Zeitsynchronisierung von verteilter Simulation

Roman Kuprat

Erstprüfer	Christoph Nebl
Zweitprüfer	Dr.-Ing. Paul Spannaus
Ausgabedatum	10.05.2020
Abgabedatum	12.07.2020

1 Einleitung

Simulationen können im groben in zwei Kategorien eingeteilt werden, zum einen die Simulation von Stochastiken, welche eingesetzt wird um Vorhersagen zu treffen. Diese benötigen stets einen Simulationsaufbau, welcher die Eingehenden Parameter hervorhebt und die zu simulierende Variable definiert. Diese Simulationen hängen von Chancen und Wahrscheinlichkeiten ab und können sich demnach von Simulation zu Simulation unterschiedlich Verhalten, somit kann keine Aussage mit 100 prozentiger Wahrscheinlichkeit getroffen werden. Anhand von Trends, best- und worst-case-szenarios kann analysiert werden, in welchem Bereich sich das Ergebnis bewegen wird. Zum anderen existieren Simulatoren wie zum Beispiel ein Fahrzeugsimulator, hierbei wird das reelle Fahrerlebnissimuliert und ein Nutzer fährt in diesem Beispiel ein Auto oder anderes Fahrzeug. Diese Art der Simulatoren kann für Menschen genutzt werden um sich mit einer Aufgabe oder Situation vertraut zu machen oder um künstliche Intelligenz in einem geschützen Umfeld zu trainieren.

Diese Arbeit befasst sich Inhaltlich mit der Zeitsynchronisation von verteilter Simulation. Hierbei soll geklärt werden, warum eine verteilte Simulation sinnvoll sein kann. Um dies zu veranschaulichen wird ein Beispiel aufgebaut, anhand dessen klar werden sollte warum auch Verteilung der Simulation bei wenig komplizierten Simulationen sinnvoll sein kann.

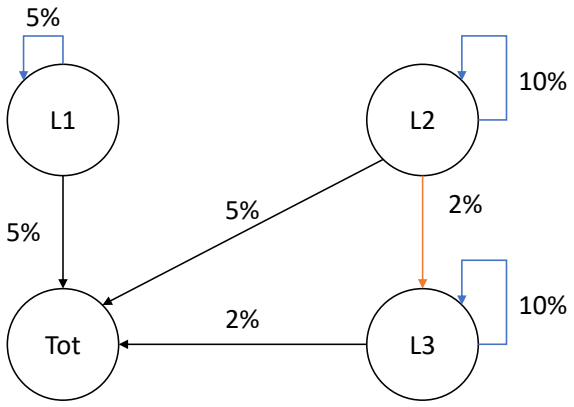
Ein weiterer Aspekt in dieser Arbeit ist die Zeitsynchronisation solcher Simulation, zum einen warum eine Zeitsynchronisation von nöten ist und welche Methoden genutzt werden können um die verschiedenen Komponenten zu synchronisieren.

2 Verteilte Simulation

Dies sind Simulationen, welche nicht mehr zentral von einem einzigen Simulationssystem gehandhabt werden. Hierfür gibt es mehrere Gründe, zum einen können dadurch zusätzliche Ressourcen für die Simulation genutzt werden. Dies reduziert die Simulationsdauer oder es werden komplexere Simulationen möglich. Sollten die Simulationen für Stochastik und Trendanlysen genutzt werden, können so mehrere Systeme die gleiche Simulation mit den selben Parametern laufen lassen um die Wiederholungsrate zu senken. Bei Simulatoren können wie beim programmieren Berechnungen aufgeteilt werden um so die Stabiltät zu verbesseren.

2.1 Beispiel Monte-Carlo-Simulation

Monte-Carlo-Simulationen sind Simulationen aus dem Bereich der Stochastik. Dies kann genutzt werden um Verhalten zu Simulieren, welches von Zufall abhängt. Es lassen sich zwar mathematisch auch Aussagen zu solchen Simulationen treffen, allerdings kann durch die Unterstützung der Simulation abgeschätzt werden ob die Annahme zutreffend war.



In diesem Beispiel gibt es 3 Lebensformen, jede hat eine Chance sich pro Tick zu reproduzieren und eine Chance zu sterben. Die 2te Lebensform hat bei einer Reproduktion eine 2% Chance zu L3 zu mutieren. Jede Lebensform ist leicht verändert gegenüber den anderen Lebensformen. Bei L1 ist die Sterblichkeit genauso hoch wie die Chance der Reproduktion, somit kann es zwar zu Schwankungen kommen allerdings sollte die Anzahl dieser Lebewesen um den Startwert sich bewegen. L2 besitzt zwar die gleiche Sterblichkeitsrate wie L1, allerdings besitzt diese eine doppelt so hohe Chance für die eigene Reproduktion, somit müsste die Anzahl der Lebensform L2 steigen. Zudem besitzt L2 als einzige Lebensform die Möglichkeit zu mutieren. L3 besitzt die gleiche Reproduktionschance wie L2, besitzt aber zudem noch eine weit niedrigere Sterberate, was dafür sorgen sollte, dass L3 prozentual betrachtet noch rapider wächst als L2.

$$L1(t+1) = 0,05 \cdot L1(t) - 0,05 \cdot L1(t) \quad (2.1)$$

$$L2(t+1) = (0,1-0,1 \cdot 0,02) \cdot L2(t) - 0,05 \cdot L2(t) \quad (2.2)$$

$$L3(t+1) = 0,1 \cdot L3(t) - 0,02 \cdot L3(t) + (0,1 \cdot 0,02) \cdot L2(t) \quad (2.3)$$

Nun können diese Gleichungen nach den entsprechenden Faktoren umgestellt werden. Sollten nur die reinen Faktoren betrachtet werden, so ergibt sich aus 2.1 folgendes

$$\Delta T = 5\% - 5\% <=> \Delta T = 0 \quad (2.4)$$

Aus 2.4 bestätigt sich die vorher gestellte Annahme, dass sich L1 in der Regel nicht verändert, natürlich existieren hierbei durch Zufälle ein Steigerung oder Reduktion der Lebensform, aber in der Regel ist keine Änderung zu erwarten.

Bei 2.2 muss die Reproduktionszahl zusätzlich um die Mutationschance reduziert werden, werden dann die entsprechenden Faktoren zusammengerechnet ergibt sich folgendes

$$\Delta T = (10\% - 10\% \cdot 2\%) - 5\% <=> \Delta T = 4,8\% \quad (2.5)$$

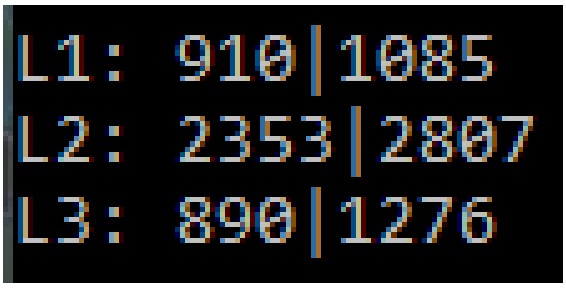
Wie erwartet sollte L2 steigen, aus 2.5 geht hervor, dass diese Lebensform im Durchschnitt um 4,8% pro Generation steigen sollte.

Wie aus 2.3 hervorgeht hängt der Wachstum von L3 nicht nur von der derzeitigen Population ab, sondern auch von der Größe von L2, da hieraus ebenfalls Lebewesen von L3 hervorgehen können. Somit ist es hierbei nicht möglich den genauen Wachstum der Lebensform in Relation zur vorherigen Generation zu bestimmen. Was allerdings gemacht werden kann, ist der Anstieg der eigenen Population ohne Mutation zu bestimmen und zu vermerken, dass der Anstieg im Durchschnitt mindestens so hoch sei.

$$\Delta T = 10\% - 2\% <=> \Delta T \geq 8\% \quad (2.6)$$

Die abgebildete Änderung von 2.6 liegt somit mindestens bei 8% pro Generation. Diese Lebensform ist somit die einzige für die keine Linearefunktion genutzt werden kann um die durchschnittliche Veränderung darzustellen.

Da es sich hierbei um Durchschnittswerte handelt wäre zu erwarten, dass die Simulationen von dem Wert abweicht, allerdings sollten die Anstiege in relation betrachtet dennoch zusammenpassen. Hierfür wurde ein kleines Skript genutzt, welches die Chancen dieser Lebensformen nutzt und diese simuliert. Es wurde über 20 Generationen durchgeführt und am Ende dieser 20 Zyklen geprüft ob es ein Minimum oder Maximum ist. Der Test wurde 100 mal ausgeführt damit sich Extremas bilden können. Eingestiegen wurde mit 1000 Vertretern von L1/L2 und 200 Vertretern von L3.



Hierbei kann beobachtet werden, dass L1 den kleinesten Ausschlag besitzt. Dies ist eine Folge von dem nicht erwartenden Wachstums. Da sich die Basis nur langsam entwickelt und somit keine Anstiege in

der Population verlangsamt oder beschleuntigt werden. Es passt auch zu dem vermuteten Durchschnittsanstieg, dass sich die Extremas um den Startwert bewegen.

Bei L2 liegen die beiden Extremas bereits weiter auseinander. Dies liegt daran, dass ein Anstieg zu erwarten ist und sich eine gute oder schlechte Generation stark auf den Anstieg auswirkt, da die Anzahl der Lebewesen deren Wachstum sehr stark beeinflusst.

L3 ist der Sonderfall, da hier die Steigung nicht nur von der eigenen Größe abhängt, sondern auch von L2, weshalb hier das größte Wachstum zu erwarten war. Aber durch diese schneller Entwicklung wirken sich wie bei L2 der Erfolg der vorherigen Generation noch stärker auf die Population in der nächsten Iteration aus, das sorgt dafür, dass es hierbei eine Abweichung von fast 50% gibt.

Eine kleine Basis der Lebensform birgt aber ein Risiko hierbei, da eine Chance besteht, dass diese Lebensform trotz prognostitierten Anstieg ausstirbt. Allgemein kann diese Wahrscheinlichkeit im gegebenen Beispiel mit folgender Formel abgebildet werden.

$$((1 - Rep + (Rep \cdot Mut)) \cdot Ste)^n \quad (2.7)$$

Rep: Reproduktionschance
Mut: Mutationschance
tS: Sterbewahrscheinlichkeit
n: Anzahl der Lebewesen

Die Formel 2.7 besteht aus der Chance, dass keine neuen Lebewesen der eigenen Form hinzukommen, dies ist wenn keine neuen entstehen und wenn zwar neue entstehen, diese aber einer anderen Lebensform zugeordnet werden. Die Anzahl der Lebewesen sorgt hierbei für einen exponentielle Minderung der Wahrscheinlichkeit, dass diese Lebensform verschwindet. Da die Population exponentiellen Einfluss bleibt dennoch immer eine geringe Restchance. Somit kann das aussterben einer Lebensform innerhalb eines Zykluses auch ein Zufall sein.

$$(1 - 0,1 + (0,1 \cdot 0,02)) \cdot 0,5 = 0,046 \quad (2.8)$$

2.8 ist eine Beispielrechnung von L2 mit einem einzigen Vertreter, für diese Situation besteht also im ersten Lebenszyklus eine Chance von 4,6%, dass diese Lebensform aussirbt. Sollte die Simulation also nur wenige male ausgeführt werden besteht eine Wahrscheinlichkeit, dass zufällig dieser Fall mehrmals getroffen wurde.

2.2 Wiederholung der Simulation parallelisiseren

Wie aus dem Beispiel hervogeht, besteht stets eine Chance für Zufälle, um diese Wahrscheinlichkeit zu reduzieren ist es sinnvoll, die Simulation häufig ablaufen zu lassen um Ausnahmeverhalten als solche zu erkennen. Den Simulationen mit vielen komplexeren Parametern mit verschiedensten Abhängigkeiten erlauben es nicht, die Erwartung an die Simulation so einfach zu berechnen. Da aber das mehrfache Ausführen die benötigte Zeit erhöht wäre es sinnvoll diese einfachen Simulationen verteilt Ablaufen zu lassen.

$$S_R = \frac{S_g}{n} \quad (2.9)$$

S_R: Simulation pro Ressource
S_g: Simulation gesamt
n: Zu Verfügung stehende Ressourcen

Dieser Zeitgewinn lässt sich auch wie in 2.9 dargestellt ausrechnen. Da jede Ressource einen Teil der benötigten Simulationen durchführt, kann der benötigte Zeitaufwand drastisch reduziert werden.

2.3 Verteilte Simulation bei Simulatoren

Ein Simulator kann unterschiedlich verteilt ausgeführt werden. Wenn die Simulation nur für einen Nutzer ablaufen soll, so kann die Simulation bei klassischer Software parallelisiert werden. Allerdings sollte eine Simulationsumgebung geschaffen werden, in welcher mehrere Nutzer gleichzeitig Aktionen tätigen gibt es verschiedene Möglichkeiten dies zu realisiseren. Veranschaulicht wird dies Anhand eines Fahrsimulators als Beispiel.

1. Übertragung der eigenen vollständigen Simulation
2. Veränderung der Position des eigenen Fahrzeuges
3. Veränderung der Parameter des eigenen Fahrzeuges
4. Tastatureingabe

Hierbei muss nun abgewägt werden, welcher Ansatz verfolgt werden soll. Der erste Ansatz ist am unabhängigsten von der Simulationsumgebung, da hier alles auf dem Rechner des Nutzer generiert wird. Diese Welt wird dann an die Umgebung übermittelt, diese muss anhand aller übermittelten Umgebungen die tatsächliche Welt generieren und an die anderen Maschinen übermitteln. Beim zweiten Ansatz wird von dem Nutzer die neue Position des Fahrzeuges ermittelt und diese Information wird dann der Umgebung bereitgestellt. Die Umgebung updated damit die Position des Fahrzeuges für alle Nutzer. Die dritte Möglichkeit wäre nur die Parameter für das Auto zu übermitteln. Hierbei würden Variablen wie zum Beispiel Geschwindigkeit und Beschleunigung an den Simulationsumgebung übergeben. Diese berechnet dann für jeden Nutzder die Position im nächsten Tick. Der letzte Ansatz verlangt dem Nutzer am wenigsten ab, da dieser nur die Tastatureingaben bereitstellt und der Server für alle Nutzer alles bestimmen und berechnen muss.

Bei diesen Ansätzen handelt es sich um verschiedene Abstraktionsebenen und je nach Wahl ändert sich der Rechenaufwand der jeweiligen Komponenten. Um also das beste aus der Verteilung der Simulation zu holen ist es wichtig den Rechnenaufwand sinnvoll zu verteilen. Es gibt hierbei kein Ansatz der pauschal besser ist, da es von der Performance der einzelnen Komponenten abhängig wie gut ein Ansatz funktioniert. Sollten KI-System nur wenig Rechenleistung überhaben ist es nicht zielführend diesen Systemen die vollständige Simulation zu überlassen.

2.4 Fehlertoleranz

Die Verteilung der Simulation kann auch die Fehlertoleranz steigern, da hierbei auf andere Rechner gesetzt wird, welche andere Vorraussetzungen besitzen. Ob nun bei der Software oder auch bei der Hardware, dies sind potentiell Dinge die auf die Simulation Einfluss nehmen könnten, obwohl diese es nicht sollten. Sollte zum Beispiel jeder Simulator seine eigene Welt erzeugen, welche dann zusammengeführt reduziert dies die Wahrscheinlichkeit eines Fehlers in der simulierten Welt.

2.5 Zeitgewinnung durch Parallelisierung einer Simulation

Allerdings kann auch der andere Simulationstyp parallelisisert werden, im Beispiel mit dem Fahrsimulator wäre es denkbar, dass einzelne Teile der Welt parallel berechnet werden. Hierfür könnten dann Objekte grupiert werden und die jeweiligen Gruppen von zum Beispiel einem anderen Thread generiert werden.

$$n_S = \frac{T}{t_S + \frac{t_p}{n_p}} \tag{2.10}$$

n_S: Speedup
T: Gesamtlaufzeit
t_S: serielle Programmzeit
t_p: parallelisier bare Programmzeit
n_p: vorhandene Prozessoren

Das Amdahlsche Gesetz 2.10 beschreibt den Zeitgewinn eine Programmes, welches vollständig paralelisisert wurde. Somit kann diese Formal auch benutzt werden um den Zeitgewinn für parallelisierte Berechnungen einer Simulation zu ermitteln. Die Steigerung der Performance der Simulation bringt einige Vorteile mit sich. Zum einen bringt dies Stabilität in die Simulation, diese erlaubt es, dass Prozesse flüssig ablaufen und es zu weniger Inputlags oder ähnlichen kommt. Dadurch fühlt sich die Simulation schließlich Realitätsnäher an. Zudem erlaubt eine stabile Simulation die Erweiterung dieser um weitere Komponenten, sollte nun der Fahrsimulator gut und flüssig laufen, so könnten neue Dinge hinzugefügt werden, wie zum Beispiel unterschiedliches Wetter, Stausituationen oder Unfallsituationen. Denn sollte eine KI anhand dieses Simulators für den realen Einsatz trainiert werden müssen allerhand Situation trainiert werden um eine hohe Sicherheit gewährleisten zu können.

3 Zeitsynchronisation

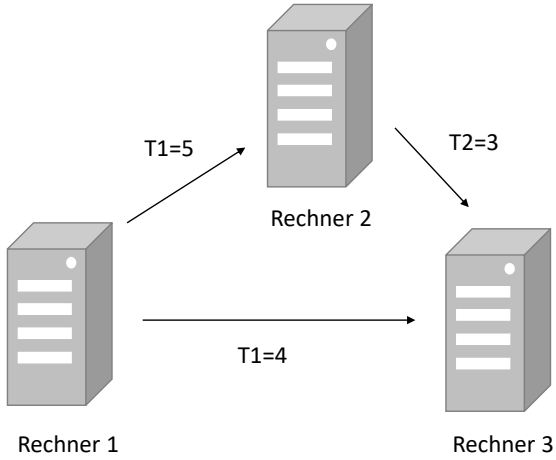
Dieses Problem stellt sich nicht, wenn Simulationen vollständig ausgeführt werden und nur deren Ergebnisse zusammengetragen werden, da hierbei nur die Endergebnisse übertragen werden und somit keine Informationen Synchronisiert werden müssen. Sollte aber eine Simulation parallel ablaufen so müssen die Komponenten Zeitsychronisiert werden. Sind Daten zum Beispiel zentral auf einem Server gelagert werden, besteht die Möglichkeit, dass diese Daten von einem System geändert werden, während ein anderes System diese Daten ebenfalls ändern möchte. Dies stellt aber ein Problem dar, da hierbei nicht klar ist, welche dieser Übertragungen den neusten Informationsstand hat.

3.1 Kommunikation aller Änderungen

Sollten bei jeder Veränderung die Maschinen ihren Informationsstand an alle Übermitteln, müsste entweder ein zentraler Server eine Änderung an alle Supporter übermitteln oder jede Komponente die neuen Informationen per Broadcast weitergeben. Bei beiden würde es eine enorme Netzauslastung bedeuten, welche dafür sorgen würde, dass die Komponenten ihre neuen Informationen nur Zeitverzögert mitteilen können, da das Netz bereits belegt ist.

3.2 Lokaler Zeitstempel

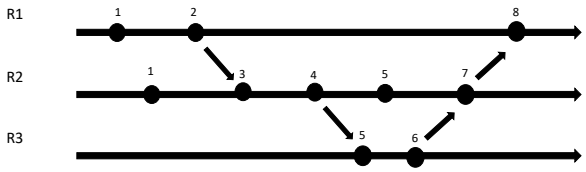
Jeder Rechner besitzt einen eigenen lokalen Zeitstempel, dieser wird bei jeder Aktion inkrementiert und zusammen mit den Daten an die anderen Rechner im Netz übergeben.



1. Rechner 1 übermittelt zu Rechner 3
2. Rechner 1 übermittelt zu Rechner 2
3. Rechner 2 übermittelt zu Rechner 3

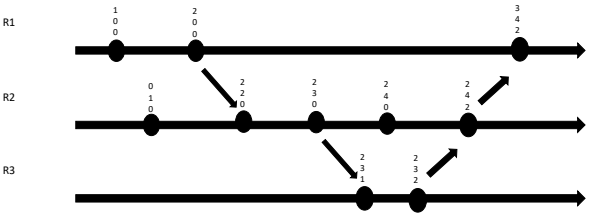
In diesem Szenario wird Rechner 3 vor ein Entscheidungsproblem gestellt, da dieser eine Nachricht von Rechner 1 mit T=4 erhält und von Rechner 2 eine Nachricht von T2=3. Sollten diese Daten zusammengesetzt werden, besteht so keine Möglichkeit die Nachrichten nach Aktualität zu unterscheiden.

3.3 Lamport-Synchronisation



Bei der Lamport-Synchronisation wird bei jeder Aktion der lokale Zeitstempel inkrementiert. Bei Kommunikation mit einem anderen Rechner wird dieser mit übertragen. Der Rechner, welcher die Nachricht mit dem Zeitstempel erhält prüft den übermittelten Zeitstempel gegen den lokalen. Bei der nächsten Aktion wird der höchste bekannte Zeitstempel inkrementiert und weiterhin als eigener genutzt beziehungsweise übertragen. Hierbei besteht nur das Problem, dass Zeitstempel nicht einem Rechner zugeordnet sind. Da ein Rechner lokal genug Aktionen tätigen könnte, dass der lokale Zeitstempel höher ist als der eingehende.

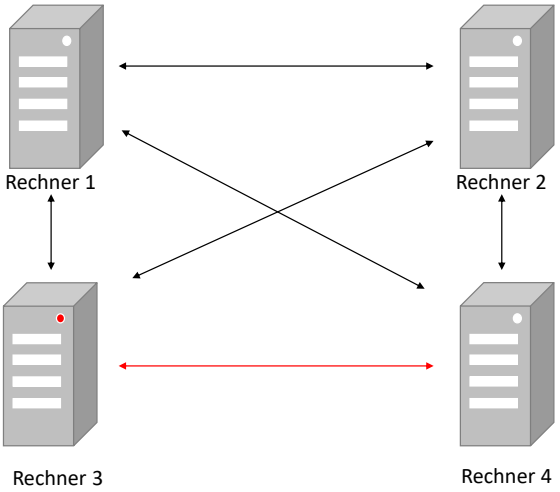
3.4 Vektorzeit



Systeme die mit Vektorzeit synchronisiert werden, besitzen den letzten bekannten Zeitstempel aller anderen Maschinen. Bei einer lokalen Aktion wird der lokale Zeitstempel inkrementiert. Übermittelt ein Rechner Information zu einem anderen Rechner, so wird hier der aktuelle Vektor mitübertragen. Die eingehende Maschine wählt dann das entsprechende Maximum und bildet daraus den neuen lokalen Vektor. Hierdurch gibt es nicht das Problem wie bei Lamport, da hier jeder Rechner seinen eigenen Zeitstempel hat. Zudem ist immer bekannt, wie aktuell die Information des entsprechenden Rechnes ist.

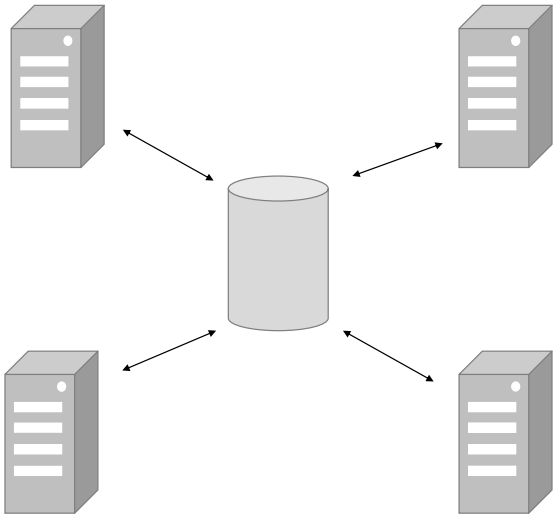
3.5 Zeitsynchronisation bei Simulatoren

Die Synchronisierung kann hierbei auf zwei Arten geschehen, hierbei verhält es sich anders, wenn die Simulationen lokal laufen, als bei den anderen 3 Möglichkeiten wo lokal höchstens einfache Berechnungen gemacht werden.



Grund hierfür ist, dass wenn eine Welt lokal simuliert wurde ist es relevant zu Wissen in welchem Stadium sich die anderen Simulationen befanden, demnach müsste bei solch einem Aufbau auf Vektorzeiten gesetzt werden, damit stets nachvollziehbar bleibt, mit welchen Informationen der jetztige Stand generiert wurde. Sollte nämlich nun in der Simulation von Rechner 3 mit dem geführten Fahrzeug und dem Fahrzeug von Rechner 4 ein Unfall passieren wird dieser Zustand den anderen Rechnern übermittelt. Allerdings passiert in der Simulation von

Rechner 4 kein Unfall, zwischen den beiden muss nachvollziehbar sein, welcher Rechner mit einem veralteten Stand gearbeitet hat. Damit dies möglich ist, können die anderen Maschinen die Vektorzeiten bewerten, aus diesen ergibt sich dann welcher Informationsstand der aktuell ist. Daraus lässt sich dann ein Konsens bilden, dass in diesem Beispiel Rechner 3 mit veralteteten Daten der anderen Rechner gearbeitet hat.



Sollte die verteilte Simulation mit einem der anderen drei Ansätze realisiert werden, so müssen keine Vektorzeit genutzt werden. Eine einfache Synchronisation ist hier ausreichend, da die Systeme nur ihre eigenen Veränderungen melden, welche dann die zentrale Instanz benutzt um die Welt zu simulieren. Es besteht auch keine Kommunikation zwischen den Clients sondern nur zwischen Client und Server. Da hierbei der Server als Koordinator eingesetzt wird und die Daten, welche die Nutzer verändern nur ihre eigenen sind, kann es nicht zu unklarheiten bezüglich der Simulation kommen, da der Server die Umgebung bereitstellt und die Nutzer nur ihre Aktionen übermitteln.

4 Ausblick und Fazit

Bei klassischen Programmen wird auch immer verstärkter auf Parallelisierung gesetzt um Performance zu gewinnen, demnach wäre anzunehmen, dass auch bei Simulationen immer verstärkter auf parallele Abläufe gesetzt wird. Dies ermöglicht eine Stabilisierung der Simulationen und erlaubt auch einen komplexeren Simulationsaufbau.Je nach Simulation ist eine andere Art von der Parallelisisierung möglich und einsetzbar.

Die Zeitsynchronisaiton kann bei Simulationen unterschiedlich angegangen werden und welche Synchronisation angewendet werden muss hängt hierbei stark von dem Grund der Synchronisation ab, da wenn vollständige Simulationen unabhängig ausgeführt werden die Komponenten nicht umfangreich Synchronisiert sein müssen wenn deren Ergebnisse nur gesammelt werden und dann gemeinsam betracht werden sollen. Sollte aber eine Simulation aufgeteilt werden ist eine Zeitsynchronisation entscheidend, da hierbei die verschiedenen Datensätze passend zusammengesetzt werden müssen.

Literatur

[1] Arne Dieckmann. „Entwicklungspfade und Zukunftschancen der verteilten Simulation in der Produktion und Logistik“. Diss. URL: http://www.itpl.mb.tu-dortmund.de/publikationen/files/FA_2015_Dieckmann.pdf.

[2] Ding-Geng (Din) Chen, John Dean Chen. *Monte-Carlo Simulation-Based Statsitical Modeling.* Singapur: Springer, 2017.

[3] Kai-Steffen Jens Hielscher. „Measurement-Based Modeling of Distributed Systems“. Diss. Erlangen: Technischen Fakultät der Universität Erlangen-Nürnberg.

[4] *Parallele und verteilte Simulation.*

[5] Stephan Höme. „Analytische Modellierung des Zeitverhaltens von verteilten industriellen Steuerungssystemen“. Dissertation. Magdeburg: Otto-von-Guericke-Universität, 1.04.2016.

[6] Christian Thiel. „Analyse von Partitionierungen und partieller Synchronisation in stark verteilten multiagentenbasierten Fußgängersimulationen“. Diss. URL: https://edoc.sub.uni-hamburg.de/haw/volltexte/2013/2059/pdf/MA_christian_thiel.pdf.

[7] Thomas Schulze, Ulrich Klein, Steffen Strassburger, Klaus Christoph Ritter, Eberhard Blümel, Marco Schumann. „HLA basierte verteilte Simulationsmodelle für die Fertigung“. Diss. URL: https://strassburger-online.de/papers/SV98Fertigung.pdf.

Anhang

```
#include <iostream>
struct Lebensform {
    int Anzahl;
    float Reproduktionschane;
    float Sterbechance;
    float Mutationschance;
};
void CalculateNextGen(Lebensform& leben)
{
    int i_change_in_anzahl = 0;
    int i_repro = (int)(1 / leben.Reproduktionschane);
    int i_sterb = (int)(1 / leben.Sterbechance);
    for (int i = 0; i < leben.Anzahl; ++i)
    {
        int rdm_repro = rand();
        int rdm_die = rand();
        if (rdm_repro % i_repro == 0)
        {
        ++i_change_in_anzahl;
    }
    if (rdm_die % i_sterb == 0)
        i_change_in_anzahl $\minuseq$ 1;
    }
    leben.Anzahl += i_change_in_anzahl;
}
void CalculateNextGen(Lebensform& leben, Lebensform& mutation)
{
    int i_change_in_anzahl = 0;
    int i_repro = (int)(1 / leben.Reproduktionschane);
    int i_sterb = (int)(1 / leben.Sterbechance);
    int i_mut = (int)(1 / leben.Mutationschance);
    for (int i = 0; i < leben.Anzahl; ++i)
    {
    int rdm_repro = rand();
    int rdm_die = rand();
    if (rdm_repro % i_repro == 0)
    {
    if (rand() % i_mut == 0)
    mutation.Anzahl += 1;
    else
    ++i_change_in_anzahl;
    }
    if (rdm_die % i_sterb == 0)
        i_change_in_anzahl $\minuseq$ 1;
    }
    leben.Anzahl += i_change_in_anzahl;
}
int main()
{
    int i_min1 = 10000;
    int i_max1 = 0;
    int i_min2 = 10000;
    int i_max2 = 0;
    int i_min3 = 10000;
    int i_max3 = 0;
    for (int j = 0; j < 100; ++j)
    {
        Lebensform L1 = { 1000, 0.05, 0.05, 0.0 };
        Lebensform L2 = { 1000, 0.1, 0.05, 0.02 };
        Lebensform L3 = { 200, 0.1, 0.02, 0.0 };
        std::cout << "{ 0 ," << L1.Anzahl << ", " << L2.Anzahl << ", " << L3.Anzahl << "}\n";
        for (int i = 0; i < 20; ++i)
        {
            CalculateNextGen(L1);
            CalculateNextGen(L3);
            CalculateNextGen(L2, L3);
            std::cout << "{ " << i+1 << " ," << L1.Anzahl << ", " <<
                L2.Anzahl << ", " << L3.Anzahl << "}\n";
        }
        std::cout << "\n\n";

        if (L1.Anzahl < i_min1)
            i_min1 = L1.Anzahl;
        else if(L1.Anzahl > i_max1)
            i_max1 = L1.Anzahl;

        if (L2.Anzahl < i_min2)
            i_min2 = L2.Anzahl;
        else if (L2.Anzahl > i_max2)
            i_max2 = L2.Anzahl;

        if (L3.Anzahl < i_min3)
            i_min3 = L3.Anzahl;
        else if (L3.Anzahl > i_max3)
            i_max3 = L3.Anzahl;
    }
    std::cout << "L1: " << i_min1 << "|" << i_max1 << "\n";
    std::cout << "L2: " << i_min2 << "|" << i_max2 << "\n";
    std::cout << "L3: " << i_min3 << "|" << i_max3 << "\n";
}
```