De Invloed van Fluctuaties in de Prestatieniveaus van Netwerk Nodes op de Runtijd van Parallelle Toepassingen

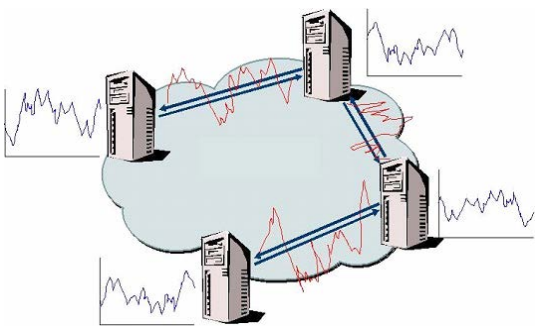
Een Prestatieanalyse van Parallelle Toepassingen in een Realistische Gridomgeving

Eric Jonker, Menno Dobber

Scriptie
Bedrijfswiskunde en -Informatica

Vrije Universiteit
Faculteit der Exacte Wetenschappen
Divisie Wiskunde
De Boelelaan 1081a
1081 HV Amsterdam

July 2006



Preface

A Dutch preface and abstract are both available immediately following the English translations.

This document is my thesis for the Business Mathematics and Informatics study at the Vrije Universteit of Amsterdam. The thesis is the final requirement to fulfil for this study, in order for me to graduate. It has to report in a professional but understandable manner on the results of an investigation conducted independently by the student. The emphasis is on the business-oriented aspect of the study, next to the mathematical and informatics related aspects.

Here, I would like to thank Menno Dobber first and foremost, PhD for the Optimization of Business Processes research group at the Vrije Universiteit, because he was willing to supervise me and for bringing the subject to live. I have been conducting this investigation with a lot of interest; writing this report has been just a tad harder. In addition, I would like to thank him for his valuable advice, for trying to make me look at the subject from a very conceptual point of view and for giving me the idea that he succeeded in that to some extend.

Next, I would like to thank Auke Pot, PhD for the Optimization of Business Processes research group at the Vrije Universiteit, for the invaluable tips he gave me, which I understand usually remains a trade secret reserved for PhD students. Thanks, it was utterly useful. Rest assured, I will not tell anyone.

Finally, I would like to thank Annemieke van Goor-Balk, coordinator business contacts and member of the graduation committee of the Faculty of Sciences of the Vrije Universiteit, for the interest she has displayed and her support during the whole period my graduation was forthcoming.

Eric Jonker
Broek op Langedijk, August 2006

Abstract

Goals of the Investigation

In this investigation, the running times of a parallel application in a realistic grid environment are analysed. The goal is to provide insight into the impact of continuous changes in the running times of jobs or tasks on the total running time of the application. Because running times are ever changing, these are said to fluctuate.

Problem Definition

A grid is a large-scale network of computers cooperating to reach unprecedented performance levels, with the goal of solving the largest computational problems in existence within the shortest possible time.

It is a well-known fact that performance levels of computers in a realistic grid environment, commonly referred to as nodes, are diverse and change over time. Running times of equal tasks differ strongly between computers as well as between subsequent evaluations on the same computer. These differences have a negative impact on the running times of parallel applications.

Principles of the Research Design

In this report, a simple method is devised that is capable of determining to what extend each cooperating computer is responsible for the total running time of an application. This method considers a given set of cooperating computers and iteratively determines which of these slows down the application the most. This computer is subsequently removed from the set and the process is repeated with the updated set, until the set is empty.

It is furthermore established that computers responsible for a large portion of the total running time are also amongst the slowest computers in many subsequent measurements of the running times. This is determined with a similar method, but now the number of measurements is counted in which the computer is the slowest, instead of computing the total increase of the running time. Computers are still removed from the set in the same order as before.

This way, a relationship between the amount of fluctuations between the slowing effect of the cooperating computers and the number of computers with a slowing effect is established.
This relationship is very strong for a random selection of cooperating computers and diminishes when those computers with the largest slowing effect are iteratively excluded.

The effects on the total running time of the application are subsequently simulated for a random selection of cooperating computers, while a different number of computers with the largest slowing effect are prohibited to cooperate in advance.

Main Research Results

A small fraction of the computers that are running a parallel application cause a substantial part of the total running time. 8 out of 100 cooperating computers are responsible for 68% of the total running time and only about one third causes 95% of the total running time.

Those computers that have the largest effect on the total running time can moreover be identified with ease, such that the total running time can immediately be reduced, for example by having these computers cooperate no longer, or by reducing the amount of work they must complete.

Hence, the total running time of an application can easily be reduced significantly. Put differently, the performance of parallel applications in a realistic grid environment can be improved upon considerably with ease.

Voorwoord

Dit document is mijn afstudeerscriptie voor de opleiding Bedrijfswiskunde en –Informatica (BWI) aan de Vrije Universiteit (VU), Amsterdam. Het schrijven van een scriptie is de laatste vereiste voor mij om deze studie te voltooien. De scriptie dient op deskundige en heldere wijze verslag te leggen van een door de student zelfstandig uitgevoerd onderzoek. De nadruk ligt hierbij op het bedrijfskundige aspect van de studie, naast de wiskunde en informatica.

Ik wil ik hier allereerst graag Menno Dobber bedanken, PhD bij de Optimalisatie van Bedrijfsprocessen groep aan de Vrije Universiteit, omdat hij mij wilde begeleiden en het onderwerp dichtbij heeft gebracht. Ik heb dit onderzoek met plezier uitgevoerd; het schrijven van dit rapport was ietsje minder. Verder wil ik hem bedanken voor zijn waardevolle adviezen, dat hij zijn zeer geconceptualiseerde inzicht in dit onderwerp op mij heeft proberen over te brengen en mij het idee heeft gegeven dat hem dat ook wel behoorlijk is gelukt.

Verder wil ik Auke Pot bedanken, PhD bij de Optimalisatie van Bedrijfsprocessen groep aan de Vrije Universiteit, voor de buitengewoon handige tips die ik van hem kreeg, en waarvan ik begrepen heb dat die gewoonlijk tot het beroepsgeheim van PhD studenten behoort. Bedankt, dit heeft echt bijgedragen. Geen nood, ik zal het niet verder vertellen.

Ook wil ik hier graag Annemieke van Goor-Balk bedanken, coördinator bedrijfscontacten en lid van de afstudeercommissie aan de Faculteit der Exacte Wetenschappen van de Vrije Universiteit, voor de door haar betoonde interesse en haar begeleiding gedurende de gehele lengte van mijn afstudeertraject.

Eric Jonker
Broek op Langedijk, Augustus 2006

Samenvatting

Doelstelling van het Onderzoek

In dit onderzoek zijn de verwerkingstijden van een parallelle toepassing in een realistische gridomgeving onderzocht. Het doel van dit onderzoek is, om inzicht te verlenen in de invloed van steeds veranderende verwerkingstijden van taken op de totale verwerkingstijd van de toepassing. Omdat verwerkingstijden voortdurend veranderen, wordt wel gezegd dat deze fluctueren.

Probleemdefinitie

Een grid is een grootschalig netwerk van computers, die samenwerken zodat ze gecombineerd een ongekend hoog prestatieniveau bereiken, met als doel de grootste rekenintensieve problemen in de kortst mogelijke tijd op te lossen.

Het is een gegeven dat de verwerkingstijden van gelijke taken op computers in een realistische gridomgeving, gewoonlijk nodes genoemd, sterk van elkaar verschillen en ook voortdurend veranderen. Ook achtereenvolgens op dezelfde computer gemeten verwerkingstijden vertonen grote verschillen.

Hoofdlijnen van het Onderzoeksontwerp

In dit rapport wordt een eenvoudige methode uiteengezet, waarmee kan worden bepaald welke toename van de totale verwerkingstijd elke computer veroorzaakt door fluctuaties van de verwerkingstijd. Deze methode beschouwt een gegeven verzameling samenwerkende computers en bepaalt iteratief welke van deze de totale verwerkingstijd van de toepassing het sterkste verhoogt. Deze computer wordt vervolgens verwijderd uit de verzameling en het proces wordt herhaald, tot de verzameling leeg is.

Verder is vastgesteld dat de computers die de grootste toename van de verwerkingstijd veroorzaken, ook vaak bij de traagsten behoren gedurende opeenvolgende metingen van de verwerkingstijd van gelijke taken die in parallel worden uitgevoerd. Dit wordt met een vergelijkbare methode vastgesteld, maar nu wordt het aantal metingen geteld waarin de computer de traagste was, in plaats van dat de totale toename van de verwerkingstijd wordt berekend. De volgorde waarin computers uit de verzameling worden verwijderd is echter nog steeds dezelfde als hierboven.

Op deze manier kan een relatie worden vastgesteld tussen de mate van fluctuatie in de vertragende effecten van de samenwerkende computers met het aantal computers dat vertragingen veroorzaakt. Deze relatie is sterk als een willekeurige selectie wordt gemaakt van samenwerkende computers, maar is steeds zwakker aanwezig als de computers die de verwerkingstijd het sterkste doen verhogen achtereenvolgens worden uitgesloten van samenwerking.

Het effect op de totale verwerkingstijd van de toepassing wordt vervolgens gesimuleerd indien een willekeurige selectie van computers samenwerkt, waarbij verschillende aantallen computers met het sterkste vertragende effect vooraf worden uitgesloten van samenwerking.

Hoofdconclusies

Een klein deel van de computers die in parallel meewerken aan een toepassing veroorzaken een belangrijk deel van de totale verwerkingstijd. 8 van elke 100 computers zijn verantwoordelijk voor 68% van de totale verwerkingstijd en slechts éénderde veroorzaakt 95% hiervan.

De computers die de totale verwerkingstijd het sterkste beïnvloeden kunnen bovendien eenvoudig worden geïdentificeerd. De totale verwerkingstijd kan daarom direct worden verlaagd, bijvoorbeeld door trage computers uit te sluiten van meewerking.

Het is dus mogelijk om op eenvoudige wijze de totale verwerkingstijd van een toepassing sterk terug te brengen. Anders gezegd, de prestaties van parallelle toepassingen in een realistische gridomgeving kunnen op eenvoudige wijze behoorlijk worden verhoogd.

Inhoudsopgave

Introductie

1. Grid Computing

1.1. De Alsmaar Stijgende Vraag naar Verwerkingscapaciteit

Computers en software zijn *commodities* geworden, goedkoop en beschikbaar voor iedereen. Dat is niet van de ene op de andere dag zo gegaan. Langzamerhand is een brede industrie gegroeid rondom de technologie, terwijl er in eerste instantie slechts een handjevol specialistische experts waren. Een kenmerkende eigenschap van de computertechnologie is dat deze zich met enorme snelheid heeft ontwikkeld en dat de enorme populariteit die de technologie geniet zowel reden is als doel van de ontwikkeling ervan. Vandaag de dag is er een enorme markt voor de technologie gegroeid en zijn de financiële belangen navenant toegenomen.

Computers worden vandaag de dag gebruikt door vrijwel alle organisaties en consumenten, voor academische doeleinden, zeer uiteenlopende administratieve doeleinden en voor ontspanning en vermaak. De hedendaagse brede toepasbaarheid is grotendeels te danken aan de ontwikkeling van de personal computer. Terwijl voorheen de computers onhandelbaar groot waren, was het plotseling mogelijk om er één onder de arm te nemen naar huis, waar deze dan ook nog kon worden gebruikt. Dit betekende een revolutie. En hoe gewoon dit vandaag de dag ook is, nog steeds wordt gezocht naar kleinere, lichtere, krachtigere ontwerpen van apparaten zoals telefoons en zakagenda's.

De ontwikkeling van andere dan personal computers, zoals supercomputers, heeft overigens sinds de ontwikkeling van de personal computer niet stilgestaan. Die ontwikkeling is echter onveranderd specialistisch gebleven. Vandaag de dag worden supercomputers bijvoorbeeld ingezet bij bedrijven, academische instellingen en overheden, daar waar buitengewone vereisten gelden.
Denk bijvoorbeeld aan real-time besturing van productierobots, werkelijk enorme databasesystemen van zeer grote bedrijven en het berekenen van weersvoorspellingen. Het is deze laatste toepassing van de computertechnologie, voor grote rekenproblemen, die in dit onderzoek centraal staat.
Eén dergelijke toepassing is bijzonder bekend, want af en toe de supercomputer gebruiken voor een schaakwedstrijd is leuk en heeft het prettige neveneffect van gratis publiciteit.

> **Kasparov verslaat supercomputer** (Sci-Tech 2003, door Madison Gray) Wereldkampioen schaken Garry Kasparov versloeg zondag uitdager Deep Blue in de eerste van zes sets in een wedstrijd tussen menselijk verstand en computerlogica.

Dat een speciaal ontworpen schaakcomputers niet kon winnen van een echt goede menselijke tegenstander is misschien niet erg belangrijk, maar wel zeer tekenend. De zoektocht naar steeds snellere verwerkingscapaciteit is in volle gang, waarbij continu vooruitgang wordt geboekt. Dit heeft tot gevolg gehad dat schaken tegen de computer in de afgelopen jaren steeds moeilijker is gebleken voor de menselijke grootmeesters.

> **Supercomputer verslaat grootmeester** (http://tournament.hydrachess.com, 2005) Grootmeester Michael Adams heeft verloren van Hydra, een supercomputer die 200 miljoen zetten per seconde kan berekenen. De scheidsrechter van de wedstrijd, Albert Vasse, zei dat hij nog nooit een mens zo goed tegen een machine had zien spelen en gaf toe dat mensen wellicht niet meer van computers zullen kunnen winnen.

Ondanks de enorme rekenkracht van hedendaagse supercomputers hebben deze belangrijke tekortkomingen. Ten eerste zijn supercomputers duur. Ook in gebruik zijn ze duur, mede omdat speciale software nodig is die geschikt moet zijn voor de supercomputer. Daarom is de aanschaf van een supercomputer zelden het overwegen waard. Verder zijn er ondanks de rekenkracht rekenproblemen genoeg die eenvoudigweg te lang duren op zelfs de snelste supercomputer.
De ontwikkeling van snellere hardware zal echter steeds moeilijker en duurder worden, terwijl de vraag naar rekenkracht voor bijvoorbeeld grote simulaties en modelleringen alsmaar toe zal nemen.

Om toch over voldoende rekenkracht te kunnen beschikken, worden soms grote aantallen servers en personal computers aangeschaft en gelijktijdig aan het werk gezet. De kosten blijven echter hoog en dat terwijl er overal op de wereld miljoenen computers zijn die slechts enkele uren per dag worden gebruikt. Het onderzoek naar grid computing heeft als doel efficiënt gebruik te kunnen maken van deze ongebruikte rekencapaciteit. Zo kan een computersysteem worden gevormd, een netwerk eigenlijk, die de prestaties van welke supercomputer dan ook kan doen verbleken.

In dit hoofdstuk wordt Grid Computing geïntroduceerd. In sectie 1.2. wordt uiteengezet hoe de prestaties van toepassingen in het algemeen kunnen worden verbeterd. De grenzen van de op dit moment haalbare prestaties worden opgezocht en er wordt verteld hoe deze kunnen worden overschreden in toekomstige systemen. Sectie 1.3. beschrijft de geschiedenis van Distributed Computing, de voorloper van Grid Computing. De laatste sectie wijst op de beperkingen van Distributed Computing, beargumenteert waarom deze beperkingen moeten worden opgeheven en introduceert Grid Computing als de oplossing.

1.2. Prestatieverhogende Maatregelen

Om de prestaties van een computersysteem te verhogen, kunnen twee soorten maatregelen worden genomen.

- Scale up – Het verhogen van de rekencapaciteit van de computers waarop de toepassing wordt uitgevoerd.
- Scale out – Het vergroten van het aantal computers waarop de toepassing wordt uitgevoerd.

Het voordeel van Scale up is, dat vrijwel elke toepassing baat heeft bij deze prestatieverhogende maatregel. Het nadeel is echter, dat er nieuwe computerhardware moet worden aangeschaft en dat de al bestaande niet meer wordt gebruikt, zodat dit een dure maatregel is. Voor grote rekenproblemen zal de rekencapaciteit van de beschikbare technologie bovendien altijd een probleem blijven vormen.

Scale out heeft als grootste nadeel dat de software hiervoor geschikt moet zijn. Echter, in gevallen waarin dit zo is, is deze maatregel vaak uitermate flexibel en efficiënt.

Scale out kan op verschillende manieren worden uitgevoerd. Vaak wordt deze maatregel in het bedrijfsleven toegepast om zogenaamde clusters te vormen. In een cluster nemen meerdere, identieke servers elk een deel van het werk voor hun rekening. Een cluster wordt beheerd door één organisatie of administratieve eenheid en bevindt zich in de regel op één enkele locatie, waardoor configuratie en beheer relatief eenvoudig is.

Hoewel clusters tot zeer hoge prestaties kunnen komen, zijn er toepassingen en situaties waarvoor de aanschaf van een voldoende groot aantal servers niet haalbaar is. In plaats daarvan kan de capaciteit van al beschikbare computers worden gebruikt, aangezien het vaak zo is dat de beschikbare rekencapaciteit op veel computers bij lange na niet gedurende de hele dag volledig wordt gebruikt – en zelfs niet over kortere periodes.

De oorzaak hiervan is, dat de meeste softwaretoepassingen slechts werk hoeven te verrichten op verzoek van de gebruiker, zij het interactief dan wel via een netwerk. De hoeveelheid werk die elke interactie met zich meebrengt is in de regel beperkt. Wanneer geen interactie plaatsvindt, wordt geen werk verzet.

> **Voorbeeld** Een secretaresse met zeer hoge typevaardigheid die zo snel mogelijk typt is slechts in staat om enkele honderden malen per minuut de inhoud van een document te veranderen. Toch zijn moderne computers in staat om zeer omvangrijke documenten ruimschoots binnen één enkele minuut in hun geheel op te maken, bijvoorbeeld voorafgaand aan het afdrukken ervan, waar toch aanzienlijk meer verwerkingscapaciteit voor nodig is.

Meestal is het zo, dat de gebruiker die de interactie initieert letterlijk wacht op antwoord. De prestatie-eisen die aan computers gesteld worden zijn dan ook vaak zó hoog, dat de wachttijden die gebruikers ervaren acceptabel kort kunnen blijven. Ook is het regelmatig zo dat het prestatieniveau voldoende moet zijn om een zekere, zware taak uit te voeren, die echter slechts af en toe hoeft te worden uitgevoerd. In het eerste geval zal de computer frequent korte periodes van inactiviteit ervaren, in het laatste geval zijn er lange periodes van inactiviteit.

Zulke periodes van inactiviteit kunnen aangewend worden om andere taken uit te voeren. Deze toepassing van verwerkingscapaciteit ligt aan de basis van Grid Computing. De eenvoudigere wijze waarop deze praktijk vandaag de dag al wordt toegepast, wordt Distributed Computing genoemd.

1.3. Distributed Computing

Distributed Computing is het aanwenden van ongebruikte verwerkingscapaciteit op computers die verbonden zijn via een netwerk, om deze computers in parallel te laten werken aan hetzelfde probleem of dezelfde taak. Distributed Computing kan gezien worden als de voorloper van Grid Computing.

Distributed Computing heeft zich ontwikkeld gelijktijdig met het Internet. Al in 1970 werd hiermee geëxperimenteerd op ARPAnet, de voorloper van het Internet. De experimentele programmaatjes genaamd Creeper and Reaper kunnen worden gezien als de eerste software voor Distributed Computing, of ook als de eerste netwerkvirussen. Creeper had geen ander doel dan zich te verspreiden via het netwerk en Reaper volgde, waarbij deze Creeper weer verwijderde van de computers. In 1973 werd vergelijkbare software voor het eerst door Shoch en Hupp van Xerox's Palo Alto Research Center ontwikkeld om complexe berekeningen uit te voeren op ongebruikte computers. (Shoch en Hupp, 1982).

Serieuze toepassing van de principes van Distributed Computing op grotere schaal liet op zich wachten tot 1988, toen het DEC System Research Center e-mail gebruikte om vrijwilligers taken te sturen, die zij dan op hun computer uitvoerden. De resultaten werden weer via e-mail teruggestuurd. In de tijd vanaf 1990 was RSA Security, Inc een belangrijke promotor van een gelijksoortige vorm van Distributed Computing. Het onderzoekscentrum van dit bedrijf, RSA Laboratories, liet consumenten wedijveren om een berekening zo snel mogelijk te voltooien en betaalde daar ook voor.

Verschillende bronnen noemen distributed.net als eerste bedrijf dat Distributed Computing op grote schaal toepaste. In 1997 werd begonnen met een project dat als doel had het breken van een bericht, versleuteld met een 56-bits geheime sleutel. Het versleutelde bericht was afkomstig van opnieuw RSA Security, Inc. De sleutel werd in ongeveer 9 maanden gevonden, met behulp van computers van vrijwilligers, aangesloten via het Internet. De prestaties van dat netwerk nam in de laatste 8 maanden van het project echter toe met maar liefst een factor 200. Een even complexe sleutel zou aan het einde van het project naar verwachting binnen 2,5 maanden zijn gevonden. Distributed.net heeft daarna meerdere, soortgelijke projecten ondernomen. (distributed.net)

Om mee te werken moet een computer expliciet op vrijwillige basis voor een project beschikbaar worden gesteld. Anderen werken op dezelfde manier. Zo wordt er gezocht naar zeer grote priemgetallen in het project GIMPS. Een zeer populair project is SETI@Home van de universiteit van Berkeley, met als doel het zoeken naar buitenaards leven. Dit wordt gedaan door enorme hoeveelheden radiosignalen uit de ruimte op te nemen, ruis eigenlijk, en op regelmatigheden te onderzoeken. Anderen doen projecten op het vlak van medisch en biologisch onderzoek.

Overigens hoeft de kracht van Distributed Computing niet alleen in rekencapaciteit te liggen, ook een enorme opslagcapaciteit of bandbreedte hebben toepassingen. Zo kunnen bijvoorbeeld peer-to-peer netwerken voor bestandsdeling, zoals Kazaa en voorheen Napster, worden gezien als Distributed Computing netwerken.

1.4. De Evolutie naar Grid Computing

De grootste Distributed Computing projecten worden echter vrijwel alle door academici uitgevoerd en hebben een beperkt praktisch nut. Zakelijke toepassing is er ook, maar voordat zakelijke dienstverlening daadwerkelijk succesvol kan zijn, zal het Distributed Computing netwerk moeten evolueren. Ten eerste zijn de huidige Distributed Computing netwerken in grote mate toepassingafhankelijk. Verder ontbreekt het op technisch, administratief en sociaal vlak aan standaarden en afspraken die een bredere en efficiëntere toepassing mogelijk zouden kunnen maken.

Elementaire zaken zoals beveiliging, beleid en betaling voor beschikbaar gestelde computercapaciteit zal onderdeel moeten worden van een architectuur. Nu moet nog voornamelijk de betrouwbaarheid van de coördinator van het netwerk worden verondersteld en moet gerekend worden op de loyaliteit en sympathie van de vrijwilliger. Op technisch vlak zijn standaarden en protocollen nodig die nog maar gedeeltelijk zijn ontwikkeld, zodat het ontwikkelen van software bemoeilijkt wordt. Een lucratieve architectuur heeft immers een rijk softwareaanbod nodig. Zo zijn er standaarden nodig voor authenticatie, autorisatie en het vinden en inzetten van bronnen en capaciteiten. Verder zijn er

oplossingen nodig op het gebied van bijvoorbeeld beschikbaarheid en prestatieniveau, die gaan spelen wanneer het netwerk wordt gebruikt door verschillende partijen tegelijkertijd, voor verschillende toepassingen.

De term Grid Computing is ontstaan in de jaren ‘90 als een metafoor voor het beschikbaar stellen van computercapaciteit op een manier die vergelijkbaar is met stroom van het elektriciteitsnetwerk en net zo eenvoudig. Foster (2002) definieert een grid als een systeem dat bronnen coördineert die niet gecentraliseerd worden beheerd, met behulp van standaarden en open protocollen die niet voor een specifieke toepassing zijn ontwikkeld, met als doel dat het systeem in staat is diensten te leveren van een kwaliteit die superieur is aan dat van een Distributed Computing netwerk, juist dankzij deze coördinatie.

Een grid is dus eenvoudig gezegd een Distributed Computing netwerk met een gestandaardiseerde software architectuur, waarop uiteenlopende toepassingen tegelijkertijd kunnen worden uitgevoerd. De prestaties en mogelijkheden van de toepassingen op dit netwerk zijn superieur ten opzichte van de prestatie op een Distributed Computing netwerk zónder een dergelijke architectuur.

2. Parallellisme en Werkdrukverdeling

In dit hoofdstuk wordt dieper ingegaan op de precieze werking van grids. Sectie 2.1. beschrijft in detail hoe ongebruikte verwerkingscapaciteit kan worden gebruikt voor gridtoepassingen en geeft een overzicht van de moderne technologie die dit mogelijk maakt. De daaropvolgende sectie beargumenteert dat softwaretoepassingen specifiek geschikt moeten worden gemaakt om efficient parallel te kunnen worden uitgevoerd. In sectie 2.3. wordt load-balancing behandeld als methode om de prestaties van parallelle toepassingen te optimaliseren terwijl ze worden uitgevoerd. Er wordt gesteld, dat een dynamische strategie noodzakelijk is om te kunnen reageren op veranderende situaties, in het bijzonder in een grid. Sectie 2.4. tenslotte behandelt complicaties van load-balancing in een gridomgeving, die bijvoorbeeld worden veroorzaakt door het ontbreken hierin van een besturingssysteem of andere centrale coördinatie. Ook worden ter illustratie enkele recente onderzoeken aangestipt, waarin oplossingen voorgesteld worden om de gridspecifieke problematiek aan te pakken.

2.1. 'Enabling Technology': Multi-Threading en Cycle Stealing

Moderne besturingssystemen staan meerdere toepassingen toe, tegelijkertijd te worden uitgevoerd. Dit is een vorm van parallelisme of *multi-tasking*. Elke toepassing bestaat uit één of meerdere taken, *threads,* die gelijktijdig kunnen worden uitgevoerd, maar in de regel frequent informatie delen. Elke keer als er informatie wordt gedeeld, moeten de taken worden gesynchroniseerd. De één wacht dan op de ander, zodat de laatste toegang heeft tot een consistente versie van de informatie.

In het bijzonder wanneer een computer slechts één centrale verwerkingseenheid heeft, is de coördinatie van bronnen door het besturingssysteem cruciaal. De actieve taken lijken tegelijkertijd te worden uitgevoerd, maar deze schijn ontstaat slechts doordat het besturingssysteem de actieve taken één voor één toestaat gebruik te maken van de centrale verwerkingseenheid, waarbij zeer snel tussen taken wordt gewisseld. Een hoeveelheid executietijd die een taak toegewezen krijgt wordt een *timeslice* genoemd.

De interactiviteit en het prestatieniveau van het systeem worden gewaarborgd doordat het besturingssysteem actief voorkomt dat één taak de centrale verwerkingseenheid bezet houdt. Zo blijft elke toepassing responsief. Vanwege de preventieve rol van het besturingssysteem wordt deze vorm van parallellisme wel *pre-emptive multi-tasking* genoemd. Als een taak wacht op voltooiing van een I/O-bewerking of op interactie van de gebruiker kent het besturingssysteem geen timeslice toe. In principe kan het besturingssysteem zo taken onbeperkt uitstellen. Daarnaast kan een taak zelf aangeven dat het voorlopig geen executietijd nodig heeft.

> **Voorbeeld** Op Microsoft® Windows NT™ gebaseerde besturingssystemen kennen aan elke taak een prioriteitsklassering toe. Taken krijgen alleen executietijd toegewezen als alle taken met hogere prioriteitsklassering aangeven voorlopig geen executietijd nodig te hebben.

Ongeacht het aantal centrale verwerkingseenheden beschikbaar in een computersysteem is pre-emptive multi-tasking noodzakelijk. Er zijn vrijwel altijd meer taken dan centrale verwerkingseenheden en dat is ook efficiënt. Bovendien maakt het mogelijk dat een grid gebruik kan maken van computers die primair voor andere doeleinden worden gebruikt, zonder ooit hinderlijk te zijn voor de primaire gebruikers van een systeem.

De toewijzing van timeslices aan een taak die is gestart door een grid gebeurt alleen als andere taken geen executietijd nodig hebben. Deze praktijk wordt wel *cycle-stealing* genoemd, omdat de tijdseenheid waarin centrale verwerkingseenheden rekenen cycles worden genoemd.

> **Voorbeeld** Door pre-emptive multi-tasking toe te passen kan het besturingssysteem garanderen dat alle taken voldoende timeslices toegewezen blijven krijgen, ongeacht de hoeveelheid werk, van lagere prioriteit, die wordt aangeboden door een grid. Hierdoor kunnen de prestaties van taken met hogere prioriteit onaangetast blijven, terwijl zo min mogelijk verwerkingscapaciteit verloren gaat.

Wanneer een computer meer dan één centrale verwerkingseenheid heeft, biedt pre-emptive multi-tasking extra voordelen. Taken die enkel wachten op een nieuwe timeslice kunnen tegen geringe kosten wisselen van centrale verwerkingseenheid. Executie ervan kan dus worden voortgezet, vóórdat er opnieuw een timeslice kan worden toegewezen op dezelfde centrale verwerkingseenheid

als voorheen. Hierdoor kan de beschikbare verwerkingscapaciteit op eenvoudige wijze min of meer optimaal worden benut.

Hoewel executietijd niet de enige systeembron is waarvoor coördinatie gewenst is, is het wel verreweg de belangrijkste. Vrijwel elk gebruik van randapparatuur impliceert namelijk gebruik van executietijd, aangezien er bijna altijd wel aansturing nodig is door een centrale verwerkingseenheid. Soms meer, soms minder. Meestal is er bijvoorbeeld transport van gegevens nodig tussen het hoofdgeheugen en de randapparatuur, in welk geval de centrale verwerkingseenheid de bronnen voor directe geheugentoegang moet toewijzen aan de randapparatuur en de overdracht van gegevens initieert.

2.2. Ontwerpen van Parallelle Toepassingen: 'Scalable by Design'

In het voorgaande hoofdstuk zijn de prestatie verhogende maatregelen Scale up en Scale out geïntroduceerd, waarmee de prestaties van een willekeurige toepassing kunnen worden verhoogd. Er werd gesteld dat Scale out een uitermate flexibele en efficiënte methode kan zijn, maar dat de toepassing ontworpen moet zijn om er baat bij te hebben.

In hoeverre een toepassing geschikt is voor parallellisme is in hoge mate afhankelijk van het probleem waarvoor de toepassing een oplossing behelst. Een toepassing die op eenvoudige wijze gebruik kan maken van beschikbare extra capaciteit wordt *scalable* genoemd. Anders gezegd, de mate waarin de prestaties toenemen bij de beschikbaarheid van extra hardware, in het bijzonder centrale verwerkingseenheden, bepaalt de schaalbaarheid van de toepassing.

Een toepassing is uitstekend schaalbaar als de prestaties onder ideale omstandigheden volledig proportioneel toenemen met de beschikbare verwerkingscapaciteit. Dit is echter in de praktijk niet altijd zo. Het is niet vanzelfsprekend dat het probleem waarvoor de toepassing wordt ontwikkeld wel een uitstekende schaalbaarheid toestaat. Afhankelijkheden tussen berekeningen kunnen ertoe leiden dat de centrale verwerkingseenheden steeds vaker op elkaar moeten wachten als er steeds meer meewerken. Het kan wel zo zijn dat een toepassing theoretisch gesproken uitstekend schaalbaar is, maar dan leiden suboptimale omstandigheden, zoals fluctuaties in verwerkingstijden, tot toenemende vertraging.

Er kan worden gesteld dat een toepassing op een tweetal manieren kan worden ontworpen om schaalbaar te zijn. Enerzijds kan het algoritme in delen worden gesplitst, anderzijds kunnen de gegevens die het algoritme verwerkt in delen worden gesplitst. Elk deel wordt dan verwerkt door verschillende centrale verwerkingseenheden, die zich mogelijk in verschillende computers bevinden.

Gelijke algoritmes kunnen dus op verschillende centrale verwerkingseenheden verschillende gegevens verwerken. Vaak is het bij deze aanpak echter nog steeds zo, dat regelmatig gegevens moeten worden uitgewisseld tussen de verwerkingseenheden. Afhankelijk van het probleem soms alleen met enkele, logischerwijs naastgelegen medewerkers, soms met meerdere of zelfs alle. In de ontwerpfase is het van belang te proberen dit aantal zoveel mogelijk te reduceren. Deze aanpak vereist bovendien een opstartfase, waarin gegevens worden gedistribueerd.

Als samenwerkende centrale verwerkingseenheden verbonden zijn door een relatief traag communicatiemedium, zoals het Internet, dan zijn verzend- en ontvangstvertragingen van invloed. Deze vertraging is de tijd, nodig voor het verzenden van een bericht zónder informatie van de verzender naar de ontvanger, of in omgekeerde richting. Deze vertragingen worden ook wel *latency* of *lag* genoemd en de som de *round-trip latency*. De latency wordt gemeten exclusief verwerkingstijd. Indien deze wordt meegerekend is sprake van *responstijd*.

De invloed van de latency op de responstijd is groter, naarmate de communicatie korter is en frequenter nodig. Ook is het van belang of de communicatie synchroon dient plaats te vinden of asynchroon kan worden uitgevoerd. In het laatste geval kan de centrale verwerkingseenheid verder werken terwijl gegevens worden ontvangen of verzonden, zolang er tenminste werk voorhanden is.

> **Voorbeeld** Toepassingen die via het Internet communiceren, presteren beter als minder frequent communicatie nodig is, aangezien verzend- en ontvangstvertragingen hier een belangrijke rol kunnen spelen.

Overigens is het niet eenvoudig om ervoor te zorgen dat niet alleen de consistentie van de gegevens wordt gewaarborgd door alle toegang tot gegevens te bewaken, maar dit ook nog zó, dat verschillende centrale verwerkingseenheden zo min mogelijk op elkaar hoeven te wachten. Als een toepassing geschikt is om op meerdere centrale verwerkingseenheden te worden uitgevoerd,

betekent dit niet automatisch dat deze ook op meerdere computers kan worden uitgevoerd. Beide situaties vereisen een expliciet ontwerp.

2.3. De Noodzaak van Dynamische Strategieën voor Werkdrukverdeling

Wanneer een toepassing op meerdere computers wordt uitgevoerd, is het optimaal om elke computer werk te laten uitvoeren naar ratio van capaciteit. Dit wordt *load-balancing* genoemd. Er is een belangrijk onderscheid tussen statische en dynamische load-balancing. Bij statische load-balancing wordt de werkverdeling vooraf bepaald, of deze is deterministisch. Bij dynamische load-balancing wordt de daadwerkelijk ervaren werkdruk in overweging genomen bij het verdelen van de werkdruk. Het doel is deze zó te verdelen dat het prestatieniveau zo hoog mogelijk is. Er zijn vele vormen van load-balancing, zie bijvoorbeeld Shekhar et al (1995) en Ledlie en Seltzer (2005).

Een eenvoudige vorm van statische load-balancing is *round-robin*, waarbij medewerkende computers om beurten taken krijgen toegekend. Zo krijgt elke computer op de lange termijn evenveel taken te verwerken, die dan gemiddeld dezelfde werkdruk vertegenwoordigen. Dit is in het bijzonder effectief als alle taken van zeer beperkte omvang zijn en de computers vrijwel identiek.

Echter, wanneer de taken omvangrijk zijn of in grootte sterk van elkaar verschillen, of als de computers van elkaar verschillen, kan statische load-balancing eenvoudigweg ontoereikend blijken. Het blijkt in veel situaties moeilijk te zijn vooraf met voldoende zekerheid te voorspellen hoe zwaar elke computer zal worden belast, zowel in een clusteromgeving als op een Distributed Computing netwerk, maar in het bijzonder in een grid.

2.4. Complicaties bij Dynamische Werkdrukverdeling in een Gridomgeving

De coördinerende softwareonderdelen van het grid zoals gedefinieerd in Foster (2002) die de kwaliteit van de dienstverlening van het grid dienen te waarborgen, in het bijzonder load-balancing oplossingen, zijn in hoge mate verschillend van de coördinerende onderdelen van moderne besturingssystemen. Niettemin is de basisopzet gelijk. Zo worden toepassingen gesplitst in taken, nu in *processen* in plaats van in threads, die na elkaar op dezelfde, of tegelijkertijd op verschillende computers kunnen worden uitgevoerd.

> **Definitie** Een *node* is een computer die verbonden is aan een Distributed Computing netwerk, een grid of ander netwerk.

De belangrijkste verschillen zijn, dat in een grid de coördinatie decentraal plaatsvindt en er geen informatie voorhanden is over alle toepassingen die op het grid worden uitgevoerd, zodat er dus geen autoriteit is die taken kan prioriteren en bronnen kan toewijzen of beschermen. De kosten van migratie, dat wil zeggen een taak op een andere node verder uitvoeren, zijn veel hoger wanneer een proces van node wisselt dan wanneer een thread van centrale verwerkingseenheid wisselt.

Het is bijvoorbeeld mogelijk dat een node in het grid gedurende enige tijd uitstekend presteert, maar dat plotseling het prestatieniveau drastisch afneemt, of dat de verbinding met het grid wordt verbroken voordat alle aan de node toegewezen taken zijn voltooid. Dit alles beperkt vooralsnog de effectiviteit van load-balancing methoden wanneer deze toegepast worden in een gridomgeving.

Een grid omgeving is fundamenteel verschillend van een cluster omgeving, aangezien de servers in een cluster prestatieniveaus hebben die in de regel zeer goed vergelijkbaar zijn. De werkdruk van de processoren fluctueert in een cluster minder en deze kan eenvoudiger worden bijgestuurd, omdat bekend is welke toepassingen worden uitgevoerd. Ook vindt de communicatie tussen nodes in de regel op een hoge en bekende snelheid plaats. Verder is het aantal servers bekend en kan deze vrijwel zonder uitzondering constant worden verondersteld. Deze verschillen hebben tot gevolg dat toepassingen die goed presteren in een cluster geregeld slecht presteren in een gridomgeving.

Een zelfde vergelijking kan worden gemaakt tussen prestaties van toepassingen op een Distributed Computing netwerk en in een gridomgeving, hoewel de resultaten minder uitgesproken zijn. De voornaamste reden is, dat Distributed Computing netwerken in de regel toepassingsgebonden zijn, zodat er slechts één toepassing tegelijkertijd wordt uitgevoerd op elke node. De capaciteiten van de nodes in een Distributed Computing netwerk blijken constanter te zijn en beter te achterhalen of te voorspellen dan dat het geval is in een realistische gridomgeving.

Een oplossing voor het ontbreken van een centrale coördinerende autoriteit is, om standaard softwareonderdelen te ontwikkelen die door ontwikkelaars van grid toepassingen aan deze toepassing zelf kunnen worden toegevoegd. Zie bijvoorbeeld Dinda (2002), waarin de toepassing

zelf verantwoordelijk wordt gesteld voor het selecteren van nodes waarop executie wordt overwogen. Ook het verzamelen van prestatiestatistieken en het voorspellen van de eigen prestatie op de nodes is aan de toepassing zelf. Op basis van deze gegevens dient de toepassing zelf de meest geschikte nodes te selecteren en verwerking erop te beginnen.

> **Voorbeeld** Deze aanpak blijkt alleen goed te werken als bij de selectie van nodes een zekere willekeurigheid geforceerd wordt. Wanneer dit niet gebeurt, is de kans groot dat verschillende toepassingen onafhankelijk van elkaar dezelfde, best presterende nodes selecteren. Wat zou kunnen leiden tot overbelasting van deze nodes. Dit is een goed voorbeeld van een probleem dat niet zou optreden indien coördinatie centraal zou kunnen plaatsvinden.

Kovács et al (2004) behandelen verder een mechanisme dat toepassingen in staat stelt zichzelf te migreren. Het mechanisme werkt geheel toepassingonafhankelijk, hoewel ook hier de noodzakelijke software onderdeel vormt van de toepassing zelf en niet van een externe toepassing of besturingssysteem. Dit mechanisme is bovendien in staat een toepassing fouttolerant te maken, door zogenaamde checkpoints op te slaan. Dit komt bijvoorbeeld van pas wanneer nodes onaangekondigd van het grid verdwijnen. De toepassing kan in dat geval een opgeslagen checkpoint gebruiken om executie voort te zetten vanuit een recente, consistente situatie.

3. Verwerkingstijden en Fluctuaties

In dit hoofdstuk wordt uiteengezet waarom het onderzoek naar verwerkingstijden en hun fluctuaties van belang is voor het onderzoek naar grids en de prestaties van toepassingen in een gridomgeving.

Ten eerste wordt in sectie 3.1. de theoretische situatie behandeld waarin fluctuaties ontbreken. Sectie 3.2. verleent vervolgens inzicht in de systematisch invloed van fluctuaties op de totale verwerkingstijd en verwoordt het hoofddoel van dit onderzoek. Sectie 3.3. plaatst dit onderzoek tenslotte binnen het grotere kader van de ontwikkeling van load-balancing strategieën.

3.1. Optimale Verwerkingstijden – Een Utopie

Zoals beschreven in het vorige hoofdstuk is het nodig een toepassing expliciet geschikt te maken zodat deze op meerdere centrale verwerkingseenheden tegelijkertijd kan worden uitgevoerd. De analyse van de prestaties van parallelle toepassingen wordt in dit onderzoek beperkt tot de verwerkingstijden. In hoofdstuk 2 is gesteld dat ook communicatietijden van invloed kunnen zijn, maar deze worden in dit onderzoek volledig buiten beschouwing gelaten.

Er werd in dit hoofdstuk gesteld dat het werk in het optimale geval zó wordt gedeeld, dat aan elke beschikbare centrale verwerkingseenheid werk wordt toebedeeld naar ratio van beschikbare capaciteit. In dit geval voltooien immers alle centrale verwerkingseenheden de aan hen toegewezen deeltaken tegelijkertijd, aangenomen dat ze ook gelijktijdig begonnen zijn, zodat geen verwerkingscapaciteit verloren gaat. Elke centrale verwerkingseenheid presteert dan naar zijn beste vermogen gedurende de gehele tijd dat er deeltaken worden uitgevoerd, zodat de totale verwerkingstijd in dit geval minimaal is.

Wanneer deeltaken toegewezen zijn aan een centrale verwerkingseenheid, maar deze heeft de taken zover voltooid dat moet worden gewacht op meewerkende centrale verwerkingseenheden, gaat rekencapaciteit verloren. De tijd die een centrale verwerkingseenheid niet besteedt aan verwerking wordt *idle-time* genoemd. Idle-time kan dus ook voortkomen uit communicatietijden.

> **Definitie** *Idle-time* is tijd gedurende welke een centrale verwerkingseenheid wacht op ontvangst van gegevens die noodzakelijk zijn om verder te kunnen werken. Dit is idle-time die van belang is voor een toepassing.

Het is overigens niet zo dat de centrale verwerkingseenheid geen ander nuttig werk kan doen gedurende idle-time. De verloren tijd verlengt echter de totale verwerkingstijd van de toepassing en dit wordt veroorzaakt doordat één of meerdere medewerkers moeten wachten op anderen.

3.2. Het Ontstaan van Fluctuaties

In dit onderzoek wordt, naast de communicatietijden, ook de optimaliteit van het toepassingsontwerp buiten beschouwing gelaten. Wel is de schaalbaarheid van belang, als abstracte eigenschap van het toepassingsontwerp. Het onderzoek is gebaseerd op verwerkingstijden van taken die voornamelijk verwerkingstijdintensief zijn en die op identieke hardware, onder ideale omstandigheden, precies even lang zouden duren. De toepassing waarmee de verwerkingstijden zijn gemeten is uitstekend schaalbaar genoemd, omdat deze in staat is om de werkdruk zeer precies naar ratio van capaciteit te verdelen, als tenminste de totale verwerkingsbehoefte groot genoeg is.

De toepassing verdeelt het werk gelijk over de meewerkende nodes, zodat de verwerkingstijden in het ideale geval gelijk zouden zijn. Toch is er sprake van idle-time. Deze komt zowel voort uit verschillen tussen de capaciteiten van de meewerkende nodes als uit hun veranderende werkdruk. Het is niet te achterhalen welke van deze redenen de idle-time veroorzaakt. Een gelijktijdige invloed van deze beide oorzaken moet echter worden verwacht bij het uitvoeren van een toepassing in een realistische, geografisch verspreide gridomgeving.

Wijzigingen in het prestatieniveau van een node over kortere of langere periodes hebben tot gevolg dat de verwerkingstijden van gelijke taken verschillend zijn als deze herhaaldelijk worden gemeten op dezelfde node. Dus, de gemeten verwerkingstijden vertonen fluctuaties en er ontstaat idle-time.

Het doel van dit onderzoek is om inzicht te verlenen in de invloed van deze fluctuaties op de verwerkingstijd van gridtoepassingen, zodat deze toepassingen in de toekomst minder gevoelig kunnen worden gemaakt voor deze fluctuaties en de prestaties ervan dus kunnen worden verbeterd.

3.3. De Noodzaak van Statistisch Onderzoek naar Verwerkingstijden en hun Fluctuaties

Dobbber et al (Mei 2006, pagina 150) identificeren de volgende doelstellingen voor onderzoek dat noodzakelijk is om toepassingen minder gevoelig te maken voor verschillende of wijzigende prestatieniveaus en veranderende beschikbaarheid van nodes in een grid.

- Inzicht verkrijgen in statistische eigenschappen van prestatieniveaus van nodes in een grid.
- Ontwikkeling van effectieve voorspellingsmethoden.
- Implementatie van dynamische load-balancing strategieën die de werkdruktoewijzing dynamisch kunnen aanpassen als antwoord op veranderende omstandigheden.

Zij noemen verder dat een goed inzicht in de karakteristieken van een grid zeer nuttig zal zijn om simulaties en berekeningen uit te voeren, waarmee de effectiviteit van voorspellingsmethoden en load-balancing strategieën kan worden beoordeeld, voordat een tijdrovend en duur realisatietraject wordt aangevangen. Deze noodzaak wordt goed geïllustreerd in Dobber et al (juni 2006), waarin wordt gesteld dat voorspellingsmethoden die bewezen hebben goed te presteren voor gesynthetiseerde verwerkingstijden of in laboratoriumexperimenten, in een realistische gridomgeving hun voorspellingskracht voor een deel verliezen.

Ook is er in de literatuur weinig aandacht besteed aan de karakteristieken van achtereenvolgende verwerkingstijden van een taak, hoewel getracht is deze te voorspellen op basis andere grideigenschappen, zoals de werkdruk. Zie bijvoorbeeld Dinda (2001) en Dinda en O'Hallaron (2002). Hoewel deze factoren in theorie een sterke relatie hebben, vinden zij dat het in de praktijk moeilijk blijkt te zijn de verwerkingstijden te voorspellen op basis van de werkdruk, omdat andere factoren, zoals de hoeveelheid beschikbaar geheugen, de relatie hiertussen verstoren. Het kan daarom zo zijn dat werkdruk en verwerkingstijd zeer verschillende karakteristieken vertonen en dus is het nodig de verwerkingstijden zelf te onderzoeken (Dobber et al, mei 2006, pagina 151)

Bovendien concluderen Dobber et al (mei 2006) dat het niet mogelijk is de verwerkingstijden te modelleren middels standaard kansverdelingen, wat simulaties en berekeningen in het bijzonder zou hebben vereenvoudigd. Dat dit wel mogelijk is voor sommige andere grideigenschappen blijkt onder meer uit Paxson en Floyd (1995), Mutka en Livny (1991), Leland en Ott (1986), Harchol-Balter en Downey (1996) en Kondo et al (2004).

Onderzoek

4. Vraagstelling en Gegevensverzameling

In dit hoofdstuk worden de onderzoeksvragen behandeld en de wijze waarop de onderzochte gegevens zijn verzameld.

In de eerstvolgende sectie worden de hoofdvragen gesteld waaraan dit onderzoek dient te beantwoorden en worden deze hoofdvragen ingeleid.

De daaropvolgende secties behandelen de verzameling van gegevens. Sectie 4.2. introduceert het onderzoek dat is verricht naar realistische verwerkingstijden van nodes in een grid. Sectie 4.2. behandelt de parallelle toepassing waarvan verwerkingstijden zijn gemeten. Sectie 4.3. tenslotte volgt een kort overzicht van enkele relevante en kenmerkende resultaten uit een voorgaand onderzoek naar de statistische eigenschappen van de gemeten verwerkingstijden.

In de laatste sectie in dit hoofdstuk tenslotte wordt uiteengezet hoe de gemeten verwerkingstijden worden geïnterpreteerd, welke aannames worden gemaakt en wordt beargumenteerd waarom de gemaakte aannames plausibel zijn.

4.1. Hoofdvragen

Zoals in de laatste sectie van het vorige hoofdstuk is uiteengezet, is het doel van het onderzoek naar de statistische eigenschappen van prestaties tweeledig. Enerzijds is inzicht hierin noodzakelijk om effectieve voorspellingsmethoden en load-balancing strategieën te kunnen ontwikkelen. Anderzijds kan dit inzicht bijdragen aan de ontwikkeling van simulatiemethoden en wiskundige modellen waarmee op efficiënte wijze onderzoek kan worden verricht naar zulke voorspellingsmethoden en strategieën vóór de implementatie ervan.

Dit onderzoek richt zich specifiek op fluctuaties van verwerkingstijden. Dit zijn veranderingen ten opzichte van een zeker gemiddelde of ander ijkpunt. De waarde van dit ijkpunt wordt dan ook als gegeven beschouwd. Het is overigens ook zo dat bijvoorbeeld een gemeten gemiddelde verwerkingstijd niet indicatief is voor het te verwachten prestatieniveau, juist vanwege fluctuaties. Zo is het minimum van een serie gemeten verwerkingstijden de beste indicatie van de snelheid van een node, maar is er geen enkele garantie dat de node niet nog veel sneller is. Hoewel het prestatieniveau aan een door de hardware bepaald maximum is gebonden, kan het hier buitengewoon sterk van afwijken, mogelijk zonder uitzondering. Ook de mate waarin de verwerkingstijden fluctueren over tijdsperiodes van verschillende lengte zijn in hoge mate verschillend op verschillende nodes (Dobber et al, mei 2006, pagina 151-152).

Wat in de vorige sectie is beargumenteerd en vaststaat, maar niettemin zal worden gedemonstreerd, is dat de fluctuaties een negatieve invloed hebben op de totale verwerkingstijd. De vraag is, in welke mate de totale verwerkingstijd wordt beïnvloed en in hoeverre de mate waarin de applicatie parallel wordt uitgevoerd hierop van invloed is. Dus, verandert de invloed van fluctuaties in positieve of negatieve zin, als het aantal nodes waarop de toepassing wordt uitgevoerd toe- of afneemt?

Wat gebeurt er als wordt gesimuleerd dat de nodes geen fluctuaties hebben in hun opeenvolgende verwerkingstijden? In dit geval presteren de nodes allemaal op een vast prestatieniveau, maar deze vaste prestatieniveaus fluctueren nog steeds tussen de nodes. Hoe verandert de totale verwerkingstijd als de nodes een vast prestatieniveau zouden hebben?

Een methode die een dynamische load-balancing strategie zou kunnen gebruiken, is het uitsluiten van trage nodes. De toepassing wordt verwerkingscapaciteit dan ontzegd. Kan er winst worden verwacht van een dergelijke aanpak, die toch op zichzelf gezien paradoxaal is? Kan er een getalsmatige indicatie worden gegeven voor de gevolgen van een dergelijke aanpak?

4.2. De Oorsprong van de Verzamelde Gegevens

Dit onderzoek is gebaseerd op gegevens verkregen gedurende het onderzoek waarvan de resultaten gepresenteerd zijn in Dobber et al (2004).

Dobber et al (2004, pagina 344) voeren uitgebreide experimenten uit in een testomgeving.
Een vereiste voor de testomgeving is, dat het de intrinsieke eigenschappen van een realistische

gridomgeving heeft. Namelijk, dat nodes uiteenlopende capaciteiten hebben en er grote fluctuaties zijn in de werkdruk zowel als de prestaties van de nodes, die zich bij voorkeur op wereldwijd verspreide locaties bevinden. De experimenten worden uitgevoerd op PlanetLab, dat aan deze vereisten voldoet. PlanetLab is een open, globaal netwerk voor het delen van verwerkingscapaciteit en voor de ontwikkeling en het uitrollen van netwerkdiensten over de gehele planeet (PlanetLab)

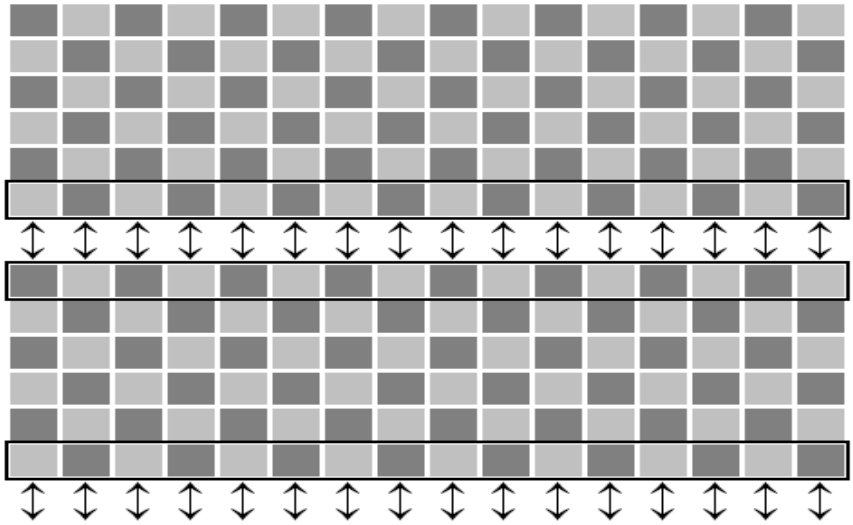
Voor dit onderzoek is een toepassing geselecteerd die aan de volgende sleutelvereisten voldoet. Enerzijds moet de toepassing afhankelijkheden hebben tussen haar opeenvolgende iteraties, daar de meeste parallelle toepassingen deze eigenschap hebben. Anderzijds moeten deze afhankelijkheden een eenvoudige structuur hebben. De gekozen toepassing omvat een implementatie van het Successive Over Relaxation algoritme. Dit algoritme is een iteratieve methode die bruikbaar is gebleken bij het oplossen van Laplace-vergelijkingen (Evans, 1984). De gebruikte parallelle implementatie van het algoritme is gebaseerd op de zwart-witte Successive Over Relaxation[1] methode uit Hageman en Young (1981).'

4.3. Gegevensverzameling met de Parallelle *Successive Over Relaxation* Toepassing

Het Successive Over Relaxation algoritme is een methode voor het oplossen van stelsels van lineaire vergelijkingen en is afgeleid van de Gauβ-Siedel methode. Het is een numerieke methode die gebruikt wordt om convergentie van de Gauβ-Siedel methode te versnellen. Vergelijkbare methodes kunnen worden gebruikt voor verschillende, traag convergerende, iteratieve processen (Wolfram Mathworld). De methode is bijvoorbeeld bruikbaar gebleken bij het oplossen van Laplace-vergelijkingen (Evans, 1984).

De gebruikte implementatie van het algoritme verwerkt een tweedimensionale toestandsruimte, ofwel een tabel. Elk punt in de tabel is een getal en heeft vier buren, maar minder als het punt aan een zijkant van de tabel ligt. Alleen naast-, onder- en bovengelegen punten worden als buren gezien. Het algoritme berekent in elke iteratie voor elk punt in de tabel een gewogen gemiddelde van de waardes van de buren en zichzelf.

In de parallelle implementatie van het algoritme wordt de tabel behandeld als een schaakbord. Alle punten in de tabel worden gezien als witte of zwarte hokjes. Hokjes van een zekere kleur hebben alleen buren van tegengestelde kleur. Elke iteratie is in twee fasen gesplitst, een witte en een zwarte fase. In de witte fase worden alléén witte hokjes gewijzigd, in de zwarte fase alléén zwarte.



Figuur 1 De verdeling in partities van de Successive Over Relaxation tabel, gekarakteriseerd voor de eerste twee medewerkers. Elke centrale verwerkingseenheid wisselt de bijgewerkte punten na elke fase met één of twee anderen uit.

Onder deze implementatie kan de tabel worden gepartitioneerd en kan elke partitie worden toegewezen aan een andere medewerkende centrale verwerkingseenheid. Deze medewerkers kunnen nu punten met dezelfde kleur gelijktijdig wijzigen. Vóórdat een medewerker aan een witte of zwarte fase begint, wisselt het de punten aan de zijkant van de aan hem toegewezen partitie uit met

[1] Parallel Red-Black Successive Over Relaxation (SOR)

alle medewerkers die aansluitende partities verwerken. (Dobber et al, 2004, pagina 344) Dit wordt geïllustreerd in Figuur 1. Alle berekeningen werden een aantal malen herhaald om de toepassing voornamelijk verwerkingstijd-intensief te maken.

De toepassing kan onbeperkt parallel worden genoemd ten aanzien van zuivere executietijd, omdat de tabel in een willekeurig aantal partities kan worden verdeeld. Een in principe onbeperkt aantal nodes zou mee kunnen werken, als tenminste de te verwerken tabel maar groot genoeg is.
De toepassing kan uitstekend schaalbaar worden genoemd ten aanzien van zuivere executietijd, omdat de verwerkingsvereisten per medewerkende node constant klein kunnen blijven, ongeacht dit aantal.

De communicatietijd benodigd voor het verdelen van de gegevens en het verzamelen van de resultaten doet hier echter aan af, aangezien dit toch op één node moet gebeuren. Echter, in dit onderzoek wordt alleen de zuivere executietijd onderzocht.

Voor meer informatie over de exacte wijze waarop verwerkingstijden werden gemeten en bijvoorbeeld de hoeveelheid herhalingen die werden uitgevoerd per executie, wordt verwezen naar Dobber et al (2004, pagina 345).

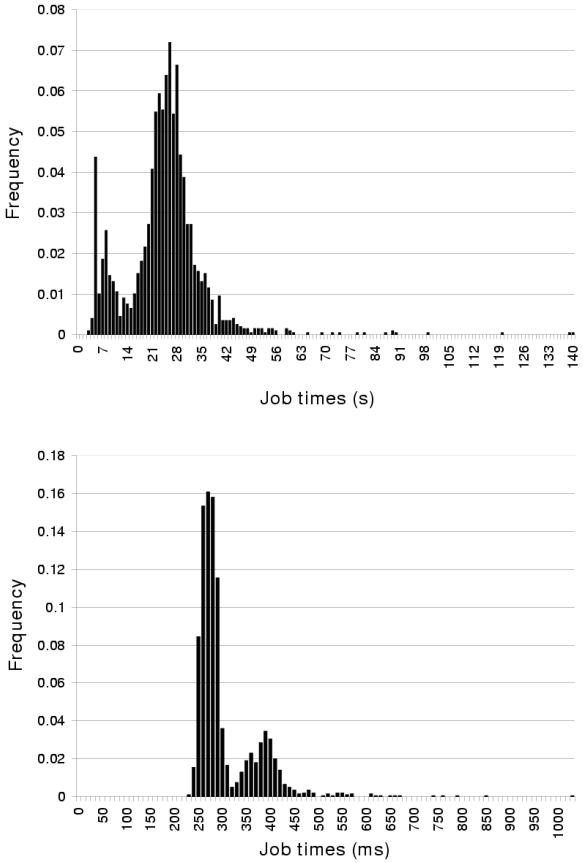
4.4. Voorafgaand Onderzoek naar Statistische Eigenschappen van deze Verwerkingstijden

In Dobber et al (mei 2006) worden de statistische eigenschappen van de met de parallelle zwart-witte Successive Over Relaxation toepassing gemeten verwerkingstijden behandeld. Hieronder volgt een kort overzicht van enkele relevante en kenmerkende resultaten uit dit onderzoek.

In Figuur 2 zijn histogrammen weergegeven van twee series verwerkingstijden, gemeten op verschillende nodes. De histogrammen laten zien hoe de verwerkingstijden zijn verdeeld ten opzichte van elkaar. De horizontale as geeft de verwerkingstijd aan, de verticale as de relatieve frequentie waarmee gemeten verwerkingstijden in een klein bereik vallen, op een schaal van nul tot één. De breedte van elk staafje in de grafieken geeft aan hoe nauwkeurig de bereiken zijn genomen en dus ook hoeveel bereiken er zijn. De verticale as geeft aan welke fractie van alle verwerkingstijden in één enkel bereik vielen.

> **Voorbeeld** In de rechtergrafiek in Figuur 2 is te zien dat de meeste verwerkingstijden op de node met de naam *Warschaw1* tussen de 225 en de 325 milliseconden vallen, en dat ook veel verwerkingstijden rondom ongeveer 400 milliseconden lagen. De bijbehorende piek ligt tussen de 390 en de 400 milliseconden. In totaal ongeveer 3,5% van de gemeten verwerkingstijden liggen tussen deze grenzen.

De figuur laat zien dat de verwerkingstijden op verschillende nodes sterk van elkaar verschillen. De linkergrafiek in de figuur is gemeten in seconden, de rechter in milliseconden. De figuur laat verder zien dat de verwerkingstijden multimodaal zijn. Beide histogrammen hebben meerdere pieken.

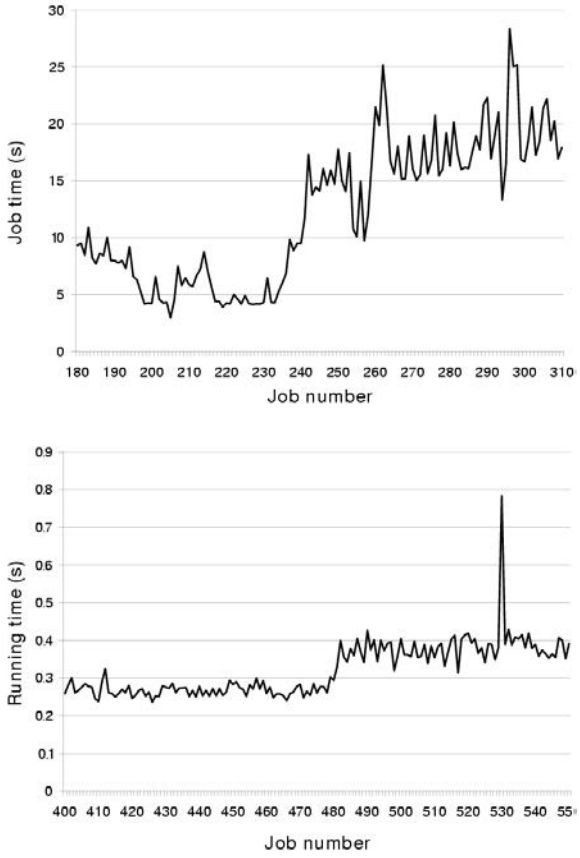


Figuur 2 Histogrammen van alle verwerkingstijden gemeten op *Arizona1* (links) en *Warschaw1* (rechts).

Dit wordt veroorzaakt doordat de verwerkingstijden fluctueren rondom een zeker korte termijn gemiddelde, die echter over een zeker tijdsbestek significant kan veranderen, om vervolgens vrijwel onveranderd te blijven gedurende aanhoudende periodes. Soms minuten lang, soms uren lang.

Figuur 3 illustreert dit uitstekend. De grafiek geeft verwerkingstijden weer die gemeten zijn op twee verschillende nodes, in opeenvolgende iteraties van de Successive Over Relaxation toepassing.
Op de horizontale as is het volgnummer van de iteratie weergegeven, op de verticale as de tijdens die iteratie gemeten verwerkingstijd. Zo duurde de taak uitgevoerd in de 531ste iteratie op de node met de naam *Warschaw1* iets minder dan 0,8 seconden.

In de linkergrafiek treedt een niveauwijziging op vanaf taak 235 tot taak 260. Daarvóór liggen de verwerkingstijden tussen de 3 en de 11 seconden, daarna liggen deze tussen de 15 en 30 seconden. Dit geeft een verklaring voor de pieken in de linkergrafiek van Figuur 2.



Figuur 3 Een aantal opeenvolgende verwerkingstijden, gemeten op de nodes *Arizona1* (links) en *Warschaw1* (rechts)

De rechtergrafieken in beide figuren, behorende bij de node met de naam *Warschaw1*, illustreren dit nogmaals zéér duidelijk. Deze grafieken laten bovendien zien dat niet alleen het korte termijn gemiddelde, maar ook de mate van fluctuatie van dit gemiddelde in belangrijke mate aan verandering onderhevig is. De eerste piek in de rechtergrafiek van Figuur 2 is extremer, wat er op duidt dat het bijbehorende korte termijn gemiddelde aanhield gedurende meer iteraties. Deze is echter bovendien abrupter, of, anders gezegd, smaller dan de tweede piek. Terwijl het bij de tweede piek behorende korte termijn gemiddelde van toepassing was, weken de gemeten verwerkingstijden hier sterker vanaf dan dat dit het geval was bij de eerste piek. Bovendien is in de rechtergrafiek van Figuur 3 een uitgesproken voorbeeld te zien van een plotselinge piek.

Dobber et al (mei 2006, pagina 153) concluderen het volgende.

- Dat de karakteristieken van de verwerkingstijden op verschillende nodes meestal behoorlijk verschillend zijn,
- dat deze over het algemeen meer fluctuaties op de lange termijn laten zien dan fluctuaties op de korte termijn,
- dat het korte termijn gemiddelde op zekere node in belangrijke mate fluctueert, waarbij de mate van fluctuatie per node verschilt,
- dat ook de standaardafwijking op zekere node in belangrijke mate fluctueert en dat ook hier de mate van fluctuatie per node verschilt
- en tenslotte dat pieken optreden die van grote invloed zijn op getalsmaten voor fluctuatie, zoals de standaardafwijking en de variantie.

4.5. Interpretatie van de Verzamelde Verwerkingstijden

Alle verwerkingstijden van reguliere iteraties die gemeten zijn tijdens alle keren dat de toepassing werd uitgevoerd, worden gebruikt in dit onderzoek. Opstart- en afkoeliteraties worden buiten beschouwing gelaten.

Aangezien de witte en zwarte fase van de Successive Over Relaxation toepassing logischerwijs equivalent zijn en een gelijke hoeveelheid werk vertegenwoordigen, worden deze fasen behandeld als iteraties. De verwerkingstijden die in dit onderzoek worden geanalyseerd zijn verwerkingstijden gemeten van een witte dan wel zwarte fase van iteraties van het parallelle zwart-witte Successive Over Relaxation algoritme. Bovenstaand aantal iteraties is op deze manier verkregen.

> **Opmerking** Waar in dit document over iteraties wordt gesproken, wordt een witte dan wel zwarte fase bedoeld van een iteratie van het parallelle zwart-witte Successive Over Relaxation algoritme en tussen deze fasen wordt geen onderscheid gemaakt.

Zodoende zijn er verwerkingstijden beschikbaar van 1988 opeenvolgende reguliere iteraties, op alle van de in totaal 130 nodes waarop de toepassing werd uitgevoerd. Dit wordt geïllustreerd in Tabel 1. Elke kolom in de tabel bevat verwerkingstijden die gemeten zijn op dezelfde node. Elke rij in de tabel bevat verwerkingstijden die gemeten zijn gedurende dezelfde iteratie. Elke cel bevat zodoende een verwerkingstijd op de bijbehorende node in de bijbehorende iteratie.

Iteratie	CPU001	CPU002	CPU003	CPU004	CPU005	CPU006	· ·	CPU130
1	3.719	1.017	220	346	1.466	1.548	· ·	2.030
2	1.672	999	262	239	1.383	1.550	· ·	1.871
3	1.804	1.164	70	88	1.384	1.548	· ·	2.069
:	:	:	:	:	:	:		:
1.988	1.876	1.134	124	346	1.532	3.055	· ·	786

Tabel 1 Een gedeelte van de verzamelde verwerkingstijden. Deze zijn in milliseconden nauwkeurig.

Het is met deze gegevens mogelijk de totale verwerkingstijd van de toepassing te simuleren ingeval een aantal van de nodes wordt geselecteerd om de toepassing op uit te voeren. Nodes die worden gebruikt om de totale verwerkingstijd van de toepassing mee te simuleren worden meewerkende nodes genoemd.

De executies werden op identieke wijze uitgevoerd en de verschillen tussen de aan de nodes toegewezen taken zijn zeer klein, zodat ze geen verschillen tussen de verwerkingstijden veroorzaken. Ook hebben deze verschillen geen enkele invloed op de fluctuaties tussen opeenvolgende verwerkingstijden van een node, aangezien elke node in elke iteratie telkens precies dezelfde verwerking verrichtte. Verder worden de communicatietijden buiten beschouwing gelaten in dit onderzoek en zijn alleen zuivere verwerkingstijden gemeten, zonder idle-time, waaruit de duur van elke iteratie en de totale verwerkingstijd eenvoudig kan worden berekend.

De hoeveelheid werk waar elke node voor verantwoordelijk is blijft gelijk als het aantal meewerkende nodes wordt verhoogd. De hoeveelheid werk die de toepassing als geheel dus verzet wordt groter. Het is niet een doelstelling van dit onderzoek om de prestatie van de toepassing bij het verwerken van een zekere hoeveelheid gegevens te analyseren, maar om inzicht te verlenen in de invloed van de fluctuaties.

Enkele series verwerkingstijden werden gemeten op dezelfde nodes, maar op verschillende dagen. Deze situatie wordt identiek verondersteld aan de situatie waarin verschillende nodes vergelijkbare hardwareconfiguraties zouden hebben, maar aan een verschillende werkdruk onderhevig waren. Het moment en de duur van de fluctuaties zijn in de meeste gevallen voldoende verschillend.

Twee paren series waren precies gelijk. Het eerste paar bestond uit node 92 met label *utah1_28347c* en node 128 met label *wash1_29993c*. Het laatste paar bestond uit node 129 met label *wash1_458* en node 130 met label *wash1_5315c*. Deze zijn desondanks beide gebruikt in dit onderzoek. De alfabetische volgorde van de nodelabels is opgenomen in appendix A.

5. Prestatiestatistieken en Vergelijkingsmethodes

In dit hoofdstuk worden de prestatiestatistieken van de Successive Over Relaxation toepassing geïntroduceerd die in dit onderzoek een sleutelrol spelen bij de analyse van de verwerkingstijden en hun fluctuaties. Waar nodig worden bovendien de gebruikte analysemethodes uiteengezet.

De volgende sectie definiëert statistieken waarmee de prestaties van de toepassing kunnen worden onderzocht en legt uit hoe deze kunnen worden berekend.

In sectie 5.2. wordt een methode uiteengezet, waarmee kan worden bepaald wat de bijdrage is van elke node aan de totale verwerkingstijd, aangenomen dat een selectie kan worden gemaakt uit een vaste verzameling nodes. Deze bijdrages worden vervolgens ook berekend en onderling vergeleken. De daaropvolgende sectie verdiept deze methode, zodat ze kan worden gebruikt om te bepalen hoeveel nodes de verwerkingstijd verlengen, en de resultaten hiervan worden gepresenteerd.

Sectie 5.4. beargumenteerd hoe het gedrag van de totale verwerkingstijd van de Successive Over Relaxation toepassing kan worden onderzocht, wanneer de toepassing op een steeds toenemend aantal nodes wordt uitgevoerd. De methode waarmee de totale verwerkingstijd later zal worden gesimuleerd wordt pas nader toegelicht in het volgende hoofdstuk.

De laatste sectie in dit hoofdstuk zet uiteen hoe de invloed van fluctuaties tussen opeenvolgende verwerkingstijden kan worden geanalyseerd door op een node gemeten opeenvolgende verwerkingstijden aan elkaar gelijk te stellen.

5.1. Prestatiestatistieken: Gemiddelde Verwerkingstijden van Iteraties en Taken

De toepassing voert 1988 iteraties uit, waar maximaal 130 centrale verwerkingseenheden in parallel aan meewerken. Het werk, dat gedurende één iteratie aan één centrale verwerkingseenheid wordt toegewezen, wordt een deeltaak of kortweg taak genoemd. De gemeten verwerkingstijden zijn de verwerkingstijden van deze taken. Elke taak kan worden gezien als behorende bij een iteratie. Alle taken behorende bij één iteratie worden in parallel uitgevoerd.

De duur van een iteratie is de totale tijdsduur gedurende welke er tenminste één centrale verwerkingseenheid taken uitvoert die tot deze iteratie behoren, vanaf het moment dat alle taken uit voorgaande iteratie volledig zijn voltooid. Dit laatste voorkomt dat verwerkingstijd meermalen wordt meegerekend. De verwerkingstijden worden zó geïnterpreteerd dat een zekere iteratie pas kan starten wanneer de vóórgaande iteratie is voltooid. Dus, de afhankelijkheden tussen de iteraties worden geïnterpreteerd alsof elke taak resultaten behoeft van alle taken in de voorgaande iteratie.

De verwerkingstijden per iteratie en per taak zijn dus verwerkingstijden inclusief idle-time.
Eerst wordt berekend hoe lang elke iteratie duurt. Vervolgens worden hieruit de gemiddelde verwerkingstijd per iteratie berekend. Tenslotte wordt de gemiddelde verwerkingstijd per taak berekend, door de gemiddelde duur van de iteraties te delen door het aantal nodes.

De totale verwerkingstijd van de toepassing is de som van alle iteratieduren. Echter, om de getallen klein te houden wordt exclusief gekeken naar de gemiddelde verwerkingstijd per iteratie en per taak. De totale verwerkingstijd kan hieruit middels een eenvoudige vermenigvuldiging worden verkregen.

5.2. De Bijdrage van Nodes aan de Gemiddelde Verwerkingstijd van de Iteraties

De tijden gedurende welke aan opeenvolgende iteraties wordt gewerkt overlappen elkaar niet en gedurende elke iteratie starten alle meewerkende nodes gelijktijdig aan hun taak. Tezamen leidt dit ertoe, dat elke iteratie precies zo lang duurt als de maximale verwerkingstijd op alle meewerkende nodes. De totale verwerkingstijd van de toepassing kan worden bepaald door dit maximum voor elke iteratie te bepalen en vervolgens deze maxima op te tellen.

> **Definitie** Een door een node *vertraagde iteratie* is een iteratie die langer duurt doordat die node de langste verwerkingstijd heeft in die iteratie.

Door bij te houden welke node in elke iteratie het langzaamste is, kan bovendien worden bepaald op welke nodes vaak moet worden gewacht, naast hoe lang er minimaal op die nodes gewacht moet worden. Dit wordt geïllustreerd in Tabel 2. Hierin staat aangegeven in welke volgorde de nodes iteraties voltooien. Zo voltooit node 57 de eerste iteratie als eerste en is node 19 in deze iteratie de langzaamste. De eerste iteratie wordt dus vertraagd door node 19.

Op deze manier kan worden bepaald dat op node 41 met label *caltech1_4321c* moet worden gewacht in 1516 van de 1988 iteraties, als alle 130 nodes meewerken. Node 41 is de meest vertragende node, ongeacht of alle nodes meewerken. De nodes, die het kortste op node 41 moesten wachten, wachtten elke iteratie gemiddeld maar liefst 39 seconden. Terwijl de toepassing de berekening in twee en een halve dag voltooid als alle nodes meewerken, duurt dit zónder node 41 nog slechts maximaal één dag en ongeveer 15 uur, een reductie van 35%.

Iteratie	Traagste	2e	3e	4e	5e	6e	· ·	Snelste
1	CPU019	CPU062	CPU087	CPU026	***CPU041***	CPU122	· ·	CPU057
2	CPU062	CPU019	CPU087	***CPU041***	CPU026	CPU122	· ·	CPU109
3	CPU062	CPU019	***CPU041***	CPU026	CPU122	CPU117	· ·	CPU107
:	:	:	:	:	:	:		:
1.988	CPU062	CPU088	CPU009	***CPU041***	CPU082	CPU92	· ·	CPU106

Tabel 2 De nodes in volgorde van hun verwerkingstijd gedurende elke iteratie. Node 41 is hierin benadrukt.

Als node 41 wordt uitgesloten van meewerking, kan opnieuw worden bepaald welke node de toepassing het meeste vertraagt.

> **Definitie** Een *vertragende node* is een node die de tenminste één iteratie vertraagt als deze deel gaat nemen aan de toepassing, ten opzicht van de tot dan toe langste iteratieduren.
>
> **Definitie** De *meest vertragende node* van een willekeurige verzameling nodes is de node waarop, over alle vertraagde iteraties gezien, het langste moet worden gewacht door de andere nodes in de verzameling.

Zo kan een ordening worden bepaald voor alle nodes in aflopende volgorde van bijdrage aan de gemiddelde iteratieduur, door telkens de meest vertragende node uit de verzameling te verwijderen en opnieuw de meest vertragende node vast te stellen van de zo bijgewerkte verzameling.

Tabel 3 laat de resultaten zien die op deze manier zijn verkregen. Hierin is te zien dat als de toepassing op 129 nodes wordt uitgevoerd, alleen zonder node 41, dan is node 28 met label *caltech1_15363c* het meest vertragend. Door verwijdering van node 28 worden 566 iteratieduren verkort. De gemiddelde iteratieduur neemt af met 5,6 seconden. Dit komt overeen met 5,1% van de oorspronkelijke gemiddelde iteratieduur van 110 seconden.

#	Node Index	Node Label	Vertraagde Iteraties	Fractie Iteraties	Toename Iteratieduur	Bijdrage Iteratieduur
1	41	caltech1_4321c	1.516	76,3 %	38,7 s	35,2 %
2	28	caltech1_15363c	566	28,5 %	5,6 s	5,1 %
3	43	caltech1_8912c	646	32,5 %	6,7 s	6,1 %
4	19	boston1_4321c	523	26,3 %	5,5 s	5,0 %
5	37	caltech1_25553c	228	11,5 %	4,0 s	3,6 %
6	9	arizona1_9736	587	29,5 %	4,5 s	4,1 %
7	38	caltech1_26193c	480	24,1 %	2,5 s	2,3 %
8	7	arizona1_4321c	498	25,1 %	3,0 s	2,8 %
9	33	caltech1_24385c	597	30,0 %	2,4 s	2,2 %
10	62	ntu1_22916	756	38,0 %	2,0 s	1,8 %

Tabel 3 Het effect van het successievelijk verwijderen van de nodes die de grootste bijdrage leveren aan de totale verwerkingstijd.

Node 28 is meer vertragend dan node 43, ondanks dat voor node 43 een grotere toename is berekend van de gemiddelde iteratieduur. Als de toepassing wordt uitgevoerd op een gegeven aantal nodes, is de gemiddelde iteratieduur minimaal als de nodes worden uitgesloten in de op bovenstaande wijze vastgestelde volgorde. De bijdrage van een zekere node is de minimale bijdrage

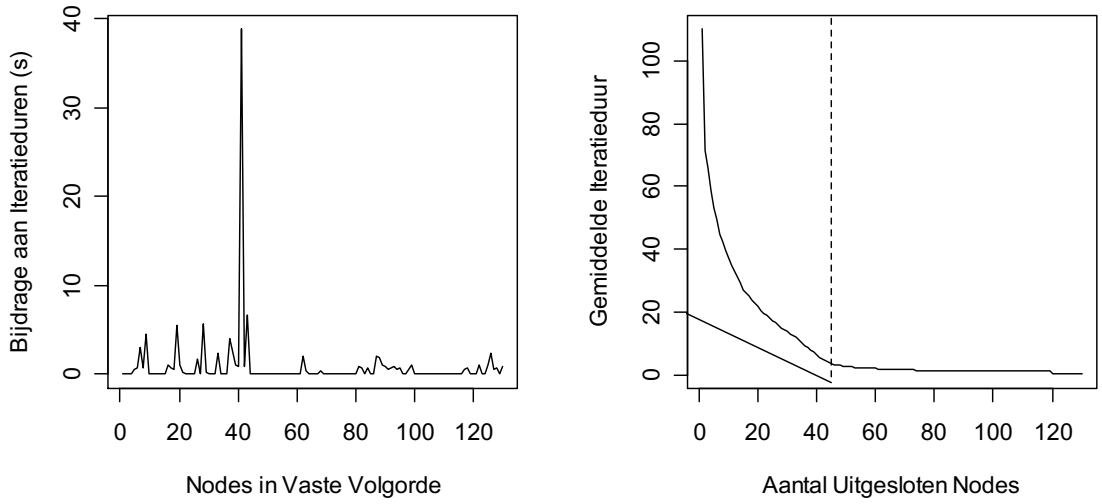
van die node aan de gemiddelde iteratieduur. Als een minder vertragende node wordt uitgesloten van meewerking, blijft deze bijdrage gelijk of neemt ze toe. Ze neemt toe als de uitgesloten node gedurende tenminste één iteratie de opeenvolgend traagste was, anders blijft ze gelijk. Optelling van de bijdrages van alle 130 nodes resulteert dan ook exact in de maximale gemiddelde iteratieduur van 110 seconden, dat gemeten wordt als alle nodes meewerken.

> **Voorbeeld** Als node 28 niet aanwezig was geweest, was de bijdrage aan de gemiddelde iteratieduur van node 41 groter geweest. De toename van de iteratieduur veroorzaakt door node 41 zou in dit geval 44,3 seconden zijn, wat gelijk is aan de som van de bijdrages van node 41 en node 28.

> **Voorbeeld** In de eerste iteratie is node 19 het traagste, node 62 de daaropvolgend traagste en node 57 de snelste. Node 19 is altijd veel trager dan node 57. Als node 62 wordt uitgesloten van meewerking stijgt de bijdrage van node 19 aan de gemiddelde iteratieduur, maar de bijdrage van node 19 neemt niet toe als node 57 wordt uitgesloten van meewerking.

Figuur 4 vat de op deze wijze verkregen gegevens samen voor alle nodes. De linkergrafiek laat per node zien welke toename van de gemiddelde iteratieduur deze veroorzaakt, als alle nodes met een grotere invloed zijn uitgesloten van meewerking. De nodes zijn in deze grafiek weergegeven in alfabetische volgorde van de node labels. Hierin is duidelijk zichtbaar dat node 41 een zeer extreme invloed heeft op de totale iteratieduur en dat de invloed van veel nodes zeer klein is.

De rechtergrafiek laat zien hoe de iteratieduur afneemt als de meest vertragende nodes in volgorde worden uitgesloten van deelname. Als alle nodes deelnemen is de gemiddelde iteratieduur 110 seconden. Als alleen node 41 wordt uitgesloten daalt deze al tot 71 seconden. Als de 45 meest vertragende nodes worden uitgesloten van deelname is de maximale gemiddelde iteratieduur nog slechts 45 seconden. De verticale onderbroken lijn is ter illustratie getekend bij dit aantal uitgesloten nodes. De minimale gemiddelde iteratieduur is 0,11 seconden.



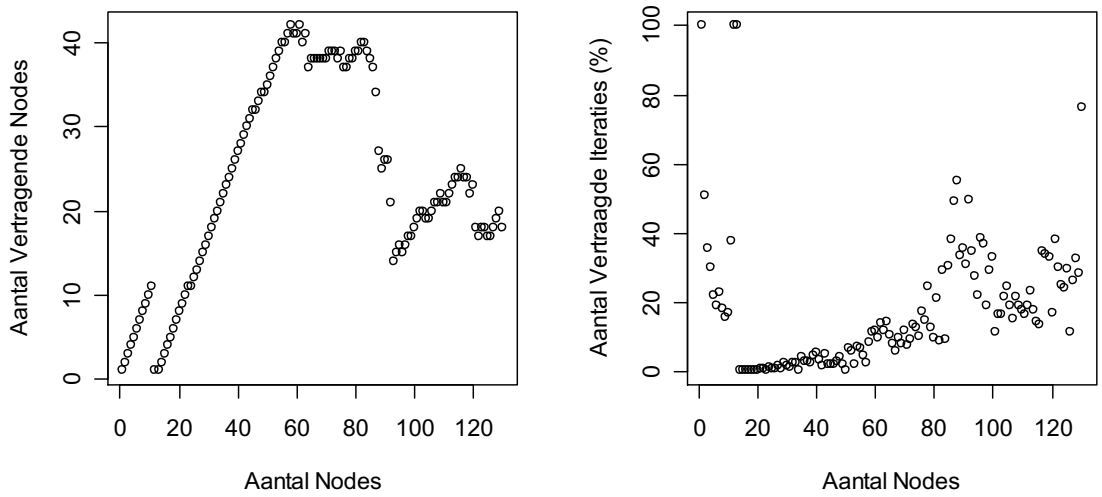
Figuur 4 De bijdrage van de nodes op de gemiddelde iteratieduur. Uit de rechterfiguur is node 41 weggelaten. [R18]

De grafieken in de figuur laten zien dat de absolute bijdrage van de nodes aan de gemiddelde iteratieduur zéér sterk uiteenlopen en dat een klein deel van de nodes een zeer groot deel van de totale verwerkingstijd veroorzaken.

5.3. Het Aantal Nodes dat de Gemiddelde Iteratieduur Verlengt

Op vergelijkbare wijze als waarop de meest vertragende nodes in de vorige sectie zijn bepaald, kan het aantal nodes worden vastgesteld dat de gemiddelde iteratieduur verlengt. Ook nu ingeval telkens de meest vertragende nodes uitgesloten worden van meewerking.

Door bij te houden welke node in elke iteratie het langzaamste is, kan een lijst worden samengesteld van nodes waarop gedurende één of meerdere iteraties moet worden gewacht. Het aantal nodes dat in deze lijst voorkomt is het aantal nodes dat de gemiddelde iteratieduur verlengt. De resultaten van deze analyse zijn weergegeven in Figuur 5.



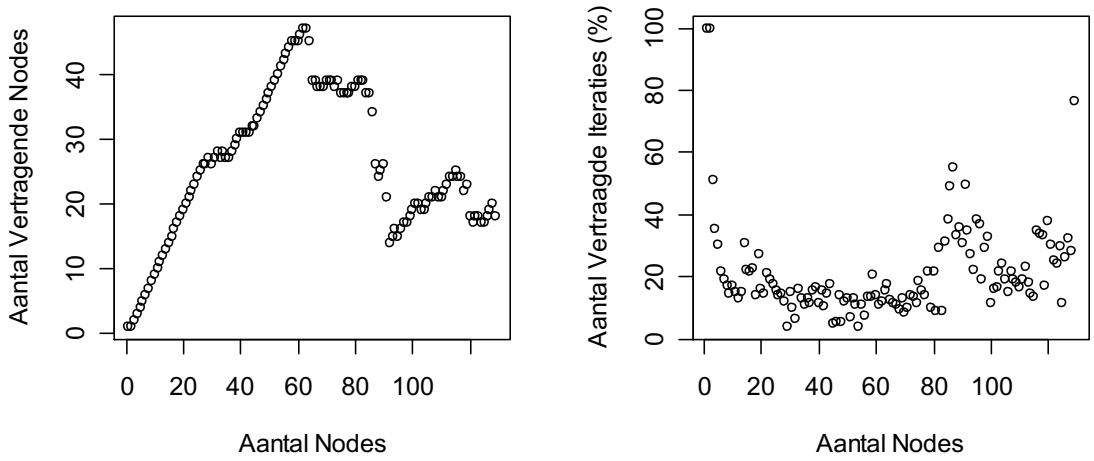
Figuur 5 Het aantal vertragende nodes (links) en de fractie iteraties vertraagd door de meest vertragende node (rechts) als functie van het aantal nodes. [R18]

De linkergrafiek in de figuur geeft het aantal vertragende nodes weer als functie van het totale aantal nodes dat meewerkt. Alleen de minst vertragende nodes wordt toegestaan mee te werken. Zo is het aantal vertragende nodes logischerwijs 1 als er 1 node meewerkt, namelijk de snelste, en heeft het toevoegen van de eerste 11 nodes telkens tot gevolg dat tenminste één iteratie wordt vertraagd. Vervolgens wordt tweemaal een node toegevoegd die de traagste is in alle iteraties. Tot ongeveer 60 nodes stijgt het aantal vertragende nodes vervolgens. Deze nodes laten blijkbaar redelijk vergelijkbare verwerkingstijden zien. Vanaf ongeveer 60 nodes verandert dit echter. Het is dan ook steeds waarschijnlijker geworden dat een aantal meer vertragende nodes het vertragende effect van een snellere node gaan maskeren. Als het aantal vertragende nodes echter met een sprong afneemt, is dit een indicatie dat tenminste de laatst toegevoegde node in zeer belangrijke mate vertragend zou kunnen zijn.

De rechtergrafiek geeft aan hoeveel iteraties worden vertraagd door de node die als laatste is toegevoegd, als percentage van het totaal aantal iteraties. Opnieuw is te zien dat de 1ste, 11e en 12e nodes die worden toegevoegd in álle iteraties de langzaamste waren. Het percentage vertraagde iteraties is klein beginnende bij 13 nodes, wat erop wijzen kan dat een aantal nodes alleen iteraties vertragen vanwege incidentele, relatief extreme verwerkingstijden ofwel pieken. Dit kunnen extreem hoge waarden zijn in de verwerkingstijden van de laatst toegevoegde node danwel extreem lage waarden in de verwerkingstijden van de eerder toegevoegde node. Het percentage neemt dan duidelijk toe met het aantal nodes. De vertragingen worden in steeds mindere mate veroorzaakt door pieken.

Dat de percentages vaak hoog zijn vanaf ongeveer 80 nodes is een aanwijzing dat de prestatieniveaus van de meest vertragende nodes vaak in belangrijke mate afwijkt van de prestatieniveaus van nodes die minder vertragend zijn.

De nodes die alle iteraties vertragen zijn node 106 met label *vu1_3736_02*, node 129 met label *wash1_458* en node 130 met label *wash1_5315c*. Node 106 is de snelste. De twee anderen zijn precies dezelfde series verwerkingstijden. Deze series hebben een hoog minimum van 0,3 seconden, een hoog gemiddelde van 1,3 seconden en een buitengewoon grote standaardafwijking vergeleken met de eerste 30 nodes in de grafieken. Dat de eerste 11 nodes nooit trager zijn is toeval. Ongeveer 7% van de op deze nodes gemeten verwerkingstijden overstijgt het minimum van 0,338 seconden gemeten op node 129 en node 130 en deze zijn verspreid over de eerste 11 nodes. Deze eerste nodes en de nodes volgend op node 129 en node 130 hebben geen wezenlijk verschillend prestatieniveau. Voor de volledigheid zijn de grafieken in Figuur 6 opnieuw weergegeven, maar nu zonder node 129 en 130.



Figuur 6 Het aantal vertragende nodes (links) en de fractie iteraties vertraagd door de meest vertragende node (rechts) als functie van het aantal nodes uitgezonderd node 129 en node 130. [R18]

Als het aantal vertragende nodes duidelijk afneemt doordat een zekere node op deze manier wordt toegevoegd, kan worden geconcludeerd dat de laatst toegevoegde node blijkbaar vaak trager is dan veel andere nodes die tot dan toe vertragend waren. Dit kan betekenen dat de prestieniveaus van de nodes nu sterker uiteen lopen.

De op 91 nodes na meest vertragende node, node 99 met label *utah3_8912c*, doet zo bijvoorbeeld het aantal vertragende nodes afnemen met 5 en blijkt in ongeveer de helft van de iteraties de traagste te zijn.

5.4. Variëren van het aantal Nodes waarop de Toepassing wordt Uitgevoerd

Wanneer het aantal nodes waarop de toepassing wordt uitgevoerd kleiner is dan het maximum van 130, dan worden alle taken die uitgevoerd werden op de resterende nodes buiten beschouwing gelaten. Dus, niet alleen het aantal nodes wordt kleiner verondersteld, maar ook de hoeveelheid werk dat de toepassing verricht, totaal zowel als per iteratie.

Wanneer het aantal nodes verandert, komt dit overeen met een wijziging van de grootte van de verwerkte *Successive Over Relaxation* tabel, maar zó, dat deze kan worden berekend door het aantal meewerkende nodes met een constante te vermenigvuldigen. Het is niet een doelstelling van dit onderzoek om de prestatie van de toepassing te analyseren bij het verwerken van een zekere hoeveelheid gegevens, maar om inzicht te verlenen in de invloed van fluctuaties op verwerkingstijden. De invloed van de hoeveelheid werk dat elke nodes verricht wordt daarom geëlimineerd, door deze te allen tijde constant te houden.

De totale verwerkingstijd blijft gelijk als de nodes waarop de toepassing wordt uitgevoerd steeds even snel hun taak voltooien. De totale verwerkingstijd kan enerzijds toenemen doordat een node wordt toegevoegd, maar dan wordt dit veroorzaakt door fluctuaties in de verwerkingstijden op een node, die niet worden veroorzaakt door de toepassing zelf. Anderszijds veroorzaakt de toepassing ook geen afname van de totale verwerkingstijd, door het werk steeds meer te verdelen.

De prestatiestatistieken worden telkens berekend als functie van het aantal meewerkende nodes. Er wordt dus herhaaldelijk een selectie gemaakt van nodes. De prestatieniveaus van de nodes verschillen onderling in belangrijke mate. Wanneer de prestatiestatistieken nu worden berekend voor een zekere selectie van nodes, is het van belang precies wélke nodes dit zijn.

In het volgende hoofdstuk wordt het gedrag van de prestatiestatistieken geanalyseerd als het aantal nodes waarop de toepassing wordt uitgevoerd toeneemt. In de eerste sectie wordt geïllustreerd hoe nodes worden geselecteerd en dat de volgorde waarin de nodes worden toegevoegd van belang is. In de sectie daarna wordt een methode uiteengezet om de invloed van deze volgorde tot een minimum te reduceren. Tenslotte worden de gedefinieerde prestatiestatistieken berekend met behulp van deze methode.

5.5. Elimineren van Fluctuaties tussen Opeenvolgende Verwerkingstijden

Een belangrijk deel van alle fluctuaties worden veroorzaakt door pieken. Uit Dobber et al (mei 2006, pagina 153) blijkt dat gemiddeld maar liefst 76% van de totale fluctuaties worden veroorzaakt door

pieken. De in dit artikel berekende statistiek geeft aan in welke mate de standaardafwijking in de verwerkingstijden gemeten op een zekere node is veroorzaakt door fluctuaties op de korte- of lange duur. Verschillen tussen verwerkingstijden van opeenvolgende taken op dezelfde node zouden dus een belangrijke oorzaak kunnen zijn van toenames van de totale verwerkingstijd, aangezien alle nodes moeten wachten op de node wiens verwerkingstijd een piek vertoont. De methode die in deze sectie wordt beschreven en beargumenteerd is bedoeld om inzicht te verlenen in de invloed van precies deze fluctuaties.

De fluctuaties in verwerkingstijden zijn toe te schrijven aan zowel verschillen in de verwerkings-capaciteiten van de nodes als de wisselende werkdruk erop. Als de gemeten verwerkingstijden consistent hoog zijn, is het toch mogelijk dat deze enkel worden veroorzaakt door een onafgebroken hoge werkdruk op die node. Een kort moment van inactiviteit kan in dit laatste geval tot gevolg hebben dat plotseling een uitzonderlijk lage verwerkingstijd wordt gemeten. Een precies tegengestelde situatie is even goed mogelijk.

De fluctuaties tussen opeenvolgende verwerkingstijden kunnen op verschillende manieren worden geëlimineerd. Er is voor gekozen om de gemiddelde verwerkingstijd te gebruiken. Dus, de verwerkingstijden van een node worden aangepast, zodat deze gedurende alle iteraties gelijk zijn aan het gemiddelde van de gemeten verwerkingstijden op die node. In dit geval blijft de totale zuivere verwerkingstijd, dus exclusief idle-time, onveranderd. Dit wordt geïllustreerd in Tabel 4. Het gemiddelde van de verwerkingstijden gemeten op node 1 met label *arizona1_10223* is bijvoorbeeld 1,8 seconden.

Iteratie	CPU001	CPU002	CPU003	CPU004	CPU005	CPU006	· ·	CPU130
1	1.868	1.346	154	150	8.142	10.580	· ·	1.258
2	1.868	1.346	154	150	8.142	10.580	· ·	1.258
3	1.868	1.346	154	150	8.142	10.580	· ·	1.258
:	:	:	:	:	:	:		:
1988	1.868	1.346	154	150	8.142	10.580	· ·	1.258

Tabel 4 Een gedeelte van de verwerkingstijden waarin de fluctuaties tussen opeenvolgende waardes zijn geëlimineerd.

In elke iteratie zal nu telkens dezelfde node de traagste zijn wanneer een selectie van nodes wordt gemaakt en de iteratieduren worden berekend op deze nodes. De andere nodes ervaren in elke iteratie ook dezelfde idle-time. De iteratieduur is immers precies gelijk aan het verschil tussen de gemiddelde verwerkingstijd op die node en het hoogste gemiddelde van alle geselecteerde nodes. Er zijn nog slechts hoogstens 130 verschillende verwerkingstijden.

Er is dus geen enkele fluctuatie meer tussen de iteratieduren, maar er blijven fluctuaties bestaan in de prestatieniveaus van de nodes. Een andere selectie van nodes zal nog wel tot verschillende iteratieduren leiden.

In Tabel 5 worden kernstatistieken van de gemeten verwerkingstijden vergeleken met de verwerkingstijden die op bovenstaande manier zijn bepaald. De kernstatistieken in de middelste kolom zijn bepaald door eerst alle oorspronkelijk gemeten, ongewijzigde verwerkingstijden samen te nemen. Die in de rechterkolom zijn bepaald door eerst gemiddelde verwerkingstijden te bereken zoals in Tabel 4 is geïllustreerd en deze samen te nemen. De getallen in Tabel 5 zijn dus alle bepaald op basis van verwerkingstijden van alle nodes in alle iteraties, ofwel 1988 maal 130 verwerkingstijden. Vervolgens is hiervan gemiddelde, standaardafwijking, minimum, maximum, mediaan, eerste en derde kwartiel berekend. De standaardafwijking is een maat voor de spreiding rondom het gemiddelde. De mediaan ligt precies midden tussen alle gemeten verwerkingstijden. De ene helft is groter, de andere helft is kleiner. Eerste en derde kwartiel liggen op een kwart respectievelijk driekwart van alle verwerkingstijden: 25% respectievelijk 75% hiervan zijn kleiner dan de weergegeven waarde.

Figuur 15 in appendix B bevat één *box-and-whisker* plot, voor elke node waarop verwerkingstijden zijn gemeten. Elke boxplot geeft in één oogopslag een beeld van de lengte van de verwerkingstijden gemeten op één node. De plots tezamen leveren een interessante bijdrage aan het inzicht in de fluctuaties tussen verwerkingstijden. In deze grafieken worden de kernstatistieken uit Tabel 5,

behalve gemiddelde en standaardafwijking, op overzichtelijke wijze gepresenteerd. Meer uitleg is opgenomen in de genoemde appendix.

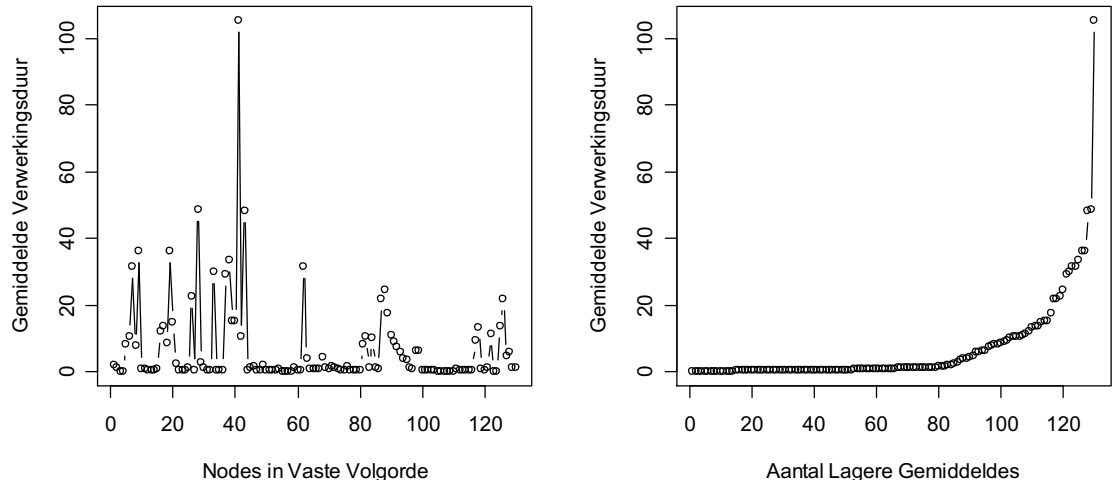
De tabel laat zien dat het gemiddelde van alle verwerkingstijden, op alle nodes, niet veranderen zou wanneer de nodes vaste verwerkingstijden zouden hebben. Ook de mediaan blijft vrijwel gelijk. Het minimum en het maximum verschuiven echter beide behoorlijk, net als het eerste en derde kwartiel.

Kernstatistiek	Gemeten Verwerkingstijden	Zonder Fluctuaties
Gemiddelde	6,65 s	6,65 s
Standaardafwijking	15,2 s	13,3 s
Minimum	34 ms	105 ms
Eerste kwartiel	39 ms	421 ms
Mediaan	919 ms	995 ms
Derde kwartiel	6,2 s	8,1 s
Maximum	593,4 s	105 s

Tabel 5 Vergelijking van de kernstatistieken van alle gemeten verwerkingstijden en de verwerkingstijden waaruit fluctuaties tussen opeenvolgende verwerkingstijden zijn geëlimineerd.

Figuur 7 geeft een overzicht van de berekende gemiddelde verwerkingstijden. De linkergrafiek in de figuur zet de nodes in alfabetische volgorde van node label op de horizontale as uit tegen het op elke node gemeten gemiddelde. Opnieuw is in deze grafiek zichtbaar dat node 41 buitengewoon traag is. De gemiddelde verwerkingstijd van deze node bedraagt 105 seconden. Dit verschilt slechts 6,2 seconden van de maximale gemiddelde iteratieduur van 110 seconden.

De rechtergrafiek in Figuur 7 geeft een overzicht van de verhoudingen tussen de gemiddelde verwerkingstijden op alle nodes. Hierin is voor elke node geteld hoeveel nodes lagere gemiddelde verwerkingstijden hebben. Zo zijn er 80 nodes wiens gemiddelde verwerkingstijd lager is dan 1,4 seconden en zijn er 116 gemiddeldes kleiner dan 20 seconden.



Figuur 7 Vergelijking van de gemiddelde verwerkingstijden op de nodes. [R16]

De vaste volgorde van de nodes in de linkergrafiek is de alfabetische volgorde van de aan de nodes toegewezen labels. Een volledige lijst is in appendix A opgenomen.

6. Analyse van de Prestatiestatistieken

In dit hoofdstuk worden de prestatiestatistieken gedefinieerd in sectie 5.1. geanalyseerd in verschillende situaties, maar eerst wordt de noodzaak van een vorm van simulatie aangetoont en de gebruikte simulatiemethode uiteengezet.

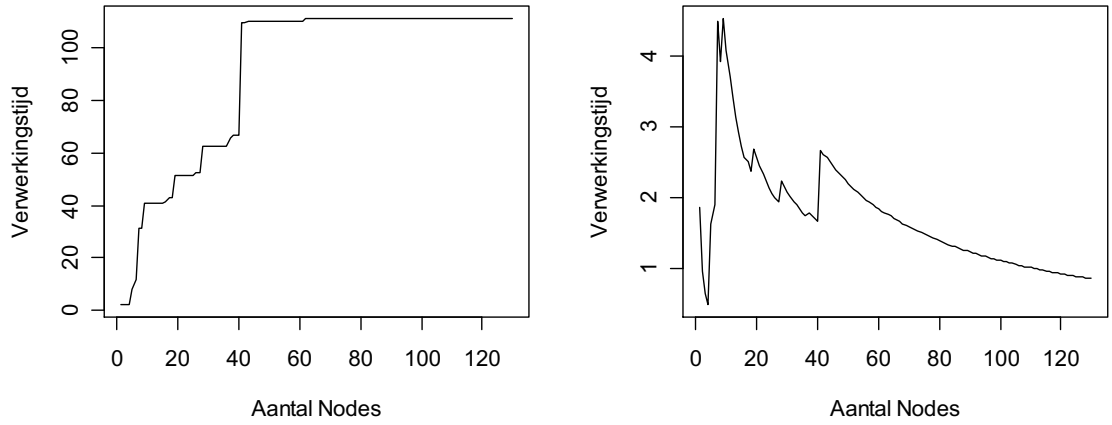
Sectie 6.1. illustreert het gedrag van de prestatiestatistieken, zoals deze zich gedragen zonder simulatie. De daaropvolgende sectie laat zien dat de grafiek in 6.1. slechts weinig informatief is en beschrijft een methode om de prestatiestatistieken systematischer te onderzoeken.

In sectie 6.3. wordt deze methode toegepast om het gedrag van de prestatiestatistieken te analyseren ingeval de toepassing een selectie kan maken uit alle van de 130 nodes waarop verwerkingstijden zijn gemeten. Ook wordt hierin uiteengezet hoe de prestatiestatistieken grafisch kunnen worden onderzocht en wat de betekenis is van de informatie die de grafieken van de prestatiestatistieken in dit hoofdstuk telkens bevatten.

De laatste twee secties behandelen verschillende scenario's, met als doel de invloed van fluctuaties tussen de prestatieniveaus van de nodes en tussen opeenvolgende verwerkingstijden te demonstreren. Dit wordt in sectie 6.4. respectievelijk sectie 6.5. gedaan.

6.1. Illustratie van het Gedrag van de Prestatiestatistieken

Figuur 8 toont de gemiddelde iteratieduur en de gemiddelde verwerkingstijd per taak als functie van het aantal nodes waarop de toepassing wordt uitgevoerd. De linkergrafiek in de figuur toont de gemiddelde verwerkingstijd per iteratie. De rechtergrafiek toont de gemiddelde verwerkingstijd per taak. De grafieken laten het prestatieverloop zien onder de aanname dat de meewerkende nodes een zekere volgorde hebben. De nodes zijn in alfabetische volgorde van node label aan de toepassing toegevoegd. Deze volgorde is van grote invloed op het gedrag van de prestatiestatistieken.



Figuur 8 De verwerkingstijd per iteratie (links) en per taak (rechts) van de parallelle Successive Over Relaxation toepassing als functie van het aantal medewerkende nodes. [R1]

Zo heeft het laten meten van node 41 in deze volgorde tot gevolg dat de verwerkingstijd per iteratie toeneemt van 67 tot 110 seconden en dat de verwerkingstijd per taak van 1,7 seconden oploopt tot 2,7 seconden. Als dezelfde node later zou zijn voorgekomen in de ordening, zou de toename van de verwerkingstijd vanwege het toevoegen van deze node wellicht gematigder zijn geweest.

De rechtergrafiek toont dat het toevoegen van de 6^{e} node een grote invloed heeft op de verwerkingstijd per taak. Deze neemt toe van 1,9 tot maar liefst 4,5 seconden. Dit grote verschil is niet goed zichtbaar in de linkergrafiek, hoewel ook de verwerkingstijd per iteratie in belangrijke mate verandert, namelijk van 11 tot 31 seconden.

6.2. Randomizeren van de Volgorde waarin de Meewerkende Nodes worden Toegevoegd

Figuur 8 in de vorige sectie laat een prestatieontwikkeling zien die mogelijkerwijs zou kunnen optreden indien de toepassing herhaaldelijk wordt uitgevoerd, telkens op één node meer.
De volgorde waarin de nodes dan worden toegevoegd bepaalt de vorm van de grafieken.
Deze precieze volgorde is echter niet van daadwerkelijk belang, enkel de prestatiestatistieken van

de nodes die meewerken. De in deze figuur zichtbare prestatieontwikkeling is dus niet karakteriserend. De invloed van de volgorde moet daarom worden geëlimineerd.

Dit kan worden bereikt door een willekeurige selectie te maken van het gewenste aantal nodes uit alle beschikbare nodes. Aangezien de prestatiestatistieken worden berekend voor verschillende aantallen nodes, is dit gedaan op basis van een willekeurige herordening van de nodes. De nodes komt precies éénmaal in elke ordening voor en worden in deze volgorde aan de toepassing toegevoegd. Elke verschillende ordening resulteert in een ander prestatieverloop. Een realistisch prestatieverloop kan worden bepaald door voor een gegeven aantal nodes het gemiddelde te bepalen van een groot aantal prestatiestatistieken berekend voor dat aantal meewerkende nodes.

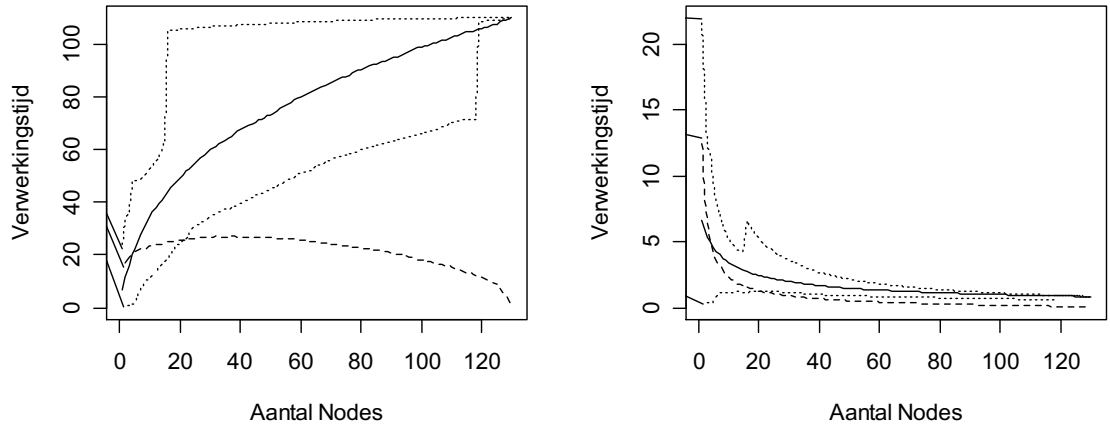
Als de toepassing op een aantal willekeurige nodes wordt uitgevoerd, kan een gemiddelde iteratieduur worden verwacht, die gelijk is aan het gemiddelde van de gemiddelde iteratieduren onder alle beschouwde ordeningen, bijvoorbeeld 10.000, berekend voor dit aantal nodes. Eén gemiddelde iteratieduur wordt berekend over de 1988 uitgevoerde iteraties, het gemiddelde daarvan wordt berekend over de 10.000 beschouwde ordeningen.

Het aantal ordeningen dient dan zó groot te zijn, dat het gemiddelde voldoende betrouwbaar is. Deze aanpak staat toe dat bovendien een betrouwbaarheidsinterval kan worden bepaald, waarbinnen een prestatiestatistiek zich met vooraf bepaalde zekerheid zal bevinden.
Een realistische, in de praktijk gemeten waarde van de prestatiestatistiek zal met de vooraf bepaalde zekerheid binnen het betrouwbaarheidsinterval vallen, vooropgesteld dat een selectie wordt gemaakt uit dezelfde of vergelijkbare nodes als die gebruikt zijn voor dit onderzoek.

> **Voorbeeld** De totale verwerkingstijd van de toepassing wordt 10.000 maal berekend als deze op een zeker aantal nodes wordt uitgevoerd. Een 80%-betrouwbaarheidsinterval kan nu worden gegeven door het minimum en het maximum te bepalen van de 8.000 verwerkingstijden die overblijven als de langste 1.000 en de kortste 1.000 verwerkingstijden worden weggelaten.
>
> Als de toepassing herhaald wordt uitgevoerd op dit aantal nodes zullen 80% van de verwerkingstijden tussen dit minimum en maximum liggen. Anders gesteld, de totale verwerkingstijd ligt met 80% zekerheid in dit bereik, telkens als de toepassing wordt uitgevoerd.

6.3. Gedrag van de Prestatiestatistieken bij Oplopend aantal Meewerkende Nodes

In Figuur 9 zijn de prestatiestatistieken weergegeven, gebaseerd op 10.000 willekeurige ordeningen van de nodes. De linkergrafiek laat zien hoe de gemiddelde iteratieduur zich ontwikkelt bij een toenemend aantal meewerkende nodes. De rechtergrafiek laat dit zien voor de gemiddelde verwerkingstijd per taak.



Figuur 9 De verwerkingstijd per iteratie (links) en per taak (rechts) als functie van het aantal meewerkende nodes. [R2]

De ononderbroken lijn in de linkergrafiek is de gemiddelde iteratieduur. Als de toepassing wordt uitgevoerd op één node is deze bijvoorbeeld 6,7 seconden. Als alle nodes meewerken is deze 110 seconden. De onderbroken lijn is de standaardafwijking van telkens 10.000 berekende gemiddelde iteratieduren. Als één node meewerkt is deze bijvoorbeeld 13,9 seconden. Dit getal geeft aan in welke mate de 10.000 waardes van het gemiddelde ervan verschillen. De twee gestippelde lijnen

geven de boven- en ondergrenzen aan van een 80%-betrouwbaarheidsinterval. 8.000 van de 10.000 gemiddelde iteratieduren liggen telkens tussen deze grenzen. Voor één node liggen de gemiddelde iteratieduren bijvoorbeeld 8.000 maal tussen de 0,24 en de 22 seconden en voor 117 nodes even zo vaak tussen de 71 en de 110 seconden.

De rechtergrafiek laat op vergelijkbare wijze zien hoe de gemiddelde verwerkingstijd per taak afhangkelijk is van het aantal meewerkende nodes. Ook hiervan zijn voor een variërend aantal nodes telkens gemiddelde, standaardafwijking en grenzen van een 80%-betrouwbaarheidsinterval berekend.

Als bijvoorbeeld 40 nodes willekeurig worden gekozen uit de 130, dan is uit de linkergrafiek op te maken dat de iteraties naar verwachting gemiddeld 67 seconden duren. Elke keer als de toepassing wordt uitgevoerd ligt de gemiddelde iteratieduur met 80% kans tussen de 40 en de 107 seconden. De standaardafwijking is 27 seconden. Dat de ondergrens van het betrouwbaarheidsinterval precies overeenkomt met het verschil tussen gemiddelde en standaardafwijking is overigens toeval.

De standaardafwijking neemt bij het maximum aantal nodes sterk af, zelfs tot nul. Dit moet worden verwacht, vanwege het gebruikte selectieproces. Zo is de ordening van nodes van geen belang als de toepassing wordt uitgevoerd op alle 130 nodes. In dit geval is namelijk de totale verwerkingstijd altijd gelijk. Het gemiddelde en de grenzen van het betrouwbaarheidsinterval zijn aan elkaar gelijk en de standaardafwijking is nul omdat alle 10.000 gemiddelde iteratieduren gelijk zijn aan het gemiddelde ervan.

De bruikbaarheid van de analyse wordt beperkt door het redelijk kleine aantal nodes waarop verwerkingstijden zijn gemeten. Het is te hopen dat een toekomstige, grootschalige gridomgeving meer keuzevrijheid biedt ten aanzien van meewerkende nodes. Voornamelijk het linkerdeel van de grafieken is inzichtverhogend. Als het aantal nodes toeneemt richting het maximum van 130 ontstaat een steeds toenemende systematiek vanwege de beperkte keuzevrijheid.

Wat opvalt, is de plotselinge toename van de bovengrenzen van de betrouwbaarheidsintervallen in beide grafieken bij toevoeging van de 13^{e} node. Ook de ondergrens in de linkergrafiek neemt sterk toe, van 71 naar 108, bij toevoeging van de 118^{e} node. In dit geval daalt de kans dat deze nodes juist niet worden geselecteerd tot beneden de 10%. Deze plotselinge stijging is echter niet goed zichtbaar in de rechtergrafiek. Een verklaring hiervoor zal later gegeven worden.

6.4. Gereduceerde Fluctuaties tussen Prestatieniveaus door Uitsluiting van Nodes

In het vorige hoofdstuk is berekend dat node 41 een zeer grote bijdrage levert aan de gemiddelde iteratieduur. Als node 41 alleen werkt, bedraagt de gemiddelde iteratieduur maar liefst 105 seconden. Wanneer node 41 niet meewerkt, bedraagt het minimum hiervan nog slechts 49 seconden, terwijl het maximum 71 seconden bedraagt.

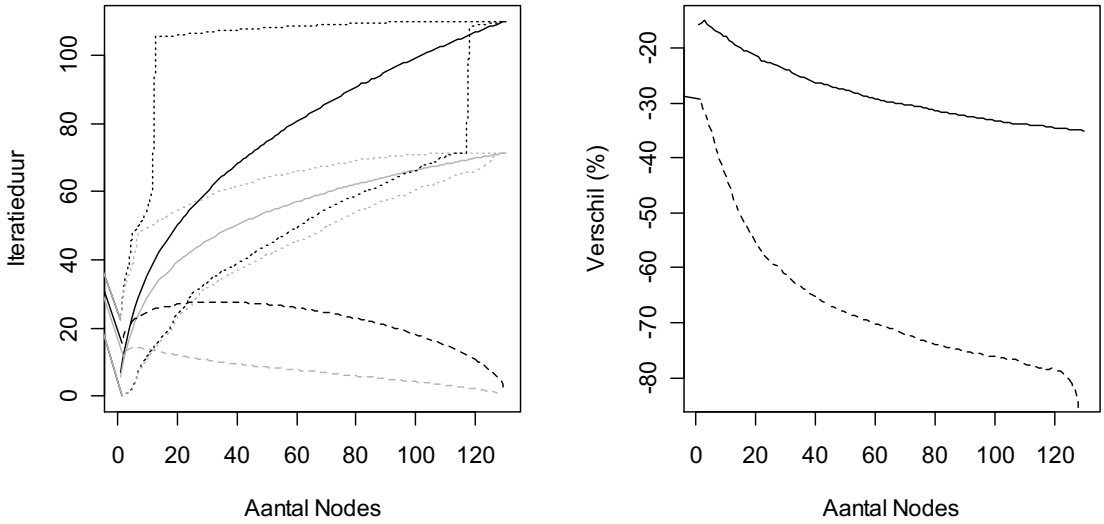
Het gedrag van het gemiddelde en de boven- en ondergrenzen van het in Figuur 12 weergegeven 80%-betrouwbaarheidsinterval kunnen hierdoor in belangrijke mate worden verklaard.
De bovengrens van het betrouwbaarheidsinterval stijgt plotseling zeer sterk bij 13 nodes, de ondergrens bij 118 meewerkende nodes.

Het betrouwbaarheidsinterval geeft aan dat de iteratieduren in 80% van de 10.000 willekeurige ordeningen tussen haar boven- en ondergrens liggen. 10% van de waarnemingen ligt dus boven de bovengrens en 10% ervan onder de ondergrens.

Wanneer op willekeurige wijze een aantal nodes worden geselecteerd uit een totaal van 130, stijgt de kans dat node 41 geselecteerd wordt tot boven de 10%, precies wanneer 13 nodes worden geselecteerd[2]. Ingeval 13 nodes meewerken zijn iets meer dan 10% van de 10.000 gemiddelde iteratieduren zeer vertraagd door node 41. De bovengrens laat dus ook precies bij 13 nodes een sterke stijging zien. Wanneer minder nodes meewerken is deze kans lager dan 10% en dus zijn iets minder dan 1.000 gemiddelde iteratieduren berekend waarop node 41 invloed heeft. De bovengrens is bij minder dan 13 nodes dus totaal niet beïnvloed door node 41. Overigens kan dit slechts met statistische zekerheid worden gesteld onder voorwaarde dat een voldoende groot aantal willekeurige ordeningen zijn beschouwd.

[2] Dit komt overeen met de kans dat het aantal gekozen witte ballen 1 bedraagt bij een aselecte keuze van 13 ballen uit 130, waarvan er slechts 1 wit is. De bijbehorende welbekende kansverdeling is de *hypergeometrische*.

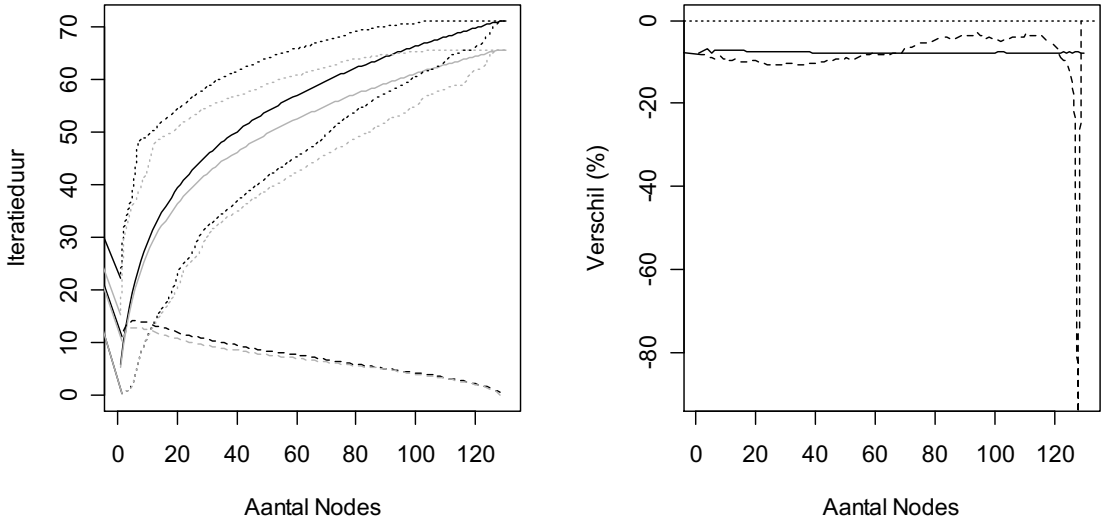
Verder daalt de kans dat node 41 niet wordt geselecteerd tot beneden de 10%, precies wanneer 118 nodes worden geselecteerd. Dit is ook precies het aantal nodes waarvoor de ondergrens van het betrouwbaarheidsinterval plotseling stijgt.



Figuur 10 Vergelijking tussen de gemiddelde iteratieduren als selectie van node 41 mogelijk is (zwart) en wanneer deze is uitgezonderd van meewerking (grijs). [R7]

De sterke invloed van node 41 op de prestatiestatistieken wordt gekwantificeerd in Figuur 10. Deze vergelijkt gemiddelde, standaardafwijking en een betrouwbaarheidsinterval van de gemiddelde iteratieduur mét en zónder node 41, op dezelfde wijze als in Figuur 9. De zwarte lijnen in de linkergrafiek zijn berekend in de situatie waarin node 41 beschikbaar is, maar dat het niet zeker is of deze meewerkt. De grijze lijnen zijn berekend als node 41 bij voorbaat is uitgezonderd van meewerking.

De rechtergrafiek in de figuur laat het verschil in procenten zien tussen de gemiddeldes en standaardafwijkingen in de linkergrafiek. De ononderbroken lijn laat zien hoe het gemiddelde afneemt, de onderbroken lijn doet hetzelfde voor de standaardafwijking.



Figuur 11 Vergelijking tussen gemiddeldes (ononderbroken) en standaardafwijkingen (onderbroken) van de gemiddelde iteratieduren, gebaseerd op alle nodes uitgezonderd 41 (zwart) en alle uitgezonderd 41 en 28 (grijs). [R7]

De grafieken laten nogmaals zien dat node 41 bij 130 meewerkende nodes verantwoordelijk is voor ongeveer 35% van de gemiddelde iteratieduur en dus ook 35% van de totale verwerkingstijd.
De grafiek laat verder zien wat de invloed is van node 41 als minder nodes meewerken. Naast het gedrag van de grenzen van het betrouwbaarheidsinterval, blijkt ook het gedrag en de grootte van de standaardafwijking grotendeels door node 41 te zijn veroorzaakt. Zonder node 41 neemt deze vrijwel lineair af onder toenemend aantal meewerkende nodes. Al bij 15 meewerkende nodes is de

standaardafwijking twee keer zo klein als wanneer node 41 meewerkt. De standaardafwijking neemt af met maar liefst ongeveer 80% bij 130 nodes, enkel door uitsluiting van node 41.

De vraag is, of het uitsluiten van extra nodes een vergelijkbare invloed heeft op de prestatiestatistieken. Figuur 11 vergelijkt daarom opnieuw het gedrag van de gemiddelde iteratieduur. Nu wordt node 41 bij voorbaat uitgezonderd en wordt een vergelijking gemaakt met de situatie waarin bovendien node 28 wordt uitgezonderd. Dit gebeurt opnieuw op dezelfde wijze als in Figuur 9. Node 28 heeft na node 41 de grootste invloed op de gemiddelde iteratieduur, zoals blijkt uit Tabel 3 en Figuur 4 op pagina 25.

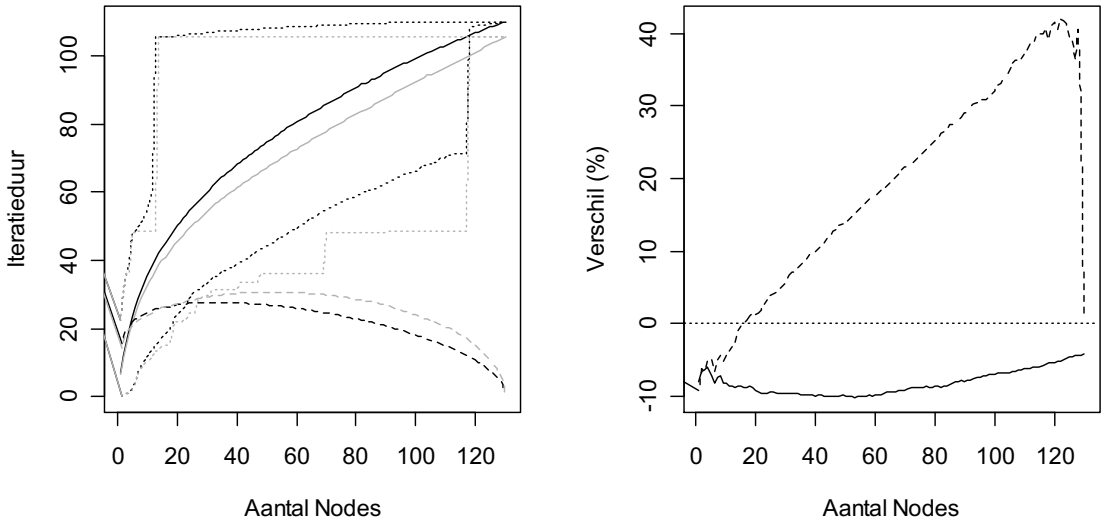
De grafieken laten zien dat de maximale gemiddelde iteratieduur zoals verwacht afneemt met 5,2 seconden. Dit is ongeveer 8% is van de gemiddelde iteratieduur van 71 seconden zonder node 41. De standaardafwijking is echter nauwelijks veranderd.

Het procentuele verschil van het gemiddelde is vrijwel constant ongeacht het aantal nodes. Ook de buitengewoon grote veranderingen in de standaardafwijking door uitsluiting van node 41 worden niet herhaald.

6.5. Het Ontbreken van Fluctuaties in Opeenvolgende Verwerkingstijden

In deze sectie wordt de gemiddelde iteratieduur vergeleken zoals die is verkregen met de ongewijzigde verwerkingstijden en de gemiddelde verwerkingstijden van de nodes.

In de linkergrafiek van Figuur 12 is het gemiddelde, de standaardafwijking en een 80%-betrouwbaarheidsinterval weergegeven van de gemiddelde iteratieduren in beide situaties.
De waardes in zwart behoren bij de ongewijzigde verwerkingstijden, de lichtere waardes behoren bij de gemiddelde verwerkingstijden. De rechtergrafiek geeft een procentueel verschil aan tussen de gemiddeldes en de standaardafwijkingen. Geen van de nodes is uitgesloten van meewerking.



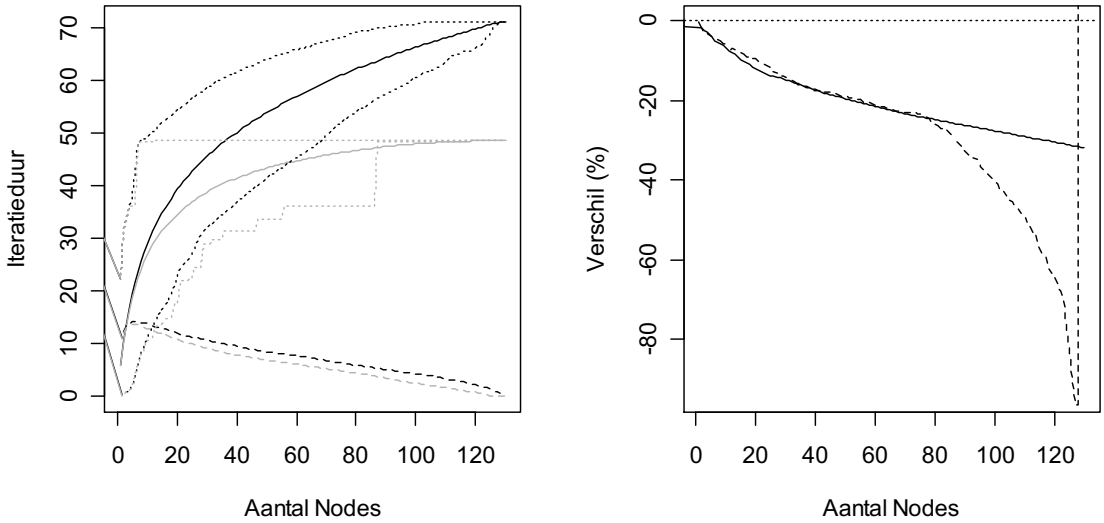
Figuur 12 Vergelijking tussen gemiddeldes (ononderbroken) en standaardafwijkingen (onderbroken) van de gemiddelde iteratieduren, gebaseerd op de oorspronkelijk gemeten verwerkingstijden (zwart) en deze zonder fluctuaties (grijs). [R7]

Uit de figuur blijkt dat de gemiddelde iteratieduur weliswaar iets afneemt door het ontbreken van fluctuaties in opeenvolgende verwerkingstijden, maar niet bijzonder veel. Bij 53 meewerkende nodes wordt het grootste procentuele verschil gemeten, namelijk 10%. Het grootste absolute verschil tussen de gemiddeldes bedraagt slechts 8 seconden en dit treedt op bij 82 nodes.

De standaardafwijking neemt over het algemeen relatief toe. Deze is tot 16 nodes iets lager, is dan bijna gelijk tot ongeveer 24 nodes, maar blijft stijgen, tot 31 bij 50 nodes. Met de oorspronkelijke verwerkingstijden bedraagt het maximum 28 seconden, bij 34 nodes.

Het procentuele verschil tussen de standaardafwijkingen is dus ook bijna nul bij 16 nodes, maar blijft vrijwel proportioneel toenemen met het aantal nodes. Bij meer dan 118 nodes wisselen stijgingen en dalingen elkaar af en verandert dit beeld. Bij 130 nodes zijn de standaardafwijkingen in beide gevallen precies 0.

In Figuur 13 wordt dezelfde vergelijking opnieuw gemaakt. Nu echter is node 41 bij voorbaat uitgesloten van meewerking.

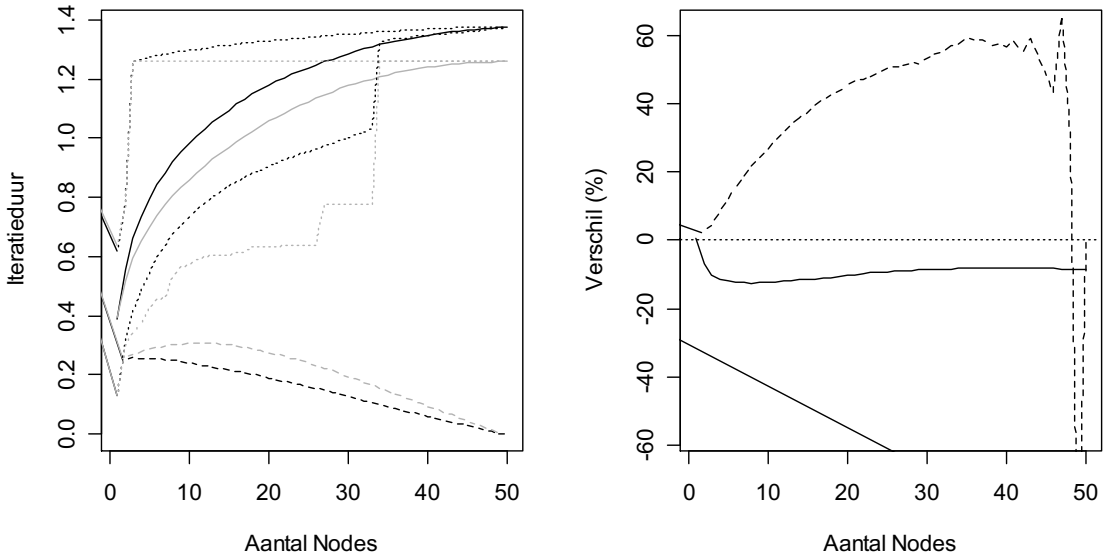


Figuur 13 Vergelijking tussen gemiddeldes (ononderbroken) en standaardafwijkingen (onderbroken) van de gemiddelde iteratieduren, gebaseerd op alle nodes uitgezonderd 41 (zwart) en deze zonder fluctuaties (grijs). [R7]

De afname van de gemiddelde verwerkingstijd verandert nu sterker bij toenemend aantal meewerkende nodes. Bij slechts één node zijn de verwerkingstijden gelijk en de invloed van de fluctuaties neemt vervolgens toe met het aantal nodes. Bij 130 meewerkende nodes is de gemiddelde iteratieduur maar liefst 32% lager door het ontbreken van fluctuaties in opeenvolgende verwerkingstijden. In plaats van tot maximaal 71 seconden, loopt nu de gemiddelde iteratieduur op tot slechts 49 seconden bij 130 meewerkende nodes, waar deze in Figuur 12 nauwelijks van de oorspronkelijke 110 seconden afweek.

In plaats van dat de standaardafwijking toeneemt door het elimineren van fluctuaties in opeenvolgende verwerkingstijden, neemt deze hierdoor nu af, ongeacht het aantal nodes. De gemiddelde iteratieduur in de grafiek is nu dus een betere indicatie voor de te verwachten gemiddelde iteratieduur. De hoge standaardafwijking werd voorheen primair veroorzaakt door de onzekerheid over het al dan niet meewerken van node 41. Nu is het ontbreken van korte termijn fluctuaties duidelijk een voordeel, doordat de voorspelde verwerkingstijd belangrijk afneemt en deze voorspelling bovendien betrouwbaarder blijkt te zijn.

Tenslotte vergelijkt Figuur 14 de prestatiestatistieken als een belangrijk aantal meest vertragende nodes zijn uitgezonderd van meewerking. De maar liefst 80 meest vertragende nodes zijn uitgesloten van meewerking. Merk op dat de maximale gemiddelde iteratieduur nog slechts 1,4 seconden bedraagt.



Figuur 14 Vergelijking tussen gemiddeldes (ononderbroken) en standaardafwijkingen (onderbroken) van de gemiddelde iteratieduren, gebaseerd op de 50 nodes met kleinste bijdrage (zwart) en deze zonder fluctuaties. (grijs). [R7]

Nu is de afname van de gemiddelde iteratieduur vanwege het elimineren van fluctuaties in opeenvolgende verwerkingstijden slechts klein. De standaardafwijking neemt echter toe. Fluctuaties in opeenvolgende verwerkingstijden hebben nu nog slechts een kleine invloed, maar deze invloed is niet zozeer negatief.

De gemiddelde iteratieduur wordt sterker beïnvloed door selectie van een node met een relatief hoge gemiddelde verwerkingstijd. Deze node zal altijd hoge verwerkingstijden laten meten, terwijl het aannemelijk is dat toch minimaal de helft van de verwerkingstijden op de node lager is dan het gemiddelde. Aangezien de prestaties van de nodes beter vergelijkbaar zijn en de iteratieduren dus steeds vaker worden bepaald door pieken, is het nu goed mogelijk dat ook een aantal iteratieduren korter zal blijken te zijn dan het hoogste gemiddelde van de geselecteerde nodes. Deze liggen dan dichter bij het gemiddelde van de 10.000 gesimuleerde gemiddelde iteratieduren.

Conclusies

7. Conclusies

7.1. Conclusies

In dit onderzoek is de totale verwerkingstijd van de Successive Over Relaxation toepassing gesimuleerd in verschillende scenario's en is een verband gelegd tussen de prestaties van deze toepassing en de prestatieniveaus van 130 nodes waarop verwerkingstijden waren gemeten.
De toepassing is in hoge mate parallel en heeft sterke afhankelijkheden tussen haar iteraties.
De nodes maken deel uit van een wereldwijde, realistische gridomgeving.

Er is bepaald hoeveel elk van deze nodes minimaal bijdraagt aan de totale verwerkingstijd van de toepassing en de nodes zijn gerangschikt in volgorde van afnemende bijdrage.

- Het overgrote deel van de totale verwerkingstijd (68%) wordt veroorzaakt door een klein aantal nodes (8%).
- 95% van de totale verwerkingstijd wordt veroorzaakt door 32% van de nodes.
- De meeste nodes (65%) dragen slechts zeer weinig bij aan de totale verwerkingstijd.
- Het overige deel (35%) van de nodes hebben prestatieniveaus die sterk verschillend zijn, zowel onderling als van de prestatieniveaus van de andere nodes.
- Er is 1 node die buitengewoon langzaam is.

Verder is bepaald hoeveel nodes de toepassing vertragen als deze wordt uitgevoerd op een steeds toenemend aantal nodes, waarbij telkens alleen die nodes meewerken die de totale verwerkingstijd zo min mogelijk verlengen.

- Nodes met een grote bijdrage aan de totale verwerkingstijd zijn ook de langzaamsten in een steeds groter aantal iteraties.
- Het aantal iteraties waarin de meest vertragende node de langzaamste is fluctueert steeds sterker als de bijdrage aan de totale verwerkingstijd groter wordt.
- Nodes die weinig bijdragen aan de totale verwerkingstijd zijn ook zelden de langzaamste.

Het gedrag van de gemiddelde iteratieduur en de gemiddelde verwerkingstijd per taak, inclusief idle-time, is middels simulatie onderzocht voor een oplopend aantal meewerkende nodes, als nodes op willekeurige wijze worden geselecteerd en geen enkele node wordt uitgesloten van meewerking.

- Beginnende bij 1 meewerkende node stijgt de totale verwerkingstijd relatief zeer snel met het aantal nodes.
- De snelheid van de stijging neemt echter gestaag af, maar stabiliseert bij ongeveer 50 nodes. De totale verwerkingstijd blijft dus ongeveer lineair stijgen.
- De gemiddelde verwerkingstijd per taak laat een omgekeerde relatie zien, maar deze convergeert naar een horizontale lijn. Het blijft dus nuttig om de toepassing in de praktijk op steeds meer nodes uit te voeren.

Vervolgens is middels simulatie onderzocht hoe deze statistieken zich gedragen als de meest vertragende nodes achtereenvolgens worden uitgesloten van meewerking.

- De meest vertragende node veroorzaakt 30% tot 80% van de grootte van de standaardafwijking van de gemeten gemiddelde iteratieduren en 15% tot 35% van het gemiddelde hiervan.
- De daaropvolgend meest vertragende node doet beide afnemen met ongeveer 10%, ongeacht het aantal nodes dat meewerkt.
- Als de prestatieniveaus van de nodes in toenemende mate vergelijkbaar zijn, daalt de totale verwerkingstijd eerst buitengewoon snel en tot zeer lage waarden. Deze is afgenomen van 110 seconden tot 35 seconden met 120 nodes en tot 1,4 seconden met 50 nodes.

Tenslotte werd de invloed van fluctuaties in opeenvolgende verwerkingstijden gedemonstreerd, door deze te elimineren. De verwerkingstijden werden hiertoe aangepast, zodat deze voor een zekere node gedurende alle iteraties gelijk was aan de gemiddelde verwerkingstijd op die node. Deze situatie werd vergeleken met de situatie waarin de verwerkingstijden ongewijzigd zijn.

- De invloed van fluctuaties in opeenvolgende verwerkingstijden lijkt groot te zijn en afhankelijk van het aantal nodes, als de prestatieniveaus van nodes sterk uiteenloopt.
- Deze lijkt sterk afgenomen te zijn en onafhankelijk van het aantal nodes, als de prestatieniveaus van nodes beter vergelijkbaar is.

7.2. Enorme Reductie van Verwerkingstijd Mogelijk met behulp van Eenvoudige Methode

Het overgrote deel van de totale verwerkingstijd (68%) wordt veroorzaakt door een klein aantal nodes (8%). Deze nodes zijn ook in de meeste iteraties vertragend, omdat ze vaak de langste verwerkingstijden laten meten. Deze nodes kunnen dus eenvoudig worden geïdentificeerd.

Dit zijn sterke argumenten voor een eenvoudige prestatieverhogende strategie. Een eenvoudige methode kan dus al een grote snelheidswinst boeken. Implementatie ervan kan ook verwerkingstijd-efficiënt, omdat de identificatiemethode snel is en eenvoudig te implementeren. De belangrijkste operaties zijn het sorteren van verwerkingstijden en het berekenen van de verschillen hiertussen.

Er kan een verband worden gelegd tussen de traagste nodes in een aantal opeenvolgende iteraties door in de praktijk zuivere verwerkingstijden te meten, exclusief idle-time, van een parallelle toepassing. Als een node relatief vaak traag is, is het ook waarschijnlijk dat deze de totale verwerkingstijd negatief beïnvloedt. Al na een beperkt aantal iteraties kunnen de trage nodes worden geïdentificeerd, ookal is een node in nog geen enkele iteratie de traagste geweest.

Waar in dit onderzoek de prestaties van de Successive Over Relaxation toepassing zijn verhoogd door nodes uit te sluiten, kan er in de praktijk voor worden gekozen om het werk dat de meest vertragende nodes krijgen toegewezen te reduceren in volgende iteraties. Tenminste, als de toepassing dit ondersteunt.

7.3. Vooruitzicht op Eventueel Vervolgonderzoek

Dit onderzoek heeft de communicatietijden volledig buiten beschouwing gelaten. Ook zijn de afhankelijkheden tussen opeenvolgende iteraties vereenvoudigd, zodat elke node telkens op alle andere nodes wacht, alvorens aan een nieuwe iteratie te beginnen. Bovendien zijn de taken die de Successive Over Relaxation toepassing de meewerkende nodes telkens laat uitvoeren precies even groot. Een dieper onderzoek zou de invloed van fluctuaties in communicatietijden kunnen laten meetellen en de invloed van fluctuaties in verwerkingstijden kunnen onderzoeken voor toepassingen die een minder eenvoudige afhankelijkheid hebben tussen hun iteraties, of die taken laten uitvoeren die niet allemaal precies even groot zijn.

Appendices

A. Indexnummers en Labels van de Gebruikte Nodes

Index	Label	Index	Label	Index	Label
1	arizona1_10223	45	caltech2_20587	88	ucsd2_26193c
2	arizona1_20587	46	cam1_28634	89	ucsd2_8912c
3	arizona1_24834_01	47	china2_18999	90	utah1_11547c
4	arizona1_24834_10	48	china2_2225	91	utah1_24385c
5	arizona1_28347c	49	dk1_19582	92	utah1_28347c
6	arizona1_29993c	50	dk1_24834_01	93	utah1_29977
7	arizona1_4321c	51	dk1_24834_02	94	utah1_29993c
8	arizona1_5315c	52	dk1_24834_09	95	utah1_5315c
9	arizona1_9736	53	dk1_24834_10	96	utah2_20587
10	au1_14228	54	inria1_23177	97	utah2_458
11	au1_2235_01	55	inria1_3736_01	98	utah3_4321c
12	au1_2235_02	56	inria1_3736_02	99	utah3_8912c
13	au1_2235_09	57	inria1_3736_03	100	vu1_2235_01
14	au1_2235_10	58	inria1_3736_10	101	vu1_2235_02
15	au2_18999	59	madrid1_14228	102	vu1_2235_09
16	boston1_15363c	60	msu1_18999	103	vu1_2235_10
17	boston1_25553c	61	msu1_2225	104	vu1_23975
18	boston1_26193c	62	ntu1_22916	105	vu1_3736_01
19	boston1_4321c	63	ntu1_2692	106	vu1_3736_02
20	boston1_8912c	64	seoul2_32445	107	vu1_3736_03
21	ca1_20312	65	singapore1_17853_01	108	vu1_3736_04
22	ca1_3736_01	66	singapore1_17853_02	109	vu1_3736_09
23	ca1_3736_09	67	singapore1_17853_10	110	vu1_3736_10
24	ca1_3736_10	68	singapore1_24604	111	vu2_14228
25	ca2_20587	69	singapore1_27986	112	vu2_2988
26	caltech1_11547c	70	telaviv1_14228	113	warsch1_2235_01
27	caltech1_14228	71	telaviv1_2235_01	114	warsch1_2235_05
28	caltech1_15363c	72	telaviv1_2235_08	115	warsch1_2235_09
29	caltech1_15370	73	telaviv1_2235_10	116	warsch1_2235_10
30	caltech1_2235_01	74	tw1_19254	117	wash1_11547c

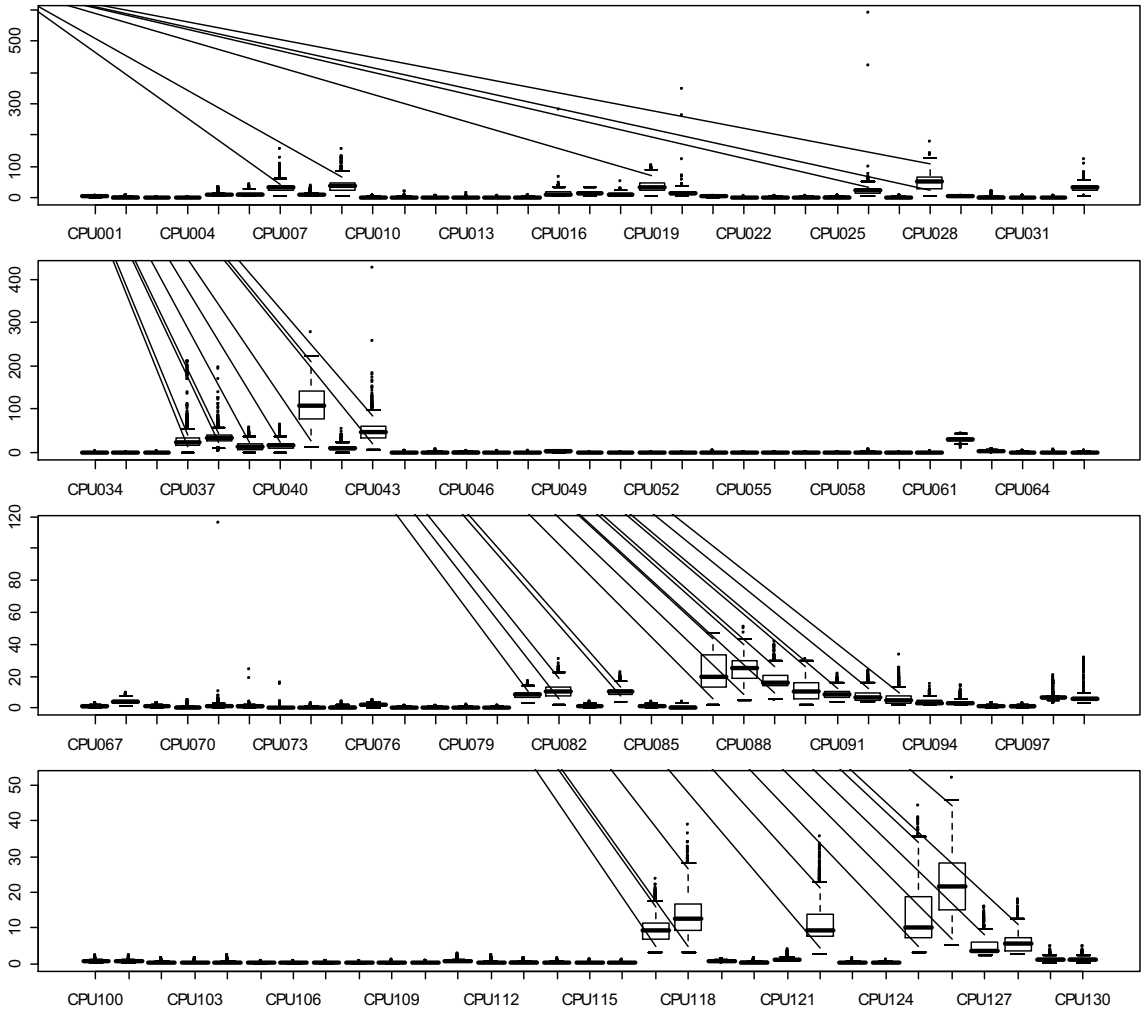
Index	Label	Index	Label	Index	Label
31	caltech1_2235_08	75	tw1_23975	118	wash1_15363c
32	caltech1_2235_10	76	ucsb1_22599	119	wash1_18947
33	caltech1_24385c	77	ucsb1_3736_01	120	wash1_18999
34	caltech1_24834_01	78	ucsb1_3736_05	121	wash1_20587
35	caltech1_24834_02	79	ucsb1_3736_09	122	wash1_24385c
36	caltech1_24834_10	80	ucsb1_3736_10	123	wash1_24834_01
37	caltech1_25553c	81	ucsd1_11547c	124	wash1_24834_09
38	caltech1_26193c	82	ucsd1_15363c	125	wash1_25553c
39	caltech1_28347c	83	ucsd1_23975	126	wash1_26193c
40	caltech1_29993c	84	ucsd1_24385c	127	wash1_28347c
41	caltech1_4321c	85	ucsd1_2988	128	wash1_29993c
42	caltech1_5315c	86	ucsd1_4286	129	wash1_458
43	caltech1_8912c	87	ucsd2_25553c	130	wash1_5315c
44	caltech2_18999				

Tabel 6 Indexnummer en labels van de gebruikte nodes.

B. Grafische Samenvatting van de Verzamelde Verwerkingstijden

Figuur 15 bevat één *box-and-whisker* plot, kortweg *boxplot*, voor elke node waarop verwerkingstijden zijn gemeten. Elke boxplot geeft in één oogopslag een beeld van de lengte van de verwerkingstijden gemeten op één node. De grafieken tezamen staan toe dat kernstatistieken van de verwerkingstijden op de nodes onderling vergeleken worden. Zie de uitleg bij Tabel 5 in sectie 5.5 voor een uitleg over mediaan, eerste kwartiel en derde kwartiel.

De box in elke grafiek omvat eerste en derde kwartiel. De horizontale lijn door de box ligt ter hoogte van de mediaan. De hoogte van de box wordt het *interkwartielbereik* genoemd. De horizontale lijnen buiten de box liggen precies anderhalf maal het interkwartielbereik buiten de box, tenzij het minimum (voor de onderste lijn) of maximum (voor de bovenste lijn) dichter bij de box ligt.
In dat geval is de horizontale lijn onder, respectievelijk boven de box weergegeven ter hoogte van minimum of maximum. De punten die nog verder dan anderhalf maal het interkwartielbereik buiten de box liggen, worden *uitbijters* genoemd. Deze worden individueel aangegeven, elk met een klein rondje.



Figuur 15 Eén box-and-whisker plot voor elke node waarop verwerkingstijden zijn gemeten. [R1]

Index

Literatuurindex

Abbate, Janet (1999) *Inventing the Internet*, Cambridge MIT Press

Dinda, Peter and O'Hallaron, David (2000) Host load prediction using linear models, *Cluster Computing, 3-4:265-280*

Dinda, Peter (2002) Online prediction of the running time of tasks, *Cluster Computing 5-3:225-236, Juli 2002*

Dinda, Peter (2002) A prediction-based real-time scheduling advisor, Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), *IEEE Computer Society ISBN 0-7695-1573-8:10-17, April 2002*.

distributed.net, *http://www.distributed.net*

Dobber, Menno, Ger Koole, and Rob van der Mei (2004) Dynamic load balancing for a grid application, Proceedings of High Performance Computing 2004 (HiPC 2004)*, Springer-Verslag ISBN 3-5402-4129-9:342-352, Januari 2005*

Dobber, Menno, Ger Koole, and Rob van der Mei (2005) Dynamic load balancing experiments in a grid, *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005) Pagina's 123-130, Mei 2005*

Dobber, Menno, Ger Koole, and Rob van der Mei (Mei 2006) Statistical Properties of Task Running Times in a Global-Scale Grid Environment, Sixth IEEE International Symposium on Cluster Computing and the Grid, 2006 (CCGRID 06)*, IEEE Press ISBN 0-7695-2585-7:150-153, Mei 2006*

Dobber, Menno, Ger Koole, and Rob van der Mei (Juni 2006) Effective Prediction of Job Processing Times in a Large-Scale Grid Environment, High Performance Distributed Computing, 2006 15th IEEE International Symposium, *IEEE Press ISBN 1-4244-0307-3:359-360, Juni 2006*

Evans, D.J. (1984) Parallel SOR iterative methods, *Parallel Computing 1:3–18*

Foster, Ian (2002), What is the Grid? A three point checklist, Argonne National Laboratory & University of Chigago, *http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf*

Hageman, L.A. en D.M. Young (1981) *Applied Iterative Methods*, Academic Press

Harchol-Balter, Mor en Allen Downey (1996) Exploiting process lifetime distributions for dynamic load balancing, SIGMETRICS '96 Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, *ACM Press ISBN 0-89791-793-6:13:24*

Hayes, Brian (1998) Collective Wisdom, *American Scientist 86-2:118-122*

Kondo, Derrick, Michela Taufer, Charles Brooks, Henri Casanova en Andrew Chien (2004) Characterizing and evaluating desktop grids: An empirical study, Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'04), *IEEE Computer Society ISBN 0-7695-2132-0:26*, April 2004

Kovács, József and Péter Kacsuk (2004) A migration framework for executing parallel programs in the Grid, *http://grid.ucy.ac.cy/axgrids04/AxGrids/papers/axgrid2004-migration-kovacs-final-draft.pdf*

Ledlie, Jonathan and Margo Seltzer (2005) Distributed, Secure Load Balancing with Skew, Heterogeneity, and Churn, *http://www.eecs.harvard.edu/~jonathan/lb/*

Leland, Will en Teunis Ott (1986) Load-balancing heuristics and process behaviour, Proceedings of the 1986 ACM SIGMETRICS joint international conference on Computer performance modelling, measurement and evaluation, *ACM Press ISBN 0-89791-184-9:54-69*

Leopold, Claudia (2000) Parallel and Distributed Computing: A Survey of Models, Paradigms, and Approaches, *John Wiley & Sons, Inc, ISBN 0-471-35831-2*

Mutka, Matt en Miron Livny (1991) The available capacity of a privately owned workstation environment, *Performance Evaluation 12-4:269-284*

Paxson, Vern en Sally Floyd (1995) Wide-Area Traffic: The Failure of Poisson Modeling, *IEEE/ACM Transactions on Networking 3-3:226-244*

PlanetLab, An open platform for developing, deploying and accessing planetary-scale services, *http://www.planet-lab.org*

Shekhar, S, S. Ravada, V. Kumar, D. Chubb, G. Turner (1995) Load-Balancing in High Performance GIS: A Summary of Results, *http://www.cs.umn.edu/research/shashi-group/paper_ps/HiPC.ps*

Shoch, John F., and Jon A. Hupp (1982) The "Worm" programs. Early experience with a distributed computation, *Communications of the ACM 25:172-180*

Wolfram Mathworld, the web's most extensive mathematics resource, *http://mathworld.wolfram.com*

Tabellen

Figuren