Parallele und verteilte Simulation

Clemens Berchtold

Institut für Technik Intelligenter Systeme (ITIS e.V.)
an der Universität der Bundeswehr München
Fakultät für Informatik

Bericht 2000-06, November 2000

Zusammenfassung

In diesem Bericht werden Aspekte der parallelen und verteilten Simulation betrachtet. Kapitel eins gibt eine Einleitung und einen Überblick über die Möglichkeiten in der parallelen und verteilten Simulation. Im zweiten Kapitel wird im speziellen die parallele und verteilte Simulation zur Leistungssteigerung behandelt. Es werden Möglichkeiten der Parallelisierung, Anwendungsfelder sowie Problemstellungen aufgezeigt. Die Synchronisation stellt das zentrale Problem in der parallelen und verteilten Simulation dar. Deshalb werden die Synchronisationsprotokolle besonders ausführlich erläutert. Fehlertoleranz in der verteilten Simulation ist das Thema des dritten Kapitels und wurde in der Literatur noch kaum untersucht. In diesem Kapitel wird eine strukturierte Beschreibung und eine systematische Betrachtung von Fehlertoleranz in der verteilten Simulation gegeben und es werden Lösungsmöglichkeiten angedacht. Zum Abschluß des Kapitels werden Ideen zur Fehlertoleranz und High Level Architecture (HLA) vorgestellt.

Dieser Bericht entstand im Rahmen einer ITIS-Studie zum Thema *Zukunftsfelder der Modellbildung und Simulation.*

Parallel and Distributed Simulation

Abstract

This report points out different aspects of parallel and distributed simulation. In section one an introduction and overview of parallel and distributed simulation are given. Then, special attention is paid to parallel and distributed simulation for reducing execution time. Here, synchronisation is the main problem, therefore synchronisation protocols are described in detail. Another potential benefit of distributed simulation is fault tolerance. In section three a systematical view of the problem and possible solution approaches are given. Ideas concerning fault tolerance and the High Level Architecture (HLA) are shown at the end of the section.

This report is part of an ITIS scientific study under the title *Future Perspectives in Modeling and Simulation.*

Inhaltsverzeichnis

1 Einleitung

Ein Modell ist immer eine idealisierte, vergröberte Darstellung der realen Welt. Eine Verfeinerung des Modells zieht in den meisten Fällen ein genaueres Ergebnis der Simulation über das Verhalten in der realen Welt nach sich. Somit besteht das Bestreben, immer komplexere, detailliertere Simulationsmodelle zu konstruieren, um der Wirklichkeit möglichst nahe zu kommen. Allerdings stößt man oft bei einem bestimmten Detailliertheitsgrad des Modells auf die Grenzen eines ausführbaren Simulationsprogramms auf herkömmlichen sequentiellen Rechnern.

In Zukunft werden sowohl die Kosten für rechenstarke Parallelcomputer weiterhin sinken als auch leistungsstarke Rechnernetze vermehrt zur Verfügung stehen. Die Verfügbarkeit von Parallelrechnern bzw. Rechnernetzen und der Wunsch nach immer genaueren, komplexeren Simulationsmodellen haben daher Überlegungen und Untersuchungen auf dem Gebiet der verteilten und parallelen Simulation zur Folge.

Es gibt im wesentlichen vier Gründe [11], eine Simulation verteilt oder parallel durchzuführen. Diese sind:

- Geographische Verteilung,
- Nutzen von vorhandenen Ressourcen,
- Leistungssteigerung,
- Fehlertoleranz.

Eine Simulation aus primär den ersten beiden Gründen werden wir in diesem Abschnitt verteilte Simulation nennen, eine Simulation aus Gründen der Leistungssteigerung parallele Simulation (in der Literatur werden diese zwei Begriffe nicht einheitlich verwendet). Die wesentlichen Zielsetzungen von verteilter Simulation und paralleler Simulation sind verschieden, wobei es auch Überlappungen in den Zielen geben kann. In beiden Richtungen der verteilten und parallelen Simulation möchte man sehr aufwendige Simulationen durchrechnen, wobei bei der verteilten Simulation die Koppelung von mehreren Modellen im Vordergrund steht, hingegen bei der parallelen Simulation ein meist einzelnes großes Simulationsmodell so schnell wie möglich durchgeführt werden soll. Ein weiterer Grund für eine Verteilung der Simulation kann das Schaffen von Redundanz durch diese Verteilung sein, wodurch Fehler toleriert werden können. Diesen Aspekt der parallelen und verteilten Simulation werden wir noch ausführlich in Kapitel 3 erläutern. Daher ergeben sich folgende verschiedene Zielsetzungen:

- Simulieren größerer Modelle,
- ein effizientes Ressourcenmanagement,
- Verkürzung der Simulationszeit,
- höhere Zuverlässigkeit.

Demnach unterscheiden sich auch die Simulationsmethoden bei der verteilten Simulation und der parallelen Simulation. In folgender Tabelle [10] sind diese wesentlichen Unterschiede zusammengefaßt:

	Verteilte Simulation	Parallele Simulation
Zeitanforderung	Realzeit	so schnell wie möglich
Typische Anwendungen	z.B.: Militärische Trainings-, simulation, Verkehrssimulation, Interaktivität	z.B.: Simulation über Strömungsverhalten, Analyse
Leistungsmaßstab	Wirklichkeitsnähe Erweiterbarkeit Wiederverwendbarkeit	Speed-up
Simulationsmodell	Föderation von Modellen	(meistens) einzelnes Modell
Verteilung	geographisch verteilt	auf Prozessoren
Netzwerk	LAN oder WAN	Mehrprozessorrechner oder LAN

Es gibt aber auch gemeinsame Problemfelder, wie zum Beispiel die Synchronisation. Für parallele Simulation entwickelte Synchronisationsmethoden können auch hilfreich sein, um Probleme der zeitlichen Korrelation zwischen den einzelnen gekoppelten Modellen der verteilten Simulation zu bewerkstelligen. Auf der anderen Seite können Techniken, die für eine Reduzierung in der Kommunikation bei der verteilten Simulation konzipiert worden sind, auch in der parallelen Simulation angewandt werden. In diesem Bericht werden wir die parallele und verteilte Simulation aus Gründen der Leistungssteigerung und der Fehlertoleranz behandeln.

2 Parallele Simulation zur Leistungssteigerung

2.1 Einleitung

Unter Leistungssteigerung verstehen wir hier eine Steigerung in der Geschwindigkeit der Simulation, das heißt eine Verkürzung der Durchlaufzeit einer Simulation. Ein anderes weiteres Motiv für eine parallele Simulation zur Leistungssteigerung ist auch, durch Verteilung (Parallelisierung) mehr Speicher für größere Modelle zur Verfügung zu haben.

Bei einem sequentiellen Algorithmus sind im wesentlichen die Ausführzeit, der Speicherbedarf und die Programmierbarkeit die Hauptkriterien für die Güte des Algorithmus. Möchte man jetzt für eine Problemstellung einen sequentiellen Algorithmus und einen parallelen Algorithmus (=geeigneter Algorithmus für eine Implementierung auf einem Parallelrechner) vergleichen, benötigt man andere Kriterien für die Einschätzung des parallelen Algorithmus. Zwei gängige Maße hierfür sind der *Speed-up* und die *Effizienz* [15].

Der *Speed-up* mißt das Verhältnis zwischen einem optimalen sequentiellen Algorithmus und einem parallelen Algorithmus für eine gegebene Problemstellung. Der Speed-up ist auch abhängig von der Anzahl der Prozessoren, auf denen der parallele Algorithmus ausgeführt wird. Mathematisch definiert ist der Speed-up für ein Problem P, der durch einen parallelen Algorithmus A_p gegenüber dem optimalen sequentiellen Algorithmus A_s erreicht wird, durch

$$S_n(P) := \frac{T(A_s)}{T_n(A_p)}$$

wobei $T(A_s)$ und $T_n(A_p)$ die Durchlaufzeiten für den sequentiellen Algorithmus bzw. für den parallelen Algorithmus implementiert auf einem Rechner mit n Prozessoren sind. Im Idealfall möchte man parallele Algorithmen konstruieren, mit denen man einen Speed-up von ungefähr n erhält, das heißt, im Idealfall läuft die Simulation auf einem Parallelrechner mit n Prozessoren mit einem geeigneten parallelen Algorithmus n-mal so schnell durch.

Ein anderes Maß für die Leistung eines parallelen Algorithmus ist die *Effizienz*. Sie ist für eine Problemstellung P definiert durch

$$E_n(P) := \frac{T_1(A_p)}{nT_n(A_p)}$$

$T_1(A_p)$ ist die Durchlaufzeit für den parallelen Algorithmus implementiert auf einem Rechner mit einem Prozessor. Die Effizienz gibt also an, wie effektiv die n Prozessoren relativ zum gegebenen parallelen Algorithmus arbeiten. Ein Effizienzwert nahe bei 1 besagt, daß der Algorithmus implementiert auf einem Rechner mit n Prozessoren ungefähr n-mal schneller ist als implementiert auf einem Rechner mit einem Prozessor. Das bedeutet weiter, daß bei einem Effizienzwert nahe bei 1 die Rechenarbeit gut auf die Prozessoren aufgeteilt ist.

Das übergeordnete Ziel bei der parallelen Simulation aus Gründen der Leistungssteigerung ist nun, sowohl einen hohen Speed-up-Wert als auch einen Effizienzwert nahe bei 1 zu erhalten. Durch die Entwicklung geeigneter paralleler Algorithmen und durch den Einsatz von Parallelrechnern (oder die Möglichkeit, den Algorithmus parallel auf mehreren Prozessoren verteilt durchzurechnen) kann man folglich eine Simulation wesentlich schneller ausführen.

Beispiel Anhand folgenden Beispieles [4] möchten wir die Perspektiven der parallelen Simulation zur Leistungssteigerung aufzeigen. In diesem Papier [4] wird ein paralleler numerischer Algorithmus zur Lösung von Randwertproblemen (gewöhnliches Differentialgleichungssystem mit vorgegebenen Randwerten) vorgestellt. Viele wichtige wissenschaftliche Probleme können als System von gewöhnlichen Differentialgleichungen mit vorgegebenen Randwerten formuliert werden wie zum Beispiel die Modellierung des Wiedereintritts der Flugobjekte vom Orbit in die Atmosphäre.

Dieser parallele Algorithmus basiert auf der Methode der *Schießverfahren*, ein weit verbreitetes Verfahren zur Lösung von Randwertaufgaben. Die Grundidee von Schießverfahren besteht darin, die Behandlung von nichtlinearen Randwertaufgaben auf Anfangswertprobleme auf Teilintervallen zurückzuführen, welche leichter zu lösen sind [22]. Berechnungen in diesen Teilintervallen können dann unter bestimmten Voraussetzungen parallel durchgeführt werden.

Im Artikel [4] wurde dieser parallele Algorithmus mittels *PVM (Parallel Virtuell Machine)*[1] realisiert und mit sequentiellen Algorithmen verglichen. Als Beispiel wurden in der Literatur bekannte „kritische“ Probleme gelöst. „Kritisch“ bedeutet hier, daß die numerische Behandlung dieser Gleichungen schwierig ist, zum Beispiel treten numerische Instabilitäten auf.

[1] PVM ist ein von *Oak Ridge National Labs* entwickeltes Softwarepaket, das erlaubt, daß ein Netzwerk von mehreren Rechnern als ein paralleler Rechner erscheint.

Bei den Testsimulationen erreichte man mit dem parallelen Algorithmus einen Speed-up von bis zu 47,3 (mit 128 Prozessoren). Werden nun diese Randwertprobleme als Modellbeschreibung in einem Simulator eingesetzt, kann man mit erheblichen Leistungssteigerungen des Simulators rechnen.

2.2 Möglichkeiten der Parallelisierung

Eine Parallelisierung kann auf verschiedene Arten geschehen, und die Möglichkeit, die auftretenden Prozesse zu parallelisieren, hängt im wesentlichen von der Simulationsmethode ab. Wir unterscheiden zwischen Parallelität:

- von unabhängigen Simulationsläufen,
- von Hilfsfunktionen des Simulators,
- auf Modellebene.

Diese drei Arten treten im Modellbildungsprozeß in jeweils verschiedenen Phasen auf, wie in Abbildung 1 gezeigt ist.

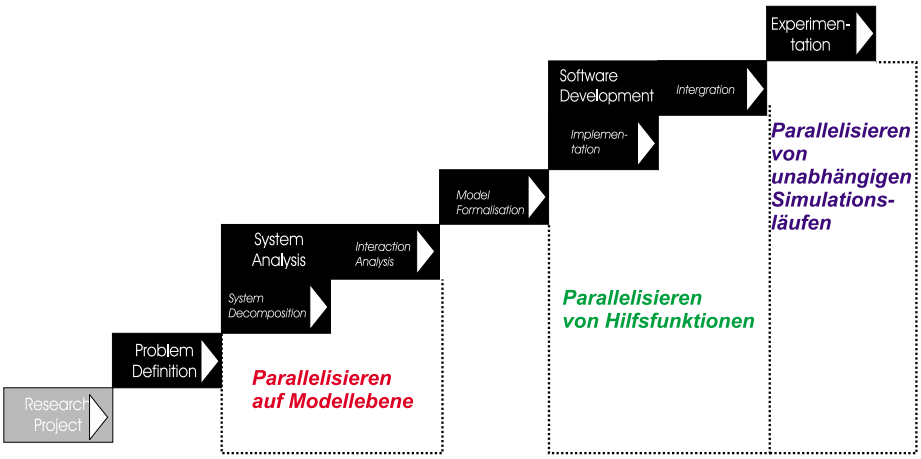

Abbildung 1: Einordnung der drei Parallelisierungsmöglichkeiten in den Modellbildungsprozeß

.

2.2.1 Parallelisieren von unabhängigen Simulationsläufen

Eine einfache Methode, um eine Beschleunigung der Simulation zu erlangen, ist das parallele Ausführen einzelner unabhängiger Simulationsläufe. Oft steht nicht ein (langer) Simulationslauf, sondern viele (kurze) Simulationsdurchgänge des gleichen Simulationsmodells mit jeweils nur verschiedenen Eingangswerten im Vordergrund, um zum Beispiel statistische Aussagen zu gewinnen. Ein Ansatz zur Parallelisierung der Berechnungen ist nun, jeden einzelnen Simulationsdurchgang, der durch die Anfangsparameter bestimmt ist, auf jeweils einem separaten Prozessor parallel ausführen zu lassen.

Sind jedoch die einzelnen Simulationsläufe nicht mehr unabhängig voneinander, zum Beispiel wenn ein Simulationslauf mit dem Ergebnis des vorhergehenden Simulationslaufes berechnet wird, so ist dieser Parallelisierungsansatz nicht mehr möglich. Ein großer Vorteil dieser Parallelisierungsmethode ist, daß ein möglicherweise bestehender sequentieller Simulator oder Simulationsprogramm benutzt werden kann, ohne Änderungen im Simulationscode vorzunehmen. Ein weiterer Vorteil ist, daß abgesehen von der Kommunikation zum Zusammenfassen der Teilergebnisse ein (beinahe) linearer Speed-up (=Speed-up-Wert entspricht der Anzahl der Prozessoren) erreicht wird.

2.2.2 Parallelisieren von Hilfsfunktionen des Simulators

Eine weitere einfache Methode, eine Simulation parallel durchlaufen zu lassen ist, häufig benötigte Hilfsfunktionen auf andere Prozessoren auszulagern. Solche typischen Hilfsfunktionen sind zum Beispiel Zufallszahlengeneratoren, Ein- und Ausgabe von Daten und das Erzeugen von Statistiken.

Auch bei dieser Parallelisierungsmethode besteht der große Vorteil darin, daß bestehende sequentielle Simulationsmodelle nicht konzeptionell verändert werden müssen.

2.2.3 Parallelisieren auf Modellebene

Der vielversprechendste Ansatz, eine Beschleunigung der Simulation durch einen parallelen Ansatz zu erreichen, ist die Parallelisierung auf Modellebene. Hier unterteilt man das Modell in möglichst viele unabhängige Teilmodelle oder Teilbereiche, um jene dann so gut wie möglich unabhängig voneinander parallel zu simulieren.

Diese Unterteilung in Submodelle kann wiederum auf verschiedene Art und Weise geschehen, vor allem unterscheidet sie sich zwischen der kontinuierlichen und der diskreten Modellbildung und Simulation. Gemeinsam ist jedoch eine wesentliche Steigerung in der Geschwindigkeit (Speed-up) und Effizienz.

Je unabhängiger die Submodelle voneinander sind, desto höher ist der potentielle Grad der Parallelität. Hängen jedoch die Teilmodelle stark voneinander ab, sind von vornherein nur geringe Leistungssteigerungen im Sinne einer Verkürzung der Durchlaufzeit der Simulation zu erwarten.

Das Hauptaugenmerk bei diesem Ansatz zur Parallelisierung der Simulation besteht also in der konzeptionellen Modellbildung und nicht so sehr in der technischen Realisierung und Implementierung. Es gibt jedoch kein allgemeines Konzept, um ein Simulationsmodell für eine konkret vorgeschriebene Fragestellung so zu konstruieren, daß es möglichst gut parallel ausführbar ist. Die Problem- oder Fragestellung, für die das Modell entwickelt wird (und somit die Struktur des Modells), bestimmt die Methode, mit der ein idealer paralleler Algorithmus gesucht wird.

In den nächsten zwei Abschnitten (parallele kontinuierliche Simulation und parallele diskrete ereignisorientierte Simulation) werden wir im großen und ganzen diesen Ansatz (Parallelität auf Modellebene) zur Parallelisierung der Simulation betrachten.

2.3 Parallele kontinuierliche Simulation

Bei der kontinuierlichen Simulation ändern sich die Zustandsgrößen des Modells kontinuierlich in der Zeit. Die Simulationen sind folglich ausschließlich zeitgesteuert. Die Beziehung zwischen den Zustandsgrößen und die Änderungen der Zustandsgrößen können größtenteils mit Gleichungssystemen und Differentialgleichungen beschrieben werden. Simulation bedeutet hier das numerische Lösen der mathematischen Problemstellung im Gegensatz zum analytischen Lösen.

Bei der parallelen kontinuierlichen Simulation besteht nun die Herausforderung darin, eine mathematische Problemstellung (z. Bsp.: Gleichungssystem, Differentialgleichungen) mit derartigen Algorithmen zu lösen, die eine parallele Bearbeitung ermöglichen. Auch hier ist natürlich das übergeordnete Ziel, möglichst rasch eine Lösung durch einen parallelen Ansatz aufzufinden.

2.3.1 Anwendungsgebiete der kontinuierlichen Simulation

Der kontinuierlichen Simulation kommt insofern eine große Bedeutung zu, da die Vorgänge in der Natur (Physik, Chemie, Biologie...) stetig ablaufen, das heißt, die Zustandsgrößen ändern sich kontinuierlich in der Zeit. Demnach ist es naheliegend, solche natürlichen Vorgänge mit Modellen für die kontinuierliche Simulation nachzubilden.

Aber auch Fragestellungen, die an und für sich mit diskreter Simulation gelöst werden können, kann man durch formale kontinuierliche Gedankengebilde beantworten. Als Beispiele seien hier logistische Probleme und die Fragestellung der Abnutzung zweier sich bekämpfender Kriegsparteien erwähnt. Letztere Fragestellung kann zum Beispiel mit den *Lanchestergleichungen* (Differentialgleichungssystem) modelliert werden.

Typische Anwendungsgebiete der kontinuierlichen Simulation sind die Meteorologie, die Strömungslehre, die Mechanik und Wärmelehre. Das Verhalten von Flugkörpern, statische Berechnungen von Brücken, von Fahrzeugen oder Einschlagskräfte von Geschossen wären Beispiele für mechanische Fragestellungen. Die Strömungslehre wird unter anderen auch bei Flugbahnberechnungen von Projektilen und bei Flugsimulatoren benötigt.

Eine sehr detaillierte Modellbildung in diesen Anwendungsgebieten zieht eine Menge von Berechnungen nach sich. Man möchte durch genaue Modellierung auch genauere, zuverlässigere Ergebnisse erhalten. Durch parallele Simulation ist es nun möglich, entweder schneller das Ergebnis auf eine Problemstellung zu erhalten oder in angemessener Zeit ein noch detaillierteres, genaueres Modell zu rechnen.

Beispiel Um dies zu untermauern, möchten wir ein Beispiel [23] herausgreifen. Darin wird ein Luftqualitätsmodell behandelt und es werden basierend auf diesem Modell numerische Simulationen (sowohl mit sequentiellen als auch mit parallelen Lösungsmöglichkeiten) durchgeführt.

Luftqualitätsmodelle werden unter anderen entwickelt, um die chemische Zusammensetzung und die Wechselwirkungen in der Atmosphäre zu verstehen. Die Bedeutung dieser Modelle nimmt vor allem im Hinblick auf die immer größer werdenden Umweltverschmutzungen zu. Die Vorgänge (Diffusion, chemische Re-

aktionen, Emission) des Modells werden durch partielle Differentialgleichungen beschrieben. Diese partiellen Differentialgleichungen werden durch numerische Simulation gelöst (an eine analytische Lösung ist aufgrund der Komplexität nicht zu denken).

Untersuchungen auf diesem Gebiet führen schnell zu Berechnungen mit Millionen von unbekannten Größen. In diesem Fall ist der Einsatz von Parallelrechnern und von den geeigneten parallelen Algorithmen sehr hilfreich. In [23] werden Parallelisierungsmöglichkeiten unter anderen für die chemischen Diffusionsprobleme aufgeführt und anschließend der sich daraus resultierende Speed-up-Wert angegeben. Für einen Rechner mit acht Prozessoren wurde zum Beispiel ein Speed-up von 6.6 erreicht, das heißt, daß mit dieser entwickelten Parallelisierungsmethode die Simulation fast sieben mal so schnell durchgerechnet werden konnte. ⋄

Zum Abschluß des Abschnittes Anwendungsgebiete der kontinuierlichen Simulation möchten wir noch auf [25] verweisen. Darin wird die Bedeutung der numerischen Simulation (bei uns kontinuierliche Simulation) und die Bedeutung der parallelen Möglichkeit hervorgehoben. In diesem Artikel wird erwähnt, daß durch leistungsstarke Parallelrechner (in Verbindung mit den dazugehörigen parallelen Algorithmen) wesentliche Durchbrüche in der Simulation erreicht wurden. Anwendungsgebiete, in denen dies zu tragen kommt, sind zum Beispiel

- Numerische Wettervorhersage,
- Crash-Simulation in der Automobilindustrie,
- Numerischer Entwurf technischer Systeme.

Zusammenfassend kann man festhalten, daß Parallelrechner und vor allem die schnellen numerischen parallelen Algorithmen heute kontinuierliche Probleme simulieren können, die vor einigen Jahren noch unlösbar erschienen.

2.3.2 Methoden zur Parallelisierung (Beispiele)

Eine einheitliche Methode, kontinuierliche Simulation zu parallelisieren, gibt es nicht. Aber das Fehlen einer einheitlichen Methode wird kompensiert durch eine Anzahl von Grundtechniken und Paradigmen, mit denen ein großer Bereich von kontinuierlichen Problemen effektiv behandelt werden kann [15]. Dort werden solche Grundtechniken vorgestellt, wir möchten hier kurz die *Divide-and-Conquer-Strategie* erwähnen, die eine typische Vorgehensweise im Parallelisieren darstellt.

Die *Divide-and-Conquer-Strategie* besteht aus drei Schritten, nämlich

1. Partition der Eingangsdaten,
2. Paralleles Lösen der durch die Partition hervorgerufenen Teilprobleme,
3. Zusammenfügen der Teillösungen zu der Lösung des Gesamtproblems.

Eine typische Anwendung dieser Strategie ist das Auffinden eines parallelen Algorithmus für das Lösen von Gleichungssystemen. Parallele Algorithmen für

spezielle Problemstellungen sind in der Literatur schon vielfach behandelt worden. In [16] wird zum Beispiel ein Überblick gegeben, wie man mit parallelen Methoden gewöhnliche Differentialgleichungen und Differentialgleichungssysteme effizient lösen kann. Auch hier gibt es keine allgemein gültige Vorgehensweise. Die Methode hängt einerseits von der mathematischen Struktur der Gleichungen ab, andererseits werden auch spezielle Algorithmen entwickelt, die auf bestimmte Rechnerarchitekturen zugeschnitten sind. Man kann diese unterschiedlichen Methoden zum parallelen Berechnen von gewöhnlichen Differentialgleichungen in drei Kategorien einteilen. Diese sind:

- Parallelisieren bezüglich der Zeit,
- Parallelisieren bezüglich des Gebietes,
- Parallelisieren bezüglich der Methode.

Aber auch für andere mathematische Operationen, mit denen in Simulationen (Hilfs-)Ergebnisse errechnet werden, wurden parallele Methoden gefunden, um damit in der Simulation Zeit zu sparen. Als Beispiele seien erwähnt: Schnelle Fouriertransformation (z. Bsp.: als Hilfsmittel zur Lösung von partiellen Differentialgleichungen), Matrizenoperationen, polynomiale Operationen, Interpolation [15].

2.4 Parallele diskrete Simulation

2.4.1 Einleitung

Im Gegensatz zur kontinuierlichen Simulation erfolgt bei der diskreten Simulation das Voranschreiten der Simulationszeit sprunghaft von einem Zeitpunkt zum nächsten. Das bedeutet, daß sich die Zustandsgrößen des Modells nur zu gewissen diskreten Zeitpunkten ändern. Je nachdem wie der Simulationsablauf bei der diskreten Simulation gesteuert wird, unterscheidet man zwischen

- zeitgesteuerter diskreter Simulation,
- ereignisgesteuerter diskreter Simulation.

Bei der *zeitgesteuerten diskreten Simulation* wird jeweils die Simulationszeit durch ein Zeitinkrement fester oder variabler Größe erhöht, und innerhalb des jeweiligen aktuellen Zeitintervalls werden dann alle Ereignisse (=Änderungen der Zustandsgrößen) ausgeführt. Der große Nachteil der zeitgesteuerten diskreten Simulation sind die *Totzeiten*, das sind jene Zeitintervalle, in denen kein Ereignis eintritt, die aber dennoch Simulationszeit kosten.

Bei der *ereignisgesteuerten diskreten Simulation* wird die Simulationszeit immer bei einem Eintreten eines Ereignisses vorangeschaltet. Somit umgeht man hier das Problem der Totzeiten.

Diese Möglichkeiten der Simulationsmethoden bezüglich Zeit- und Zustandsstruktur ist in Abbildung 2 noch einmal zusammengefaßt. Die meisten Untersuchungen in der Literatur, diskrete Simulation zu parallelisieren, erfolgen in der ereignisorientierten diskreten Simulation. Auch hier ist der vielversprechendste Ansatz das Parallelisieren auf Modellebene. Wir werden uns daher im folgenden

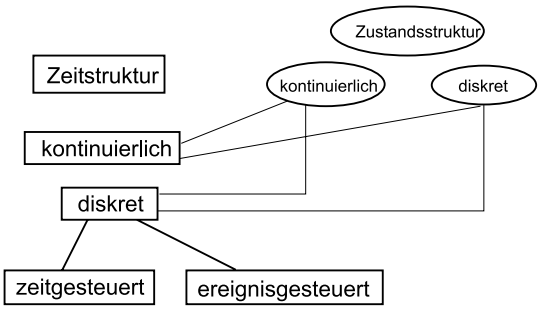

Abbildung 2: Möglichkeiten der Simulationsmethoden bezüglich Zeit- und Zustandsstruktur

.

auf diesen Fall beschränken. Das Modell wird hier also in Teilmodelle oder Teilbereiche untergliedert, die konkurrent ausgeführt werden. Solche eigenständig simulierten Teilmodelle werden *logische Prozesse* (LP) genannt. Jeder logische Prozeß hat eine Modellregion R (zum Beispiel die jeweiligen Zustandsvariablen), einen Simulationsmotor SM, der u.a. zur Ausführung der Ereignisse dient und ein Kommunikationsinterface KI, das den Austausch der Nachrichten zwischen den logischen Prozessen handhabt. Die Nachrichten werden über ein Kommunikationssystem gesendet. Diese Architektur ist in Abbildung 3 dargelegt. Ein

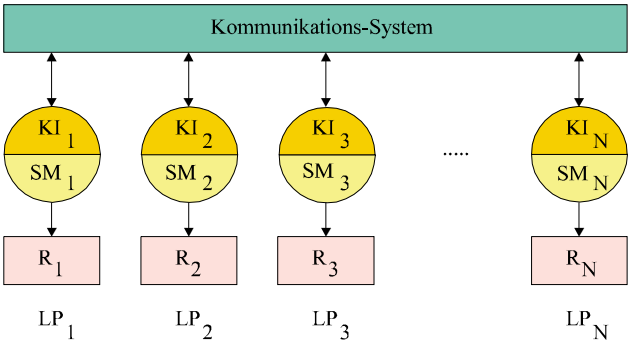

Abbildung 3: Architektur der parallelen und verteilten diskreten ereignisgesteuerten Simulation [8]

.

Beispiel für eine ereignisorientierte diskrete Simulation ist die Modellierung und Simulation eines Straßenverkehrsverlaufs einer Großstadt. Wenn man jede Kreuzung durch einen logischen Prozeß simuliert, kann man das Verkehrsmodell parallel simulieren [18].

Bei der Parallelisierung entstehen aber gegenüber der sequentiellen Simulation neue Probleme, die wir im folgenden Abschnitt erläutern möchten.

2.4.2 Problemstellungen (bei Parallelisierung auf Modellebene)

Bei der parallelen ereignisgesteuerten diskreten Simulation entstehen beim Parallelisierungsansatz auf Modellebene im wesentlichen drei Problemstellungen, nämlich

- Partitionierung,
- Mapping and Scheduling,
- Synchronisation.

Bei der *Partitionierung* wird das Gesamtmodell in möglichst voneinander unabhängige Teilmodelle unterteilt, die jeweils von einem logischen Prozeß ausgeführt werden. Der Partitionierung kommt insofern eine gewichtige Rolle zu, da sie durch das Auffinden von unabhängigen Teilmodellen (und somit die Möglichkeit, diese Teilmodelle parallel zu verarbeiten) die Geschwindigkeit der Simulation mitbestimmt.

Mapping ist die Zuordnung von Prozessoren auf logische Prozessoren. In den meisten Fällen sind jedoch weniger (physische) Prozessoren zur Verfügung als logische Prozesse. Für einen Prozessor, der mehrere logische Prozesse bedient, ist dann eine geeignete Auswahlstrategie (*Scheduling*) erforderlich, das heißt eine Strategie, die angibt, mit welchen Prioritäten die einzelnen logischen Prozesse ausgeführt werden sollen. Außerdem muß es nicht optimal sein (bei mindestens so vielen Prozessoren wie logischen Prozessen), je einen Prozessor auf einen logischen Prozeß anzusetzen. Zum Beispiel kann es sinnvoll sein, stark kooperierende Teilmodelle zusammenzufassen und auf dem gleichen Prozessor ausführen zu lassen.

Man kann bei Mapping- bzw. Scheduling-Verfahren unterscheiden in

- statische Mapping- und Scheduling-Verfahren,
- dynamische Mapping- und Scheduling-Verfahren.

Bei *statischem Mapping* ist und bleibt die Zuordnung von Prozessoren auf die logischen Prozesse fest, während beim *dynamischen Mapping* einzelne logische Prozesse von einem Prozessor zum anderen migrieren können. Beim dynamischen Mapping bestimmt das vergangene Systemverhalten (z. Bsp.: Lasten an Rechnerknoten) die Strategie, welcher Prozessor welchen logischen Prozeß zu welcher Zeit ausführt.

Eine dynamische Mapping-Strategie ist auch erforderlich, wenn während einer Simulation durch das Ausführen von Ereignissen neue logische Prozesse erzeugt werden. In diesem Fall versagen statische Mapping-Strategien. Typische Anwendungsbeispiele, in denen dynamische Mapping-Strategien angewendet werden, sind [18]:

- Gefechtssimulation, in der neue Gefechte hinzukommen oder wieder wegfallen,
- Simulation von Mobilfunknetzen.

Die *Synchronisation* ist das zentrale Problem in der parallelen ereignisorientierten diskreten Simulation. Im nächsten Abschnitt werden Methoden vorgestellt, die dieses Problem zu lösen versuchen.

2.4.3 Synchronisationsmethoden bei paralleler ereignisorientierter diskreter Simulation

Nachdem es nur in den allerwenigsten Fällen möglich ist, völlig voneinander unabhängige logische Prozesse zu erzeugen, muß zwischen den einzelnen logischen Prozessen eine Kommunikation stattfinden. Da aber jeder einzelne logische Prozeß eigenständig in der Zeit voranschreitet (jeder logische Prozeß besitzt seine eigene lokale Simulationszeit), muß bei einer Kommunikation eine Synchronisation der Ereignisausführungen erfolgen, damit die Kausalordnung zwischen den auszuführenden Ereignissen eingehalten wird.

Kausalität bedeutet, daß jeder logische Prozeß seine Ereignisse in korrekter Zeitreihenfolge ausführt. Eine Verletzung der Kausalität tritt ein, wenn ein logischer Prozeß LP_1 ein Ereignis mit lokaler Simulationszeit t_1 ausführt, jedoch eine Nachricht m_{21} von einem logischen Prozeß LP_2 bekommt, die ein neues Ereignis für LP_1 mit einem Zeitstempel $t_2 < t_1$ auslöst. Die Verletzung der Kausalität ist das kritische Problem bei der parallelen ereignisorientierten diskreten Simulation. Je nach Strategie, ob man jetzt Verletzungen der Kausalität a priori verhindert oder ob man sie zunächst zuläßt und erst später behebt (oder eine Kombination davon), unterscheidet man bei den Synchronisationsmethoden zwischen

- konservativen Synchronisationsmethoden,
- optimistischen Synchronisationsmethoden,
- hybriden Synchronisationsmethoden.

Eine Einteilung hinsichtlich dieser Unterscheidung der Synchronisationsmethoden ist in Abbildung 4 veranschaulicht. In den nächsten drei Abschnitten werden wir nun die Eigenschaften der einzelnen Synchronisationsmethoden (oder auch Synchronisationsprotokolle) näher betrachten.

2.4.4 Konservative Protokolle

Bei konservativen Synchronisationsmethoden werden nur sichere Ereignisse ausgeführt. Ein Ereignis gilt als sicher, wenn ein logischer Prozeß entscheiden kann, ob er kein anderes Ereignis mit früheren Zeitstempel erhalten kann. Kausalitätsverletzungen werden dadurch von vornherein vermieden.

Kann ein logischer Prozeß kein sicheres Ereignis ausführen, so muß er blockieren. Blockieren aufgrund unsicherer Ereignisse alle logischen Prozesse, entsteht eine Situation, die *Deadlock* bezeichnet wird, und der Simulationslauf steht. Deadlocks sind das wesentliche Problem bei konservativen Synchronisationsprotokollen. Deadlocks entstehen, wenn es einen Zyklus von blockierenden logischen Prozessen gibt, und jeder logische Prozeß in bezug auf einen anderen logischen Prozeß in diesem Zyklus blockiert. Hinsichtlich der Art, wie dieses Problem behoben wird, unterscheidet man bei konservativen Synchronisationsmethoden zwischen

- Deadlock-vermeidenden Synchronisationsprotokollen,
- Deadlock-erkennenden und -behebenden Synchronisationsprotokollen.

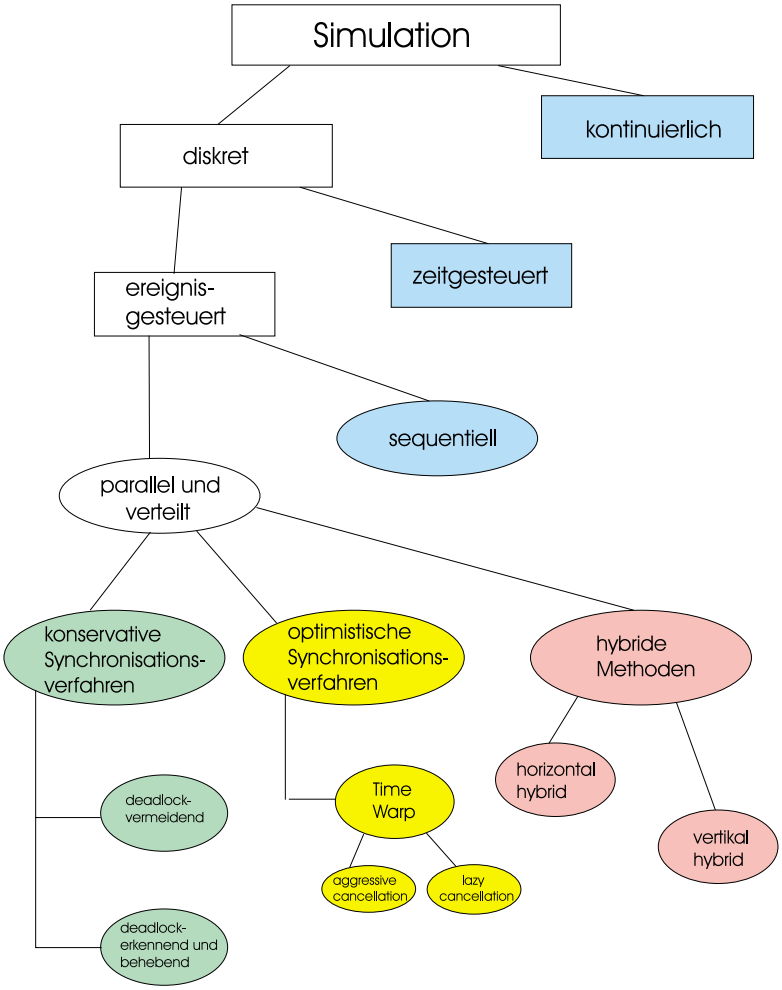

Abbildung 4: Unterteilung der Synchronisationsmethoden zur parallelen diskreten ereignisorientierten Simulation

.

Deadlock-vermeidende Synchronisationsprotokolle sind mit einem Mechanismus versehen, der es a priori nicht zuläßt, daß einzelne logische Prozesse blockieren. Dies wird durch das Versenden von sogenannten *Null-Nachrichten (Null-messages)* erreicht. Eine Null-Nachricht m_{NULL} mit einem Zeitstempel t_{NULL} von einem logischen Prozeß LP_1 an einen logischen Prozeß LP_2 teilt LP_2 mit, daß LP_2 keine weiteren Nachrichten vom logischen Prozeß LP_1 mit einem Zeitstempel kleiner als t_{NULL} empfängt. Eine Null-Nachricht bewirkt keinen Eintritt eines Ereignisses (=keine Änderung der Zustandsvariablen), sie gibt jedoch die Garantie, daß alle Ereignisse mit einem kleineren Zeitstempel als t_{NULL} als sicher angesehen werden können und somit ausgeführt werden können. Dadurch wird auch gewährleistet, daß immer sichere Ereignisse in der Ereignisliste ste-

hen, sodaß Deadlocks verhindert werden. Dieser Mechanismus setzt voraus, daß das Zeitinkrement größer als 0 ist. Ist in der Simulation ein Zeitfortschritt auch von 0 gefordert, ist diese Methode nicht mehr anwendbar.

Deadlock-erkennende und -behebende Synchronisationsprotokolle bestehen aus zwei Phasen in der Simulation, nämlich

1. Phase bis zum Deadlock,
2. Phase des Einschreitens.

Derartige Protokolle benötigen einen Kontrollprozeß, der erkennt, wann eine Deadlock-Situation eingetreten ist (1.Phase) und der dann entscheiden kann, welche Ereignisse global sicher sind (2.Phase). Das Ereignis mit dem global gesehen kleinsten Zeitstempel ist auf alle Fälle sicher.

2.4.5 Optimistische Protokolle

Optimistische Synchronisationsmethoden lassen zunächst mögliche Verletzungen der Kausalität zu (auch unsichere Ereignisse werden ausgeführt) und beheben sie gegebenenfalls später. Dazu ist die Fähigkeit notwendig, in der Zeit zurückzuspringen, um falsche Ereignisausführungen zu annullieren. Diese Fähigkeit wird *Time-Warp oder Rollback* genannt.

Jeder logische Prozeß führt zunächst unabhängig von den anderen logischen Prozessen die Ereignisse in seiner Ereignisliste aus. Im Gegensatz zu konservativen Synchronisationsprotokollen wartet hier also der logische Prozeß nicht auf die Garantie eines sicheren Ereignisses. Bekommt nun ein logischer Prozeß eine Nachricht mit einem kleineren Zeitstempel t^* als seine lokale Simulationszeit, so springt er in der Zeit bis zu diesem Zeitstempel t^* zurück und löscht alle bereits nach t^* ausgeführten Ereignisse aus. Um jedoch auch die Auswirkungen dieser falschen Ereignisse rückgängig zu machen, muß er einerseits Nachrichten an die jeweiligen anderen logischen Prozesse versenden, damit diese wiederum ihre falschen Ereignisse unwirksam machen können. Dies wird durch das Versenden von sogenannten *Antiereignissen* (=Kopie des ursprünglichen falschen Ereignisses) bewerkstelligt. Um also ein erzeugtes Ereignis, welches für einen logischen Prozeß LP_i vorgesehen wurde, später rückgängig zu machen, wird einfach diese Kopie dieses Ereignisses an LP_i gesendet. (Damit ein logischer Prozeß Originalereignis und Antiereignis unterscheiden kann, wird das Originalereignis konzeptionell mit einem „+“, das Antiereignis mit einem „-“ versehen). Nach welcher Strategie Antiereignisse versendet werden, unterscheidet man zwischen

- Aggressive-cancellation-Methoden,
- Lazy-cancellation-Methoden.

Bei *Aggressive-cancellation-Methoden* versendet ein logischer Prozeß, nachdem er in der Zeit zurück gesprungen ist, sofort das Antiereignis zum jeweiligen schon ausgeführten falschen Ereignis. Im Gegensatz dazu verschickt der logische Prozeß bei *Lazy-cancellation-Methoden* zunächst keine Antiereignisse, sondern führt alle Ereignisse nach dem Zeitsprung erneut aus. Erst wenn sich die erneut ausgeführten Ereignisse von den ursprünglichen Ereignissen unterscheiden, versendet der logische Prozeß die entsprechenden Antiereignisse.

Bei der Lazy-cancellation-Methode werden möglicherweise viele Rollbacks (=Zeitsprünge) eingespart, die bei Aggressive-cancellation unnötigerweise ausgelöst wurden. Andererseits wird ein falsches Ereignis bei Lazy-cancellation auch erst später annulliert, sodaß sich falsche Folgeereignisse schneller ausbreiten können. Welches Verfahren im konkreten Anwendungsfall besser ist, hängt vom Modell ab.

Ist nun ein logischer Prozeß gezwungen, in der Zeit zurückzuspringen, muß er in der Lage sein, den entsprechenden früheren Zustand wiederherzustellen. Deshalb muß jeder logische Prozeß jeweils eine Kopie des Zustandes anlegen und abspeichern. Dies führt zum Hauptproblem bei optimistischen Synchronisationsmethoden, nämlich zum hohen Speicherbedarf, der im schlimmsten Fall ein Simulationsprogramm zwingt, den Simulationslauf zu stoppen. Dieses und andere typische Probleme bei optimistischen Methoden sind

- Hoher Speicherbedarf,
- Berechnung der GVT (Global Virtual Time),
- Fehlerfortpflanzung (Rollback-Kaskaden).

Ist kein Speicherplatz mehr vorhanden (und somit eine Blockade der Simulation), so muß wieder neuer freigegeben werden. Es existieren mehrere Methoden dies zu erreichen. Eine ist die *Fossil-collection*, ein Garbage-collection-Protokoll. Bei der Fossil-collection werden alle Zustandskopien mit einem früheren Zeitstempel als der GVT (siehe weiter unten) gelöscht, und es wird somit Speicher gewonnen. Diese Methode reicht jedoch nicht immer aus, um genügend Speicher wieder zu gewinnen. Daher sind unter Umständen auch Speicherengpaßprotokolle notwendig. Speicherengpaßprotokolle identifizieren von den noch benötigten Daten solche, die sich wieder generieren lassen und auf die bei einem Speicherengpaß verzichtet werden kann.

Die *Global Virtual Time (GVT)* ist das Minimum der lokalen Simulationszeiten der einzelnen logischen Prozesse sowie der Zeiten der versendeten aber noch nicht abgearbeiteten Nachrichten. Die GVT wird zum Beispiel benötigt bei der Fossil-collection (siehe oben), bei der Behandlung von nicht mehr zurücksetzbaren Aktionen und für das Erkennen des Simulationsendes. Der genaue Wert der GVT läßt sich jedoch im allgmeinen nicht exakt berechnen, da die verschiedenen logischen Prozesse verteilt ablaufen und sie damit keinen Zugriff auf eine gemeinsame Realzeit haben. Daher stellt die Berechnung der GVT ein weiteres Problem bei optimistischen Synchronisationsprotokollen dar. Näherungswerte für die GVT können jedoch von GVT-Algorithmen berechnet werden.

Die Fehlerfortpflanzung ist das dritte Hauptproblem bei optimistischen Synchronisationsprotokollen. Die Anhäufung von Rollback-Kaskaden (die durch die Fehlerfortpflanzung entstanden sind) kann durch *Moving-time-window Verfahren (MTW)* eingedämmt werden. Hier wird ein Zeitfenster konstruiert, das heißt ein Zeitintervall, in dem die Simulation stattfindet, sodaß ein logischer Prozeß seine lokale Simulationszeit nur bis zum Endpunkt dieses Intervalls voranschreiten lassen darf. Damit kann das Auftreten von Rollback-Kaskaden großteils vermieden werden.

2.4.6 Hybride Methoden

Hybride Synchronisationsmethoden sind eine Kombination aus konservativen und optimistischen Ansätzen. Hier möchte man bei einer parallelen Simulation die Vorteile sowohl der konservativen als auch der optimistischen Synchronisationsprotokolle nützen und zugleich möglichst die jeweiligen Nachteile vermeiden. Man unterteilt die hybriden Methoden in

- horizontal hybrid,
- vertikal hybrid.

Horizontal hybride Methoden verwenden in der gleichen Simulation mehrere verschiedene Synchronisationsverfahren nebeneinander. Hier werden ein Teil der logischen Prozesse mit konservativen Methoden synchronisiert, während der andere Teil nach optimistischen Protokollen arbeitet.

Bei *vertikal hybriden Methoden* werden alle logischen Prozesse mit dem gleichen hybriden Schema synchronisiert. Dieses stellt eine Mischung aus konservativen und optimistischen Verfahren dar.

2.5 Beispiel und Eigenschaften der Synchronisationsmethoden

2.5.1 Beispiel

In diesem Beispiel wird kurz veranschaulicht, wie die einzelnen Synchronisationsprotokolle im Falle des schon oben erwähnten Beispiels der Straßenverkehrssimulation eingesetzt werden können. Wir nehmen an, wir modellieren jede Kreuzung eines Straßennetzes einer (Groß-)Stadt als eigenständigen logischen Prozeß und simulieren demnach jeden Verkehrsfluß an einer Kreuzung parallel. Nachdem die Fahrzeuge, die eine Kreuzung verlassen haben, auch andere Kreuzungen befahren können, muß eine Kommunikation zwischen den logischen Prozessen (Kreuzungen) stattfinden.

Bei konservativen Protokollen werden nur Ereignisse (Verlassen der Kreuzung eines Fahrzeuges, Ankunft eines Fahrzeuges) ausgeführt, wenn sie sicher sind. In diesem Fall wird ständig gewartet, ob nicht ein fremdes Fahrzeug (Fahrzeug von einer anderen Kreuzung) das Verlassen oder die Ankunft eines Fahrzeuges dieser Kreuzung stört.

Bei optimistischen Protokollen werden zunächst alle Ereignisse ausgeführt. Erhält der logische Prozeß, der eine Kreuzung simuliert, eine Nachricht (aus der Vergangenheit), daß ein fremdes Fahrzeug in den Wirkungskreis der Kreuzung kommt, überprüft er, ob das Miteinbeziehen dieses Fahrzeuges eine Änderung der Berechnungen zur Folge hat und setzt gegebenenfalls seine Simulationszeit bis zum entsprechenden Zeitpunkt und Ausgangszustand zurück.

2.5.2 Eigenschaften

Ein direkter Vergleich, ob konservative oder optimistische Synchronisationsprotokolle besser sind, ist nicht möglich, da die Eignung eines Protokolls vom speziellen Modell abhängt. Welche Synchronisationsmethode für ein gegebenes Simulationsmodell effizienter ist, kann von vornherein nicht entschieden werden.

Dennoch lassen sich einige Vor- und Nachteile der einzelnen Synchronisationsprotokolle zusammenfassen.

Optimistische Protokolle können gut eingesetzt werden, wenn nur wenige Rücksetzungen in der Zeit notwendig sind, und wenn der Aufwand für diese Rücksetzungen gering gehalten werden kann. Gegenüber konservativen Methoden nützen optimistische die Parallelität (hervorgerufen durch die Struktur des Modells) besser aus. Wenn sich beispielsweise zwei Ereignisse nur möglicherweise beeinflussen, aber ihre Berechnungen sich nicht, führen optimistische Methoden die Ereignisse aus, während konservative immer und somit auch in diesem Fall auf sichere Ereignisse warten [10].

Der große Nachteil bei den optimistischen Synchronisationsmethoden ist allerdings der hohe Speicherbedarf, der durch das Anlegen der jeweiligen Zustandskopien entsteht. Daher eignen sich optimistische Protokolle auch schlecht, wenn der Zustandsraum des Modells groß ist, und es folglich dadurch zu einem Overhead in den Speicherungen kommt.

Bei konservativen Protokollen kann man mit guten Leistungssteigerungen rechnen, wenn der Modellierer viel (externes) Wissen über das Modell an den Simulator weitergeben kann (zum Beispiel in welchen Zeitspannen Ereignisse als sicher gelten). Zusammenfassend kann man festhalten, daß bei diskreter ereignisgesteuerter Simulation in den meisten Simulationsmodellen durch Parallelisierung auf Modellebene ein Speed-up erreicht werden kann. Zur Synchronisation können verschiedenen Synchronisationsprotokolle eingesetzt werden, wobei man nicht prinzipiell von vornherein entscheiden kann, welche Synchronisationsmethode am effizientesten ist.

3 Fehlertoleranz in der verteilten Simulation

3.1 Einleitung

Im obigen Kapitel haben wir Möglichkeiten aufgezeigt, mit denen die Simulationszeit verkürzt werden kann. Dennoch kann bei komplexen Simulationsmodellen die Simulation eine erhebliche Zeit bis zur Beendigung benötigen, sogar auf einer verteilten Architektur. Ein Aussetzen eines logischen Prozesses, eines Rechnerknotens oder einer Kommunikationsverbindung bewirkt ein Abbruch der gesamten Simulation, und die Simulation muß von Beginn an wieder gestartet werden. Abhängig vom Anwendungsgebiet der Simulation kann dies zu katastrophalen Auswirkungen führen. Zwei Beispiele sollen dies verdeutlichen. Simulationsläufe in der Forschung oder Industrie können mehrere Tage dauern bis Ergebnisse geliefert werden. Der Zeitverlust, der durch einen Fehler entsteht, bewirkt einen Anstieg in den Kosten.

Wird die Simulation im militärischen Bereich zum Beispiel als Hilfsmittel zur Entscheidungsfindung eingesetzt, kann ein Aussetzen der Simulation zu gefährlichen ja sogar zu lebensbedrohlichen Situationen führen.

Diese zwei Beispiele motivieren die Bedeutung von Fehlertoleranzmechanismen im Simulationsmodell. Durch das Schaffen von Fehlertoleranz in der verteilten und parallelen Simulation wird also eine Steigerung in der Zuverlässigkeit erreicht, und zwar

- eine bessere Kosteneffizienz und
- ein Verhindern von gefährlichen sicherheitskritischen Situationen.

Der aktuelle Stand der High Level Architecture (HLA) beeinhaltet kein formales Fehlermodell [5]. Da die HLA ein Standard der verteilten Simulation ist, wird es fruchtbar sein, Ideen über Fehlertoleranzmöglichkeiten mit der HLA zu verfolgen (vgl. auch [13]). In Kapitel 3.5 werden wir ein paar Ansätze anregen. Fehlertoleranz in allgemeinen verteilten Systemen sowie einzelnen Spezialgebieten wie verteilten Datenbanken wurde in der Forschung und Entwicklung schon viel Aufmerksamkeit gewidmet. In der verteilten Simulation hingegen ist dieses Thema noch kaum behandelt worden. In [1, 6] werden Spezialfälle und spezielle Mechanismen zur Datenwiederherstellung in Time Warp Simulationen vorgestellt. Wir hingegen möchten in diesem Kapitel eine systematische und strukturierte Betrachtung von Fehlertoleranz und verteilter Simulation geben und aufzeigen, was für Anstrengungen unternommen werden müssen, um die in Zukunft wahrscheinlich nötige Fehlertoleranz zu erzielen (siehe auch [17]).

3.1.1 Untersuchungsrahmen

Wir konzentrieren uns bei den Betrachtungen auf parallele und verteilte Simulation zur Leistungssteigerung und hier auf den Fall der Parallelisierung auf Modellebene in der diskreten ereignisgesteuerten Simulation, wie sie in Kapitel 2.4 kurz beschrieben wurde. Für die Fehlertoleranzbetrachtungen führen wir eine detailliertere Architektur ein, wie sie in Abbildung 5 dargestellt wird. Hier

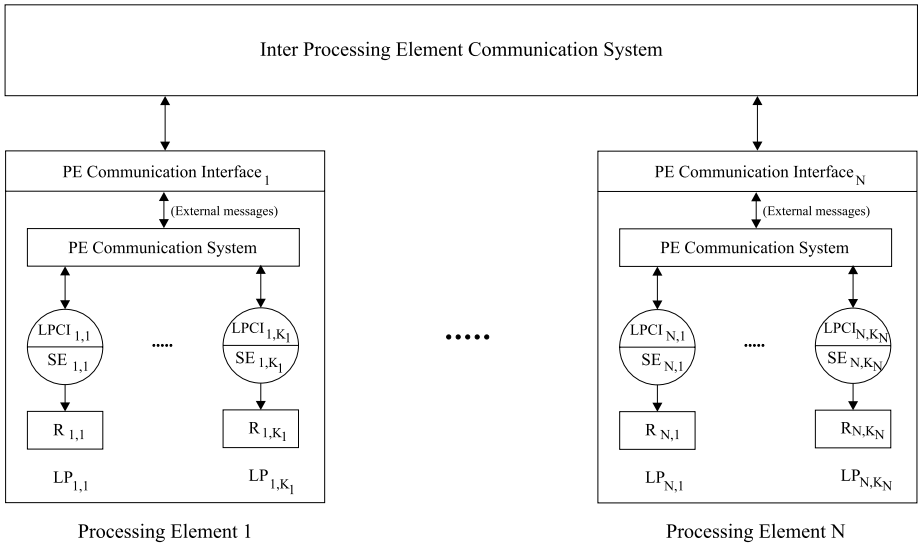

Abbildung 5: Architektur zur Betrachtung von Fehlertoleranz in der verteilten Simulation

.

wird das Simulationsmodell auf N Rechnerknoten (im folgenden abgekürzt mit

PE für processing element) verteilt, jeder Knoten bedient eine gewisse Anzahl von logischen Prozessen, die wiederum jeweils eine Modellregion R, einen Simulationsmotor SE und ein Kommunikationsinterface $LPCI$ besitzen. Das Kommunikationsinterface ist für die Verwaltung der Nachrichten zwischen den LPs auf demselben Knoten verantwortlich, welche über das PE Kommunikationssystem übermittelt werden. Sogenannte externe Nachrichten, das heißt Nachrichten zwischen LPs verschiedener Knoten, werden über das Inter-Prozessor-Kommunikationssystem übermittelt und vom PE Kommunikationsinterface verwaltet. Diese genauere Betrachtung der Architektur ist notwendig, um Fehler von logischen Prozessen, Rechnerknoten oder Kommunikationsverbindungen zu unterscheiden und um entsprechende Mechanismen zu realisieren.

3.1.2 Fehlertypen

Wir beschränken uns bei unseren Betrachtungen hier auf sogenannte *fail-stop Fehler* im Gegensatz zu sogenannten *Byzantinischen Fehlern*. Während bei fail-stop Fehler bis zum Abbruch der Simulation korrekte Ergebnisse geliefert werden, können bei Byzantinischen Fehlern verfälschte Ergebnisse auftreten. Byzantinische Fehler benötigen daher andere Fehlertoleranzmechanismen, wie Replikationen mit sog. distributed voting.

Bei fail-stop Fehlern unterscheiden wir zwischen Fehlern

- eines logischen Prozesses,
- eines Rechnerknotens oder
- einer Kommunikationsverbindung.

Zudem können die Fehler permanent oder temporär sein. Bei temporären Fehlern ist zu entscheiden, ob der Aufwand der Lastverteilung das eventuell kurze Aussetzen der Simulation rechtfertigt. Weiters können Fehler unkontrolliert oder kontrolliert, wie zum Beispiel das Herunterfahren einer Workstation, sein. Um auf die verschiedenen Fehlertypen reagieren zu können, sind unterschiedliche Mechanismen der Fehlererkennung und Datenwiedergewinnung notwendig.

3.1.3 Zwei grundsätzliche Ansätze

Wir sehen zwei grundsätzliche Ansätze, um eine verteilte und parallele Simulation fehlertolerant zu gestalten, wobei sich diese zwei Ansätze nicht ausschließen und sich zum Teil ergänzen können. Diese zwei grundsätzlichen Ansätze sind:

(1) Anwendung von etablierten Fehlertoleranzmethoden im allgemeinen verteilten Rechnen auf den Spezialfall verteilte Simulation und

(2) Adaption von Techniken und Methoden der parallelen und verteilten Simulation, damit Fehlertoleranz unterstützt wird.

Beispiele, wie diese grundsätzlichen Ansätze eingesetzt werden können, werden in den nächsten Kapiteln der Fehlererkennung, Datenwiedergewinnung und Lastumverteilung genauer beschrieben (cf. Abbildung 6).

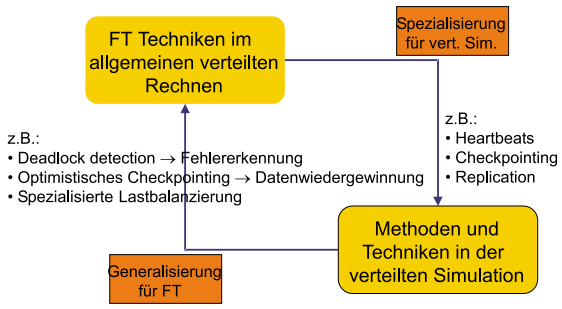

Abbildung 6: Zwei grundsätzliche Ansätze zur Fehlertoleranz in der verteilten Simulation

.

3.1.4 Voraussetzungen für Fehlertoleranz

Für eine fehlertolerante verteilte und parallele Simulation sind drei Anforderungen nötig, für die geeignete Mechanismen gefunden werden müssen. Diese Voraussetzungen für Fehlertoleranz und die Anforderungen an die verteilte Simulation sind:

- Fehlererkennung,
- Datenwiedergewinnung und
- Lastumverteilung.

Bei der Fehlererkennung müssen die Fehler sowie die Art des Fehlers in einer vertretbaren Zeit erkannt werden, damit eine geeignete Reaktion auf den Fehler eingeleitet werden kann. Im Falle eines Fehlers können Simulationsdaten verloren gehen, es ist daher nötig, diese Daten wieder herzustellen. Das ist die Aufgabe der Mechanismen zur Datenwiedergewinnung. Bei Fehlern von Rechnerknoten oder bei Fehlern der Kommunikationsverbindungen müssen die verlorengegangenen oder unerreichbaren LPs auf die verbleibenden Knoten umverteilt werden. Hier kommen Strategien der Lastumverteilung zum Einsatz. Im folgenden gehen wir im Detail auf die einzelnen Anforderungen ein.

3.2 Fehlererkennung

Die erste Voraussetzung an eine fehlertolerante verteilte und parallele Simulation ist die Fehlerkennung. Ein nicht erkannter Fehler kann zu katastrophalen Auswirkungen führen, wie schon in der Einleitung erwähnt wurde. Im verteilten Rechnen ist Fehlertoleranz weit verbreitet. Da die verteilte und parallele Simulation als ein Spezialfall verteilten Rechnens angesehen werden kann, erscheint es auch sinnvoll, einen Blick auf Methoden der Fehlererkennung im verteilten Rechnen zu werfen. Für die verteilte und parallele Simulation hilfreiche etablierte Methoden zur Fehlererkennung werden wir im folgenden Kapitel erläutern.

3.2.1 Methoden zur Fehlererkennung im verteilten Rechnen

Eine verbreitete Methode der Fehlererkennung im verteilten Rechnen ist die Technik mit sogenannten *Heartbeats*. Hier senden die Prozesse Heartbeats zu einem Prozeß-Monitor, der die anderen Prozesse überwacht. Erhält er von einem anderen Prozeß in einer vorgegebenen Zeit keinen Heartbeat, so erklärt er diesen Prozeß für nicht rechenbereit.

In [3] werden zwei Strategien vorgestellt, bei denen Prozesse überwacht werden und bei denen auch Heartbeats zum Einsatz kommen. Diese zwei Strategien sind:

- Zentralisierte Fehlererkennung und
- Verteilte Fehlererkennung.

In der zentralisierten Fehlererkennung überwacht ein ausgewählter Manager zur Fehlererkennung die anderen Prozesse, indem er in periodischen Abständen Heartbeats an alle anderen Prozesse schickt und auf Antwort von diesen wartet. Erhält er keine Antwort, deutet dies auf ein Absturz des Prozesses hin. Fällt der Manager aus, muß ein neuer gewählt werden, um weiterhin Fehlererkennung zu garantieren. Die grundlegende Idee bei der verteilten Fehlererkennung ist das Erstellen einer virtuellen Ringes, in dem eine Marke (Token) von einem Prozeß zum nächsten in diesem virtuellen Ring weitergereicht wird. Ein Fehler wird erkannt, wenn die Marke innerhalb einer festgesetzten Zeit nicht ankommt. Geht die Marke verloren, so wird auch der virtuelle Ring zerstört. Es muß ein Ring Manager gewählt werden, der erkennt, welcher Prozeß ausgefallen ist, folglich einen neuen virtuellen Ring erzeugt und eine neue Marke auf den Weg schickt. Sowohl in der zentralisierten als auch in der verteilten Fehlererkennung ist das primäre Problem die Auswahl des Managers. Auswahlalgorithmen, die dieses Problem beseitigen sind ebenfalls in [3] angegeben.

Eine andere Art der Fehlererkennung kommt in präventiven Methoden zur Anwendung, wie zum Beispiel die Methode der *Verjüngung (rejuvenation)* (zum Beispiel [14]). Hier werden Prozesse die anfällig für Fehler erscheinen als solche identifiziert und auf kontrollierte Weise neu gestartet, um einen möglichen Fehler zu entgehen.

3.2.2 Fehlererkennung in der verteilten und parallelen Simulation

Da ja die verteilte und parallele Simulation als ein Spezialfall verteilten Rechnens angesehen werden kann, werden wir nun versuchen, die oben beschriebenen Methoden zur Fehlererkennung in der verteilten und parallelen Simulation anzuwenden. In der verteilten und parallelen Simulation übernehmen die logischen Prozesse die Aufgabe des Managers zur Fehlererkennung (bei der zentralisierten Variante) oder des Ring Mangers (in der verteilten Variante). Als Heartbeats können solche Nachrichten wie die Null-Nachrichten im konservativen Synchronisationsprotokoll dienen (siehe Kapitel 2.4.3). Wir werden hier kurz erläutern, wie ein Vorgehen zur Fehlererkennung ablaufen soll, damit die verschiedenen Fehler erkannt werden können, das heißt, damit erkannt wird, ob ein logischer Prozeß, ein Rechnerknoten oder eine Kommunikationsverbindung ausgefallen ist.

Fehler von logischen Prozessen Es muß unterschieden werden zwischen Erkennen von Fehlern von LPs auf einem Rechnerknoten oder auf verschiedenen Rechnerknoten. Im ersten Fall sind sowohl die zentralisierte als auch die verteilte Fehlererkennung anwendbar. Sollen Fehler von einem LP auf einem anderen Rechnerknoten erkannt werden, so wird auf jedem Knoten ein LP als Repräsentant gewählt, der dann mit den anderen Repräsentanten kommuniziert. Zum Beispiel kann der Manager zur Fehlererkennung (im zentralisiertem Fall) oder der Ring Manager (im verteilten Fall) jedes Rechnerknotens die Rolle dieses Repräsentanten übernehmen, nachdem diese Manager ja schon für die Erkennung von Fehlern von LPs auf dem Rechnerknoten selbst bestimmt werden müssen. Erkennt nun ein Repräsentant einen LP Fehler auf seinem Rechnerknoten so sendet er eine Nachricht an alle anderen Repräsentanten. Um Ausfälle von Repräsentanten zu erkennen, wird unter den Repräsentanten ein Manager zur Fehlererkennung oder ein Ring Manager gewählt und ein virtueller Ring erstellt. Für das Erkennen von Ausfällen von LPs ist ein sicheres, zuverlässiges Kommunikationssystem Voraussetzung.

Fehler von Rechnerknoten Auch hier müssen LPs als Repräsentanten jedes Rechnerknotens gewählt werden, die dann untereinander in Verbindung stehen. Wird ein geforderter Heartbeat eines solchen Repräsentanten nicht erhalten, so wird der Ausfall jenes Rechnerknotens angenommen, auf dem dieser Repräsentant sitzt. Durch die Wahl eines Repräsentanten auf einem Rechnerknoten wird garantiert, daß zumindest ein LP auf diesem Rechnerknoten rechnet. Folglich ist der Repräsentant ein Hinweis darauf, ob der Rechnerknoten ausgefallen ist oder nicht. Auch hier wird ein sicheres, zuverlässiges Kommunikationssystem angenommen.

Fehler von Kommunikationsverbindungen Im Gegensatz zu Fehlern von LPs und Rechnerknoten muß zum Erkennen von Fehlern der Kommunikationsverbindungen mehr Aufwand unternommen werden. Hier werden wir nur mögliche Mechanismen für die Fehlererkennung im Inter-Prozessor-Kommunikationssystem angeben. Fehler im PE Kommunikationssystem werden vom Betriebssystem jedes PE toleriert. Bei der Fehlererkennung im Inter-Prozessor-Kommunikationssystem stoßen wir auf die Schwierigkeit, daß ein ausbleibender Heartbeat eines Repräsentanten wegen eines PE Fehlers verursacht wird, oder daß der Heartbeat im Kommunikationssystem verloren gegangen ist. Ein Repräsentant $LP_i^{\mathcal{R}}$ vom Rechnerknoten PE_i muß folglich folgende Vorgehensweise verfolgen. Wenn er keinen Heartbeat von dem beobachteten Repräsentanten $LP_k^{\mathcal{R}}$ erhalten hat, sendet er eine Nachricht zu einem dritten Repräsentanten $LP_l^{\mathcal{R}}$ mit der Anfrage ob $LP_l^{\mathcal{R}}$ einen Hearbeat von $LP_k^{\mathcal{R}}$ erhalten kann. Erhält $LP_l^{\mathcal{R}}$ einen Heartbeat von $LP_k^{\mathcal{R}}$, so liegt der Fehler in der Kommunikationsverbindung zwischen PE_i und PE_k. Falls auch $LP_l^{\mathcal{R}}$ keinen Heartbeat von $LP_k^{\mathcal{R}}$ erhält, so muß $LP_i^{\mathcal{R}}$ dieses Vorgehen solange auf alle anderen PEs anwenden, bis alle Kommunikationsverbindung zwischen PE_k und allen anderen PEs getested wurden. Wird von keinem Repräsentanten ein Hearbeat von $LP_k^{\mathcal{R}}$ erhalten, so ist PE_k nicht mehr erreichbar oder PE_k ist ausgefallen.

3.3 Datenwiedergewinnung

Bei Fehlern kann es zu Datenverlusten kommen. Es müssen daher Mechanismen geschaffen werden, die es gestatten, diese für die Simulation wichtigen Daten wieder herzustellen. Auch hier werden wir zunächst Techniken zu Datenwiedergewinnung im allgemeinen verteilten Rechnen betrachten, und dann versuchen, diese aus dem Blickwinkel des Spezialfalls der verteilten und parallelen Simulation zu betrachten. Weiters wird noch aufgezeigt, wie zum Beispiel Techniken zur optimistischen Simulation verwendet werden können, um Fehlertoleranz zu unterstützen.

3.3.1 Datenwiedergewinnung im verteilten Rechnen

Wir betrachten zwei Arten der Datenwiedergewinnung, nämlich Checkpointing und Replikation. Bei Checkpointing Techniken benötigt man einen *stabilen Speicher (stable storage)*. Stabiler Speicher ist ein Terminus technicus, ein abstrakter Begriff, es wird angenommen, daß Daten, die auf dem stabilen Speicher gespeichert sind, Abstürze von Rechnerknoten und Kommunikationsverbindungen überstehen, und daher kann immer auf sie zugegriffen werden. Die Realisierung eines stabilen Speichers bedarf jedoch einigen Aufwand bei der Implementation. Stabiler Speicher kann zum Beispiel ein verläßliches Speichermedium sein, das durch das Betriebssystem gewährleistet ist. Ein stabiler Speicher kann auch aus mehreren *volatilen (=vergänglich) Speichern* anderer Prozesse oder LPs bestehen. Die Daten eines LPs werden in diesem Fall auch auf dem lokalen Speicher eines anderen LPs gespeichert. In den folgenden Überlegungen über Checkpointing wird die Verfügbarkeit von einem stabilen Speicher vorausgesetzt.

Checkpointing Als *stabilen Checkpoint (stable checkpoint)* bezeichnen wir eine Momentaufnahme des Zustandes eines Prozesses, die auf stabilem Speicher geschrieben wird. In diesem Kapitel wird zwischen stabilen Checkpoints (für Fehlertoleranz) und Checkpoints in der Time Warp Simulation unterschieden. Mit $c_{i,x}$ wird der x-te stabile Checkpoint des Prozesses i bezeichnet, x heißt der Checkpoint Index. Mit $I_{i,x}$ wird das stabile Checkpoint Intervall zwischen $c_{i,x-1}$ und $c_{i,x}$ definiert. Ein *lokaler Checkpoint* ist ein stabiler Checkpoint eines speziellen Prozesses. Ein *globaler Checkpoint* besteht aus einer Menge von lokalen Checkpoints, von jedem Prozeß ein lokaler Checkpoint, sodaß das System einen konsistenten Zustand bildet. Ein globaler Checkpoint heißt *konsistent*, wenn keine Nachricht nach einem lokalen Checkpoint ci,x gesendet ist und vor einem anderen lokalen Checkpoint $c_{j,y}$ erhalten wird [27], wobei $c_{i,x}$ und $c_{j,y}$ zum globalen Checkpoint gehören. Nach einem Ausfall eines Prozesses oder Rechnerknotens kann auf diesem globalen Checkpoint aufgesetzt werden. Der aktuellste globale Checkpoint, also der globale Checkpoint, der aus den lokalen Checkpoints besteht, die am weitesten in der Zeit fortgeschritten sind, wird mit *Recovery Line* bezeichnet. In Abbildung 7 ist eine Recovery Line gezeichnet, wobei mit Pfeilen die Nachrichten m_i dargestellt sind. Allerdings bedarf die Konstruktion und das Bestimmen der Recovery Line einigen Rechenaufwand, und kann zum Beispiel mit sogenannten *Rollback Abhängigkeitsgraphen (rollback-dependency graph)* bestimmt werden [27].

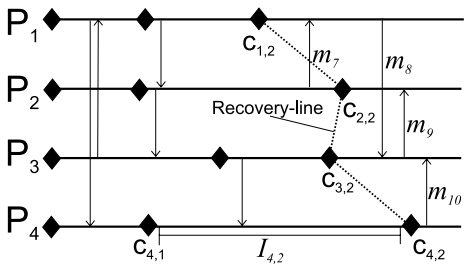

Abbildung 7: Globaler Checkpoint und Recovery Line

.

Es gibt im wesentlichen drei Arten von Techniken mit Checkpoints [7], nämlich

- unkoordiniertes,
- koordiniertes und
- kommunikationsinduziertes Checkpointing.

Beim unkoordinierten Checkpointing speichern die Prozesse ihre lokalen Checkpoints unabhängig voneinander. Das Hauptproblem bei der unkoordinierten Variante ist die Anfälligkeit für den sogenannten *Dominoeffekt*. Im Falle des Dominoeffekts ist es möglich, daß durch das Zurückspringen eines Prozesses i zu einem lokalen Checkpoint $c_{i,x}$ ein weiteres Zurückspringen eines anderen Prozesses im Zustand des lokalen Checkpoints $c_{j,y}$ nötig ist, wenn eine Nachricht des Prozesses i nach dem lokalen Checkpoint $c_{i,x}$ gesendet wurde, aber vor dem $c_{j,y}$ empfangen wurde. In Abbildung 8 kann die Situation eines Dominoeffektes nachvollzogen werden. Tritt ein Dominoeffekt auf, gehen viele Berechnungen durch

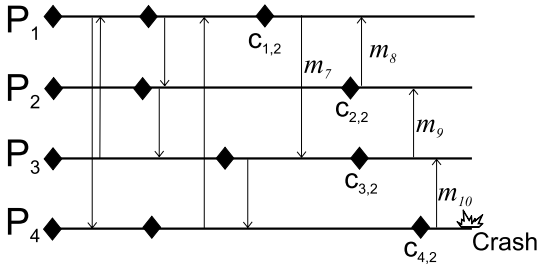

Abbildung 8: Dominoeffekt

.

die Rollback Kaskaden verloren. Möglichkeiten, den Dominoeffekt zu verhindern, sind das koordinierte und das kommunikationsinduzierte Checkpointing. Bei beiden Möglichkeiten schreiben die Prozesse in geeigneter Übereinstimmung

mit den anderen ihre lokalen Checkpoints auf stabilen Speicher, damit eine Recovery Line gefunden werden kann. Beim koordinierten Checkpointing geschieht dies durch Kommunikation zwischen den Prozessen über Nachrichten, während beim kommunikationsinduzierten Checkpointing eine systemweite Schranke das Fortschreiten der Recovery Line garantiert.

Eine Variante von Checkpointing sind Techniken, bei denen zusätzlich zu den Checkpoints Nachrichten gespeichert werden (*message logging*). Solche Nachrichten können zum Beispiel nichtdeterministische Ereignisse sein, die nach dem Zurückspringen in der Zeit neuerlich ausgeführt werden können. Diese Techniken lassen sich unterteilen in pessimistische, optimistische und kausale [2]. Sie können in der verteilten und parallelen Simulation vor allem bei optimistischen Protokollen zum Speichern der Eingangs- und Ausgangslisten der versendeten und empfangenen Nachrichten, die für den Rollback nötig sind, angewendet werden.

Replikation Bei der Replikation werden Kopien von Prozessen angelegt. Das Hauptproblem bei der Replikation ist die Schwierigkeit, die einzelnen Replikate in einem konsistenten Zustand zu halten, das heißt in einem Zustand, in dem sie mit dem Original übereinstimmen.

Es kann unterschieden werden zwischen zwei Techniken, nämlich der

- passiven Technik (*primary-backup*) und
- aktiven Technik.

In der passiven Variante führt ein Hauptreplikat (primary) die Berechnungen aus und verschickt Kopien der Zustände an die anderen Replikate (backup). Fällt der Hauptreplikat aus, übernimmt einer der anderen Replikate die Rolle des Hauptreplikats. In der aktiven Variante arbeiten alle Replikate unabhängig voneinander und führen die gleichen Berechnungen parallel aus.

Eine andere Art einer Replikationstechnik ist die RAID (*Redundant Arrays of Inexpensive Disks*) Architektur, die vor allem für zuverlässige Speichersysteme eingesetzt wird, aber auch für die verteilte und parallele Simulation hilfreich sein kann. Eine genauere Beschreibung von RAID kann man zum Beispiel in [19] finden. Bei RAID kann jedoch nur immer ein Fehler und nicht mehrere Fehler gleichzeitig toleriert werden. In [20] werden koordiniertes Checkpointing und RAID Techniken miteinander verbunden. Zwei RAID Strategien werden angewandt, nämlich Strategien mit Spiegelung (disk mirroring, level 1 RAID) und Paritätsüberlegungen (parity disk, level 5 RAID). Bei level 5 RAID wird ein Paritätscheckpoint erzeugt, der sich aus der bitweisen Oder-Verknüpfung (exclusive-or) jedes lokalen koordinierten Checkpoints berechnen läßt. Fällt ein Prozeß aus, kann dieser aus den lokalen Checkpoints der anderen Prozesse rekonstruiert werden. Folglich kann durch diesen Paritätscheckpoint der Speicheraufwand verringert werden.

3.3.2 Datenwiedergewinnung in der verteilten und parallelen Simulation

Wenn Techniken zur Datenwiedergewinnung aus dem allgemeinen verteilten Rechnen auf den Spezialfall der verteilten und parallelen Simulation angewendet

werden, so muß in Erwägung gezogen werden, welche Simulationsdaten wichtig und notwendig für die Checkpoints und somit für die Fortsetzung der Simulation sind. Die Datenstruktur für die Checkpoints sind bei der konservativen und optimistischen Simulation verschieden. In der konservativen verteilten Simulation besteht ein stabiler Checkpoint eines logischen Prozesses aus den Zustandsvariablen, der Ereignisliste und der virtuellen Zeit des LPs. Beim optimistischen Synchronisationsprotokoll müssen zusätzlich die Eingangs- und Ausgangslisten der versendeten und empfangenen Nachrichten gespeichert werden, um das Zurückspringen in der Zeit zu ermöglichen.

Die optimistische verteilte Simulation scheint sich gut für eine fehlertolerante verteilte Simulation zu eignen. Zwei Eigenschaften können hier vorteilhaft verwendet werden, nämlich zum einen werden bei optimistischen Protokollen Checkpoints zum Zurückspringen in der Zeit gespeichert, zum anderen wird die *Global Virtual Time* (GVT) (siehe Kapitel 2.4.3) berechnet. Beides kann für die Fehlertoleranz genutzt werden und zwar auf folgende Art. Die Checkpoints werden auf stabilen Speicher geschrieben, und die lokalen Checkpoints müssen derart gespeichert werden, damit konsistente globale Checkpoints oder eine Recovery Line gefunden werden können. Verfahren, die optimale Checkpointintervalle für die optimistische Simulation erstellen, wurden schon gefunden (zum Beispiel [9]). Diese Verfahren müssen in Bezug auf Fehlertoleranz nun derart adaptiert werden, damit auch eine Recovery Line gefunden werden kann. Simulationsdaten vor dieser Recovery Line sind für den weiteren Verlauf der Simulation nicht mehr von Bedeutung und können gelöscht werden. In der optimistischen Simulation wird die GVT berechnet, damit die Daten vor der GVT gelöscht werden können, um wiederum mehr Speicherplatz zu gewinnen. Auch die Daten vor der Recovery Line werden für den weiteren Verlauf der Simulation nicht mehr benötigt. Algorithmen zur Berechnung der GVT und Verfahren zur Bestimmung der Recovery Line stellen daher analoge Vorgehensweisen dar. Folglich können Methoden zur Berechnungen und Annäherungen der GVT auch zum Bestimmen der Recovery Line hilfreich sein.

Replikation von logischen Prozessen ist eine andere Idee, um in der verteilten und parallelen Simualtion verlorengegangene Daten wiederzugewinnen. In der passiven Variante führt nur der LP, der als Hauptreplikat dient, die Ereignisse aus und sendet den aktuellen Zustand den anderen Replikaten. Damit ein Ausfall eines Knotens PE toleriert werden kann, auf dem der Hauptreplikat sitzt, müssen die anderen Replikate auf anderen PEs sitzen. Ein Replikat muß dann bestimmt werden, der die Rolle des Hauptreplikaten übernimmt. In der aktiven Variante führen alle Replikate der logischen Prozesse, die vorzugsweise wieder auf verschiedenen PEs arbeiten, die Ausführungen selbständig und parallel aus. Hier wird vorausgesetzt, daß alle Replikate exakt dieselben Ergebnisse liefern. Wenn in einer Simulation Zufallszahlen zur Anwendung kommen, dann muß jeder Replikat die gleichen Startwerte der Zufallszahlengeneratoren setzen.

In der passiven Variante werden viele Nachrichten vom Hauptreplikat an die restlichen Replikate gesendet. Somit ist diese Variante für Synchronisationsprotokolle geeignet, bei denen die Aktivität des Nachrichtensendens nicht hoch ist, wie es in der optimistischen Simulation der Fall ist. Bei der aktiven Variante wird viel Rechenkapazität verbraucht, da ja alle Replikate die Berechnungen durchführen. Im Gegensatz zur optimistischen Simulation ist bei der

konservativen Simulation nicht die Rechenkapazität sondern die Kommunikation mit Nachrichten der Flaschenhals. Folglich ist die aktive Variante für die konservative Simulation besser geeignet, um Fehlertoleranz zu erzielen.

3.4 Lastumverteilung

Bei einem permanenten Ausfall eines Rechnerknotens müssen die LPs, die von diesem Rechnerknoten bedient wurden, auf die anderen verbleibenden PEs umverteilt werden, damit weiter gerechnet werden kann. Bei einem temporären Ausfall ist zu entscheiden, ob die Kosten der Lastmigration die Verzögerung der Simulation aufwiegen. Im Falle von kontrollierten Fehlern (wie das Herunterfahren einer Workstation in einem Workstationcluster) können ja die Simulationsdaten vorher versendet werden. Lastumverteilung ist in diesem Fall daher die einzige Anforderung an die verteilte Simulation, um Fehlertoleranz zu erzielen. Bei unkontrollierten Fehlern sind Fehlererkennung und Datenwiedergewinnung Bedingungen für Lastumverteilung.

Bei den Überlegungen in diesem Kapitel gehen wir davon aus, daß das Simulationsmodell in Teilmodelle, logischen Prozesse, unterteilt ist. Ein nötige Lastumverteilung ist daher ein Mapping auf die verbleibenden PEs. Für die Lastumverteilung können zwei Arten von Lastbalancierung in Betracht gezogen werden,

- statische Lastbalancierung und
- dynamische Lastbalancierung.

3.4.1 Statische Lastbalancierung

Bei der statischen Lastbalancierung wird vor dem Simulationsbeginn die Strategie der Lastumverteilung festgelegt. Um Lastumverteilung für Fehlertoleranz mit statischen Strategien zu erhalten, müssen also für mehrere Ausfallsituationen geeignete Mapping-Strategien vorbereitet werden. Wenn n PEs zu Beginn zur Verfügung stehen, und der Ausfall von maximal k PEs verkraftet werden soll, müssen Mapping-Strategien für $n, n-1, n-2, \ldots, n-k$ PEs vorbereitet werden. Techniken der statischen Lastbalancierung sind jedoch nur für eine beschränkte Anzahl von Ausfällen von PEs geeignet.

3.4.2 Dynamische Lastbalancierung

Bei der dynamischen Lastbalancierung werden die Entscheidung über ein Lastverteilung und die Strategien zur Lastumverteilung während der Simulation getroffen und festgelegt. Spezielle Methoden für Lastbalancierung in der verteilten und parallelen Simulation sind zum Beispiel in [21, 26, 12] entwickelt worden, bei denen aufgrund von Lasten der einzelnen PEs entschieden wird, ob migriert wird oder nicht. Diese Methoden können auch für die Lastumverteilung nach einem Fehler eingesetzt werden, indem der fehlerhafte Rechnerknoten mit einem hohen Lastwert belegt wird.

3.5 Fehlertoleranz und die HLA

Die HLA ist ein Standard für verteilte Simulation in militärischen Bereichen, um Interoperabilität und Wiederverwendbarkeit von Simulationsmodellen zu ermöglichen und zu erleichtern. Die HLA und vor allem die RTI können aber auch als Kommunikationsmedium zur verteilten und parallelen Simulation zur Leistungssteigerung, wie sie in diesem Kapitel betrachtet wird, dienen [24]. Daher scheint es wichtig, Überlegungen über HLA und Fehlertoleranz anzustellen, da ja gerade im militärischen Bereich fehlertoleranten, zuverlässigen Simulationssystemen enorme Bedeutung zukommt. Wir sehen zwei grundlegende Möglichkeiten, HLA und Fehlertoleranz zu betrachten, wobei eine Möglichkeit sich auf die RTI, der Implementation der HLA Interface Spezifikation bezieht. Diese Ansätze sind:

- Fehlertolerante RTI Implementation,
- fehlertolerante Föderationen:
  - Fehlertoleranz innerhalb eines Föderaten,
  - Föderaten für Fehlertoleranz und
  - Erweiterung der HLA in bezug auf Fehlertoleranz.

3.5.1 Fehlertolerante RTI Implementation

Bei diesem Ansatz besteht die Idee darin, wichtige Prozesse der RTI gegen Ausfälle zu schützen. Zwei wichtige Prozesse sind der RTIexec Prozeß und der FedExec Prozeß. Der RTIexec Prozeß ist der Startprozeß einer HLA Föderation, und jeder Föderat dieser Föderation muß sich über diesen RTIexec registrieren. Der FedExec Prozeß ist verantwortlich für die Kommunikation zwischen den Föderaten und für die Besitzerrechte der Objekte und Attribute. Ohne FedExec ist daher eine Interaktion zwischen den Föderaten nicht möglich. Beide wichtigen Prozesse sind zentral auf einem PE und es existieren keine Replikate. Fällt einer dieser Prozesse aus, ist ein Fortfahren der verteilten Simulation nicht mehr möglich. Um die Fortsetzung der verteilten Simulation auch nach einem Ausfall eines dieser Prozesse zu ermöglichen, können diese beiden Prozesse repliziert werden, dezentral ausgeführt werden und ihre Zustände auf stabilen Speicher geschrieben werden.

3.5.2 Fehlertolerante Föderationen

Wir geben drei Strategien an, fehlertolerante Föderationen zu erzeugen.

Fehlertoleranz innerhalb eines Föderaten Hier werden Mechanismen zur Fehlertoleranz innerhalb des Föderaten implementiert, das sind Mechanismen zur Fehlererkennung, zur Datenwiedergewinnung und zur Lastumverteilung, wie sie weiter oben besprochen wurden. Die HLA dient nur als Kommunikationsmedium.

Föderaten für Fehlertoleranz Eine weitere Idee, die HLA für Fehlertoleranz zu nützen, ist eigenständige Föderaten zu konstruieren, die Mechanismen zur Unterstützung von Fehlertoleranz beinhalten. Zum Beispiel könnte in einer Föderation ein Föderat nur zur Fehlererkennung entwickelt werden, der die anderen Föderaten mit Methoden aus Kapitel 3.2 beobachtet.

Erweiterung der HLA in bezug auf Fehlertoleranz Bei dieser Idee wird die HLA selbst bezüglich Fehlertoleranz erweitert. Zusätzliche Services können eingebaut werden, die Fehlertoleranz unterstützen. Zum Beispiel können die save/restore Services erweitert werden. Es kann dadurch eine Automatisierung der Speicherung der Checkpoints erfolgen, sodaß ein konsistenter globaler Checkpoint erstellt wird. Ein weiteres Beispiel wäre, Services für ein Management zur Replikation von Föderaten einzurichten, die dann dafür verantwortlich zeichnen, daß die Replikate in einem konsistenten Zustand gehalten werden.

4 Zusammenfassung und Ausblick

In diesem Kapitel wurden vier Gründe angegeben, eine Simulation zu verteilen und parallel ablaufen zu lassen, diese Gründe sind:

- Geographische Verteilung,
- Nutzen von vorhandenen Ressourcen,
- Leistungssteigerung,
- Fehlertoleranz.

Daraus lassen sich folgende Zielsetzungen schlußfolgern:

- Simulieren größerer Modelle,
- ein effizientes Ressourcenmanagement,
- Verkürzung der Simulationszeit,
- höhere Zuverlässigkeit.

Bei der verteilten und parallelen Simulation zur Leistungssteigerung ist eine effektive Möglichkeit eine Parallelisierung auf Modellebene, daß heißt, das Simulationsmodell selbst wird in möglichst voneinander unabhängige Teilmodelle unterteilt. In der diskreten ereignisgesteuerten Simulation (die Schwerpunkt der Überlegungen dieses Berichtes ist) heißen diese Teilmodelle logische Prozesse (LP). Diese logischen Prozesse führen nun parallel die Ereignisse in ihren Ereignislisten aus. Da in den meisten Fällen jedoch nicht vollständig unabhängige Teilmodelle gefunden werden können, kann es durch die parallele Ausführung und durch das verschieden schnelle Fortschreiten in der Simulationszeit der einzelnen LPs zu Kausalitätsverletzungen kommen. Um diese Kausalitäsverletzungen zu verhindern, muß zwischen den LPs eine Kommunikation durch Nachrichten stattfinden, die LPs müssen synchronisiert werden. Je nachdem, ob die

Kausalitätsverletzungen a priori verhindert werden, oder ob zunächst Kausalitätsverletzungen zugelassen werden und erst anschließend behoben werden, unterscheidet man zwischen

- konservativen Synchronisationsprotokollen und
- optimistischen Synchronisationsprotokollen.

Hybride Synchronisationsprotokolle sind Mischformen von konservativen und optimistischen Verfahren. Bei einem Simulationsmodell kann jedoch oft nicht von vornherein entschieden werden, welche Art von Synchronisation vorteilhaft ist.

Ein weiterer Vorteil einer Verteilung einer Simulation kann Fehlertoleranz sein. In der herkömmlichen Simulation (sowohl sequentiell als auch verteilt und parallel) führt ein Ausfall eines logischen Prozesses, eines Rechnerknotens (PE) oder einer Kommunikationsverbindung zum Stillstand und Abbruch der Simulation. Ein Ausfall einer Simulation kann jedoch gerade im militärischen Bereich zu bedrohlichen Situation führen, zum Beispiel wenn die Simulation als DV-Unterstützung der Operationsführung eingesetzt wird. Deshalb sind in der verteilten und parallelen Simulation Mechanismen wünschenswert, die einen Ausfall eines logischen Prozesses, eines PE oder einer Kommunikationsverbindung tolerieren können. Fehlertolerante verteilte und parallele Simulation führt zu einer Steigerung der Zuverlässigkeit, und zwar wird erreicht:

- eine bessere Kosteneffizienz und
- ein Verhindern von gefährlichen sicherheitskritischen Situationen.

Drei Anforderungen muß die verteilte und parallele Simulation genügen, um Fehlertoleranz zu erzielen, das sind:

Fehlererkennung Fehler sowie die Art des Fehlers müssen in einer vertretbaren Zeit erkannt werden, damit eine geeignete Reaktion auf den Fehler eingeleitet werden kann.

Datenwiedergewinnung Damit die Simulation fortgesetzt werden kann, müssen die nach einem Fehler verlorengegangene Daten wieder gewonnen werden.

Lastumverteilung Bei nicht temporären Fehlern von Rechnerknoten oder bei Fehlern der Kommunikationsverbindungen müssen die verlorengegangenen und unerreichbaren LPs auf die verbleibenden Rechnerknoten umverteilt werden.

Wir schlagen hier zwei grundlegende Ansätze vor, verteilte und parallele Simulation fehlertolerant zu gestalten, diese sind:

- Anwendung von etablierten Fehlertoleranzmethoden im allgemeinen verteilten Rechnen auf den Spezialfall verteilte Simulation
- Adaption von Techniken und Methoden der parallelen und verteilten Simulation, damit Fehlertoleranz unterstützt wird.

In diesem Kapitel werden demnach Methoden zur Fehlertoleranz im allgemeinen verteilten Rechnen vorgestellt und es wird vorgeschlagen, wie diese Methoden auf den Spezialfall der verteilten und parallelen Simulation angewendet werden können. Ebenfalls werden Ideen dargelegt, wie Techniken in der verteilten und parallelen Simulation genutzt und adaptiert werden können, um Fehlertoleranz in der verteilten und parallelen Simulation zu unterstützen.

Besonderes Augenmerk wird in diesem Abschnitt auch Gedanken über HLA und Fehlertoleranz geschenkt. Wir geben vier Ansätze an, und zwar

- Fehlertolerante RTI Implementation,
- Fehlertoleranz innerhalb eines Föderaten,
- Föderaten für Fehlertoleranz und
- Erweiterung der HLA in bezug auf Fehlertoleranz.

In der Literatur und Forschung wurde dieser Thematik (Fehlertoleranz und verteilte Simulation) noch wenig Beachtung geschenkt. In diesem Kapitel werden daher vor allem Konzepte, Ideen und mögliche Lösungsansätze zu einer fehlertoleranten verteilten und parallelen Simulation gegeben. Das Erstellen einer prototypischen Implementation eines fehlertoleranten verteilten Simulators sowie Leistungs- und Zuverlässigkeitsmessungen darüber sind Gegenstand von zukünftiger Forschungsarbeit. Es wird immer wichtiger werden, eine Steigerung in der Zuverlässigkeit der Simulation zu erzielen, z. Bsp. wenn die Simulation als Werkzeug zur Entscheidungsfindung im militärischen Einsatz verwendet wird. Es wird daher in Zukunft von Bedeutung sein, Überlegungen über Fehlertoleranz in der verteilten und parallelen Simulation anzustellen.

Literatur

[1] D. Agrawal, J.R. Agre. *Replicated Objects in Time Warp Simulations.* Proceedings of the 1992 Winter Simulation Conference. 1992, pp.657-664.

[2] L. Alvisi, K. Marzullo. *Message Logging: Pessimistic, Optimistic, Causal and Optimal.* IEEE Transactions on Software Enginieering 24(2). 1998, pp.149-159.

[3] T. Becker. *Keeping Processes under Surveillance.* IEEE 16^{th} Symposium on Reliable Distributed Systems. 1991, pp.198-205.

[4] K.L. Chow, W.H. Einright. *Distributed Parallel Shooting for BVODES.* High Performance Computing 1998, pp. 203-209.

[5] J.S. Dahmann. *The High Level Architecture and beyond: Technology Challenges.* Proceedings of the 13^{th} Workshop on Parallel and Distributed Simulation (PADS'99). 1999, pp.64-70.

[6] O.P. Damani, V.K. Garg. *Fault-tolerant Distributed Simulation.* Proceedings of the 12^{th} Workshop on Parallel and Distributed Simulation (PADS'98). 1998, pp.38-45.

[7] E.N. Elnozahy, D.B. Johnson and Y.M. Wang. *A Survey of Rollback Recovery Protocols in Message Passing Systems.* School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, Technical Report, CMU-CS-96-181, http://ftp.cs.cmu.edu/users/mootaz/papers/S.ps, 1996.

[8] A. Ferscha. *Parallel and Distributed Simulation of Discrete Event Systems.* in Handbook of Parallel and Distributed Computing, McGraw-Hill, 1995, pp. 1004-1041.

[9] J. Fleischmann, P.A. Wilsey. *Comparative Analysis of Periodic State Saving Techniques in Time Warp Simulators.* Proceedings of the 9^{th} Workshop on Parallel and Distributed Simulation (PADS'95). 1995, pp.50-58.

[10] R.M. Fujimoto. *Parallel and Distributed Simulation.* Proceedings of the 1995 Winter Simulation Conference. 1995

[11] R.M. Fujimoto. *Parallel and Distributed Simulation.* Proceedings of the 1999 Winter Simulation Conference. 1999, pp.122-131.

[12] Boon Ping Gan, Yoke Hean Low, Sanjay Jain, Stephan J. Turner, Wentong Cai, Wen Jing Hsu, Shell Ying Huang. *Load Balancing for Conservative Simulation on Shared Memory Multiprocessor Systems.* Proceedings of the 14^{th} Workshop on Parallel and Distributed Simulation (PADS2000). 2000, pp.139-146.

[13] Competence Center Informatik GmbH. *Abschlußbericht: „GE-RTI Bewertung einer nationalen RTI-Entwicklung".* Bearb.-Nr: M/GSPO/Z0063/Z9963, 2000.

[14] Y. Huang, C. Kintala, N. Kolettis, D.N. Fulton. *Software Rejuvenation: Analysis, Module and Applications.* IEEE 25^{th} Symposium on Fault-tolerant Computing. 1995, pp.381-390.

[15] J. JàJà. *An Introduction to Parallel Algorithms.* Addison-Wesley New York, 1992

[16] Journal of Applied Numerical Mathematics, Vol. 11, 1993

[17] J. Lüthi, C. Berchtold. *Concepts for Dependable Distributed Discrete Event Simulation.* Proceedings of the 14^{th} European Simulation Multi-Conference 2000 (ESM2000). 2000, pp. 59-66.

[18] H. Mehl. *Methoden verteilter Simulation.* Vieweg Braunschweig, 1994

[19] D.A. Patterson, P. Chen, G. Gibson, R.H. Katz. *Introduction to Redundant Arrays of Inexpensive Disks (RAID).* IEEE Spring 89 COMPCON. 1989, pp.112-117.

[20] J.S. Plank. *Improving the Performance of Coordinated Checkpointers on Networks of Workstations using RAID Techniques.* IEEE 15^{th} Symposium on Reliable Distributed Systems. 1996, pp.76-85.

[21] R. Schlagenhaft, M. Ruhwandl, C. Sporrer, H. Bauer. *Dynamic Load Balancing of a Multi-Cluster Simulator on a Network of Workstations.* Proceedings of the 9^{th} Workshop on Parallel and Distributed Simulation (PADS'95). 1995, pp.175-180.

[22] H. R. Schwarz. *Numerische Mathematik.* B.G. Teubner Stuttgart, 1997

[23] E.J. Spee, J.G. Verwer, P.M. de Zeeuw, J.G. Blom, W. Hundsdorfer. *A Numerical Study for Global Atmospheric Transport-Chemistry Problems.* Mathematics and Computers in Simulation 48(1998), p. 177-204.

[24] J.S. Steinman, G. Berliner, G.E. Blank, J.S. Brutocao, J. Burckhardt, M. Peckham, S. Shupe, K. Stadsklev, T. Tran, R. Van Iwaarden, L. Yu. *The SPEEDES-based Run-Time Infrastructure for the High-Level Architecture in High-Performance Computers.* Proceedings of the High Performance Computing Symposium - HPC'99. 1999, pp.255-266.

[25] U. Trottenberg. *Quantensprünge in der numerischen Simulation.* Der GMD-Spiegel 3/4-1998. 1998

[26] Voon-Yee Vee, Wen-Jing Hsu. *Locality-Preserving Load-Balancing Mechanisms for Synchronous Simulations on Shared-Memory Multiprocessors.* Proceedings of the 14^{th} Workshop on Parallel and Distributed Simulation (PADS2000). 2000, pp.131-138.

[27] Y.-M. Wang. *Maximum and Minimum Consistent Global Checkpoints and their Applications.* IEEE 14^{th} Symposium on Reliable Distributed Systems. 1995, pp.86-95.