Het Synchronisatieprobleem bij Parallelle Discrete Simulatie

BWI Werkstuk

Februari 2006

Stefan Reijmers

Begeleider: drs. Menno Dobber

Voorwoord

Dit werkstuk, het zogehete BWI-werkstuk, is een verplicht onderdeel van de studie bedrijfswiskunde en informatica. Naar aanleiding van het gekozen onderwerp doet de student een onderzoek. Vaak zullen de bronnen van het onderzoek literatuur zijn, maar het kan ook een computerprogramma zijn. Het onderwerp van dit werkstuk is ontstaan uit een stage tijdens deze studie. Het werkstuk beschrijft een probleem wat ontstaat bij parallel simuleren met daarbij de mogelijke oplossingen voor dit probleem. Het doel van het werkstuk is dat de student voor een deskundige manager op een heldere wijze een probleem beschrijft. Dit vereist enige toelichting:

- deskundige manager: de student moet er vanuit gaan dat de manager een algemene deskundigheid heeft op het gebied van het onderwerp;
- heldere wijze: een manager heeft slechts beperkt tijd, het werkstuk zal daarom beknopt moeten zijn.

Het werkstuk heeft als doelstellingen:

- nadruk leggen op het bedrijfsgerichte aspect van de studie, naast wiskunde en informatica aspecten;
- een eenvoudig (literatuur)onderzoek zelfstandig uitvoeren (coproducties zijn niet toegestaan);
- op adequate wijze, zowel wat betreft wetenschappelijkheid als de manier waarop (schriftelijk en mondeling), presenteren van de resultaten;
- binnen een bepaalde periode voltooien van het onderzoek.

In het werkstuk refereer ik naar mijn stage bij Roto Smeets Deventer. Ik heb daar een simulatieprogramma geschreven. Ik wil hun graag bedanken dat ik mede door de stage het onderwerp van mijn werkstuk heb kunnen bedenken en dat ik hun ter referentie heb mogen gebruiken voor dit werkstuk. Tot slot wil graag mijn begeleider, Menno Dobber, bedanken voor zijn inzet tijdens het schrijven van dit werkstuk.

Inhoud

1. Simulatie

In dit hoofdstuk wordt kennis gemaakt met simulatie. In de eerste paragraaf wordt uitgelegd hoe simulatie is ontstaan en hoe de computer als rekenmachine hierbij gebruikt kan worden. De tweede paragraaf geeft uitleg over de manier waarop processors communiceren om bepaalde processen te versnellen. De twee daarop volgende paragrafen geven een beeld van de verschillende vormen van simulatie. De laatste paragraaf gaat dieper in op de verschillende analytische typen.

1.1 Waarom simulatie?

Nog geen eeuw geleden werd de computer uitgevonden met als doel ponskaarten te lezen en te verwerken, maar al snel werd duidelijk dat de computer voor veel meer doeleinden te gebruiken was. Eén van deze doeleinden is simuleren.

Simulatie is een handige *tool* om voorspellingen mee te doen over mogelijke toekomstige gebeurtenissen binnen aanzienlijke tijd. Met aanzienlijke tijd wordt bedoeld dat er redelijkerwijs op te wachten is. Het is eveneens de bedoeling dat de tijd van het simulatieproces, ook wel de gesimuleerde tijd genoemd, sneller verloopt dan de werkelijke tijd (simulatietijd), want anders zou het geen voorspelling zijn, maar eerder een bevestiging van de werkelijkheid.
De meest voorkomende vorm van simuleren is gebaseerd op gebeurtenissen, ook wel *events* genoemd. Elk moment dat er een gebeurtenis plaatsvindt, verandert de staat van de simulatieomgeving en past deze zich aan om verder te simuleren. Dit gaat door tot alle *events* hebben plaatsgevonden waarna de simulatie stopt. Hoofdstuk 2 gaat hier uitgebreider op in.

Vrij eenvoudige data structuren zorgen er voor dat deze *events* sequentieel worden uitgevoerd. Deze vorm van simuleren vergt bij complexe analytische problemen zeer veel tijd. Een oplossing om tijd te winnen is om parallel te gaan simuleren. Parallelle data structuren verdelen de berekeningen over meerdere processors, waardoor er een grotere rekencapaciteit ontstaat en de simulatie sneller verloopt. Een vraag die kan worden gesteld is in hoeverre parallelle simulatie sneller is dan simuleren op één enkele computer? In het derde hoofdstuk wordt hier nader op ingegaan. In de volgende paragraaf wordt uitgelegd op welke manier er gecommuniceerd wordt tussen meerdere processors en vervolgens wordt ingegaan op de verschillende simulatietechnieken.

1.2 Parallelle versus gedistribueerde computers

Het feitelijke verschil tussen parallelle en gedistribueerde computers wordt veroorzaakt door de afstand tussen de communicerende computerprocessors. De parallelle computerprocessors bevinden zich vaak in één behuizing of kamer, terwijl gedistribueerde computers zijn verdeeld over het gehele land of zelfs wereldwijd.
De manier waarop de computers communiceren verschilt eveneens. De parallelle computers communiceren meestal door middel van een bus of switch en de gedistribueerde computers via een lokaal netwerk (LAN) of het Internet (WAN). Het gevolg hiervan is dat er een verschil optreedt in de snelheid waarmee wordt gecommuniceerd. Bij de parallelle computers is dit vaak minder dan honderd microseconden en bij gedistribueerde computers vele honderden microseconden per berichten uitwisseling.

1.3 Virtuele omgevingssimulatie

Een van de bekendste vormen van simuleren is de virtuele omgevingssimulatie, waarbij een realistische representatie van de omgeving wordt gesimuleerd in *real-time*. De gesimuleerde tijd van de simulatie loopt gelijk aan de simulatietijd. De simulatie wordt door de actief deelnemende persoon in deze virtuele wereld telkens aangepast door bepaalde handelingen die hij verricht. Tot slot heeft virtuele omgevingssimulatie het kenmerk alleen die elementen te simuleren die door de deelnemer worden waargenomen. Als de deelnemer een ander vliegtuig ziet vliegen is het alleen interessant om te zien waar deze heen vliegt. Wat er in dat andere vliegtuig gebeurt, is op dat moment niet interessant.
Een bekend voorbeeld van een virtuele omgevingssimulator is de vliegsimulator. Deze simulator wordt gebruikt om piloten te trainen tijdens hun opleiding. Het is een zeer bruikbare simulator, omdat het beginnende piloten de mogelijkheid biedt verschillende scenario's mee te maken en het traint hun op een effectieve manier hoe te reageren in verschillende situaties. Ook het leger gebruikt soortgelijke simulators om een oorlogsscenario na te bootsen in een virtuele wereld. Dit soort simulators wordt gebruikt om mensen te trainen zonder mogelijke ernstige gevolgen. Door middel van *best practice* zal de deelnemer de simulatie na een aantal keer oefenen steeds beter uitvoeren en de missie tot een goed einde brengen.
Omdat de piloten en militairen uiteindelijk een grote verantwoordelijkheid dragen in de *real-world* moet de virtuele omgevingssimulatie een goed beeld geven van de werkelijkheid. In enkele gevallen

zal de gesimuleerde tijd iets sneller of langzamer lopen, om bijvoorbeeld tijd te winnen of iets in slowmotion te laten zien.

1.4 Analytische simulatie

Een totaal andere vorm van simuleren is de analytische simulatie. Bij analytische simulatie gaat het eveneens om het nabootsen van de werkelijkheid. In tegenstelling tot de virtuele omgevingssimulatie moet er echter zo snel mogelijk antwoord gegeven worden op vragen over mogelijke toekomstige situaties. De simulatietijd loopt daarom vaak sneller dan de gesimuleerde tijd. Dit spreekt voor zich, omdat het niet praktisch is een voorspelling te geven over wat gelijktijdig plaats vindt of reeds in het verleden heeft plaatsgevonden. Statistiek speelt hierin een belangrijke rol: het helpt de antwoorden te interpreteren om een voorspelling te doen over mogelijk te ontstane toekomstige situaties. Bij de analytische vorm blijft de rol van de mens beperkt tot een externe waarnemer en is de simulatie puur een kwantitatieve analyse van de werkelijkheid. De analytische simulatie reproduceert zo nauwkeurig mogelijk de toestand voor en na gebeurtenissen.

Een praktisch voorbeeld is de stage die ik gelopen heb bij Roto Smeets in Deventer. Zij hebben een machinepark van diepdrukpersen en nabewerkingsmachines. Deze twee afdelingen hebben een gezamenlijke opslag. De diepdrukpersen vullen deze opslag met drukwerk en de nabewerkingsmachines legen deze opslag en verwerken het drukwerk tot tijdschriften. Roto Smeets wilde weten wat er gebeurt als de samenstelling van hun drukwerk zou verandert. Hierbij valt te denken aan een toename in volume, maar ook de verhouding tussen retailbladen en tijdschriften. Dit soort kwantitatieve waarden zijn sprekende voorbeelden als het gaat om discrete *event* simulatie (zie paragraaf 1.5). Naast het aanpassen van dit soort kwantitatieve waarden is het ook mogelijk diepdrukpersen en nabewerkingsmachines toe te voegen en te verwijderen binnen het simulatieprogramma. Het simulatieprogramma kan verschillende toekomstige scenario's uitwerken, waardoor Roto Smeets aan de hand van de uitkomsten in de werkelijkheid handig hierop kan anticiperen.
Zoals bij een *event* gestuurde analytische simulatie de bedoeling is, duurde een simulatie van drie maanden in werkelijkheid ongeveer twaalf seconden op een pentium IV, 3.0 GHz met 512 Mb werkgeheugen. Met één druk op de knop starten de runs. Roto Smeets kan daarna binnen enkele tellen over de resultaten beschikken. Als alle runs gedaan gedaan zijn, kan de data uit een bestand worden gelezen en aan de hand daarvan een aantal grafieken geplot worden. Deze waarden worden vervolgens vergeleken met de werkelijkheid om op deze manier de simulatieruns te toetsen. De

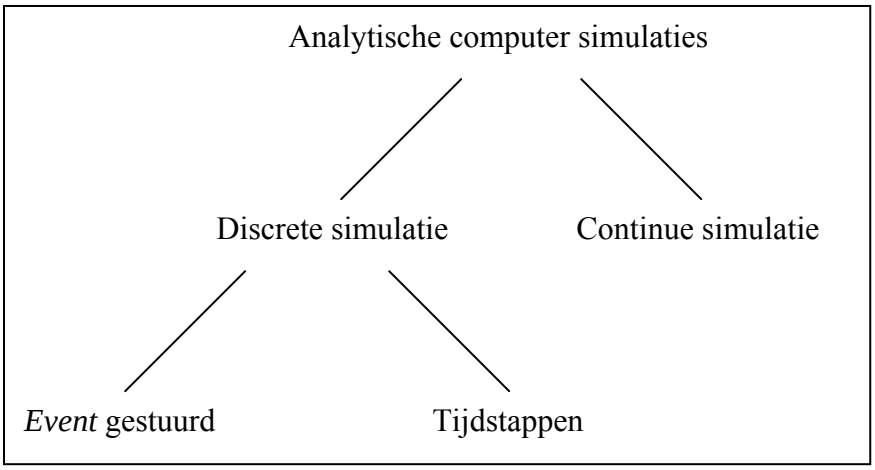
waarden kwamen sterk overeen met de werkelijkheid en statistische berekeningen laten zien dat ze significant correct zijn.

1.5 Typen analytische simulaties

In de vorige twee paragrafen is gesproken over mogelijke simulatievormen. Binnen de analytische vorm is onderscheid te maken tussen continue en discrete simulatie. De continue simulatie is te omschrijven als een simulatie waarbij op ieder tijdstip, met andere woorden continu, gegevens veranderen. In de praktijk wordt deze simulatie nauwelijks toegepast, omdat de kosten van continue simulatie de baten niet overstijgen. Bij discrete simulatie zijn er uitsluitend veranderingen in de toestand op vaste tijdstippen. Er zijn twee typen discrete analytische simulaties. Om een beeld te krijgen bij de typen analytische discrete simulaties volgt er een tweetal voorbeeld.

Als eerste wil bijvoorbeeld een aandeelhouder de waarde van een aandeel over de tijd simuleren. Het simuleren van de waarde van een aandeel is een continu proces en nagenoeg niet te berekenen op ieder tijdstip. Een mogelijkheid om rekentijd te besparen is om de werkelijkheid met een discreet model te benaderen. Men kan dit doen aan de hand van vaste tijdstappen van bijvoorbeeld een seconde. Op deze manier wordt er om de seconde een voorspelling gedaan naar de waarde van het aandeel.
Een ander voorbeeld waarbij *event* gestuurde simulatie goed werkt is het simuleren van verschillende kassarijen bij de supermarkt. Een *event* is dan bijvoorbeeld de aankomst van een klant of het vertrekken van een klant. Afhankelijk van hoe gedetailleerd de simulatie aan de werkelijkheid moet voldoen kunnen er meer of minder *events* zijn waarmee rekening gehouden dient te worden. Het is vanzelfsrpekend dat de hoeveelheid *events* bepalend zijn voor de snelheid van de simulatie.

In figuur 1.1 wordt het classificatieschema van de verschillende typen nog eens schematisch weergegeven. In dit werkstuk gaat de aandacht uitsluitend verder uit naar discrete *event* simulatie.



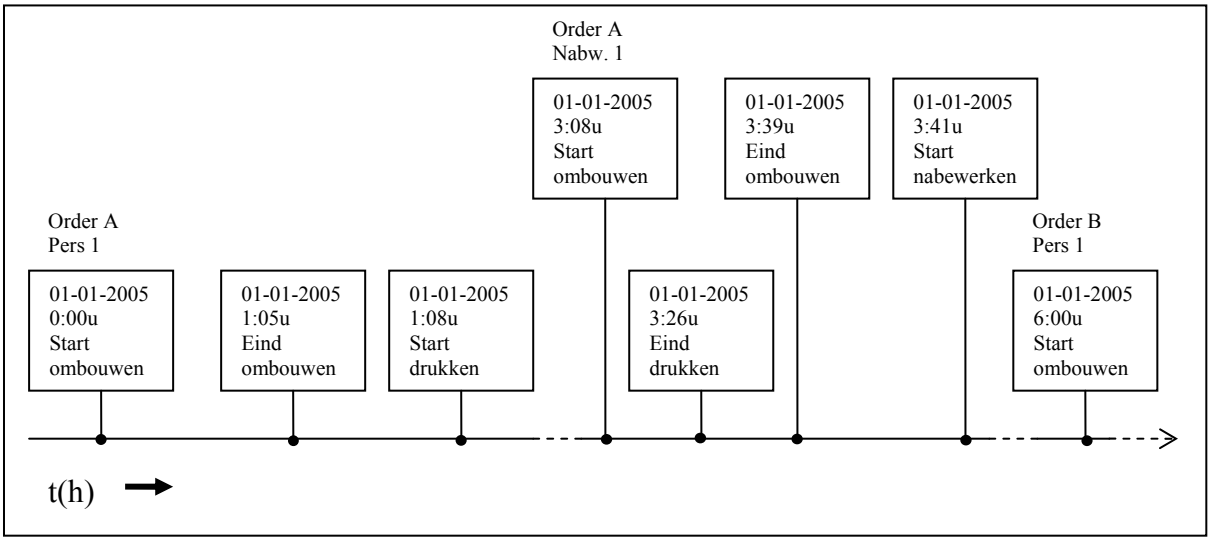
Figuur 1.1 Classificatie van simulatie typen.

2 Discrete event simulatie

Dit hoofdstuk gaat verder in op analytische discrete *event* simulatie. Uit ervaring is gebleken dat een simulatie met een groot tijdsbestek veel tijd vergt om tot een uiteindelijk resultaat te komen. In de eerste paragraaf komt een simulatie met een groot tijdsgebrek aan de orde en wordt er uitgelegd hoe dit *event-driven* simuleren in zijn werk gaat. De tweede paragraaf geeft een oplossing om de tijd waarin de berekeningen gedaan worden te reduceren met behulp van parallelle simulatie.

2.1 Hoe werkt discrete event-driven sequentiële simulatie?

Bij simuleren is het niet praktisch om bij elke tijdseenheid te controleren of er veranderingen hebben plaatsgevonden. Het is handiger om het om te draaien en te kijken wanneer er een belangrijke gebeurtenis plaatsvindt en hieraan een tijdspunt toe te wijzen. Dit is de sleutelgedachte achter discrete *event-driven* sequentiële simulatie.
Het is mogelijk alles in het dagelijkse leven op deze manier te simuleren, zoals bedrijfsprocessen van productiebedrijven, het vliegverkeer op Schiphol of het treinverkeer van de NS, door ProRail. Om beter inzicht te krijgen hoe discrete *event-driven* sequentiële simulatie praktisch kan worden toegepast wordt er gekeken hoe het bij Roto Smeet Deventer in de praktijk gaat.
In figuur 2.1 is een schematische weergave van het bedrijfsproces van Roto Smeets weergegeven. Aan de hand van deze schematische weergave wordt uitgelegd hoe discrete *event-driven* sequentiële simulatie in zijn werk gaat. De schematische weergave is enigszins vereenvoudigd en de afstand tussen de *events* is iets vertekend als er gekeken wordt naar de tijdspunten. In de werkelijkheid draaien er zes persen te gelijk, maar om overzicht te bewaren is in figuur 2.1 slecht één pers weergegeven.



Figuur 2.1 Een aantal *events* van gebeurtenissen in discrete *event* simulatie

De simulatie start op één januari 2005 middennacht met het ombouwen van pers 1. Dit is een belangrijke *event* binnen het bedrijfsproces en wordt daarom opgenomen in de *event-list*. De *event-list* houdt bij wanneer een *event* start en wanneer volgens verwachting deze weer eindigt. Elk blok in figuur 2.1 is een *event* dat wordt opgenomen in de lijst van belangrijke gebeurtenissen. Als er een *event* plaatsvindt worden alle variabelen binnen het bedrijfsproces aangepast en kan het volgende *event* plaatsvinden. Met de variabelen kan gedacht worden aan de breedte van het papier, het aantal te drukken exemplaren of het aantal te drukken pagina's. Het bedrijfsproces in dit voorbeeld is een vereenvoudigde weergave van de werkelijkheid. Het is verstandig om de essentiële *events* van een bedrijfsproces vast te leggen en niet te gedetailleerd het bedrijfsproces proberen na te bootsen. Dit is ten eerste om het simulatieproces iets te versnellen. Bij veel *events* moet er namelijk veel opgeslagen worden in het geheugen en veel berekeningen gedaan worden. Ten tweede is het om de simulatieomgeving af te bakenen om op deze manier het overzicht te bewaren en de belangrijke informatie naar voren te krijgen. In het voorbeeld van hierboven is het bijvoorbeeld niet interessant te weten wanneer een werknemer een kopje koffie drinkt, wel is het van belang om te weten wanneer er een storing optreedt zodat hier effectief op in gespeeld kan worden.

De *events* zoals in figuur 2.1 weergegeven kunnen uiteraard uitgebreid worden met andere gebeurtenissen die moeten plaatsvinden bij het drukken en nabewerken. Het is mogelijk andere bedrijfsprocessen erbij te betrekken. Er moet echter voor gewaakt worden dat de simulatie niet te gedetailleerd gaat worden. De kans bestaat dan immers dat de rekentijd snel oploopt en is het wachten op resultaat van de simulatie niet rendabel, omdat de simulatietijd bijvoorbeeld langzamer loopt dan de gesimuleerde tijd. Het is daarom verstandig een goede afbakening van het model op te stellen. Indien binnen een bedrijfsproces wel alle aspecten essentieel zijn voor een voorspelling van de werkelijkheid, maar de rekentijd duurt veel te lang, dan zou een oplossing hiervoor parallelle simulatie kunnen zijn. In de volgende paragraaf wordt verder ingegaan op deze vorm van simuleren.

2.2. Hoe werkt discrete event-driven parallelle simulatie?

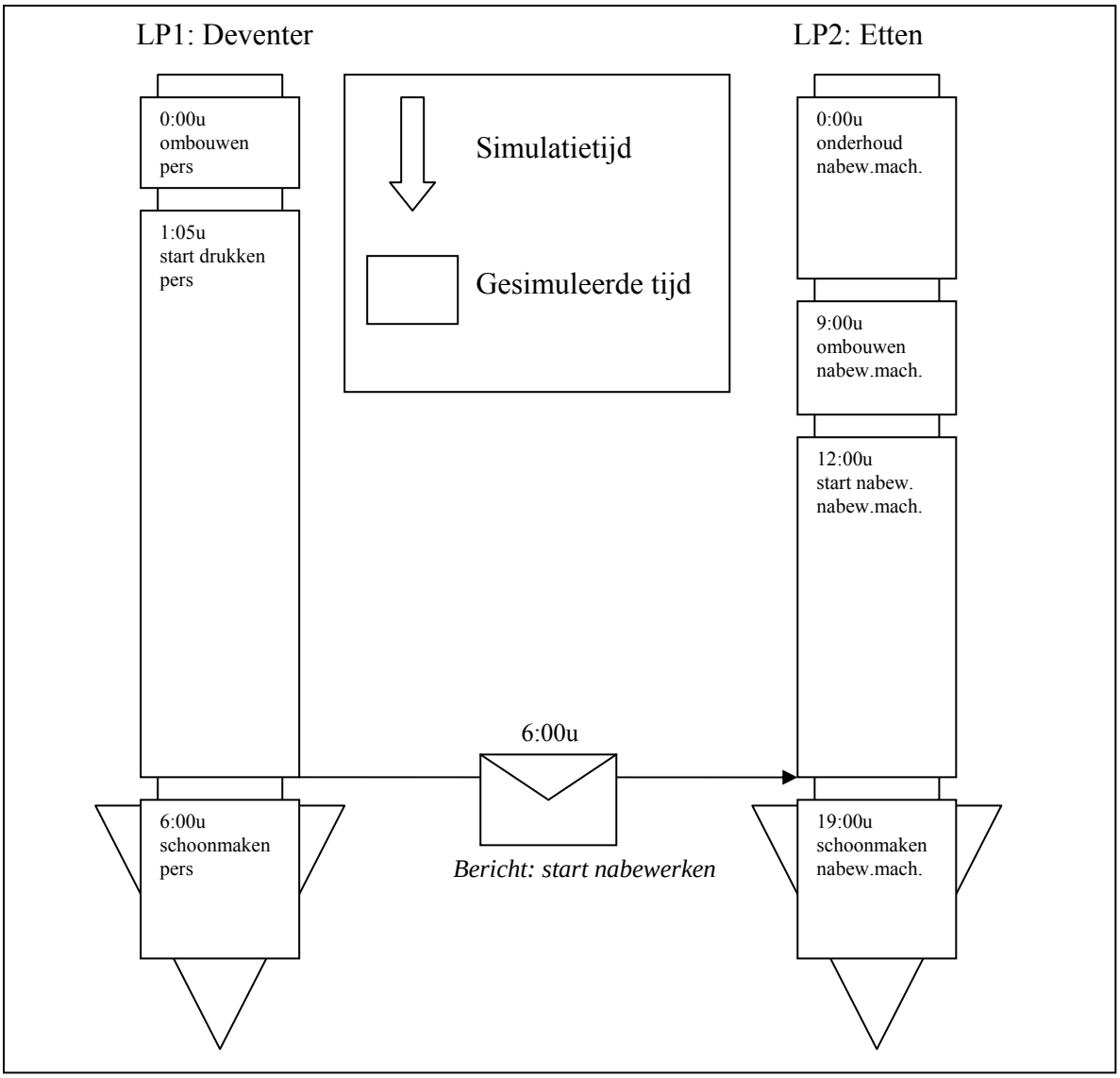
Anders dan bij discrete *event-driven* sequentiële simulatie wordt er bij discrete *event-driven* parallelle simulatie op meerdere computers tegelijk gesimuleerd met als doel de simulatietijd te reduceren.

Parallelle simulatie kan gezien worden als meerdere sequentiële simulaties, waarbij iedere sequentiële simulatie op een aparte computer draait. De aparte computers hebben ieder zogehete

logische processen die tijdsgebonden berichten met elkaar uitwisselen. Elke berekening die gedaan wordt door elke aparte computer, door middel van de logische processen, zorgt voor een reeks *events* waardoor de staat van de simulatieomgeving verandert. De verandering kan plaats vinden binnen de individuele computer zelf, maar ook op een andere computer. Op het moment dat de verandering op een andere computer plaatsvindt, wordt een bericht gestuurd naar die computer en wordt deze op de hoogte gebracht door de andere computer.

Deze constructie lijkt eenvoudig, maar het tegendeel is waar. Bij discrete *event-driven* sequentiële simulatie wordt een gecentraliseerde *eventlist* bij gehouden waardoor alle *events* in de juiste tijdsvolgorde verlopen. Bij discrete *event-driven* parallelle simulatie bestaat zowel een gecentraliseerde *eventlist* als een globale *eventlist*, waardoor het kan zijn dat een bericht van de ene naar de andere computer te laat aan komt, doordat de ene computer verder in de gesimuleerde tijd is dan de andere. Het probleem van het in de juiste volgorde laten verlopen van *events* wordt het synchronisatieprobleem genoemd. In het volgende hoofdstuk wordt hier dieper op in gegaan. Er wordt nu eerst een voorbeeld gegeven waarbij parallelle simulatie een rol kan spelen. Zoals in de vorige paragraaf wordt Roto Smeets aangehouden in het voorbeeld. Roto Smeets heeft twee drukkerijen in Nederland er in Deventer en in Etten. Sommige orders hebben specifieke bewerkingen nog die alleen gedaan kunnen worden bij één van de twee drukkerijen.

In figuur 2.2 is dit schematisch weergegeven. Het ene logische proces, LP1, representeert een pers in Deventer en het andere logische proces, LP2, een nabewerkingsmachine in Etten. Zodra er een order vanuit Deventer naar Etten moet worden verstuurd, wordt er een *event* verzonden van de ene naar de andere computer. De computer waarop Etten wordt gesimuleerd ontvangt het bericht en weet dat er een order vanuit Deventer ingeroosterd moet worden en wil deze er op het juiste moment tussen zetten. Het grootste probleem dat kan ontstaan is dat het bericht van een *event* uit Deventer te laat aankomt zoals in figuur 2.2 te zien is. Het *event* had al plaats moeten vinden, maar de simulatie is inmiddels in Etten in een verder gevorderd stadium. Kortom hoe is het mogelijk om de *events* van meerdere computers synchroon te laten verlopen. Dit probleem wordt het synchronisatieprobleem genoemd. In het volgende hoofdstuk wordt hier verder op ingegaan



Figuur 2.2 Twee logische processen LP1 en LP2

3 Het synchronisatieprobleem

In dit hoofdstuk wordt het probleem behandeld dat ontstaat bij parallel simuleren. De eerste paragraaf beschrijft het probleem en geeft een aantal voorwaarden waaraan voldaan moet worden bij het synchronisatieprobleem. Paragrafen 3.2 en 3.3 geven elk een oplossing voor dit probleem.

3.1 Wat is het synchronisatieprobleem?

Een discrete *event-driven* parallelle simulatie wordt gekenmerkt door het synchronisatieprobleem. Het doel van het synchronisatiealgoritme is de simulatie dusdanig te laten verlopen dat het net lijkt of de simulatie op een *stand-alone* computer draait. Kortom, dat alle *events* in de juiste tijdsvolgorde verlopen zonder te merken dat er wellicht tien computers bezig zijn met de berekeningen.
Het synchronisatieprobleem is niet een probleem dat uitsluitend bij analytische simulatie een rol kan spelen. Bij de virtuele omgevingssimulatie speelt dit probleem eveneens. Een goed voorbeeld hierbij is een virtuele militaire omgeving waarin meerdere personen een rol hebben als soldaat. Iedere soldaat heeft een bepaalde positie in de virtuele omgeving van waaruit de simulatie van start gaat. Als de simulatie van start gaat voor de soldaat mogelijk om verschillende richtingen op te gaan. Er kan een moment ontstaan dat twee of meer individuen zich op exact dezelfde positie willen manoeuvreren. De verschillende computers moeten daarom onderling berichten uitwisselen zodat zij van elkaar op de hoogte zijn waar de individuen zich bevinden en de soldaten niet tegelijk op exact dezelfde positie terecht kunnen komen. De simulatie zou dan een botsing moeten simuleren in plaats van twee personen die dwars door elkaar heen lopen.
Een gouden regel om het synchronisatieproblemen te ondervangen bij analytische simulatie is als volgt:

> **Local Causality Constraint**
> Een discreet *event-driven* simulatie, bestaande uit meerdere logische processen die een interactie hebben middels berichten uitwisseling, houden zich aan het *local causality constraint* als ieder LP alle *events*, die of in het eigen LP of door een ander LP gegenereerd zijn, in oplopende volgorde uitvoert.
> [R.M. Fujimoto, 2000]

Als alle logische processen zich aan deze *constraint* houden dan kan er geen synchronisatieprobleem optreden. De vraag blijft echter hoe dit in zijn werk gaat en welke problemen zich daarbij kunnen voordoen. In de volgende twee paragrafen komen twee verschillende methoden aan de orde: de

conservatieve en de optimistische methode. Deze methoden verschillen volledig van elkaar. De conservatieve methode houdt rekening met de *causality constraint* en de optimistische methode negeert deze volledig en hanteert een *roll back* mechanisme.

3.2 De conservatieve methode

De *events* zijn bij elk logisch proces gesorteerd in oplopende tijdsvolgorde. Elk logisch proces verwerkt het *event* met de kleinste tijdseenheid als eerst. Als een *eventlist* leeg raakt dan moet het logische proces wachten tot een nieuw *event* binnenkomt, omdat er een *event* binnen zou kunnen komen met een tijdseenheid minimaal zo klein als het laatst verwerkte *event*. Het logische proces kan in andere *eventlists* wel *events* verwerken met hetzelfde tijdstip als het laatst verwerkte *event* uit de andere *eventlist*, maar niet *events* die een later tijdstip hebben. Als dit gebeurt bij alle logische processen binnen de simulatie dan stopt de simulatie, omdat deze eventueel in strijd zou kunnen komen met het *causality contraint.* Het systeem is dan in *deadlock*.
Vooral simulaties met weinig *events* die nog verwerkt moeten worden en simulaties waarbij één logisch proces geclusterde *events* heeft zal *deadlock* vaker voor komen. *Lookahead* wordt gebruikt om dit probleem te ondervangen.

Laat T_s de huidige simulatietijd zijn van het logische proces met de vroegste tijdswaarde binnen de gehele simulatie. Als elk logisch proces een *lookahead* van minimaal L heeft, geeft dit de garantie dat elk nieuw *event* verzonden door een ander logisch proces een minimale tijdswaarde van T_s+L heeft. Dit betekent dat elk *event* binnen het interval [T_s, T_s+L] veilig kunnen worden verwerkt. De *lookahead* wordt als volgt gedefinieerd.

> **Lookahead** Als op simulatietijd T_s het eerst volgende *event* dat ingeroosterd kan worden op een LP minimaal simulatietijd T_s+L heeft, dan is L het zogehete *lookahead* tijdstip van het logische proces.

De communicatie tussen de verschillende logische processen om de *lookahead* te bepalen wordt aan de hand van *null-messages* gedaan. Dit mechanisme is noodzakelijk om voor een LP_X een indicatie te geven aan andere LPs over het tijdstip van een bericht dat LP_X in de toekomst zou kunnen sturen. *Null-messages* worden alleen gebruikt ter ondersteuning van het synchronisatieproces en worden niet gezien als *events.* Het werkt als volgt, een *null-message* met een tijdstip T_{null} dat verzonden is van LP_A naar LP_B is een belofte van LP_A dat het geen *event* verstuurd naar LP_B met een tijdstip kleiner dan T_{null}. Op deze manier kan LP_B dan de *events* verwerken kleiner dan T_{null}.

Null Message Methode:

Als er geen bericht kan worden verzonden door een LP om een *event* te starten bij een ander LP, dan stuurt het logische proces een *null message*. Dit is een garantie dat er een geen onderbreking in de simulatie zal plaatsvinden, omdat het ontvangende LP op de hoogte is welke *events* deze veilig kan uitvoeren.

Chandy/Misra/Bryant *null-message* algoritme [K.M. Chandy, 1989, J. Misra, 1986, R.E. Bryant 1977]

De snelheid van het *null message* algoritme is sterk afhankelijk van de *lookahead* mogelijkheid. Met de *lookahead* mogelijkheid wordt bedoeld de mogelijkheid in hoeverre je in de toekomst kan kijken. Bij een kleine *lookahead* moeten er veel *null-messages* verstuurd worden. Dit gaat ten kosten van de snelheid van het simuleren.

3.3 De optimistische methode

De conservatieve methode is niet altijd de beste oplossing voor parallel simuleren. Als de *lookahead* mogelijkheid beperkt is, kortom als er dus niet ver genoeg in de toekomst gekeken kan worden, is het een onhandige methode, omdat er dan veel *null-messages* moeten worden verstuurd voordat er een *event* wordt gestuurd. De berichtenwisseling tussen de LPs zal toenemen en veel tijd innemen en dat is onwenselijk. Parallel simuleren zou immers een oplossing moeten zijn om sneller te kunnen simuleren dan bij simulatie met één enkele processor. Een oplossing voor dit probleem is de optimistische methode.

De optimistische methode ontdekt een fout dat in strijd is met het *causality constraint*, omdat er een bericht in het verleden bij een LP aankomt. Dit kan uiteraard niet en herstelt deze met behulp van het *roll back* mechanisme wat de simulatie terugbrengt in de situatie waarin het bericht niet in het verleden aankomt. Als een *event* wordt ontdekt die te laat zou worden uitgevoerd wordt het systeem teruggedraaid en worden de *events* opnieuw in chronologische volgorde uitgevoerd. Eens in de zoveel keer wordt de staat van de simulatie bewaard, zodat er een *roll back* kan plaats vinden tot aan dat punt.

Precies zoals bij de conservatieve methode moet ook hier een balans gevonden worden, maar dit maal over hoe vaak de staat van de simulatie moet worden bewaard. Te vaak de staat bewaren betekent dat *overhead* veel tijd in beslag gaat nemen. Te weinig daarentegen betekent dat de *roll back* ver terug in de gesimuleerde tijd zal plaats vinden.

Terwijl de conservatieve methode tracht strikt genomen het *causality constaint* te ontlopen, verwerkt de optimistische methode wel de *events* die in verkeerde tijdsvolgorde binnenkomen. Op de tijdstippen dat de conservatieve methode de processors dwingt te wachten, dwingt de optimistische methode de processors continu berekeningen en *state-updates* te maken. De theoretische benutting van de processors bij de optimistische methode is dus maximaal. Het is echter onvermijdbaar dat sommige *events* in de reeks verwerkte *events* uit de *eventlist* voorbarig worden uitgevoerd door het karakter van de optimistische methode, welke het risico neemt *events* uit te voeren die niet zeker zijn. Als er in een reeks tijdens een *update* een processor een *straggler message* (een bericht uit het "verleden") ontvangt, dan moeten alle *events* die ervoor zijn verwerkt worden geannuleerd tot de *roll-back*. De processor verzendt dan annuleringsberichten, ook wel *anti-messages* genoemd, naar de andere processors om een *roll back* te verwezenlijken en alle voorbarige *events* terug te draaien. Met de optimistische methode, waarbij de processors nooit onbenut blijven, worden de processors niet volledig efficiënt benut, dit als gevolg van handelingen zonder betekenis (i.e. verwezenlijken en uitvoeren van de *roll back*). Een deel van de handelingen wordt echter wel gebruikt voor berekeningen die werkelijk iets bijdragen aan de voortgang van de simulatie.
Er zijn uiteraard verschillende optimistische updatevarianten, te denken valt aan modellen waarbij de efficiency en geheugenverdelingen beter zijn dan bij andere optimistische methoden. Het hoofdkenmerk van het updatemechanisme, zoals hierboven beschreven en als eerst geïntroduceerd door Jefferson's Time Warp [D.A. Jefferson, 1985], kan worden beschouwd als generieke kenmerk van de optimistische methode. In het volgende hoofdstuk worden beide methoden met elkaar vergeleken.

4 Conservatief versus optimistisch

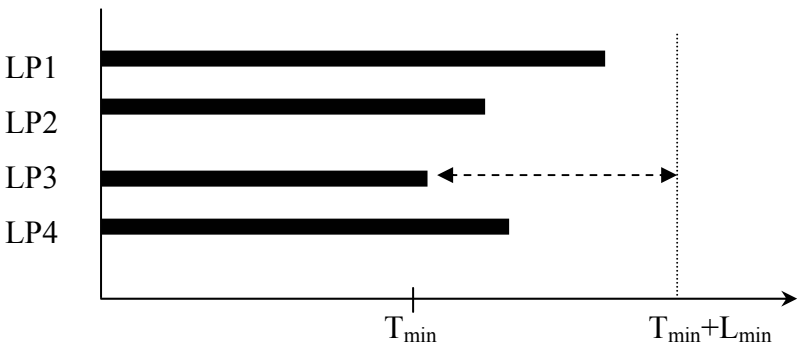
In het vorige hoofdstuk zijn de methoden voor het ondervangen van het synchronisatieprobleem besproken. De twee methoden hebben beide zowel voor- als nadelen. In dit hoofdstuk zal moeten gaan blijken welke van de beide methoden het beste te gebruiken is binnen verschillende scenario's. In eerste instantie worden de beide methoden in een wiskundig model gebracht om een goed beeld te scheppen over de variabelen waar rekening mee moet worden gehouden. Eerst zal de conservatieve methode worden gemodelleerd en tot slot de optimistische. Aan de hand van deze modellen wordt in paragraaf 4.3 tot en met 4.6 bekeken in welke situatie welke methode de voorkeur krijgt.

4.1 Het model: conservatief

Laat het model bestaan uit N logische processen (processors), dan is LP_i het i-de logische proces met de volgende variabelen:

T_i is het tijdstip van een eerst volgend niet verwerkt *event* van LP_i
L_i is de *lookahead* van LP_i
LP_{min} is het LP dat van alle LP_i het verst terug in de gesimuleerde tijd is (in figuur 3.1 LP3)
T_{min} is het tijdstip van LP_{min}
L_{min} is de *lookahead* van LP_{min}

Als dit geldt dan is het veilig alle *events* te verwerken welke binnen het interval $[T_{min}, T_{min} + L_{min}]$ liggen. Nadat deze *events* verwerkt zijn, wordt opnieuw de *lookahead* berekend voor LP_{min}. Ter illustratie is figuur 3.1 ingevoegd. Hierin zijn de zwarte balken de tijd waarin het betreffende LP is gevorderd. Dit is gelijk aan de gesimuleerde tijd. De verticale stippellijn geeft aan tot waar het veilig is om de *events* uit te verwerken.



Figuur 3.1 Het interval van de pijl is veilig om uit te voeren

Na het verwerken van de veilige *events* moet, om uit te rekenen welke *events* de eerst volgende veilige *events* zijn, opnieuw de *lookahead* worden berekend. Het berekenen van de *lookahead* neemt een bepaalde hoeveelheid tijd in beslag, zeg K_t. Het verwerken van de veilige *events* neemt eveneens een bepaalde hoeveelheid tijd in beslag, zeg V_t. De totale tijd dat nodig is om de simulatie af te ronden aan de hand van de conservatieve methode is dan gelijk aan $T_{con}= K + V$ met $K = K_1 + ... + K_T$ en $V = V_1 + ... + V_T$ waarin T het aantal berekeningen is.

De hoeveelheid berekeningen van de *lookaheads* is voor een groot deel bepalend voor de duur van de totale simulatietijd. De tijd van de berekening van een *lookahead* is afhankelijk van het aantal processors dat wordt gebruikt en uiteraard de snelheid van de processors. Het is gewenst een niet te kleine *lookahead* te hebben bij de conservatieve methode. Een te kleine *lookahead* zou zorgen voor te hoge kosten aan *overhead*, omdat er zeer frequent berekeningen gemaakt moeten worden. Een *lookahead* van 0 is uitgesloten, want dit zou betekenen dat de *events* alleen op sequentiële manier kunnen worden verwerkt en dat zou het nut van parallel simuleren wegnemen.

4.2 Het model: optimistisch

Laat het model eveneens bestaan uit N logische processen met LP_i het i-de logische proces. De volgende variabelen spelen dan een rol:

T_s het tijdstip waarop de staat van de simulatie wordt opgeslagen
R_y het tijdstip waarop er een *roll back* plaats moet gaan vinden

Een staat kan op verschillende manieren opgeslagen worden. Zo is het mogelijk om de gehele staat van de simulatie op te slaan. Daarnaast kunnen echter alleen de variabelen die veranderd zijn tijdens het *event* opgeslagen worden. De tweede mogelijkheid valt te overwegen als er weinig variabelen veranderen na een *event.* Als meer dan 20% van de variabelen veranderen tijdens een *event* blijkt het raadzaam om alle variabelen van de staat op te slaan [R.M. Fujimoto, 2000].
De staat van de simulatie hoeft niet na elk *event* te worden opgeslagen, omdat niet na elk *event* de *causality constraint* wordt verbroken. Het zogehete infrequent opslaan van de staat is een goede oplossing voor het sparen van geheugen. Als het fout gaat kan het betekenen dat de *roll back* meer tijd in gaat nemen. Onderstaande formule biedt uitkomst om te bepalen hoe frequent de staat van de simulatie moet worden opgeslagen [Lin Preiss, 1993].

$$\left\lfloor \sqrt{\frac{(\alpha-1)\Delta}{e}} \right\rfloor < m_{opt} < \left\lceil \sqrt{\frac{(2\alpha+1)\Delta}{e}} \right\rceil$$

m_{opt} := het aantal *events* verwerkt tussen elke *state-save*

α := het aantal *events* tussen elke *roll back* als *state-saving* plaatsvindt na elk *event*

Δ:= de kosten van de tijd om de *state-save* uit te voeren

e := is de verwachte tijd hoelang het duurt een staat op te slaan

Het interval geeft een indicatie over na hoeveel *events* de staat moet worden bewaard. Naast het verwerken van de *events* (P_t) zal het opslaan van de staat een bepaalde hoeveelheid tijd innemen, zeg O_t. De *roll back* zal eveneens een bepaalde hoeveelheid tijd innemen, zeg B_t. De totale tijd dat nodig zal zijn om de gehele simulatie af te ronden aan de hand van de optimistische methode is dan gelijk $T_{opt} = P + O + B$ waarin $P = P_1 + ... + P_T$, $O = O_1 + ... + O_T$ en $B = B_1 + ... + B_T$ en T het aantal berekeningen is.

4.3 De vergelijking

In de vorige twee paragrafen zijn beide modellen om het synchronisatieprobleem te ondervangen beschreven. De tijden die het duurt om de simulatie af te ronden bij de conservatieve en optimistische methoden zijn gedefinieerd als respectievelijk T_{con} en T_{opt}. Om te bepalen welke methode optimaal is voor een bepaalde simulatie moet de vergelijking $T_{con} = T_{opt}$ worden opgelost. De vergelijking doet vermoeden dat het eenvoudig te berekenen is. Deze vergelijking is echter lastig op te lossen. Dat wordt bevestigd door de overigens beperkte literatuur die hierover te vinden is. Er is wel een poging gewaagd P_t te berekenen; P_t is de gemiddelde tijd dat een bepaalde hoeveelheid *event* zal innemen tot een *state saving point*. Dit is sterk afhankelijk van de hoeveelheid processors die aan de simulatie mee rekenen. Om het model niet te omslachtig te maken werd aangenomen dat er maar twee processors mee zouden rekenen.
Een volgend probleem wat een belangrijke rol speelt in de berekening zijn de snelheden van beide processors. Zijn deze even snel of is de één sneller dan de ander? In eerste instantie leek de aanname om één sneller te laten zijn dan de ander een plausibele aanname om bij de langzamere processor de kans zeer klein te laten op een *roll back*, echter een kans op een *roll back* bij de langzamere processor is hiermee uiteraard nog steeds niet uitgesloten. Al snel bleek, door de verschillende aannames die gedaan moesten worden om het wiskundige model eenvoudig genoeg te laten om het

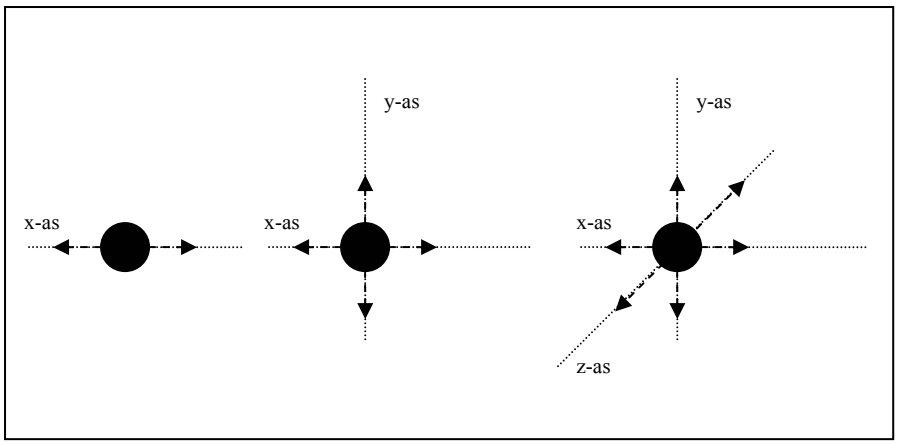
binnen de scoop van het werkstuk te kunnen berekenen, dat hiermee de realiteit uit het oog verloren was en er geen conclusie aan verbonden mochten worden.

4.3.1 Analytische vergelijking

De eerste vergelijking tussen de conservatieve en optimistische methode betreft een analytische benadering in het artikel van Nicol [1991]. In het algemeen kan er uitgegaan worden van twee mogelijke manieren waarop een LP zijn staat verandert. De eerste manier wordt samengevat, zoals reeds vermeld in hoofdstuk 3 door middel van het sturen van een bericht van het ene naar het andere LP. Het bericht doet het ontvangende LP zijn staat veranderen. Dit model wordt het *message-initiating model* genoemd. Een andere mogelijkheid is het *self-initiating model*. Hierin bepaalt een LP zelf wanneer zijn staat verandert. Het LP stuurt eveneens een bericht naar andere LP's, maar hierin vermeldt het LP de verandering van zijn eigen staat. Het bericht verandert niet per definitie de staat van het ontvangende LP. Het komt er op neer dat de *state messages* een verandering versturen en dit melden bij de ontvangende LP's.

Het model

Het model komt voort uit simulatie met behulp van het Ising spin model [B.D. Lubachevsky, 1987]. Het kan gezien worden als een soort rooster met op de roosterpunten positieve danwel negatieve magnetische *spins*. Een *spin* is te vergelijken met een kompas naald. Bepaalde effecten zullen de *spin* doen veranderen en dit kan gezien worden als een discreet *event*. De reeks veranderingen van een bepaald roosterpunt wordt gezien als een Poisson proces; hierin zijn de veranderingen van alle roosterpunten onderling onafhankelijk verdeeld. De *spin value* van een roosterpunt gevolgd door een verandering op tijdstip *t* is afhankelijk van de *spin value* van zijn buurt roosterpunten vlak voor tijdstip *t*. De informatie die nodig is om de *spin value* te berekenen is pas beschikbaar als het roosterpunt een bericht verzend naar al zijn buurt roosterpunten. Een buurt roosterpunt ligt binnen een van te voren bepaalde straal van een roosterpunt. De *spin value* is te vergelijken met de waarden waarin de staat van de simulatie zich bevindt. Verschillende roosterpunten hebben verschillende waarden op een bepaald tijdstip (*state value*). Het rooster kan tot in drie dimensies worden gekozen en laat het afhangen van *K*. *K* is het aantal berichten dat een LP stuurt naar nabij gelegen andere LP's. Als voorbeeld zou K=2 in 1-dimensie, K=4 in 2-dimensies en K=6 in 3-dimensies kunnen voorkomen (zie figuur 4.1.). De roosterpunten hebben vaste afstanden ten opzichte van elkaar, maar deze afstanden zijn niet allemaal even groot tussen alle roosterpunten waardoor de richting per definitie niet zoals in figuur 4.1 hoeft te zijn. Alle punten in de dimensie zijn mogelijk. Denk hierbij aan het heelal met daarin een heleboel sterren.



Figuur 4.1 Drie verschillende waarden voor *K* in drie verschillende dimensies.

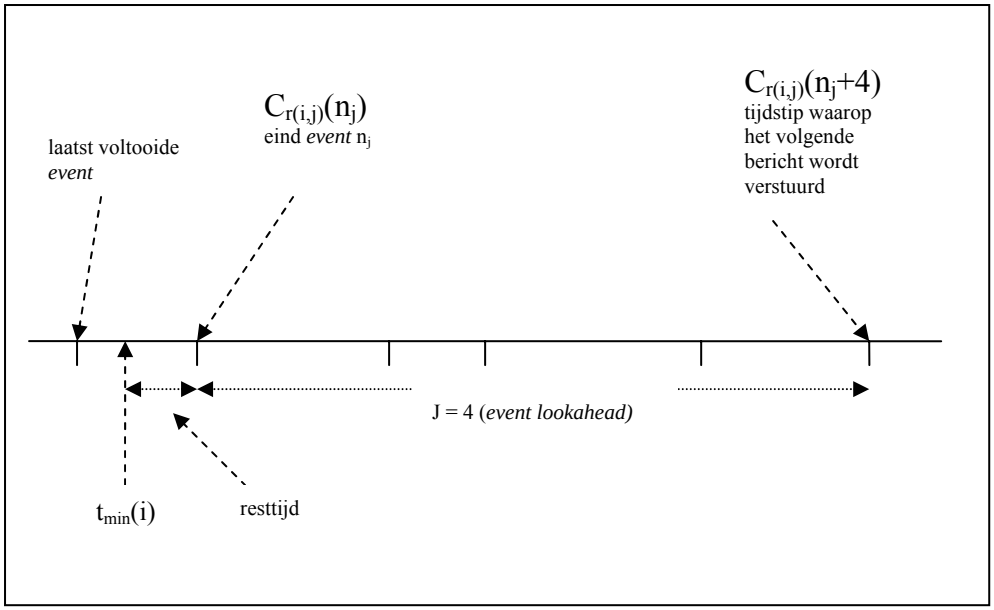
Dit model kent twee kenmerken die van belang zijn. Het eerste kenmerk is de frequentie waarin de roosterpunten veranderingen ondergaan, dit is een random proces, welke onafhankelijk is van ieder ander roosterpunt en iedere *spin value*. Het *self-initiating* model kan gezien worden als een model waarin ieder roosterpunt onafhankelijk voor zichzelf bepaald wanneer zijn *spin value* te veranderen. Het tweede kenmerk is het communicatie principe van het model. Afhankelijk van het roostermodel ontvangen verschillende buurt roosterpunten een bericht van de nieuwe *spin value* van een bepaald roosterpunt. In parallelle simulatie zal een roosterpunt, welke op deze manier een bericht zal ontvangen, nooit vlak voor tijdstip *t* zijn eigen gedrag volledig nauwkeurig kunnen berekenen. Dit betekent dat hoe meer buurt roosterpunten een roosterpunt heeft hoe meer roosterpunten op elkaar wachten door het synchronisatieprobleem. Een interessante vraag die hieruit zou voort kunnen komen is: hoe het kwantificeren van de afhankelijkheid van de prestatie van een roosterpunt met een verschillend aantal buurt roosterpunten?

Laat $C_i(j)$ de waarde zijn van LP_i's tijdstip van de gesimuleerde tijd aan het eind van *event* j. De lengte van de hoeveelheid gesimuleerde tijd waarmee LP_i vooruit gaat door het uitvoeren van *event* j is een random getal X_{ij} met verdeling ζ. Dit betekent voor iedere LP_i en *event* j dat geldt: $C_i(j) = \sum_{k=1}^{J} X_{ik}$. Met de aanname dat alle X_{ij} onafhankelijk zijn, wordt $C_i(j)$ gezien als het tijdstip van de j-de vernieuwing in een vernieuwingsproces met intervernieuwingsverdeling ζ. Er wordt aangenomen dat iedere LP_i een aantal van *K* berichten verzend naar buurt roosterpunten, wat de *message fanout* wordt genoemd. De *K* berichten informeren de buurt roosterpunten over de veranderende *state* (*spin value*) van LP_i. De *message fanout* is onderdeel van het model, maar is onafhankelijk van de gekozen synchronisatie methode.

Aanname: twee vormen van verdelingen voor ζ.
Binnen het model worden twee verschillende verdelingen voor ζ aangenomen. In één vorm is ζ continu cumulatief verdeeld en in de andere vorm geometrisch 1/p met $0 < p \leq 1$.

Het verkrijgen van een bovengrens voor de prestatie kan worden bereikt met beide methoden en door middel van het hierboven beschreven model. Het verkrijgen van een ondergrens kan worden bereikt met behulp van de conservatieve methode. De grens voor de prestatie aan de hand van de optimistische methode is onafhankelijk van het gekozen *roll-back* mechanisme. De bovengrens is een maximale grens die niet overschreden kan worden en de ondergrens is een bepaald minimum waarvoor hetzelfde geldt.

De bovengrens (conservatief)
Het basisprincipe om een nontriviale bovengrens te contrueren is redelijk eenvoudig. De *Global Virtual Time* (GVT) op tijdstip i wordt aangeduid met GVT(i), dit is het tijdstip van het logische proces dat het minst ver gevorderd is in de gesimuleerde tijd. Het is wenselijk de grens waarin de simulatie is gevorderd te bepalen volgens, $\lim_{i \to \infty} GVT(i)/i$. Het bepalen van deze grens voor GVT(i) gaat volgens een functie N(i), met N(i) : = de minimale tijdseenheid van alle berichten verzonden door LP's die op tijdstip i-1 de minimale tijdseenheid ontvangen hadden. Er wordt een beroep gedaan op het asymptotisch redenering om $\lim_{i \to \infty} N(i)/i$ te bepalen en dus de bovengrens te bepalen. De bovengrens is als volgt te bepalen:
Laat $t_{min}(i)$ het vroegste tijdstip zijn van alle verzonden berichten op tijdstip i en laat r(i,j) de index van het j-de LP van alle *K* LP's zijn. Neem een willekeurige $LP_{r(i,j)}$ en neem aan dat $t_{min}(i)$ binnen het tijdsbestek van zijn n_j-de *event* valt. Het volgende bericht dat $LP_{r(i,j)}$ verzendt kan niet een tijdseenheid hebben dat groter is dan $C_{r(i,j)}(n_j+J)$, hierin is J de *event lookahead* (het aantal *events* wat vooruit gekeken kan worden). Het gat van de simulatietijd tussen $t_{min}(i)$ en $C_{r(i,j)}(n_j+J)$ is samengesteld door de som van een aantal *random* variabelen; het verschil $C_{r(i,j)}(n_j+J) - t_{min}(i)$ plus J *lookahead events* zoals aangegeven in figuur 4.2.



Figuur 4.2 *Event lookahead* voor $LP_{r(i,j)}$ met *lookahead value* 4

$t_{min}(i+1)$ kan niet groter zijn dan de vroegste tijdseenheid van alle berichten verzonden op tijdstip i+1 door één van de $LP_{r(i,j)},...,LP_{r(i,K)}$. Noem deze tijdseenheid N(i+1). Dan mag geschreven worden:

$$N(i+1) = t_{\min}(i) + M_{K,J}(i+1)$$

hierin geldt

$$M_{K,J}(i+1) = \min_{1 \le j \le K} \{C_{r(i,j)}(n_j) - t_{\min}(i) + \zeta_J(J)\}$$

hierin is $\zeta_j(J)$ de *lookahead value* van *random* variabelen uit de verdeling ζ.

Er is waar te nemen dat $N(i+1) \geq GVT(i+1)$ en omdat er gezocht wordt $\lim_{i \to \infty} GVT(i)/i$ te begrensen voldoet $\lim_{i \to \infty} N(i)/i$ en hieruit volgt:

$$N(i)/i = \sum_{J=1}^{i} (N(j) - N(j-1))/i$$

$$= \sum_{j=1}^{i} (t_{\min}(j-1) + M_{K,J}(j) - N(j-1))/i$$

$$\leq \sum_{j=1}^{i} (N(j-1) + M_{K,J}(j) - N(j-1))/i$$

$$= \sum_{j=1}^{i} M_{K,J}(j)/i$$

Volgens het artikel mag er worden aangenomen dat de staarten van ζ niet dik zijn en dat dan de verwachting van de reeks $\{M_{J,K}(j)\}$ convergeert tot eindige correlatie tijd en vervolgens geldt: $\psi(K,J)= \lim_{i\to\infty} E[M_{J,K}(j)]$ en een begrenzing van GVT(i). Tot slot laat μ het gemiddelde van ζ zijn dan is de gemiddelde benutting van iedere processor van de parallelle simulatie, met in achtneming van het model, maximaal gelijk aan $\psi(K,J)/\mu$.

De ondergrens (conservatief)

De methode voor het vinden van een ondergrens wordt gezien als conservatief synchronisatie protocol. Doordat deze suboptimaal is vormt dit protocol een ondergrens. Stel: een simulatiemodel met *J-event lookahead* met $J \geq 1$. De strategie van het protocol wordt aan de hand van de volgende drie stappen doorlopen. De stappen worden net zo lang herhaald tot het simulatieproces is beëindigd.

(1) Bepaal voor elk LP de tijd van het volgende bericht dat zal worden verzonden. Als LP_i *event* m als laatst heeft verwerkt dan is de tijd van het volgende bericht dat zal worden verzonden gelijk aan $C_i(m + J + 1)$. Bereken het minimum c_{min} onder alle LP's; c_{min} wordt het plafond genoemd.

(2) Elk LP berekent al zijn *events* met eindtijd kleiner dan c_{min}. Voor elk *event* dat zo is berekend voorspelt en verzendt het LP berichten met *lookahead* van n + J.

(3) Elk LP accepteert de berichten die in de vorige stap zijn verzonden.

De prestatie van het mechanisme zal als volgt worden verkregen. Laat $c_{min}(j)$ het plafond zijn tijdens het j-de tijdstip. Neem nu dit j-de tijdstip: elk LP_i berekent alle *events* met eindtijd kleiner dan die van $c_{min}(j)$ en laat m_i het laatste *event* zijn dat berekend is door LP_i. Het volgende bericht dat zal worden verzonden door LP_i kan worden uitgedrukt in de som van $c_{min}(j)$, de resttijd van het *event* waarin $c_{min}(j)$ ligt plus de *J lookahead*. Kort samengevat komt dit neer op het verschil van tijd tussen LP_i's volgende bericht en $c_{min}(j)$ resttijd van het *event* plus de *J event lookaheads.* Dit was reeds te zien in figuur 4.2. Nu kan $c_{min}(j+1)$ worden uitgedrukt in $c_{min}(j)$ plus het minimum van N random variabelen wat resulteert in: $E\left[c_{\min}(j+1) - c_{\min}(j)\right] \to \psi(N,J) \quad als\ j \to \infty$

Als aan al deze voorwaarden wordt voldaan volgens het model dan is de gemiddelde benutting van de processors minimaal gelijk aan $\psi(N,J)/(J\mu)$. Er kan nu worden geconcludeerd dat de gemiddelde processor benutting bij de conservatieve methode binnen het volgende interval komt te liggen $[\psi(N,J)/(J\mu), \psi(N,J)/\mu]$.

De bovengrens en ondergrens (optimistisch)

Ditzelfde is te bepalen voor de optimistische methode, maar hierbij moet rekening gehouden worden met de tijd die nodig is voor het opslaan van de *state* C_S en de tijd nodig voor de *roll-back* C_R. Met *null lookahead* geldt het volgende:

$$\lim_{i\to\infty} GVT(i)/i \le \begin{cases} \dfrac{\psi(K,0)}{C_S(C_{Rh}+1)} & \text{als } \zeta \text{ niet discreet met } C_{Rh} = C_R/2 \\ \dfrac{\psi(K,0)}{C_S} & \text{als } \zeta \text{ discreet} \end{cases}$$

De gemiddelde benutting van de processors is gelijk aan $\dfrac{\psi(K,0)}{\mu C_S(C_{Rh}+1)}$ als ζ niet discreet verdeeld is, anders is de gemiddelde benutting gelijk aan $\dfrac{\psi(K,0)}{\mu C_S}$. Als ζ discreet verdeeld is, dan is het mogelijk voor een LP om een bericht te ontvangen met het kleinste tijdstip dat vervolgens gelijk is aan GVT en een *roll-back* onnodig maakt. Een *roll-back* is eveneens onnodig als er bij de optimistische methode een *lookahead* is. De gemiddelde benutting is dan gelijk aan $\dfrac{\psi(K,J)}{J\mu C_S}$. De ondergrens van de benutting van de processors ziet er als volgt uit: $\dfrac{\psi(K,J)}{J\mu C_G}$, hierin is C_G het synchronisatie *overhead* percentage.

Met dit stuk van de *paper* van [D.M. Nicol, 1991] is een analytische benadering voor het synchronisatieprobleem van een grootschalige discrete *event* simulatie gemodelleerd. Er zijn non-triviale boven- en ondergrenzen gevonden voor optimale prestatie van bepaalde klasse van simulaties. De resultaten, zoals vermeld op in de tabellen 1 en 2, laten zien dat de conservatieve methode beter werkt dan de optimistische methode volgens het aangenomen model. Het is één van de weinige analytische vergelijkingen tussen beide methoden. Dit model werkt in het voordeel van de conservatieve methode, maar er zijn andere modellen waarin de uitslag andersom kan zijn. Tevens zijn er modellen waarbij de *lookahead* niet voldoet of te niet voorspellen is. Dit is echter moeilijk aan te tonen, omdat het modelleren van parallelle simulatie niet eenvoudig is. In alle stukken die zijn doorgenomen om dit werkstuk te schrijven en te onderbouwen komt telkens naar voren dat een vergelijking tussen beide methoden op velerlei manieren kan uitpakken. Het is daarom lastig een harde uitspraak te doen. Precieze modellen zijn onmogelijk vanwege de complexiteit van parallelle simulatie, maar met aannames in dit model kan er redelijkerwijs een gefundeerde uitspraak gedaan worden over de hoofdvraag van dit werkstuk.

Tabel 1.a Vergelijking Conservatieve en Optimistische methode met ζ als δ + exp{μ_x} verdeling; μ_x=1; N=65536 en C_{Rh}=0; Gemodelleerd met hoge *state saving costs* C_S=10

K \ δ	0.00	0.10	0.50	1.00	5.0	10.00
2	0.049	0.954	3.366	5.024	8.341	9.095
4	0.098	1.907	6.732	10.049	16.683	18.191
8	0.196	3.814	13.464	20.098	33.366	36.381
16	0.392	7.629	26.928	40.196	66.732	72.763
32	0.783	15.258	53.856	80.392	133.464	145.526

(Conservatief/Optimistisch) δ + exp{μ_x} verdeling

Tabel 1.b Vergelijking Conservatieve en Optimistische methode met ζ als δ + exp{μ_x} verdeling; μ_x=1; N=65536 en C_{Rh}=0; Gemodelleerd met lage *state saving costs* C_S=2

K \ δ	0.00	0.10	0.50	1.00	5.0	10.00
2	0.010	0.191	0.673	1.005	1.668	1.819
4	0.020	0.381	1.346	2.010	3.337	3.638
8	0.039	0.763	2.693	4.020	6.673	7.276
16	0.078	1.526	5.386	8.039	13.346	14.553
32	0.157	3.052	10.771	16.078	26.693	29.105

(Conservatief/Optimistisch) δ + exp{μ_x} verdeling

Tabel 2.a Vergelijking Conservatieve en Optimistische methode met ζ als geometrische verdeling; μ_x=1; N=65536 en C_{Rh}=0; Gemodelleerd met hoge *state saving costs* C_S=10

K \ δ	1 / 1	1 / 1.1	1 / 1.5	1 / 2	1 / 6	1 / 11
2	5.000	4.959	4.444	3.750	1.528	0.868
4	5.000	5.000	4.938	4.688	2.589	1.585
8	5.000	5.000	4.999	4.980	3.837	2.667
16	5.000	5.000	5.000	5.000	4.730	3.912
32	5.000	5.000	5.000	5.000	4.985	4.763

(Conservatief/Optimistisch) δ + exp{μ_x} verdeling

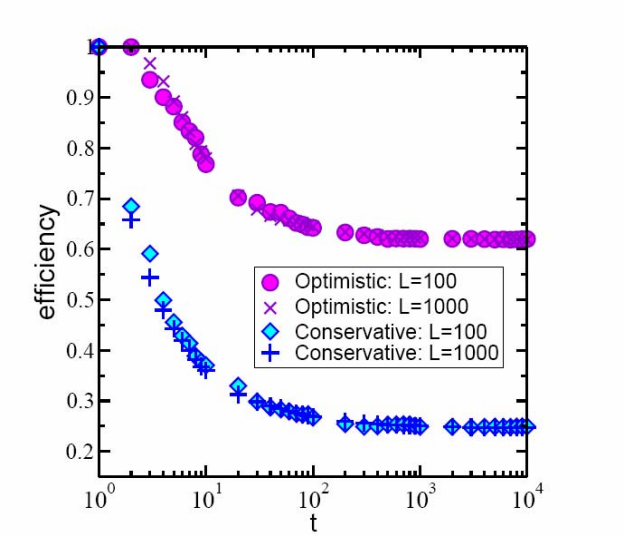
Tabel 2.b Vergelijking Conservatieve en Optimistische methode met ζ geometrische verdeling; μ_x=1; N=65536 en C_{Rh}=0; Gemodelleerd met lage *state saving costs* C_S=2

K \ δ	1 / 1	1 / 1.1	1 / 1.5	1 / 2	1 / 6	1 / 11
2	1.000	.0992	0.889	0.750	0.306	0.174
4	1.000	1.000	0.988	0.938	0.518	0.317
8	1.000	1.000	1.000	0.996	0.767	0.533
16	1.000	1.000	1.000	1.000	0.946	0.782
32	1.000	1.000	1.000	1.000	0.997	0.953

(Conservatief/Optimistisch) δ + exp{μ_x} verdeling

4.3.2 Proefondervindelijke vergelijking

De tweede vergelijking tussen de conservatieve en optimistische methode vindt plaats door middel van simulatie in het onderzoek van Kolakowska en Novotny [2005]. In dit onderzoek wordt aangenomen dat elke *update* cyclus van iedere processor *l* met *l* = (1,...,L) bestaat uit *N events*, en dat de integer *t* de tijdsindex van de *update* cyclus representeert. Het grote verschil tussen de optimistische en conservatieve methode is dat op het moment dat de conservatieve methode de processor laat stoppen met verwerken van *events* de optimistische methode een *random* gok neemt en door gaat met het verwerken van *events*. In de meest eenvoudige vorm van het totale onbevooroordeelde gokken in elke cyclus, is het aantal correct gekozen gokken (de keer dat een *event* niet in het verleden had moeten plaatsvinden) uniform verdeeld. De cumulatieve lokale virtuele tijd (LVT: de tijd van een processor), waarin ***alle*** verwerkte *events* worden meegenomen, correspondeert met de optimistische virtuele tijdshorizon (VTH). De cumulatieve LVT, waarin ***alleen de correct*** verwekte *events* worden meegenomen, correspondeert met de daadwerkelijke voortgangs-VTH. Het verschil tussen de optimistische VTH en de voortgangs-VTH representeert de cumulatieve tijd waarin foutieve *events* zijn verwerkt plus de tijd van het *roll back* mechanisme. De totale efficiency van de optimistische methode wordt gedefinieerd als de ratio tussen de totale *voortgangstijd (ook wel de voortgangs-VTH)* en de *totale berekeningstijd* (*ook wel de optimistische VTH*). Analogisch aan de totale efficiency van de optimistische methode wordt de totale efficiency van de conservatieve methode als volgt gedefinieerd: de ratio tussen de *totale berekeningstijd* en de tijd nodig voor *berekeningen van verwerkte events + de tijd dat een processor niets doet*. Deze *efficiencies* zijn verwerkt in figuur 4.1 met de *worst case* scenario met een minimale *load* per processor en *N* = 1.



Figuur 4.3 Efficiency prestatie tussen conservatieve en optimistische methode.

De berekeningen die gedaan zijn, maar door de *roll back* terug worden gedraaid, blijven in het geheugen om eventueel extra rekentijd te doen vermijden. Er wordt getracht zo min mogelijk overbodige berekeningen te maken. Alleen berekeningen die door een *strangler message* opnieuw gedaan moeten worden, zullen worden verwerkt na een *roll back* en de rest van de berekeningen worden uit het geheugen meegenomen. Hierdoor zou de feitelijke efficiency van de optimistische methode iets lager komen te liggen. Helaas is dit niet meegenomen in het onderzoek. De rede hiervan ontbreekt echter, maar gezien het grote verschil van bijna veertig procent kan worden aangenomen dat de simulatie in het onderzoek aantoont dat de optimistische methode de conservatieve methode qua efficiency overtreft. De optimistische methode heeft duidelijk een betere benutting van het parallellisme en krijgt bij een groot aantal processors (minimaal 100, zie figuur 4.3) de voorkeur boven de conservatieve methode. Helaas is niet terug te vinden in het onderzoek wat de resultaten zijn bij een klein aantal processors van bijvoorbeeld tien of twintig. Strikt genomen is dat alleen interessant voor kleinschalige parallelle simulatiestudies, waarbij budget een grote rol speelt. Tegenwoordig is het vrij eenvoudig vele gedistribueerde processors over het Internet met elkaar te laten communiceren en op deze manier een groot aantal processors te laten deelnemen in een parallelle simulatiestudie. Het is echter niet eenvoudig en zelfs af te raden om telkens opnieuw simulatie te moeten doen om te kijken welke van beide methoden beter en sneller is. Het kost namelijk veel tijd om voor beide methoden een groot aantal simulatie runs te doen om dan tot slot te concluderen dat één van twee sneller werkt en vervolgens een groot aantal runs te doen. Daarom is er een aantal vuistregels waardoor de keuze eenvoudiger kan worden gemaakt. Aan de hand van deze vuistregels kan één van beide methoden per definitie al afvallen. De vuistregels komen in de volgende paragraaf aan bod.

4.4 Vuistregels

De analytische en proefondervindelijke vergelijking geven beide een andere methode als optimale oplossing voor het synchronisatieprobleem. Welke van de twee methoden het beste kan worden gebruikt in verschillende scenario's kan aan de hand van een aantal vuistregels worden versterk. Regels over superioriteit kunnen moeilijk worden geformuleerd, omdat prestaties – door een hoge graad aan beïnvloedbare factoren – niet voldoende binnen modellen kunnen worden gebracht McGraw – Hill [1995]. De volgende alinea wijdt zich aan de vuistregels, welke de keuze enkel kunnen versterken voor het kiezen tussen één van beide methoden. Een keuze aan de hand van één van deze vuistregels garandeert niet dat daadwerkelijk één van beide methoden beter is dan de ander.

Vuistregels:

- Als de *lookahead* klein is, dan is de *overhead* bij de conservatieve methode te groot. Er moet te vaak een berekening gedaan worden om te bekijken welke *events* veilig zijn om uit te voeren.
- Als de simulatie heel veel variabelen bevat, die tijdens de simulatie regelmatig veranderen, dan is de *overhead* bij de optimistische methode te groot. Telkens als de staat moet worden bewaard neemt dit veel geheugen in, waardoor de kans bestaat dat de simulatie vastloopt door geheugen gebrek. Mocht de keuze vallen op de optimistische methode, dan is het raadzaam om niet na elk *event* een *state-save* te doen, maar na een bepaald aantal *events*.
- De conservatieve methode heeft de voorkeur als het de implementatie van de simulatie betreft. Dat komt door de vrij eenvoudige *data structures*. De simulatie van de optimistische methode is moeilijk te implementeren en zeker zo lastig om *bug* vrij te krijgen.
- De optimistische methode is gevoeliger voor *speedup* ten opzichte van de conservatieve methode, dat wil zeggen dat de prestatie van de simulatie door toename van het aantal processors beter zal gaan verlopen.

5 Conclusie

Voor diverse beslissingsprocessen in het bedrijfsleven wordt gebruik gemaakt van simulatie om in de toekomst te kunnen kijken. Hierdoor is het van belang dat de simulatietijd, van bijvoorbeeld discrete *event* simulatie, sneller verloopt dan de werkelijke tijd. Het is mogelijk het simulatieproces van discrete *event* simulatie nog meer te versnellen. Dit kan met behulp van parallelle processors: een keten van aaneengeschakelde computers. Een probleem dat hierbij optreedt en kenmerkend is voor parallelle simulatie is het synchronisatieprobleem. Er bestaan twee methoden om dit probleem te ondervangen. De eerste methode is de conservatieve methode, welke rekening houdt met het *causaltiy constraint* en alleen veilige *events* verwerkt. De twee methode is de optimistische methode, welke dit *constraint* niet in acht neemt, maar een *roll back* mechanisme hanteert wanneer een *event* in het verleden is gegenereerd. Een eenduidig antwoord op de vraag welke van beide methoden beter is, is niet expliciet te geven. Dit komt doordat dit sterk afhangt van het simulatiemodel en de technologie (hardware/software) die wordt gebruikt bij het implementeren en het uitvoeren van de simulatie.

Er is een aantal studies gedaan naar de keuze tussen beide methoden. Deze blijven echter beperkt tot slechts twee processors of bevatten een dusdanige hoeveelheid aannames dat de realiteit geheel uit het oog verloren is gegaan. In het model van Nicol [1991] is een analytisch model opgesteld waarbij naar voren komt dat de conservatieve methode beter presteert dan de optimistische methode. Ook dit model heeft echter beperkingen door de aannames die genomen zijn.

Naast deze studie hebben Kolakowska en Novotny [2005] een poging gedaan om proefondervindelijk te onderzoeken of één van beide methoden superieur is ten opzichte van de ander. Zij tonen aan dat bij een groot aantal processors, minimaal honderd, de voorkeur per definitie uitgaat naar de optimistische methode. De efficiency van de benutting van de processors ligt namelijk bij de optimistische methode vele malen hoger dan bij de conservatieve methode waardoor het parallellisme beter werkt.

In het huidige tijdperk en met de huidige technologie is het echter vrij eenvoudig en niet al te duur meer om berekeningen op veel snelle processors te laten uitvoeren. De conservatieve methode zal tegenwoordig de voorkeur krijgen als het een kleinschalige simulatiestudie betreft, waarbij de implementatiesnelheid van het simulatiemodel een belangrijkere rol heeft. Communicatie over grotere afstanden, denk aan *strangles messages,* kunnen immers zorgen voor dataverlies en hierdoor

neemt de betrouwbaarheid van een grootschalige simulatiestudie af. Bij de verschillende scenario's moet, om tot een juiste keuze te komen tussen beide methoden, telkens opnieuw bepaalde afwegingen worden gemaakt. Aan de hand van een aantal vuistregels kan de keuze tussen beide methoden worden vereenvoudigd.

Uit mijn onderzoek blijkt dat op dit moment hooguit indicaties gegeven kunnen worden over de vraag welke van beide methoden beter is in welke situatie. Een goed onderbouwde wiskundig, algemeen geldende vergelijking van de beide methoden zou het antwoord op deze vraag moeten geven. Het vergt echter aanzienlijke inspanning om een dergelijke vergelijking op te stellen. Hierdoor bestaat snel de neiging meerdere aannames te doen. Het is immers niet reëel om geen enkele aanname te doen, omdat het opstellen van een wiskundige vergelijking in deze situatie erg lastig, zo niet onmogelijk is. Aan de andere kant is het echter verstandig om bij het maken van aannames goed de afwegingen te onderbouwen en kritisch te onderzoeken. Het gevaar bestaat namelijk dat al snel de realiteit uit het oog wordt verloren. Daarnaast hangt de superioriteit van de ene methode ten opzichte van de andere ook sterk af van het gekozen simulatiemodel en de technologie (hardware/software) die wordt gebruikt bij het implementeren en het uitvoeren van de simulatie.

Verder onderzoek is noodzakelijk om duidelijkheid te krijgen over de mogelijkheid om een dergelijke wiskundige vergelijking op te lossen en daarmee het antwoord te geven op de vraag of het mogelijk is om in alle situaties een optimale keuze te maken tussen de beide methoden. Indien het opstellen van een wiskundige vergelijking onmogelijk blijkt, kan een oplossing gevonden worden met behulp van simulaties zoals reeds gedeeltelijk in paragraaf 4.3.2 gedaan is. Het nadeel van dit soort simulaties is de implementatietijd die het kost. Beide methoden moeten immers worden geprogrammeerd en veelvuldig worden gedraaid om tot een uitspraak te komen. Wellicht kan een collega BWI student een uitdagende vervolgstudie naar een wiskundige vergelijking tussen de twee methoden doen. De uitdaging zit in het vinden van een wiskundig correcte vergelijking en het oplossen hiervan met zo min mogelijk aannames.

Referentielijst

Bryant, R.E.
Simulation of Packet Communication Architecture Computer Systems (1977)

Chandy, K.M. en Sherman, R.
Asynchronous distributed simulation via a sequence of parallel computation (1989)

Fujimoto, R.M.
Parallel and Distributed Simulation Systems (2000)

Jefferson, D.A.
Virtual Time (1985)

Kolakowska, A. en Novotny, M.A. ,
Desynchronization and Speedup in an Asynchonous Conservatice Parallel Update Protocol (2005)

Lin, Y.B. en Preiss, B.R.
Selecting the checkpoint interval in Time Warp simulation (1993)

Lubachevsky, B.D.
Efficient parallel simulations of asynchronous cellular arrays. (1987)

McGraw-Hill
Parallel and Distributed Simulation of Discrete Event Systems (1995)

Misra, J.
Distributed discrete event simulation (1986)

Nicol, D.M.
Performance bounds on parallel self-initiating discrete-event simulations (1991)

Preiss, B.R.
Performance of Discrete Event Simulation on a Multiprocessor using Optimistic and Conservative Synchronization (1990)