Telecommunications Conference, 1997. GLOBECOM'97., IEEE*, volume 3, pages 1924–1931. IEEE, 1997.

[13] D. Mosberger and T. Jin. httperf—a tool for measuring web server performance. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):31–37, 1998.

[14] J. Dilley, R. Friedrich, T. Jin, and J. Rolia. Web server performance measurement and modeling techniques. *Performance Evaluation*, 33(1):5–26, 1998.

[15] D.A. Menascé, D. Barbará, and R. Dodge. Preserving qos of e-commerce sites through self-tuning: A performance model approach. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 224–234. ACM, 2001.

[16] C. Lever, S.N. Alliance, and S.P. Molloy. An analysis of the tux web server. *Ann Arbor*, 1001:48103–4943, 2000.

[17] T. Voigt and P. Gunningberg. Kernel-based control of persistent web server connections. *ACM SIGMETRICS Performance Evaluation Review*, 29(2):20–25, 2001.

[18] A. Bosque, P. Ibañez, V. Viñals, P. Stenström, and J.M. Llabería. Characterization of apache web server with specweb2005. In *Proceedings of the 2007 workshop on Memory performance: Dealing with Applications, systems and architecture*, pages 65–72. ACM, 2007.

[19] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. In *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*, pages 315–326. IEEE, 2008.

[20] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *ACM SIGMETRICS Performance Evaluation Review*, volume 26, pages 151–160. ACM, 1998.

[21] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega. Cotson: infrastructure for full system simulation. *ACM SIGOPS Operating Systems Review*, 43(1):52–61, 2009.

[22] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[23] R. Bedichek. Simnow: Fast platform simulation purely in software. In *Hot Chips*, volume 16, 2004.

[24] F. Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[25] M. Rosenblum. Vmware's virtual platform^{TM}. In *Proceedings of Hot Chips*, pages 185–196, 1999.

[26] J. Nieh and O.C. Leonard. Examining vmware. *Dr. Dobb's Journal*, 25(8):70, 2000.

[27] Gaurav Banga and Peter Druschel. Measuring the capacity of a web server under realistic loads. *World Wide Web*, 2(1-2):69–83, 1999.

[28] H.F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H.W. Lie, and C. Lilley. Network performance effects of http/1.1, css1, and png. In *ACM SIGCOMM Computer Communication Review*, volume 27, pages 155–166. ACM, 1997.

[29] C. MacCárthaigh. Scaling apache 2. x beyond 20,000 concurrent downloads. *ApacheCon EU*, 116, 2005.

— Boeken —

[30] R. Nelson. *Probability, stochastic processes, and queueing theory: the mathematics of computer performance modelling.* Springer, 1995.

[31] D.A. Menascé and V.A.F. Almeida. *Capacity Planning for Web Services: metrics, models, and methods.* Prentice Hall, 2002.

— RFC's —

[32] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol–http/1.1. `http://www.ietf.org/rfc/rfc2616.txt`, 1999.

— Theses —

[33] L. Deboosere. Experimentele karakterisatie van webservers, 2005.

[34] A. Hagsten and F. Neis. Crisis request generator for internet servers, 2006.

[35] A. Hidalgo Barea et al. Analysis and evaluation of high performance web servers, 2011.

[36] A.S. Harji. *Performance comparison of uniprocessor and multiprocessor web server architectures.* PhD thesis, University of Waterloo, 2010.

— Webpagina's —

[37] Netcraft. September 2011 Web Server Survey. `http://news.netcraft.com/archives/2011/09/06/september-2011-web-server-survey.html`, Sep 2011.

[38] Igor Sysoev. Nginx. `http://www.nginx.org`.

[39] Dan Kegel. The c10k problem. `http://www.kegel.com/c10k.html`, 2006.

[40] Jones, M.T. Boost application performance using asynchronous I/O. `http://www.ibm.com/developerworks/linux/library/l-async/?ca=dgr-lnxw02aUsingPOISIXAIOAPI`, 2006.

[41] W3Techs Web Technology Surveys. Usage of operating systems for websites. `http://w3techs.com/technologies/overview/operating_system/all`, Aug 2011.

[42] W3Techs Web Technology Surveys. Usage statistics and market share of Unix for websites. `http://w3techs.com/technologies/details/os-unix/all/all`, Aug 2011.

[43] The Apache Software Foundation. Apache mpm prefork. `http://httpd.apache.org/docs/2.2/mod/prefork.html`.

[44] Jan Kneschke. lighttpd fly light. `http://www.lighttpd.net/`.

[45] The Apache Software Foundation. Apache mpm worker. `http://httpd.apache.org/docs/2.2/mod/worker.html`.

[46] University of Waterloo. userver project. `http://userver.uwaterloo.ca`, 2007.

[47] L.P. Hewlett-Packard Development Company. the userver project. `http://www.hpl.hp.com/research/linux/userver/`.

[48] The PHP Group. Php installation - manual.

[49] khttpd - linux http accelerator. `http://www.fenrus.demon.nl/`.

[50] The World Wide Web Consortium. World wide web server software. `http://www.w3.org/Servers.html`.

[51] Wikipedia. Comparison of web server software — wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Comparison_of_web_server_software`, 2012.

[52] The Apache Software Foundation. ab - Apache HTTP server benchmarking tool. `http://httpd.apache.org/docs/2.2/programs/ab.html`.

[53] Sourceforge. Hammerhead. `http://sourceforge.net/projects/hammerhead/`, 2011.

[54] Jan Kneschke. Start - weighttp - lighty labs. `http://redmine.lighttpd.net/projects/weighttp/wiki`.

[55] Main page - gem5. `http://www.m5sim.org/Main_Page`.

[56] Virtualbox. Virtualbox. https://www.virtualbox.org/wiki/VirtualBox.

[57] Andreas Gohr and the DokuWiki Community. Dokuwiki [dokuwiki]. http://www.dokuwiki.org/dokuwiki.

A Code

A.1 Gegevens verzamelen

A.1.1 setup.sh

```
#!/bin/bash

if [ $# -ne 2 ]; then
        echo "Aanroep: setup.sh <vm naam> <webserver>"
        exit
fi

# laad variabelen
. vars.sh

# verwijder accounting bestand
rm -rf $ACCT
# maak leeg accounting bestand aan
touch $ACCT
#start proces accounting
/etc/init.d/acct start

if [ ! -d $LOGDIR ]; then
        mkdir $LOGDIR
fi


# log geheugen gebruik elke seconde in 10 minuten (en breng proces naar de achtergrond
vmstat -a -n 1 > $LOGDIR./memory-$1 &

# start de webserver
/etc/init.d/$2 start
```

A.1.2 close.sh

```
#!/bin/sh

if [ $# -ne 2 ]; then
        echo "Aanroep: close.sh <vm naam> <webserver>"
        exit
fi

#laad variabelen
. ./vars.sh
```

```
# stop de webserver
/etc/init.d/$2 stop;

# stop vmstat
killall vmstat

# haal processorgebruik op
sa --print-seconds -t | grep $2 > $LOGDIR./proc-$1

# logs tarren voor verzending
cd $LOGDIR;
tar -cf $1.tar *$1
```

A.1.3 clean.sh

```
#!/bin/bash

rm -rf ./results/
```

A.1.4 runtests.sh

```
#!/bin/bash

# statische variabelen
HOST=10.0.1.60
USER=root
LOGDIR=tests
# lijst van VMs

vms="apache2-prefork-fastcgi-static apache2-prefork-module-static nginx-fastcgi-static
    lighttpd-fastcgi-static apache2-workerlimit-fastcgi-static apache2-worker-fastcgi-
    static apache2-prefork-fastcgi-dynamic apache2-prefork-module-dynamic nginx-fastcgi-
    dynamic lighttpd-fastcgi-dynamic apache2-workerlimit-fastcgi-dynamic apache2-worker-
    fastcgi-dynamic apache2-prefork-fastcgi-ajax apache2-prefork-module-ajax nginx-fastcgi
    -ajax lighttpd-fastcgi-ajax apache2-workerlimit-fastcgi-ajax apache2-worker-fastcgi-
    ajax"
for vm in $vms
do
        #start de VM
        VBoxManage startvm $vm

        #webserver naam uit vm afleiden
        webserver=‘echo $vm | cut -d \- -f 1‘

        # wacht 30 sec op het opstarten van de VM.
        sleep 30

        # draai met verschillende verzoeken. Beginnend bij 1 minuut, oplopend tot 5 minuten
            in stappen van 1 minuut
        for i in {60..300..60}
        do
                for j in 1 2 3
                do
```

```
        # logfirs aanmaken
        if [ ! -d $LOGDIR ]; then
               mkdir $LOGDIR
        fi
        if [ ! -d "$LOGDIR/$vm" ]; then
               mkdir "$LOGDIR/$vm"
        fi

        if [ ! -d "$LOGDIR/$vm/$i" ]; then
               mkdir "$LOGDIR/$vm/$i"
        fi
        if [ ! -d "$LOGDIR/$vm/$i/$j" ]; then
               mkdir "$LOGDIR/$vm/$i/$j"
        fi
        # setup draaien
        ssh $USER@$HOST "/root/setup.sh $vm $webserver"

        # server even laten opstarten
        sleep 10

        # test draaien
        hammerhead -c /home/xander/scenarios/static/hh.conf -o $LOGDIR/$vm/$i/$j/hh.
            log -s $i

        # test afsluiten
        ssh $USER@$HOST "/root/close.sh $vm $webserver"



        #resultaten ophalen
        scp $USER@$HOST:/root/results/$vm.tar ./$LOGDIR/$vm/$i/$j/
        cd $LOGDIR/$vm/$i/$j/
        tar -xf $vm.tar
        cd ~

        ssh $USER@$HOST "/root/clean.sh"

        ssh $USER@$HOST "/sbin/reboot"

        # wacht 1 minuut op het opnieuw opstarten van de VM
        sleep 60
        done
done

# stop de virtuele machine

ssh $USER@$HOST "/sbin/poweroff"


done
```

A.2 generate-graphdata.sh

```
#!/bin/bash

LOGDIR=tests
highest_mem_total[1]=0
highest_mem_total[2]=0
highest_mem_total[3]=0


proc_total[1]=0
proc_total[2]=0
proc_total[3]=0

# lijst van VMs
vms="apache2-prefork-fastcgi-static apache2-prefork-module-static nginx-fastcgi-static
    lighttpd-fastcgi-static apache-workerlimit-fastcgi-static apache-worker-fastcgi-static
     apache2-prefork-fastcgi-dynamic apache2-prefork-module-dynamic nginx-fastcgi-dynamic
    lighttpd-fastcgi-dynamic apache-workerlimit-fastcgi-dynamic apache-worker-fastcgi-
    dynamic apache2-prefork-fastcgi-ajax apache2-prefork-module-ajax nginx-fastcgi-ajax
    lighttpd-fastcgi-ajax apache-workerlimit-fastcgi-ajax apache-worker-fastcgi-ajax"

for vm in $vms
do

        # loop door alle requests heen
        for i in {60..300..60}
        do
                for j in 1 2 3
                do
                        inactive=0
                        highest_mem_total[$j]=0
                        active=0

                        proc_total=0
                        first=0
                        second=0
                        first_double=0
                        second_double=0

                        k=0;
                        # lees geheugenfile, zoek de grootste
                        while read line
                        do
                                # eerste regel overslaan, dat is een kop.
                                if [ $k -eq 0 ]; then
                                        # verhoog teller
                                        k=‘expr $k + 1‘
                                        continue
                                fi

                                # inactief geheugen is nog niet vrijgemaakt
                                inactive=‘echo $line | cut -d \ -f 5‘
                                # actief geheugen
                                active=‘echo $line | cut -d \ -f 6‘
```

```
                # optellen naar totaal
                total=‘echo $inactive + $active | bc‘

                # vergelijken met hoogste totaal tot nu toe
                if [ $total -gt ${highest_mem_total[$j]} ]; then
                        # hoogste totaal verhogen als totaal hoger is
                        highest_mem_total[$j]=‘expr $total‘
                fi

        done < $LOGDIR/$vm/$i/$j/memory-$vm

        # lees processorbestand
        while read line
        do

                first=‘echo $line | cut -d \ -f 2‘
                second=‘echo $line | cut -d \ -f 3‘


                first_double=‘echo $first | egrep -o ^\([0-9]*\.?[0-9]*\)‘
                second_double=‘echo $second | egrep -o ^\([0-9]*\.?[0-9]*\)‘

                line_total=‘echo $first_double + $second_double | bc‘

                proc_total[$j]=‘echo $line_total + ${proc_total[$j]} | bc‘



        done < $LOGDIR/$vm/$i/$j/proc-$vm


        # lees responsietijd en aantal verzoeken in
        requestsline=‘grep "Total Requests Served" $LOGDIR/$vm/$i/$j/hh.log‘
        requests[$j]=‘echo $requestsline | cut -d \ -f 5‘

        responseline=‘grep "Average Response Time" $LOGDIR/$vm/$i/$j/hh.log‘
        response[$j]=‘echo $responseline | cut -d \ -f 6‘
done

# het gemiddelde van de 3 topgeheugengebruiken
avg_mem=‘echo \(${highest_mem_total[1]}/1024+${highest_mem_total[2]}/1024+${
    highest_mem_total[3]}/1024\)/3 | bc‘

echo "$i $avg_mem" >> $LOGDIR/$vm-mem


avg_proc=‘echo "scale=2; (${proc_total[1]}+${proc_total[2]}+${proc_total
    [3]})/3;" | bc‘

echo "$i $avg_proc" >> $LOGDIR/$vm-proc
```

```
            avg_response=‘echo "(${response[1]}+${response[2]}+${response[3]})/3" | bc‘
            echo "$i $avg_response" >> $LOGDIR/$vm-responses

            avg_requests=‘echo "(${requests[1]}+${requests[2]}+${requests[3]})/3" | bc‘
            echo "$i $avg_response" >> $LOGDIR/$vm-requests

        done


done
```

A.3 Generatie statische bestanden

```
dd if=/dev/urandom of=1.html bs=1k count=24
dd if=/dev/urandom of=2.html bs=1k count=48
dd if=/dev/urandom of=1.js bs=1k count=10
dd if=/dev/urandom of=1.png bs=1k count=150
dd if=/dev/urandom of=2.png bs=1k count=800
dd if=/dev/urandom of=1.css bs=1k count=40
```

A.4 AJAX

A.4.1 ajax.php

```
<?php


if($_POST['default'] == 1) {

echo '
{
    "glossary": {
        "title": "example glossary",
                "GlossDiv": {
            "title": "S",
                        "GlossList": {
                "GlossEntry": {
                    "ID": "SGML",
                                        "SortAs": "SGML",
                                        "GlossTerm": "Standard Generalized Markup Language",
                                        "Acronym": "SGML",
                                        "Abbrev": "ISO 8879:1986",
                                        "GlossDef": {
                        "para": "A meta-markup language, used to create markup languages
                            such as DocBook.",
                                                "GlossSeeAlso": ["GML", "XML"]
                    },
                                        "GlossSee": "markup"
                }
            }
        }
    }
}
```

```
';

} else if($_POST['action'] == 'app' && $_POST['env'] == '' && $_POST['test'] == true) {
echo '{"web-app": {
  "servlet": [
    {
      "servlet-name": "cofaxCDS",
      "servlet-class": "org.cofax.cds.CDSServlet",
      "init-param": {
        "configGlossary:installationAt": "Philadelphia, PA",
        "configGlossary:adminEmail": "ksm@pobox.com",
        "configGlossary:poweredBy": "Cofax",
        "configGlossary:poweredByIcon": "/images/cofax.gif",
        "configGlossary:staticPath": "/content/static",
        "templateProcessorClass": "org.cofax.WysiwygTemplate",
        "templateLoaderClass": "org.cofax.FilesTemplateLoader",
        "templatePath": "templates",
        "templateOverridePath": "",
        "defaultListTemplate": "listTemplate.htm",
        "defaultFileTemplate": "articleTemplate.htm",
        "useJSP": false,
        "jspListTemplate": "listTemplate.jsp",
        "jspFileTemplate": "articleTemplate.jsp",
        "cachePackageTagsTrack": 200,
        "cachePackageTagsStore": 200,
        "cachePackageTagsRefresh": 60,
        "cacheTemplatesTrack": 100,
        "cacheTemplatesStore": 50,
        "cacheTemplatesRefresh": 15,
        "cachePagesTrack": 200,
        "cachePagesStore": 100,
        "cachePagesRefresh": 10,
        "cachePagesDirtyRead": 10,
        "searchEngineListTemplate": "forSearchEnginesList.htm",
        "searchEngineFileTemplate": "forSearchEngines.htm",
        "searchEngineRobotsDb": "WEB-INF/robots.db",
        "useDataStore": true,
        "dataStoreClass": "org.cofax.SqlDataStore",
        "redirectionClass": "org.cofax.SqlRedirection",
        "dataStoreName": "cofax",
        "dataStoreDriver": "com.microsoft.jdbc.sqlserver.SQLServerDriver",
        "dataStoreUrl": "jdbc:microsoft:sqlserver://LOCALHOST:1433;DatabaseName=goon",
        "dataStoreUser": "sa",
        "dataStorePassword": "dataStoreTestQuery",
        "dataStoreTestQuery": "SET NOCOUNT ON;select test=\'test\';",
        "dataStoreLogFile": "/usr/local/tomcat/logs/datastore.log",
        "dataStoreInitConns": 10,
        "dataStoreMaxConns": 100,
        "dataStoreConnUsageLimit": 100,
        "dataStoreLogLevel": "debug",
        "maxUrlLength": 500}},
    {
      "servlet-name": "cofaxEmail",
```

```
      "servlet-class": "org.cofax.cds.EmailServlet",
      "init-param": {
      "mailHost": "mail1",
      "mailHostOverride": "mail2"}},
    {
      "servlet-name": "cofaxAdmin",
      "servlet-class": "org.cofax.cds.AdminServlet"},

    {
      "servlet-name": "fileServlet",
      "servlet-class": "org.cofax.cds.FileServlet"},
    {
      "servlet-name": "cofaxTools",
      "servlet-class": "org.cofax.cms.CofaxToolsServlet",
      "init-param": {
        "templatePath": "toolstemplates/",
        "log": 1,
        "logLocation": "/usr/local/tomcat/logs/CofaxTools.log",
        "logMaxSize": "",
        "dataLog": 1,
        "dataLogLocation": "/usr/local/tomcat/logs/dataLog.log",
        "dataLogMaxSize": "",
        "removePageCache": "/content/admin/remove?cache=pages&id=",
        "removeTemplateCache": "/content/admin/remove?cache=templates&id=",
        "fileTransferFolder": "/usr/local/tomcat/webapps/content/fileTransferFolder",
        "lookInContext": 1,
        "adminGroupID": 4,
        "betaServer": true}}],
  "servlet-mapping": {
    "cofaxCDS": "/",
    "cofaxEmail": "/cofaxutil/aemail/*",
    "cofaxAdmin": "/admin/*",
    "fileServlet": "/static/*",
    "cofaxTools": "/tools/*"},

  "taglib": {
    "taglib-uri": "cofax.tld",
    "taglib-location": "/WEB-INF/tlds/cofax.tld"}}}';
} else {
echo '{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}';
}
```

B Configuratie

B.1 Hammerhead

B.1.1 Algemeen

Bij onderstaande configuratie dient aangetekend te worden dat Scenario_Directory per dynamisch karakter anders is was. De volgende waardes zijn gebruikt: `/home/test/scenarios/static/`, `/home/test/scenarios/dynamic` en `/home/test/scenarios/ajax/`

```
Scenario_Directory /home/test/scenarios/static/
Log_Filename /tmp/hammer.log
Sessions 256
Machine_IP 10.0.1.60:80
Sleep_time 10
Sequence_probability 100
Load_Images off
Run_time 600
Ip_Alias 10.0.0.60-254
```

B.1.2 Scenario's

Statisch karakter (static.scn)

```
NStatic1
RGET /1.html
T0
.
NStatic2
RGET /1.css
T0
.
NStatic3
RGET /1.js
T0
.
NStatic3
RGET /1.png
T0
.
NStatic4
RGET /2.html
T0
.
```

```
NStatic5
RGET /2.png
T0 _exit
.
```

Dynamisch karakter (dynamic.scn)

```
NDynamic1
RGET /doku.php
T0
.
NDynamic2
RGET /doku.php?id=start
T0
.
NDynamic3
RGET /
T0
.
NDynamic4
RPOST /doku.php
Bdo=edit&rev=&id=start
T0
.
NDynamic5
RGET /doku.php?id=playground:playground
T0 _exit
.
```

AJAX (ajax.scn)

```
NAjax1
RGET /doku.php
T0
.
NAjax2
RGET /doku.php?id=start
T0
.
NAjax3
RPOST /ajax.php
Bdefault=1
T0
.
NAjax4
RPOST /ajax.php
Baction=app&env=&test=1
T0
.
NAjax5
RGET /doku.php?id=playground:playground
T0 _exit
.
```