technische universität
dortmund

Masterarbeit

Platzierung von echtzeitfähigen virtuellen Maschinen als praktisches Optimierungsproblem

Ulrich Thomas Gabor
10. August 2015

Betreuer:
Prof. Dr.-Ing. Olaf Spinczyk
Dipl.-Inf. Boguslaw Jablkowski

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl 12
Arbeitsgruppe Eingebettete Systemsoftware
http://ess.cs.tu-dortmund.de

Errata

Dies ist eine korrigierte Fassung der eingereichten Master-Arbeit. Einfügungen gegenüber der eingereichten Version sind blau markiert, Streichungen ~~rot~~.

Liste der Änderungen

- Abschnitt 3.1.3.1: C_k fehlte in zwei Formeln.
- Fehlerhafte Darstellung der Autoren von [Dra+14].

Letzte Änderung: 4. März 2016

Inhaltsverzeichnis

Kapitel 1

Einleitung und Motivation

Mit der zunehmenden Globalisierung und komplexer werdenden Prozessen steigen auch die Anforderungen an unterstützende Technologien. Innerhalb der letzten 10 Jahre festigte sich für derartige komplexe Systeme der Begriff CPS (*cyber-physical systems*), welcher mittlerweile der Oberbegriff für all jene Systeme ist, die zusätzlich zu softwaretechnischen Komponenten auch aus mechanischen oder elektronischen Komponenten bestehen, darüber hinaus verteilt angeordnet sind und somit zwangsweise miteinander kommunizieren müssen. In den letzten Jahren zeigte sich auch die Relevanz für den deutschen Raum, z. B. im Rahmen von intelligenten Stromnetzen [JS15] und Verkehrssteuerungssystemen.

Gerade kritische Systeme, deren Ausfall eine Gefahr für Leib und Leben darstellt, müssen oft echtzeitfähig sein, also Ergebnisse nicht nur korrekt sondern auch zeitig berechnen. Diesen Anforderungen begegnet man zumeist mit einer übermäßigen Bereitstellung von Redundanz und Rechenleistung, um auch bei Engpässen eine zeitnahe Reaktion des Systems gewährleisten zu können. Dies ist jedoch mit überdurchschnittlichen Kosten verbunden, da es sich bei den Komponenten regelmäßig um Spezialhardware handelt, und ist somit hinsichtlich des allgegenwärtigen Wunsches Kosten zu sparen nicht erstrebenswert.

Eine Technologie, die in den letzten Jahren verstärkt Aufmerksamkeit erregt hat, ist Virtualisierung. Sie ermöglicht das Betriebssystem von der realen zu entkoppeln und sie stattdessen auf virtueller Hardware zu platzieren – man schafft somit eine VM (virtuelle Maschine). Dies ist nicht nur ein interessantes informationstechnisches Konzept, sondern bringt auch wünschenswerte Eigenschaften hinsichtlich Fehlertoleranz, Verfügbarkeit und optimale Ausnutzung von Ressourcen und somit Kosteneinsparungen mit. Gerade unter Berücksichtigung kürzlicher Veröffentlichungen, die besagen, dass in professionellen Rechenzentren selten mehr als 6% der zur Verfügung stehenden Rechenleistung genutzt wird, und bis zu 30% der laufenden Server überhaupt keine nutzbringenden Dienste mehr erbringen [KT15], lohnt sich eine detailliertere Betrachtung und Nutzung von Virtualisierung in weiteren Bereichen. Die Verwendung dieser Technologie im Kontext von CPS erscheint somit lohnenswert, jedoch

wurde bisher wenig Wert auf Echtzeitfähigkeit gelegt. Die vorliegende Arbeit soll deshalb einen weiteren Aspekt der Echtzeitfähigkeit bei Nutzung von Virtualisierung untersuchen.

Abgesehen von bereits betrachteten Problemen, die sich auf einer einzelnen realen Maschine ergeben, z. B. wie man die echtzeitfähige Ausführung einer virtuellen Maschine garantiert [JS12], ergibt sich über mehrere reale Maschinen hinweg ein zu optimierendes Platzierungsproblem:

Berechnung einer Platzierung von echtzeitfähigen virtuellen Maschinen auf Virtualisierungshosts unter Einhaltung nichtfunktionaler Eigenschaften

1.1. Ziele der Arbeit

Das vorgestellte Problem soll in der vorliegenden Arbeit untersucht werden. Dabei sollen folgende Ziele erreicht werden:

- Erstellung einer Anforderungsanalyse des Platzierungsproblems.
- Modellierung des Platzierungsproblems im Kontext von CPS.
- Ermittlung von Qualitätskriterien von Platzierungen.
- Erstellung eines Verfahrens zur Berechnung einer Platzierung unter Zuhilfenahme gängiger Optimierungsmethoden.
- Gewährleisten von Garantien für eine berechnete Platzierung, z. B. unter Ausnutzung eines formalen Leistungsbewertungsverfahrens.
- Evaluation des entwickelten Verfahrens.

1.2. Aufbau der Arbeit

Die Arbeit beginnt mit einer Reihe von Vorüberlegungen, dazu zählen unter anderem die Anforderungsanalyse, ein Blick in verwandte Arbeiten, die Charakterisierung des vorliegenden Problems und eine Abschätzung der Komplexität. In Kapitel 3 wird ein erster Blick auf mögliche Lösungswege geworfen, wobei lineare Programmierung, formale Leistungsbewertungsverfahren und evolutionäre Algorithmen vorgestellt und auf Anwendbarkeit untersucht werden. Folgend wird in Kapitel 4 ein evolutionärer Algorithmus unter Zuhilfenahme von formalen Leistungsbewertungsverfahren entwickelt, welcher schließlich in Kapitel 5 evaluiert wird. Den Abschluss der Arbeit bilden ein Fazit und ein Ausblick, über weitere Forschungsmöglichkeiten.

Kapitel 2

Vorüberlegungen

In diesem Kapitel werden Vorüberlegungen zum Problem vorgestellt, die unabhängig von Lösungsverfahren sind. Dazu wird zuerst ein kurzer Überblick über Möglichkeiten der Virtualisierung gegeben, um anschließend eine Anforderungsanalyse des zu lösenden Problems vorzunehmen. Unter Berücksichtigung der Anforderungen wird verwandte Literatur begutachtet und weitere Untersuchung im Rahmen dieser Ausarbeitung begründet. Den Mittelteil des Kapitels bilden Modell-Annahmen und ein formalisiertes Modell, welches den Grundbaustein für die restliche Arbeit bildet. Schließlich endet das Kapitel mit der Vorstellung möglicher Kriterien zur Bewertung einer Platzierung und einer kurzen Einschätzung der Komplexität des Problems.

Wie in der Einleitung bereits genannt, sollen die informationsverarbeitenden Komponenten von CPS virtualisiert werden, um die Vorteile von Virtualisierung nutzen zu können. Das Konzept Virtualisierung geht bis in die 1960er Jahre zurück, als es genutzt wurde um einen IBM Mainframe mehreren Benutzern isoliert zur Verfügung zu stellen [BG73; Gol73]. In den letzten 15 Jahren erlebte die Virtualisierung ganzer Betriebssysteminstanzen durch gestiegene Rechnerleistung und gestiegenem Bedarf an den angebotenen Vorteile eine Renaissance. Im x86-Umfeld sorgten vor allem die im Jahr 2005/2006 veröffentlichten Hardware-Unterstützungen durch AMDs SVM und Intels VT-x Techniken für ein breiteres Interesse an diesem Hilfsmittel. Seitdem ist es auf *commodity*-Hardware möglich Virtualisierung der gleichen Architektur mit sehr geringem zusätzlichen Aufwand zu nutzen, während vorher komplizierte und teils aufwändige Softwarelösungen genutzt werden mussten. Auch müssen seitdem keine Anpassungen mehr an den auszuführenden Betriebssystemen vorgenommen werden.

Durch die Nutzung von virtuellen Maschinen ergeben sich viele Vorteile. Im Gegensatz zu anderen Möglichkeiten mehrere Software-Komponenten auf einer gemeinsamen Ressource auszuführen, sind virtuelle Maschinen so wie auf realer Hardware isoliert. Da die Komponenten unabhängig von der Hardware sind, lassen sich virtuelle Maschinen als normale Dateien behandeln und entsprechend einfach vervielfältigen und mehrmals parallel starten, was die Ver-

waltung erheblich vereinfacht. Mittlerweile gibt es die Möglichkeit eine VM im laufenden Betrieb auf eine andere Hardware zu verschieben, indem der benutzte Speicher mit kopiert wird [Cla+05], es wird also ein *Checkpoint* übertragen. Diese Technik lässt sich dahingehend ausweiten mehrere Checkpoints pro Sekunde zu übertragen, um Hochverfügbarkeit (HA, *high availability*) zu ermöglichen [Cul+08]. Im Fall eines Ausfalls einer realen Maschine springt der aktuelle Checkpoint der Kopie ein und führt die Ausführung fort.

Ein CPS besteht im Allgemeinen aus verteilten Rechensystemen, zuzüglich mechanischen oder elektronischen Komponenten, zur Interaktion mit der physischen Umgebung. Die informationsverarbeitenden Komponenten müssen dabei regelmäßig Echtzeitanforderungen genügen, könnten ansonsten aber jeweils in einer virtuellen Maschine ausgeführt werden. Ein möglicher Aufbau für eine derartige virtualisierte Infrastruktur findet sich in Abbildung 2.1.

Zu sehen sind auf der untersten Ebene Aktuatoren, die mit der physischen Welt interagieren, oder Sensoren, z. B. A/D-Wandler, die Daten der Umwelt sammeln, um sie innerhalb des verteilten Systems zur Verfügung zu stellen. Um Kommunikation innerhalb der virtualisierten Umgebung zu ermöglichen, sind zwei Netzwerke eingeplant, eines zur „klassischen“ Kommunikation innerhalb der verteilten Umgebung, also zwischen den nun virtualisierten CPS-Komponenten und den verbliebenen physikalischen Geräten, und ein zweites Netzwerk zur virtualisierungsspezifischen Kommunikation. Zu letzterer zählen zum Beispiel Migration oder auch HA-Checkpointing von VMs. Die eigentliche VM-Umgebung besteht in der Grafik aus n physikalischen Hosts, die jeweils eine bestimmte Anzahl an CPS Applikationen virtuell ausführen. Zum koordinierten Hardware-Zugriff läuft direkt auf den physikalischen Hosts der sogenannte Hypervisor, der für die Ressourcen-Verteilung zwischen den VMs zuständig ist. Darüber hinaus wird bei Nutzung der Virtualisierungssoftware Xen als ein Gast-System immer eine privilegierte Domäne 0 gestartet, welche zur Interaktion mit dem Hypervisor genutzt wird. Die Domäne 0 ist außerdem für bestimmte Operationen (z. B. Netzwerk- und Festplattenzugriffe) unabdingbar, denn die Treiber der anderen Gastsysteme funktionieren nur in Symbiose mit den hier angesiedelten Gerätetreibern.

In dieser Infrastruktur sind zwei Abhängigkeiten möglich. Einerseits können virtuelle Maschinen von physikalischen Geräten abhängen, z. B. Sensordaten, erwarten also regelmäßig eine Eingabe über das Netzwerk bevor eine Berechnung ausgeführt werden kann. Andererseits können virtuelle Maschinen von Ausgaben anderer virtueller Maschinen abhängen.

Mit dem Wissen um CPS und Virtualisierung lassen sich nun Anforderungen an ein Platzierungsverfahren feststellen.

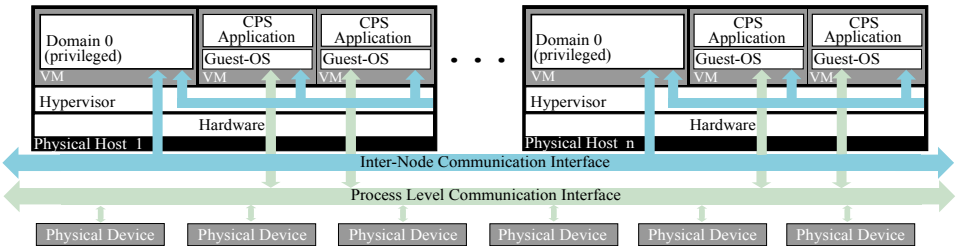

Abbildung 2.1.: Darstellung eines abstrakten virtualisierten CPS nach [JS15].

2.1. Anforderungsanalyse

Die Anforderungen an ein Platzierungsverfahren sind vielfältig. Allem voran soll natürlich eine Platzierung berechnet werden. Um im Fehlerfall, z. B. Ausfall eines Hosts, besonders schnell reagieren zu können, soll es auch möglich sein Ersatz-Platzierungen für mögliche Ausfälle proaktiv zu bestimmen.

Eine berechnete Platzierung muss außerdem weiteren Anforderungen genügen. Besonders hervorzuheben ist hier das Aussprechen von Garantien, um sicherzustellen, dass eine Platzierung die Echtzeitanforderungen der virtualisierten Komponenten erfüllt. Dafür müssen nicht nur eventuelle Datenabhängigkeiten zwischen den virtualisierten Komponenten berücksichtigt werden, sondern auch deren Kommunikationsbedarf. Um die Vorteile bezüglich Fehlertoleranz und Verfügbarkeit von Virtualisierung nutzen zu können, ist darüber hinaus weitere Kommunikationsaufwand erforderlich, der bei einer Platzierung berücksichtigt werden muss.

Über diese grundlegenden Anforderungen hinaus, besteht noch Optimierungspotential, z. B. möchte man vermutlich aus zwei Lösungen die mit möglichst wenig verwendeten Hosts nutzen, wenn sonst alle geforderten Bedingungen gleichermaßen erfüllt werden. Vielleicht möchte ein Benutzer jedoch auch eine Platzierung erhalten, in der die Ende-zu-Ende-Latenz einer jeden Komponente möglichst gering ist. Die Bandbreite der Optimierungskriterien ist groß und nicht selten führt eine Verbesserung eines Kriteriums zu einer Verschlechterung eines anderen Kriteriums. Dieser Umstand muss bei der Lösungsfindung besonders berücksichtigt werden. Die im Rahmen dieser Arbeit berücksichtigten Optimierungskriterien werden in Abschnitt 2.5 erläutert.

Alle Anforderungen im Überblick:

- Effiziente Berechnung einer Platzierung, und auch proaktive Berechnung einer neuen Platzierung für einen möglichen Fehlerfall
- Aussprechen von Garantien für Echtzeitfähigkeit

- Berücksichtigung von Datenabhängigkeiten
- Berücksichtigung von Kommunikationsbedarf
- Berücksichtigung von Aufwänden einer Hochverfügbarkeit
- Bestmögliche Erfüllung weiterer Kriterien unter Berücksichtigung ihrer Gegenläufigkeit

2.2. Verwandte Arbeiten

Das vorgestellte Problem ähnelt in seinen Grundzügen Scheduling-Problemen, die in unterschiedlichen Variationen bereits seit mehreren Jahrzehnten untersucht werden. Insbesondere lassen sich von Aufsätzen, die sich mit Multi-Core- bzw. Many-Core-Prozessoren beschäftigen, möglicherweise Lösungsmethoden oder Ideen übernehmen, denn die Varianten unterscheiden sich vor allem hinsichtlich der Behandlung von Kommunikation. Abhängig von den gewählten Modellierungen können diese Ergebnisse mit unterschiedlichem Aufwand auf das vorliegende Problem übertragen werden. Darüber hinaus ist jedoch die gewünschte Echtzeitfähigkeit ein Kriterium, welches die Übernahme vieler Ergebnisse von Aufsätzen verhindert, weil es in solchen Aufsätzen seltener berücksichtigt wird.

Zhu u. a. präsentierten 2013 eine echtzeitfähige LP-Modellierung, die Task-Zuweisung und den Versand von Nachrichten optimiert, entsprechend Prioritäten zuweisen kann, und dabei Zeitschranken und Ende-zu-Ende-Latenzen minimiert [Zhu+13]. Die LP-Formulierung ist sehr umfangreich, sodass die Autoren für Praxisprobleme vorschlagen das Lösungsverfahren in zwei Schritte zu teilen: Zuerst eine Task-Verteilung berechnen und in einem zweiten Schritt den Versand von Nachrichten optimieren. Dieses zweigeteilte Verfahren ist dann jedoch eine Approximation, wohingegen die LP-Modellierung die optimale Lösung berechnen kann. Eine weitere Herausforderung bei dieser Lösungsvariante ist die Integration mehrerer Optimierungskriterien in eine kombinierte Zielfunktion, welche in diesem Aufsatz nicht angesprochen wird, und auch die Formalisierung weiterer über den Aufsatz hinaus gehender Optimierungskriterien ist nicht trivial.

Einen anderen Ansatz, um echtzeitfähige Systeme auf Multi-Core-Hardware zu verteilen, verfolgen Feinbube u. a. unter Zuhilfenahme eines evolutionären Algorithmus [Fei+15]. Dafür optimieren sie abstrakte Syntax-Bäume, welche eine Berechnungsvorschrift für die Prioritätenzuweisung der Tasks darstellen.

Prinzipiell kann eine Multi-Core-Lösung auch für Multi-Host-Probleme interessant sein, in diesem Aufsatz wird jedoch ein globaler Scheduling-Ansatz verfolgt, der eine Task-Migration erlaubt, abhängig von den Anforderungen kann dies für Host-Scheduling unerwünscht sein. Sie berücksichtigen als mögliche Optimierungskriterien die Migrationsnotwendigkeit von Tasks sowie die maximale Auslastung, und kombinieren diese in einen gemeinsamen skalaren Wert, was eine weitere Herausforderung darstellen kann.

Ebenfalls verwenden Oh und Wu einen multikriteriellen genetischen Algorithmus, um echtzeitfähige System auf Multi-Core-Hardware zu verteilen [OW04]. Im Gegensatz zum vorigen Ansatz optimieren die Autoren keine allgemeine Berechnungsvorschrift für Prioritäten, sondern genau ein gegebenes Tasksystem. Optimiert wird die Anzahl der benötigten Kerne und die maximale Verspätung der Tasks, wobei die Kriterien separat im Algorithmus verarbeitet werden. Zusätzlich zu erwarteten evolutionären Operatoren kommt eine Heuristik zur lokalen Verbesserung von gefundenen Lösung zum Einsatz. Kommunikation zwischen den Tasks wird nicht berücksichtigt.

iPlace von Caglar, Shekhar und Gokhale ist eine Cloud-Middleware, die energie- und ressourcengewahr mithilfe eines neuronalen Netzwerks Platzierungen errechnet. Das neuronale Netzwerk wird benutzt, um die CPU-Auslastung und darauf aufbauend den Energieverbrauch und die Performanz eines Hosts zu schätzen. Das gesamte Verfahren wird als reaktives Verfahren vorgestellt, z. B. reagiert es auf das Problem, wenn ein Host überlastet ist, oder wenn entschieden wird alle VMs zu migrieren, und berechnet in diesen Fällen den besten neuen Host für die betroffenen VMs. Das Verfahren erfüllt „soft real-time application QoS“ [CSG14, S. 49].

Masrur u. a. untersuchten die Platzierung von echtzeitfähigen VMs unter Nutzung von hierarchischen Schedulern [Mas+11]. Sie erlauben mehrere Tasks pro VM, die dort nach der Deadline-Monotonic-Policy ausgeführt werden. Den VMs selbst werden mithilfe der Rate-Monotonic-Policy Abschnitte der Rechenzeit zugeteilt, außerdem wird für jede VM eine optimale Periode berechnet. Eventueller Kommunikationsbedarf wird von den Autoren nicht berücksichtigt und weitere Optimierungskriterien sind nicht vorgesehen.

Keine der vorgestellten Arbeiten erfüllt alle in Abschnitt 2.1 ermittelten Anforderungen vollständig, vor allem hinsichtlich der Berücksichtigung von Kommunikationsbedarf und der Aussprache von Garantien für Echtzeitanforderungen. Wir sehen somit Bedarf den aktuellen Stand der Forschung im Rahmen der vorliegenden Arbeit zu erweitern.

2.3. Annahmen

Neben dem bereits vorgestellten Hypervisor Xen gibt es noch weitere professionelle Virtualisierungs-Lösungen und auch die Breite der nutzbaren Hardware ist groß. Um ein Platzierungsproblem allgemein lösen zu können, muss von den vielen praktischen Möglichkeiten abstrahiert werden, wofür Annahmen festgelegt werden müssen. Die folgenden Annahmen werden für den Rest dieser Arbeit zugrunde gelegt:

- Im Folgenden gehen wir von homogener Hardware aus, also davon, dass alle physikalische Hosts identisch sind, was mit Blick auf Rechenzentren eine realistische Annahme ist.

- Wir gehen davon aus, dass alle Hosts Einkern-Systeme sind, um eine elegante Modellierung und prägnante Notationen zu ermöglichen. Multi-Core-Systeme können in unserem gewählten Modell für viele Fälle simuliert werden, siehe dazu Abschnitt 3.2.1.

- Die Scheduler der Prozessoren arbeiten präemptiv, um eine zeitnahe Bearbeitung von Aufgaben mit höheren Prioritäten zu ermöglichen.

- Bezüglich der Kommunikationskanäle betrachten wir die zur Verfügung stehende Bandbreite als tatsächlich rein für Nutzlast zur Verfügung stehende Bandbreite. Regelmäßige Verringerungen durch Protokoll-Rahmen, Übertragungsfehler, Prüfsummen, etc. werden nicht beachtet.

- Zu Zeiten in denen 1 Gbit-LAN in *commodity*-Hardware weit verbreitet ist, bietet es sich an diese verfügbare Bandbreite als Grundlage zu nutzen. Durch unterschiedliche Faktoren lässt sich regelmäßig nur ca. 50% dieser Bandbreite tatsächlich nutzen, sodass wir von einem 500 Mbit/s bzw. 62.5 MB/s-LAN ausgehen.

- Die Ressourcen-Anforderungen einer VM ergeben sich vor allem durch die durch sie ausgeführten Aufgaben. Auch an dieser Stelle vollziehen wir eine deutliche Abstraktion und gehen davon aus, dass jede VM genau eine Aufgabe periodisch ausführt. Weder parallele Ausführungen noch nebenläufige Programme werden berücksichtigt, vor allem um eine Brücke zu klassischen Scheduling-Algorithmen schlagen zu können. Daraus ergibt sich, dass eine VM direkt die Spezifikationen (z. B. WCET, Periode, ...) der auszuführenden Aufgabe übernehmen kann.

- Die Rechen-Anforderungen einer jeden CPS-Anwendung können darüber hinaus mindestens erfüllt werden, wenn die jeweilige VM alleine von einem realen Host ausgeführt wird. Andernfalls wäre die Berechnung einer Platzierung überflüssig, wenn eine CPS-Anwendung in keiner Konstellation die Anforderungen erfüllen kann.

- Die aktuelle Aufgabe muss vor der nächsten periodischen Aufgabe erledigt sein, es muss also für jede Aufgabe gelten, dass die Zeitbeschränkung kleiner oder gleich der Periode ist. In der klassischen Scheduling-Literatur sind derartige Systeme als *beschränkt* bekannt. Diese Einschränkung vereinfacht die Darstellung von Problem und Lösung, außerdem lassen sich viele Lösungen uneingeschränkt auf unbeschränkte Systeme übertragen [BB08].

- Wir gehen von fixen Prioritäten aus, wobei VMs mit kleinerer Periodendauer eine hohe Priorität erhalten, dies entspricht *Rate-Monotonic-Scheduling*. Diese Wahl der Prioritäten ist im Rahmen von Algorithmen für fixe Prioritäten optimal, d. h. wenn es einen Schedule gibt, der alle Zeitbeschränkungen einhält, dann wird dieser mit RM-Scheduling gefunden. Unsere Wahl folgt damit [JS15].

- Abhängigkeiten schränken wir so ein, dass nur einfache Abhängigkeiten modellierbar sind. Es kann nicht vorkommen, dass eine VM von zwei anderen VMs abhängt, oder eine VM für zwei andere VMs Daten generiert. Eine einfache Kette voneinander abhängiger VMs ist jedoch möglich.

- Innerhalb einer Kette müssen alle abhängigen Berechnungen mindestens mit der gleichen Periode wie die vorige Berechnung ausgeführt werden, ansonsten generiert die erste Berechnung schneller neue Ausgaben als die zweite Berechnung an Eingaben verarbeiten kann.

- Zur Beachtung von Hochverfügbarkeit legen wir eine Variante der von Remus verwendeten Techniken zugrunde [Cul+08]. In Remus wird regelmäßig ein Checkpoint mit den Änderungen seit dem letzten Checkpoint übermittelt. Abbildung 2.2 zeigt den Ablauf einer derartigen Synchronisierung. Die VM wird kurz pausiert und der aktuelle Zustand gespeichert (1), ab dann werden Ausgaben gepuffert. Danach erfolgt die Übertragung der Änderungen an die Kopie (2), währenddessen läuft die Ausführung der VM spekulativ weiter, also ohne, dass dies von Außen ersichtlich ist. Erst wenn die Kopie den Erhalt des Checkpoints bestätigt (3), werden die gepufferten Ausgaben freigegeben (4). Insgesamt ergibt dies die

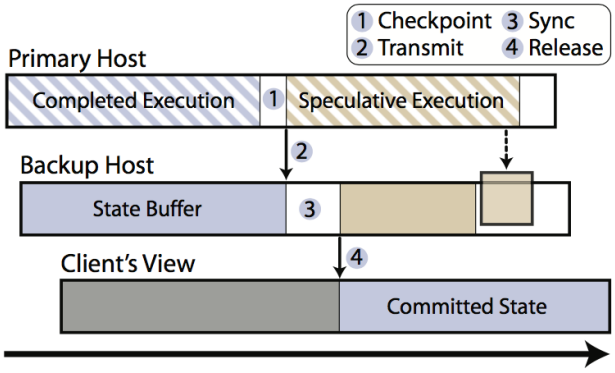

Abbildung 2.2.: Replikation in Remus nach [Cul+08].

Möglichkeit schnelles Checkpointing unter Wahrung von Konsistenzen zu ermöglichen.

Um das Netzwerk so wenig wie möglich mit Checkpointing zu belasten, gehen wir davon aus, dass nur nach einer Berechnung ein Checkpoint übermittelt wird, da sich nur durch Berechnungen der Zustand einer VM bemerkenswert verändert. Um außerdem die zeitkritische Reaktion einer VM nicht zu gefährden, gehen wir davon aus, dass die Ausgabe nicht gepuffert wird, d. h. Schritt (4) in Abbildung 2.2 fällt weg.

- Wir gehen außerdem davon aus, dass es immer einen zusätzlichen Host gibt, der die HA-Kopien beherbergt. Da der Zielhost die kopierten VMs nicht ausführt, benötigt er kaum Rechenzeit, um eine Kopie vorzuhalten [Cul+08]. Wir berücksichtigen deshalb im weiteren Verlauf der Arbeit vor allem den notwendigen Kommunikationsbedarf, um einen Checkpoint zu diesem zusätzlichen Host zu transportieren. Fällt ein Host aus, kann dieser zusätzliche Host die Ausführung übernehmen, danach muss entweder eine neue Platzierung ermittelt werden, die wieder einen freien Host schafft oder es muss ein weiterer Host in das System eingebracht werden, der als neue Notreserve fungiert.

2.4. Modell

Unter Berücksichtigung der vorgestellten Annahmen kann eine Abstraktion für den weiteren Verlauf dieser Arbeit definiert werden.

2.4.1 Definition. Eine **vCPS-Anwendung** (*virtualized CPS-Application*) ist ein Tupel

$$\tau_i = (C_i, T_i, B_i^{\text{in}}, B_i^{\text{out}}, D_i, H_i^{\text{init}}, H_i^{\text{regular}}), \tag{2.1}$$

wobei die Anwendung mit Periode T_i Aufwand zu erledigen hat, welcher die maximale Ausführungszeit C_i hat, dabei B_i^{in} Bits zur Eingabe erwartet werden und B_i^{out} Bits als Ausgabe erstellt werden. Jeder periodische Aufwand muss innerhalb der Zeitbeschränkung D_i abgearbeitet worden sein, wobei $D_i \leq T_i$ gelten muss. Sofern es eine aktuelle Platzierung gibt, wird bei einer Verschiebung außerdem die Migration von H_i^{init} Bits notwendig, welche binnen 5 Sekunden erfolgen muss. Regulär, sprich nach jeder Ausführung der VM, müssen außerdem H_i^{regular} Bits an die Schatten-Kopie übermittelt werden. Es gilt $C_i, T_i, D_i \in \mathbb{Q}^+$ und $B_i^{\text{in}}, B_i^{\text{out}}, H_i^{\text{init}}, H_i^{\text{regular}} \in \mathbb{N}_0^+$. ■

2.4.2 Definition. Ein **vCPS** (*virtualized Cyber-Physical-System*) ist ein Tupel

$$\tau = (\{\tau_1, \ldots, \tau_n\}, \phi) \tag{2.2}$$

von vCPS-Anwendungen τ_i mit $1 \leq i \leq n$ und einer Nachfolgerabbildung ϕ:

$$\phi(\tau_j) = \begin{cases} \tau_k & \text{falls } \tau_k \text{ von } \tau_j \text{ datenabhängig ist} \\ \perp & \text{sonst.} \end{cases} \tag{2.3}$$

Für alle vCPS-Anwendungen τ_f mit $1 \leq f \leq n$ muss $B_f^{\text{in}} = 0$ gelten, wenn es ein τ_e mit $\phi(\tau_e) = \tau_f$ gibt. In diesem Fall ist die Ausgabe von τ_e die Eingabe von τ_f, und da der Kommunikationskanal nur ein Mal belastet werden soll, fordern wir, dass die erneute Eingabe keinem Aufwand entspricht. ■

2.4.3 Bemerkung. Die Einschränkung auf rationale Zahlen für die Spezifikation von CPS-Anwendungen stellt sicher, dass die meisten Verfahren auch tatsächlich angewendet werden können. Manche Verfahren verwenden das kleinste gemeinsame Vielfache von Perioden [BMR90], dieses ist für zwei rationale Zahlen definiert im Gegensatz zu irrationalen Zahlen.

Dem geneigten Leser wird auffallen, dass die Definition eines vCPS der klassischen Definitionen von Task-Systemen gleicht. Dies ist beabsichtigt, und erlaubt einen Rückgriff auf mehrere Jahrzehnte Literatur in diesem Bereich.

Ein vCPS ist ein Tupel, wir werden es jedoch manchmal als Menge benutzen und meinen damit die Menge der vCPS-Anwendungen, z. B. $\forall \tau_j \in \tau$. ■

Abbildung 2.3 zeigt eine schematische vCPS-Anwendung i mit ihren Kennzahlen. Zu sehen ist im oberen Fluss ein Migrations-Event als Startpunkt, dem

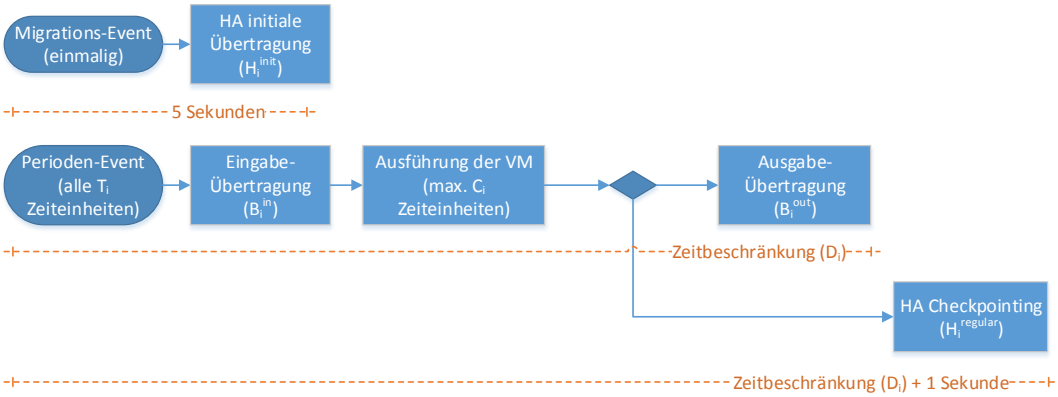

Abbildung 2.3.: Schematische Darstellung einer vCPS-Anwendung.

folgend H_i^{init} Bits innerhalb von 5 Sekunden übertragen werden müssen. Der zweite Fluss zeigt die periodische Ausführung der vCPS-Anwendung. Alle T_i Sekunden wird der Fluss gestartet, folgend müssen zuerst die Eingabedaten B_i^{in} Bits über den Kommunikationskanal übertragen werden, die dann innerhalb von maximal C_i Sekunden verarbeitet werden müssen. Anschließend teilt sich der Fluss und beide verbliebenen Zweige werden parallel durchlaufen. Einerseits müssen die B_i^{out} Ausgabe-Bits bis zum Ablauf der zeitlichen Frist D_i übertragen werden. Andererseits beginnt mit der Freigabe der ausgehenden Pakete das Checkpointing der Hochverfügbarkeit [Cul+08], wobei H_i^{regular} Bits übertragen werden müssen. Für diesen Pfad wird eine Zeitbeschränkung von $D_i + 1$ Sekunden festgelegt.

Für den Fall, dass eine vCPS-Anwendung j von der vCPS-Anwendung i datenabhängig ist, beträgt der Eingabe-Kommunikationsaufwand der folgenden VM 0. Nach der Ausgabeübertragung B_i^{out} würde also die Ausführung der vCPS-Anwendung j mit dem Zeitverbrauch C_j starten. Auch nach dieser Ausführung würde sich der Fluss wieder teilen, und sowohl die Ausgabe-Kommunikation B_j^{out} als auch den Checkpoint H_j^{regular} übermitteln. Die Zeitbeschränkungen für die vCPS-Anwendung j ergeben sich durch Aufsummierung mit denen der vorigen VMs. In diesem Fall betragen die Zeitbeschränkungen also $D_i + D_j$ und $D_i + D_j + 1$.

Für ein vCPS suchen wir im Rahmen dieser Arbeit eine Platzierung, die unter bestimmten Kriterien optimal ist.

2.4.4 Definition. Eine **Platzierung** eines vCPS $\tau = (\{\tau_1, \ldots, \tau_n\}, \phi)$ ist eine disjunkte Partitionierung $P_1 \uplus \cdots \uplus P_k$ der vCPS-Anwendungen $\{\tau_1, \ldots, \tau_n\}$ auf k Hosts. ■

2.5. Qualität einer Platzierung

Die Qualität einer Platzierung lässt sich anhand vieler Kriterien bewerten, die sich in harte und weiche Kriterien unterteilen lassen. Weiche Qualitätskriterien sind Kriterien, die eine Ordnung in die Güte von nützlichen Lösungen bringen. Harte Qualitätskriterien zeichnen sich hingegen dadurch aus, dass bei Unterschreiten eines Schwellenwerts die Platzierung in der Praxis nicht mehr nutzbar ist.

Im Rahmen dieser Arbeit gibt es einige **harte Qualitätskriterien**, die sich aus dem Kontext der vCPS ergeben:

Echtzeitfähigkeit Es geht bei der Arbeit um die Platzierung von echtzeitfähigen CPS, sodass die Einhaltung der Echtzeitbedingungen notwendig für eine brauchbare Lösung ist.

Anzahl der benötigten Hosts nicht überschritten Wenn nur eine beschränkte Menge an Geräten zur Verfügung steht, ist eine Platzierung auch nur dann nützlich, wenn sie mit den zur Verfügung stehenden Geräten auskommt.

Sicherheitspuffer bei Host-Auslastung eingehalten Wenn ein gewünschter freier Puffer an Host-Auslastung auf jedem Host gewünscht ist, ist eine Platzierung nur sinnvoll, wenn dieser Puffer immer frei bleibt.

Darüber hinaus lassen sich **weiche Kriterien** identifizieren, die sich zum Teil als Pendant der harten Kriterien erweisen:

Anzahl der benötigten Hosts Unabhängig von der Nützlichkeit einer Platzierung unterhalb einer Schwelle, möchte man natürlich möglichst wenig Geräte in Anspruch nehmen. Freie Geräte bedeuten gesparte Kosten.

Ende-zu-Ende-Latenz Ein weiteres Qualitätskriterium kann es sein die Ende-zu-Ende-Latenzen der vCPS-Anwendungen zu minimieren. Dies ist das Pendant zum harten Kriterium der Echtzeitfähigkeit. Dort müssen bestimmte Grenzen der Ende-zu-Ende-Latenz eingehalten werden, um überhaupt echtzeitfähig zu sein, unterhalb dieser Schwelle kann man dann noch an möglichst geringen Latenzen interessiert sein.

Umfang notwendiger Migrationen Falls es eine Ausgangs-Platzierung gibt, möchte man den Umfang der notwendigen Migrationen vermutlich so gering wie möglich halten, da sie eine beachtenswerte Last auf das Netzwerk ausüben.

Gerade bei den weichen Kriterien fällt auf, dass sie sich gegenläufig verhalten können. Die Minimierung von der Anzahl der benötigten Hosts kann es zum Beispiel notwendig machen mehr Migrationen auszuführen. Für einen Algorithmus gibt es somit innerhalb des Optimierungsproblems nicht eine beste Lösung, sondern es kann mehrere gleichwertige Lösungen geben, sofern keine weiteren Annahmen getroffen werden. Diesem Problem widmen wir uns allgemein im Abschnitt 3.3.4.2 und besprechen passende Algorithmen im Abschnitt 4.3.6.

Die Formalisierung der Kriterien und die Berechnung der notwendigen Kennzahlen wird in Abschnitt 4.3.5 beschrieben.

2.6. Komplexität

Um die Problemschwierigkeit einschätzen zu können, lohnt es sich die Komplexität eines Optimierungsproblems zu untersuchen. Abstrahiert suchen wir für ein vCPS eine Partition der enthaltenen vCPS-Anwendungen mit bestimmten Eigenschaften, z. B. die Einhaltung der Echtzeitbedingungen. Im schlechtesten Fall müssen wir also davon ausgehen, dass wir jede mögliche Partition bewerten und mit den restlichen Partitionen vergleichen müssen.

2.6.1 Definition. [GKP94] Sei $V = \{v_1, \ldots, v_n\}$ eine Menge, die in k disjunkte nicht-leere ununterscheidbare Partitionen zerlegt werden soll, dann bezeichnet die Stirling-Zahl zweiter Art $\left\{ {n \atop k} \right\}$ die Anzahl der Möglichkeiten die Menge V auf k Partitionen aufzuteilen. ■

2.6.2 Theorem. *[Tem93] Für die Stirling-Zahl zweiter Art* $\left\{ {n \atop k} \right\}$ *gilt die folgende geschlossene Form:*

$$\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n \tag{2.4}$$

Wenn wir als Menge V die Menge der vCPS-Anwendungen eines Systems einsetzen, gibt uns diese Formel Aufschluss über die maximale Anzahl an Partitionen, die wir zu bewerten haben. Im Vorfeld wissen wir jedoch nicht, aus wie vielen Partitionen die optimale Lösung besteht. Wir müssen also für alle $1 \leq k \leq n$ die Partitionen betrachten.

2.6.3 Definition. [Wil06, S. 23] Sei $V = \{v_1, \ldots, v_n\}$ eine Menge, dann bezeichnet die Bellsche Zahl

$$B_n = \sum_{k=0}^{n} \left\{ {n \atop k} \right\} = \sum_{k=0}^{n-1} \binom{n-1}{k} B_k \tag{2.5}$$

die Gesamtanzahl aller Möglichkeiten die Menge V zu partitionieren. ■

Nun bleibt noch die Bellsche Zahl abzuschätzen. Eine untere Grenze für Stirling-Zahlen zweiter Art wurde von Rennie und Dobson gefunden.

2.6.4 Theorem. *[RD69] Für die Stirling-Zahl zweiter Art* $\left\{ {n \atop k} \right\}$ *mit* $n \geq 2$ *und* $1 \leq k \leq n-1$ *gilt:*

$$\left\{ {n \atop k} \right\} \geq \frac{1}{2}(k^2 + k + 2)k^{n-k-1} - 1 \tag{2.6}$$

Damit ergibt sich für eine Bellsche Zahl eine exponentielle Größe und somit für das Optimierungsproblem eine Partition zu finden ein exponentiell großer Suchraum. Auch für kleine vCPS ist eine Lösung durch Aufzählen schon nicht mehr praktikabel, z. B. für ein moderates System mit 15 vCPS-Anwendungen beträgt $B_{15} = 1\,382\,958\,545$. Wenn man zur Bewertung jeder Partition eine Sekunde veranschlagt, braucht man zur Bewertung des gesamten Suchraums bereits 43 Jahre.

Mit der abgeschlossenen Charakterisierung des Problems widmet sich das nächste Kapitel möglichen Lösungswegen.

Kapitel 3

Lösungsansätze

In diesem Kapitel werden mögliche Lösungsansätze für das vorgestellte Platzierungsproblem aufgezeigt. Ein Optimierungsverfahren, welches in der Praxis regelmäßig gut anwendbar ist, ist lineare Optimierung, weshalb der erste Teil des Kapitels dieses Verfahren vorstellt und auf das vorliegende Problem anwendet. Wie sich im Verlauf des Abschnitts zeigen wird, ist eine derartige Modellierung für das Platzierungsproblem jedoch exponentiell lang, und die Berechnung einer Lösung für ein Modell der Größe $n = 30$ liegt bereits im Bereich von Tagen.

Eine hauptsächliche Anforderung an unser Problem ist die Gewährleistung von Aussprache für Echtzeitanforderungen. Derartige Garantien werden z. B. von formalen Leistungsbewertungsverfahren ausgesprochen, sodass sich der Mittelteil des Kapitels diesen Lösungsverfahren widmet. Ein Problem ist, dass ein formales Leistungsbewertungsverfahren zwar Eigenschaften einer Platzierung berechnen kann, allerdings nicht die Platzierung an sich. In dieser Arbeit wird deshalb die Kombination einer formalen Leistungsbewertung mit einem evolutionären Algorithmus (EA) untersucht.

Ein evolutionärer Algorithmus ist ein Approximationsverfahren, das auf der Bewertung generierter Lösungen basiert. An dieser Stelle wollen wir ein formales Leistungsbewertungsverfahren integrieren, um die Vorteile beider Lösungsverfahren zu kombinieren. Die für die Umsetzung dieser Idee notwendigen Grundlagen, bezüglich evolutionärer Algorithmen, werden im letzten Abschnitt dieses Kapitels gelegt. Die eigentliche Entwicklung und Beschreibung der Implementierung folgt dann in Kapitel 4.

Da ein EA nur eine approximative Lösung berechnet, ist es interessant zu evaluieren wie nah der EA einer optimalen Lösung kommt. In Kapitel 5 erfolgt deshalb unter anderem eine Evaluierung des EA durch kleine Modelle, die mithilfe der linearen Optimierung noch optimal gelöst werden können.

3.1. Lineare Optimierung

Bei der linearen Optimierung geht es um die Optimierung einer linearen Zielfunktion unter linearen Nebenbedingungen, z. B.:

$$\begin{aligned} &\text{maximize} && 3x_1 + 5x_2 \\ &\text{subject to} && 2x_1 \leq 5 \\ & && 0.5x_2 \leq 20 \end{aligned} \tag{3.1}$$

Eine derartige Menge von Gleichungen wird zusammen als lineares Programm (kurz LP) bezeichnet. In einer allgemeinen Form können die Koeffizienten der Optimierungsfunktion zu einem Vektor c zusammengefasst werden, alle Variablen x_i zu einem Vektor x, die rechten Seiten der Nebenbedingungen zu einem Vektor b und die Koeffizienten auf den linken Seiten zu einer Matrix A. Damit ergibt sich z. B. die folgende allgemeine Definition.

3.1.1 Definition. Sei $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ und die Matrix $A \in \mathbb{R}^{m,n}$, dann ist ein **lineares Programm** (LP) das Problem den Vektor $x \in \mathbb{R}^n$ zu finden, der die Zielfunktion $c^T x$ minimiert und dabei die Nebenbedingung $Ax \leq b$ einhält.

Kompakt lässt sich dies schreiben als:

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to} && Ax \leq b \\ & && x \in \mathbb{R}^n \end{aligned} \tag{3.2}$$

■

3.1.2 Bemerkung. In der Literatur gibt es keine eindeutige allgemeine Form eines linearen Programms. Insbesondere lassen sich durch Multiplikation mit (-1) Minimierungs- in Maximierungs-Probleme umwandeln, und die Vergleichszeichen der Nebenbedingungen lassen sich mit dem gleichen Trick zwischen $\leq$ und $\geq$ wechseln. Auch beschränkten sich manche Autoren in der Standardform auf $=$ als Vergleichsoperator bei den Nebenbedingungen, und Nebenbedingungen mit anderen Relationen werden durch Einführung neuer Variablen umgewandelt. ■

Die Variablen können auch auf engere Wertebereiche eingeschränkt sein. Abhängig von diesen Einschränkungen werden die Programme dann auch als ILP (*integer linear program*) bezeichnet, wenn alle Variablen nur ganzzahlige Werte annehmen dürfen, oder als BILP (*binary integer linear programm*), wenn alle Variablen nur die Werte 0 und 1 annehmen können, oder auch als MILP (*mixed integer linear programm*), wenn nur manche Variablen ganzzahligen

Einschränkungen unterliegen. Da gängige Solver, also Software, welche LPs lösen, mit allen Variablentypen gleichermaßen umgehen können, bezeichnet man alle Formulierungen regelmäßig auch einfach nur als LP.

Lineare Programme lassen sich mithilfe der Ellipsoid-Methode in polynomieller Laufzeit, abhängig von der Anzahl der Variablen, der Anzahl der Nebenbedingungen und der Größe der Koeffizienten, lösen. Durch Anwendung von Separation, lässt sich die Komplexität für manche Fälle sogar noch verbessern: Existiert ein Algorithmus, der in polynomieller Zeit entscheiden kann, welche Nebenbedingung, durch eine aktuelle Zwischenlösung verletzt wird, kann die Ellipsoid-Methode nur mit einer Teilmenge der Nebenbedingungen gestartet werden, um diese Menge nach jeder Iteration mithilfe des Separations-Algorithmus um eine eventuell verletzte Nebenbedingung zu erweitern. Die Laufzeit der Ellipsoid-Methode hängt dann polynomiell nur noch von der Anzahl der Variablen und der Größe der Koeffizienten ab [GLS88, S. 157ff], was insbesondere für LPs mit exponentiell vielen Nebenbedingungen relevant ist.

Das Ellipsoid-Verfahren ist in der Praxis jedoch merklich langsamer als der wohlbekannte Simplex-Algorithmus, weshalb dieser bei praktischen Problemen größtenteils zum Einsatz kommt. Der Simplex-Algorithmus hat dafür im schlechtesten Fall jedoch eine exponentielle Laufzeit. [GLS88, S. 64f]

Für die praktische Anwendung existieren ausgereifte Solver. Zu nennen sind vor allem CPLEX Optimizer[1], ein kommerzieller Solver von IBM, der für akademische Zwecke auch frei verfügbar ist, außerdem GLPK (*GNU Linear Programming Kit*)[2], ein freier Solver von der Free Software Foundation. Im Verlauf dieser Arbeit nutzen wir CPLEX.

Der Solver CPLEX erwartet als Dateiformat z. B. das CPLEX LP Format. Ein beispielhaftes LP im CPLEX LP Format ist in Listing 3.1 zu sehen. Ein CPLEX LP besteht aus mehreren Bereichen. Zuerst die zu optimierende Zielfunktion, welche entweder minimiert oder maximiert werden kann, außerdem ein Bereich mit Nebenbedingungen, der z. B. mit dem Schlüsselwort *subject to* eingeleitet werden kann. Alle linearen Funktionen können mit Bezeichnern (und einem Doppelpunkt) voran zur einfachen Wiedererkennung benannt werden und auch Kommentarzeilen können beginnend mit einem Backslash eingefügt werden. Abschließend kann noch der Wertebereich der Variablen spezifiziert werden. Das vollständige Programm endet mit dem Schlüsselwort *End*.

Insgesamt ist die lineare Optimierung eine wichtige Standard-Methode für Optimierung in der Praxis, weil sich viele Probleme ohne Programmierkenntnisse in linearer Form modellieren lassen und anschließen von robusten Solvern

[1] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/
[2] https://www.gnu.org/software/glpk/

```
\ Beispiel-LP
Maximize
  value: 5 x1 + 2 x2 - 500 bin1
Subject To
   identifier1 : x1 - 5x2 <= 30
Integer
  x1
  x2
Binary
  bin1
End
```

Listing 3.1: Beispiel LP

lösen lassen. Die Überführung eines Problems in ein lineares Programm stellt im Allgemeinen jedoch eine nicht-triviale Aufgabe dar.

3.1.1. Mögliche Modellierungen

LP-Formulierungen für Scheduling-Probleme basieren regelmäßig auf einem Partitions-basierten Ansatz. Es werden binäre Indikator-Variablen $X_{i,j}$ genutzt, die *wahr* sind, wenn Aufgabe i auf Ressource j ausgeführt werden soll, und sonst *falsch*. Außerdem müssen Nebenbedingungen eingeführt werden, um sicherzustellen, dass nicht nur die Indikatorvariablen sinnvoll belegt werden, sondern dass das Ergebnis auch eine gültige Platzierung ist.

Zhu u. a. veröffentlichten im Jahr 2012 einen umfangreichen allgemein gehaltenen Aufsatz, der zeigt wie nicht nur Tasks, sondern auch Nachrichten mithilfe von Variablen und linearen Gleichungen modelliert werden können [Zhu+13]. Darüber hinaus erlaubt das LP die Ermittlung von Ende-zu-Ende-Latenzen und von Prioritäten für Tasks, und die Berücksichtigung von Zeitbegrenzungen. Auf unser Problem bezogen, ergeben sich für die Nutzung dieser Formulierung jedoch bestimmte Herausforderungen.

Vor allem die Größe der erzeugten LPs steht einer praktischen Anwendung im Weg, dies stellen die Autoren selbst fest: „However, the complexity is typically too high for the sizes of industrial applications because of the large number of integer [...] and binary [...] variables. [...] we propose a faster approximated two-step synthesis method [...]“ [Zhu+13, S. 21]. Sie schlagen als mögliche Lösung also eine Aufteilung des LPs in zwei Abschnitte vor. Zuerst sollen die Tasks verteilt werden und in einem zweiten Schritt soll der Versand von Nachrichten optimiert werden. Gegen diese Idee ist prinzipiell nichts einzuwenden, da jedoch im weiteren Verlauf dieser Arbeit die LP-Lösung als

optimale Vergleichslösung zur Evaluation des EA genutzt werden soll, ist eine Approximation nicht hilfreich.

Darüber hinaus beschränken die Autoren die Größe von Nachrichten auf CAN-typische 64 Bits Nutzlast und schlagen Fragmentation als mögliche Lösung vor. Im Kontext von virtuellen Maschinen, in dem innerhalb der Ausführung eines Checkpointing-Vorgangs der gesamte Speicher einer VM übertragen werden muss, erscheint das als gravierende Einschränkung. Eine Berücksichtigung von Fragmentierung wird im Aufsatz explizit ausgelassen, sodass man diese Problematik erst lösen müsste. Hinsichtlich der sowieso schon komplexen LP-Formulierung, eine weitere Herausforderung.

Insgesamt ergibt sich hier weiteres Forschungs-Potential, was an dieser Stelle jedoch zurückgestellt wird. Stattdessen wird im Rahmen dieser Arbeit auf eine einfachere LP-Modellierung zurückgegriffen. Diese Modellierung konzentriert sich auf die Verteilung von vCPS-Anwendungen auf Hosts ohne Berücksichtigung von Kommunikation und Abhängigkeiten, sodass während der Evaluation zumindest die Lösungen des EA für eingeschränkte Modelle mit einer optimalen Lösung verglichen werden kann.

3.1.2. Charakterisierung von Rate-Monotonic-Scheduling

Um zu einer LP-Formulierung zu gelangen, müssen wir das Rate-Monotonic-Scheduling genauer charakterisieren, denn diese Scheduling-Strategie soll beim Platzierungsproblem verwendet werden. Liu und Layland untersuchten schon 1973 RM-Scheduling und darauf aufbauend entwarfen Lehoczky, Sha und Ding eine Bedingung, die genau dann erfüllt ist, wenn ein Tasksystem schedulbar ist [LSD89]. Aus dieser Bedingung konstruierten Baruah und Bini eine LP-Formulierung [BB08]. Im aktuellen Abschnitt übertragen wir die Ergebnisse zuerst auf unser Platzierungsproblem, um schließlich die LP-Formulierung vorzustellen.[3]

3.1.3 Definition. Sei $\tau = \{\tau_1, \ldots, \tau_n\}$ ein vCPS, o. B. d. A. gehen wir davon aus, dass die vCPS-Anwendungen nach Priorität nummeriert sind (τ_1 hat

[3] In [LL73] wird auch der wohlbekannte Begriff *kritischer Moment* für den Zeitpunkt des gleichzeitigen Auslösens aller Aufgaben eingeführt, welches der *Worst-Case* für ein Tasksystem ist. Da in der vorliegenden Arbeit Phasen nicht eingeführt wurden und somit alle Aufgaben immer gleichzeitig ausgelöst werden, wird auf die Definition dieses Begriffs verzichtet.

höchste Priorität), dann ist die **kumulierte Auslastung** (*Workload*) für eine Zeitspanne $[0, t]$ und die vCPS-Anwendung i mit $1 \leq i \leq n$:

$$W_i(t) = C_i + \sum_{\substack{j=1 \\ \tau_j \in P(\tau_i)}}^{i-1} C_j \cdot \left\lceil \frac{t}{T_j} \right\rceil \tag{3.3}$$

$P(\tau_i)$ sei dabei die Partition einer Platzierung, die τ_i und alle auf dem gleichen Host platzierten vCPS-Anwendungen enthält. ■

Die kumulierte Auslastung ist also eine Kennzahl dafür, wie stark eine vCPS-Anwendung inklusive der vCPS-Anwendungen mit höherer Priorität auf dem gleichen Host diesen in einer Zeitspanne $[0, t]$ auslastet. Dies lässt sich anwenden, um heraus zu finden, ob eine Partition platzierbar ist.

3.1.4 Theorem. *Eine Partition P_i einer Platzierung $P_1 \cup \cdots \cup P_l$ mit $1 \leq i \leq l$ ist genau dann gültig, wenn es für alle $\tau_k \in P_i$ einen Zeitpunkt $t_k < D_k$ gibt, sodass*

$$W_k(t_k) = t_k \tag{3.4}$$

Beweis. Die vCPS-Anwendung τ_k beendet genau dann erfolgreich seine Berechnung im Zeitpunkt t_k, wenn zu diesem Zeitpunkt die Aufwände aller vCPS-Anwendungen mit höherer Priorität auf dem gleichen Host und der Aufwand der vCPS-Anwendung C_k abgearbeitet wurden. Dies entspricht gerade $W_k(t_k)$. Gilt darüber hinaus $t_k < D_k$, dann wurde die Zeitbeschränkung der vCPS-Anwendung eingehalten.

Wenn dies für alle vCPS-Anwendungen der Fall ist, ist die gesamte Platzierung gültig. Nach [LL73] reicht es die Einhaltung der ersten Zeitbeschränkung aller vCPS-Anwendungen zu prüfen, um die Gültigkeit festzustellen. □

Um dieses Theorem in eine LP-Formulierung zu überführen, muss das Problem gelöst werden, dass die zu testenden Zeitpunkte einer großen Teilmenge des reellen Zahlenraums entsprechen. Eine interessante Beobachtung ist, dass die Funktionen W_k Treppenfunktionen sind, es gibt also Zeitabschnitte, die es sich nicht zu betrachten lohnt. Im Detail sind nur die Vielfachen der Perioden interessant, da an diesen Punkten lokale Minima sind, die linksseitig stetig und rechtsseitig auf einen höheren Wert springen. Darüber hinaus sind für eine vCPS-Anwendung nur die Zeitpunkte $t_k < D_k$ relevant, denn nach D_k würde die Echtzeitbedingung verletzt. [LSD89; BB08]

Mit dieser Beobachtung ergibt sich eine sogenannte Test-Menge an Zeitpunkten für die vCPS-Anwendung i:

$$AS(i) = \{D_i\} \cup \{t \mid t = l \cdot T_k, \tau_k \in P(\tau_i), l \in \mathbb{N}_0^+ \mid t < D_i\} \tag{3.5}$$

Diese Test-Menge enthält nun nur noch diskret viele Zeitpunkte, die es zu testen gilt, nämlich die Vielfachen aller Perioden der vCPS-Anwendungen in der gleichen Partition wie τ_i. Im Rahmen einer LP-Formulierung, in der die Verteilung von vCPS-Anwendungen auf Hosts noch nicht feststeht, und somit sowieso eine hohe Anzahl an Möglichkeiten getestet werden muss, ist jedoch das Interesse groß diese Test-Menge so klein wie möglich zu halten.

Im Jahr 2004 stellten Bini und Buttazzo eine neue Test-Menge $\mathcal{P}_{i-1}(D_i)$ für (auf unser Problem übertragen) eine vCPS-Anwendung i mit folgender Rekursionsvorschrift vor:

$$\begin{cases} \mathcal{P}_0(t) = \{t\} \\ \mathcal{P}_i(t) = \mathcal{P}_{i-1}\left(\left\lfloor \frac{t}{T_i} \right\rfloor T_i\right) \cup \mathcal{P}_{i-1}(t) \end{cases} \tag{3.6}$$

Die Kardinalität einer Menge $\mathcal{P}_i$ beträgt im schlechtesten Fall nur noch 2^i, was sich aus dem Aufbau der Definition ergibt. Diese Test-Menge ist gegenüber dem einfachen Ansatz deutlich kleiner. Der Beweis über die Korrektheit der kleineren Test-Menge wird hier aus Platzgründen ausgelassen, stattdessen verweisen wir auf [BB04, S. 4-6].

3.1.3. LP-Formulierung

Mit der bisher vorgestellten Charakterisierung lässt sich nach [BB08] eine LP-Formulierung angeben, die auf der Grundidee basiert eine Partitionierung einer Platzierung zu ermitteln, unter der Nebenbedingung, dass die Platzierung gültig ist. Eine gültige Platzierung erfüllt die Echtzeitbedingungen des vCPS und weist jeder vCPS-Anwendung genau einem Host zu.

Um eine Partitionierung zu repräsentieren werden binäre Variablen der folgenden Art eingeführt:

$$X_{i,j} = \begin{cases} 1 & \text{falls vCPS-Anwendung } i \text{ auf Host } j \text{ ausgeführt werden soll} \\ 0 & \text{sonst} \end{cases} \tag{3.7}$$

Da wir voraussetzen, dass jede vCPS-Anwendung mindestens alleine auf einem Host ausführbar ist, benötigt ein vCPS mit n vCPS-Anwendungen höchstens n Hosts. Die Kardinalität der Menge der binären Variablen beschränkt sich somit auf n^2.

Mithilfe der binären Variablen sind prinzipiell auch ungültige Modellierungen möglich sind. Das muss mithilfe von Nebenbedingungen unterbunden wer-

den, um sicherzustellen, dass jede vCPS-Anwendung genau einer Partition angehört:

$$\sum_{j=1}^{n} X_{i,j} = 1 \quad \forall 1 \leq i \leq n \tag{3.8}$$

3.1.3.1. Echtzeitbedingung

Die oben vorgestellte Gültigkeitsbedingung des RM-Scheduling muss für jede vCPS-Anwendung i mit $1 \leq i \leq n$ erfüllt sein. Die folgende LP-Bedingung[4] muss also für *ein* $t_0 \in \mathcal{P}_{i-1}(D_i)$ erfüllt sein:

$$t_0 \geq X_{i,j} \left(C_i + \sum_{k=1}^{i-1} X_{k,j} \left\lceil \frac{t_0}{T_k} \right\rceil \cdot C_k \right) \tag{3.9}$$

Damit die Formel in einem linearen Programm angegeben werden kann, muss noch eine Umformung vorgenommen werden, die dafür sorgt, dass die Formel linear wird und den Lösungsraum nur einschränkt, wenn $X_{i,j} = 1$ gilt:

$$X_{i,j} \left(C_i + \sum_{k=1}^{i-1} X_{k,j} \left\lceil \frac{t_0}{T_k} \right\rceil \cdot C_k \right) \tag{3.10}$$

$$\geq X_{i,j} C_i + \sum_{k=1}^{i-1} \left(X_{k,j} \left\lceil \frac{t_0}{T_k} \right\rceil C_k - (1 - X_{i,j}) \left\lceil \frac{t_0}{T_k} \right\rceil \cdot C_k \right) \tag{3.11}$$

Vom Prinzip her ist die LP-Formulierung damit vollständig, praktisch bereitet die Formulierung „muss [...] für *ein* $t_0 \in \mathcal{P}_{i-1}(D_i)$ erfüllt sein“ jedoch Probleme. Innerhalb einer LP-Formulierung müssen standardmäßig *alle* Nebenbedingungen erfüllt sein. Bei den vorgestellten Nebenbedingungen ist eine Lösung jedoch regelmäßig gültig, wenn nur wenige der Nebenbedingungen erfüllt sind.

Eine mögliche Lösung findet sich im Anhang von [BB08]. Die Grundidee liegt darin weitere binäre Variablen in der LP-Formulierung einzuführen und diese so den betroffenen Nebenbedingungen hinzuzufügen, dass egal welche Belegung für diese binären Variablen gewählt wird, alle bis auf eine Nebenbedingung automatisch erfüllt sind.

[4] Diese Formel mit den Nummern 11 und 12 in [BB08] enthält in dem Aufsatz fälschlicherweise $\leq$ statt $\geq$. Dass es sich dabei um einen Tippfehler handelt, haben beide Autoren bestätigt. Dass entgegen der Charakterisierung nicht einfach nur $=$ genutzt wird, ist vermutlich dem Gedanken geschuldet dem Solver etwas mehr Spielraum bei der Lösungsfindung zu gewähren.

Darüber hinaus hat sich herausgestellt, dass der innere Term der Formel geklammert werden muss, ansonsten werden manchmal gültige Lösungen ausgeschlossen.

Angenommen die folgenden Nebenbedingungen existieren im LP, wobei es ausreichen soll, wenn eine erfüllt ist:

$$\begin{aligned} A_1X &\leq b_1 \\ A_2X &\leq b_2 \\ A_3X &\leq b_3 \\ A_4X &\leq b_4 \end{aligned} \tag{3.12}$$

Dann werden zwei binäre Variablen Y_1, Y_2 und eine möglichst große ganzzahlige Zahl Z wie folgt hinzugefügt:

$$\begin{aligned} A_1X &\leq b_1 + ((1-Y_1)+(1-Y_2))Z \\ A_2X &\leq b_2 + (Y_1+(1-Y_2))Z \\ A_3X &\leq b_3 + ((1-Y_1)+Y_2)Z \\ A_4X &\leq b_4 + (Y_1+Y_2)Z \end{aligned} \tag{3.13}$$

Wir betrachten beispielhaft die Belegung $Y_1 = 1, Y_2 = 1$. Bei der ersten Nebenbedingung entsteht $0 \cdot Z$, sodass diese Nebenbedingung ihre ursprüngliche Aussagekraft erhält. Bei den anderen Nebenbedingungen steht auf der rechten Seite durch Z eine große Summe, sodass die restlichen Nebenbedingungen der Form $A_iX \leq b_i + \mathcal{O}(Z)$ immer erfüllt sind.

Im Allgemeinen werden für x Nebenbedingungen, von denen nur eine erfüllt sein muss, $\lceil \log_2 x \rceil$ weitere Binärvariablen Y_l benötigt. Damit diese Variante funktioniert müssen allerdings alle $2^{\lceil \log_2 x \rceil}$ Kombinationen von Y_l und $(1-Y_l)$ in der LP-Formulierung vorhanden sein, ansonsten könnte der Solver eine Belegung der Y_l-Variablen festlegen, sodass alle Nebenbedingungen automatisch erfüllt sind. Für den Fall, dass $x \neq 2^{\lceil \log_2 x \rceil}$ müssen also weitere Nebenbedingungen hinzugefügt werden, um alle Kombinationen der Y_l-Variablen abzudecken. Zur Lösung kann eine Nebenbedingung mit unterschiedlichen Y_l-Kombinationen mehrmals hinzugefügt werden.

Bei der Nutzung von GLPK kommt noch eine weiteres Praxisproblem hinzu: Auf der rechten Seite von Nebenbedingungen dürfen keine Variablen stehen. Auch wird ein Ausklammern von Z nicht unterstützt. Die Terme müssen also zuerst ausmultipliziert werden, verbliebene Terme, die eine binäre Variable enthalten, müssen danach durch Addition/Subtraktion auf die linke Seite verschoben werden.

3.1.3.2. Optimierungsfunktion und weitere Kriterien

Durch die Nebenbedingungen wird sichergestellt, dass die gefundene Lösung eine gültige Partitionierung ist, ohne Zielfunktion findet jedoch keine Optimie-

rung statt. Sowohl für die Zielfunktion, als auch für weitere Nebenbedingungen, eignen sich die in Abschnitt 2.5 vorgestellten Kriterien. Diese lassen sich wie folgt innerhalb eines LPs formulieren:

Anzahl der benötigten Hosts Als Zielfunktion kann folgende Formel verwendet werden:

$$\text{minimize} \sum_{i=1}^{n} \sum_{j=1}^{n} j \cdot X_{i,j} \tag{3.14}$$

Es werden möglichst viele vCPS-Anwendungen auf Host 1 platziert, danach so viele wie möglich auf Host 2, ...

Anzahl der benötigten Hosts nicht überschritten Möchte man die Anzahl der benötigten Hosts als hartes Kriterium beschränken, kann die Anzahl der binären Variablen $X_{i,j}$ reduziert werden.

Echtzeitfähigkeit Da die Echtzeitbedingungen bereits durch die Nebenbedingungen abgedeckt sind, benötigt dieses Kriterium keine Optimierungsfunktion.

Ende-zu-Ende-Latenz In der vorliegenden LP-Formulierung wurden Nachrichten ausgelassen, sodass sich die Ende-zu-Ende-Latenz nur durch die Rechenzeiten ergibt. Eine mögliche Modellierung im Rahmen der bisherigen LP-Formulierung kann so aussehen, dass pro vCPS-Anwendung i für alle $t_0 \in \mathcal{P}_{i-1}(D_i)$ weitere binäre Variablen $E_i^{t_0}$ eingeführt werden und die Nebenbedingungen zum RM-Scheduling wie folgt verändert werden:

$$t_0 \geq Z_s(1 - E_i^{t_0}) + X_{i,j} \left(C_i + \sum_{k=1}^{i-1} X_{k,j} \left\lceil \frac{t_0}{T_k} \right\rceil C_k \right) \tag{3.15}$$

wobei $Z_s = \left\lceil \frac{Z}{100} \right\rceil$ gesetzt wird.

Da von den RM-Scheduling-Nebenbedingungen welche erfüllt werden müssen, um das LP überhaupt zu lösen, werden zwangsläufig einige der $E_i^{t_0}$-Variablen auf 1 gesetzt werden, ansonsten sorgt bereits $t_0 \geq Z_s(1-0)$ für eine Verletzung der Nebenbedingung.

Eine gesetzte Variable lässt nun auf das Zeitfenster schließen, in dem die Berechnung von vCPS-Anwendung i abgeschlossen wird. Da wir die Testmenge $\mathcal{P}_{i-1}(D_i)$ im vorigen Abschnitt von allen Zeitpunkten befreit haben, die kein lokales Minimum sind, können wir nicht mehr feststellen, ob die Berechnung erst zum Zeitpunkt t_0 abgeschlossen wird oder bereits früher in diesem Zeitabschnitt. Nichtsdestotrotz berechnet die

LP-Formulierung so die optimale Obergrenze für den Abschluss der Berechnung in der vorliegenden Formulierung.

Von den möglichen Zeitfenstern, in denen die Berechnung einer vCPS-Anwendung abschließen kann, ist jedoch nur eines relevant und zusätzlich muss man verhindern, dass der Solver alle Variablen $E_i^{t_0}$ einfach auf 1 setzt. Um dies sicherzustellen, müssen für jede vCPS-Anwendung i weitere Nebenbedingungen der folgenden Art hinzugefügt werden:

$$\sum_{t_0 \in \mathcal{P}_{i-1}(D_i)} E_i^{t_0} = 1 \tag{3.16}$$

Für die Optimierungsfunktion kann also nun mithilfe der Variablen $E_i^{t_0}$ eine genaue obere Schranke für die Fertigstellung einer vCPS-Anwendung ermittelt werden. Um mit diesem Wissen eine Optimierungsfunktion zu bilden, kann z. B. die jeweilige Variable mit ihrem Zeitpunkt multipliziert werden, um sicherzustellen, dass möglichst die binären Variablen der kleinsten Zeitpunkte gewählt werden:

$$\text{minimize} \sum_{i=1}^{n} \sum_{t_0 \in \mathcal{P}_{i-1}(D_i)} E_i^{t_0} \cdot t_0 \tag{3.17}$$

Problematisch an dieser Variante ist die Notwendigkeit exponentiell vieler weiterer binärer Variablen und Nebenbedingungen, sogar schon um nur eine genaue obere Schranke für den Zeitpunkt der Fertigstellung zu ermitteln.

Auch zu beachten ist, dass die Größen der Variablen Z und Z_s aufeinander abgestimmt werden müssen. Einerseits muss Z_s so klein gewählt werden, dass eine durch Z erfüllte Gleichung auf jeden Fall erfüllt bleibt. Andererseits muss Z_s groß genug gewählt werden, um in der Lage zu sein die Erfüllung von Gleichungen sicher zu verhindern.

Umfang notwendiger Migration Angenommen $\text{preset}(i)$ entspricht der Host-Nummer der Vorbelegung der vCPS-Anwendung i, dann kann folgende Formulierung als Optimierungsfunktion genutzt werden:

$$\text{minimize} \sum_{i=1}^{n} H_i^{\text{init}} \cdot (1 - X_{i,\text{preset}(i)}) \tag{3.18}$$

Für jede VM wird der Aufwand zur Migration addiert und nur wieder subtrahiert, falls $X_{i,\text{preset}(i)} = 1$ gilt, also wenn die VM in der berechneten Platzierung X immer noch auf dem Host liegt, auf dem sie vorher lag.

Es besteht die Möglichkeit eine neue VM in das System zu bringen, indem $\text{preset}(i) = -1$ gesetzt wird. Damit die Optimierungsfunktion das korrekte Ergebnis liefert, muss in dem Fall die Nebenbedingung $X_{i,\text{preset}(i)} = 0$ erfüllt sein (und die binäre Variable noch eingeführt werden).

Sicherheitspuffer bei Host-Auslastung einhalten Sei $0 \leq c \leq 1$ der benutzbare Faktor der Rechenleistung und $1 - c$ der Sicherheitspuffer-Faktor, dann ist die einfachste Variante alle C_i durch $C_i \cdot \frac{1}{c}$ zu ersetzen. Dadurch belegt jede vCPS-Anwendung zusätzliche Zeit vom Umfang des angefragten Sicherheitspuffers, sodass dieser bei der Berechnung der Platzierung berücksichtigt wird. Die eigentliche Einhaltung des Puffers obliegt dann dem realen Scheduler.

Die Kombination mehrerer Optimierungsfunktionen in eine LP-Formulierung ist tendenziell schwierig und zwar nicht nur im quantitativen Sinne, sondern auch die gewählten Modellierungen der Zielfunktionen verhindern eine einfache Kombination. Vor allem unsere Optimierungsfunktion „Anzahl der benötigten Hosts“ lässt sich ohne weitere Anpassung nicht mit „Umfang notwendiger Migrationen“ kombinieren, denn die vorgestellte Funktion zur Optimierung der Anzahl der benötigten Hosts verringert nicht nur die Anzahl der Hosts, sondern sorgt für eine treppenhafte Befüllung der Hosts (so viel wie möglich auf Host 0, dann soviel wie möglich auf Host 1, ...). Ohne weitere Optimierungsterme ist das nicht relevant, für den Umfang der notwendigen Migrationen macht es jedoch einen Unterschied. Im Allgemeinen ist auch die quantitative Integration verschiedener Optimierungswerte in unterschiedlichen Skalierungen schwierig und macht eine bewusste Gewichtung der unterschiedlichen Terme notwendig.

3.1.3.3. Komplexität der LP-Formulierung

Wie in Kapitel 3 erwähnt lässt sich eine LP-Formulierung zumindest theoretisch in polynomieller Laufzeit, abhängig von der Anzahl der Variablen, der Anzahl der Nebenbedingungen und der Größe der Koeffizienten lösen, weshalb es sinnvoll ist zumindest kurz die Größe der LP-Formulierung zu untersuchen.

Die Anzahl der Nebenbedingungen liegt im Bereich $\mathcal{O}(2^n)$, was vor allem durch die RM-Charakterisierung bestimmt wird, denn pro vCPS-Anwendung i müssen $|\mathcal{P}_{i-1}(D_i)| = 2^{i-1}$ Nebenbedingungen erstellt werden.

Die Anzahl der Variablen liegt in der grundlegenden Variante noch im Bereich $\mathcal{O}(n^2)$, denn es gibt n^2 viele Variablen $X_{i,j}$ und maximal $\lceil \log_2 2^{i-1} \rceil = i-1$ viele Y_l für alle $1 \leq i \leq n$, um von den Nebenbedingungen nur die Erfüllung *einer* statt *aller* notwendig zu machen. Wird die vorgeschlagene angepasste LP-Formulierung zur Optimierung der Ende-zu-Ende-Latenz genutzt,

vergrößert sich die Komplexität auf $\mathcal{O}(2^n)$, denn pro Nebenbedingung der RM-Charakterisierung wird eine weitere binäre Variable benötigt.

Insgesamt ist nicht zu erwarten das Problem in polynomieller Laufzeit lösen zu können, was den geneigten Leser an dieser Stelle auch nicht überraschen sollte.

Zum Abschluss dieses Abschnitts haben wir also die Möglichkeit das Platzierungsproblem in eingeschränkter Form als lineares Programm zu modellieren und unter der Annahme von RM-Scheduling eine optimale Lösung zu berechnen. Diese optimalen Lösungen werden im weiteren Verlauf dieser Arbeit als Vergleichswert zur (eingeschränkten) Bewertung anderer Verfahren genutzt, siehe dazu auch Kapitel 5.

3.2. Leistungsbewertungsverfahren

Für Echtzeitsysteme sind Garantien gewünscht, deren Einhaltung durch Berechnungen überprüft werden kann. Entsprechende Algorithmen sind für verteilte Systeme bereits untersucht worden, die auch weitere Eigenschaften eines Systems berechnen können. Perathoner, Wandeler und Thiele geben einen guten Überblick, welche Implementierungen es gibt und wie diese zu bewerten sind [PWT06]. Die verfügbaren Methoden lassen sich vor allem mithilfe von zwei Kriterien klassifizieren: holistisch gegenüber modular und formal gegenüber simulationsbasiert.

Die Unterscheidung zwischen holistischen und modularen Verfahren stellt die unterschiedliche Betrachtung des Systems während der Berechnung heraus. Einerseits können die Komponenten modular analysiert werden, um die Ergebnisse dann im System zu propagieren und ein Gesamtergebnis zu berechnen, andererseits kann ein System direkt als Ganzes untersucht werden.

Die Zugehörigkeit zu den formalen oder simulationsbasierten Verfahren hat große Auswirkungen auf das Ergebnis. Während formale Verfahren allgemeine Garantien aussprechen, berechnen simulationsbasierte Verfahren nur den Zeitbedarf der Pfade, die im Rahmen der jeweiligen Simulation durchlaufen wurden. Das Ergebnis hängt also maßgeblich von den Eingaben ab und gibt regelmäßig nur einen Eindruck des notwendigen Aufwands.

Für die vorliegende Arbeit wurden folgende Verfahren genauer begutachtet:

- Klassische Scheduling-Probleme werden schon seit mehreren Jahrzehnten in der Literatur untersucht. Die daraus resultierenden Analysemethoden

wurden zu weiten Teilen in dem Tool MAST (*Modeling and Analysis Suite for Real-Time Applications*)[5] implementiert.

- Eine weitere Möglichkeit ist die Modellierung von Systemen als *timed automata*, also endlichen Automaten, die um reelwertige Uhren erweitert wurden, und mit deren Hilfe die verfügbaren Transitionen gesteuert werden können. Diese Richtung gehört zum Teilbereich *Model-Checking* und eine umfangreiche Implementierung ist unter dem Namen UPPAAL (vermutlich ein Kunstwort aus den beiden beteiligten Universitäten Uppsala und Aalborg)[6] verfügbar. UPPAAL bietet die Möglichkeit eine vollständige Verifikation eines erstellten Modells vorzunehmen, um Garantien über Systemeigenschaften zu erhalten. Außerdem kann eine Simulation einzelner Pfade ausgeführt werden, was ressourcenfreundlicher ist als die vollständige Verifikation des gesamten Modells.

- Die letzte begutachtete Möglichkeit ist die Modellierung von Systemen mithilfe von VCCs (*Variability Characterization Curves*) im Rahmen von RTC (*Real-Time Calculus*) [TCN00; CKT03]. Für diese Variante existiert eine Implementierung als Matlab-Plugin namens *RTC Toolbox*[7], welche die RTC-spezifischen Operatoren zusammen mit weiteren Funktionen für eine modulare Leistungsbewertung mitbringt.

Eine Entscheidung wurde basierend auf den beiden Kriterien Laufzeit und Modellierung getroffen.

Laufzeit Simulationsbasierte Verfahren garantieren keine vollständige Abdeckung aller Ausführungspfade, somit hängt die Ergebnis-Qualität maßgeblich von der Eingabe ab. Die Eingaben zu ermitteln, für die das getestete System den höchsten Aufwandsbedarf hat, ist für große Systeme jedoch praktisch unmöglich [PWT06, S. 10]. Die Simulation vieler oder aller Pfade ist darüber hinaus rechenintensiv. Demgegenüber bietet ein Model-Checking-basierter-Ansatz Garantien für alle Fälle, der Aufwand dafür ist jedoch begründet durch das *State-Explosion*-Problem hoch [PWT06, S. 32]. Beide von UPPAAL unterstützten Ausführungsvarianten benötigen somit potentiell viel Rechenzeit.

Im Rahmen eines evolutionären Algorithmus müssen pro Ausführung mehrere tausend Modelle bewertet werden. Eine wiederholte Instanzi-

[5] http://mast.unican.es/
[6] http://www.it.uu.se/research/group/darts/uppaal/
[7] http://www.mpa.ethz.ch/Rtctoolbox/Overview

ierung von Matlab zur Nutzung der RTC Toolbox erschien unter dem Aspekt nicht erfolgversprechend.

Unter der hohen Rate an notwendigen Evaluierungen leidet natürlich nicht nur die Bewertung durch RTC Toolbox. Für alle vorgeschlagenen Varianten muss zu jeder Evaluation das Modell der Daten im EA in eine vom Leistungsbewertungsverfahren verständliche Modellierung überführt werden, und das Verfahren darauf angewandt werden; siehe dazu auch Abschnitt 4.4.

Modellierung Die Überführung eines vCPS in *timed automata* oder VCCs ist ein nicht-trivialer Prozess.

Während RTC Toolbox die genauen Berechnungen abstrahiert, muss trotzdem ein Verständnis der dahinterliegenden Theorie vorhanden sein, um eine einwandfreie Modellierung zu gewährleisten. Die Erstellung eines Modells für die Ausführung in Matlab kann sich als problematisch erweisen: „To analyze a distributed embedded system with the RTC Toolbox, the user is required to write a MATLAB program that invokes appropriate commands for the creation and analysis of performance networks. For large systems this may result in a considerable modeling effort.“ [PWT06, S. 54]

Die Überführung von Systemen in *timed automata* ist ebenso mit Aufwänden verbunden, da eine automatische Überführung schwierig ist, und sich der Aufwand mit steigender Größe des zu modellierenden Systems erhöht: „The use of predefined templates may help, but the major issue is scalability: with growing system dimensions the modeling effort may increase dramatically.“ [PWT06, S. 55]

Demgegenüber lässt sich ein derartiges Systems (unter Verwendung der korrekten Syntax) in MAST einfacher modellieren. Im Gegensatz zu den anderen Varianten muss hier nämlich keine schwierige Transformation des Modells in ein anderes Modell stattfinden, sondern es genügt das gegebene Modell in der MAST-eigenen Syntax zu beschreiben. Für ein vollständiges Verständnis der angewandten Methoden stellt die dahinterliegenden Theorie allerdings eine „very steep learning curve for non-specialists“[PWT06, S. 55] dar, allein schon aufgrund der Masse der integrierten Aufsätze. Circa 14 Aufsätze werden direkt von den MAST-Entwicklern zitiert, deren Veröffentlichungsdaten sich über 35 Jahre erstrecken [Gut+14]. Unter der Berücksichtigung, dass für das Verständnis eines Aufsatzes regelmäßig weitere Aufsätze unabdingbar sind, kann man das in der Tat als steile Lernkurve bezeichnen.

```
Processing_Resource (
   Type                    => Regular_Processor,
   Name                    => host_0,
   Max_Interrupt_Priority => 32767,
   Min_Interrupt_Priority => 1,
   Worst_ISR_Switch        => 0.00,
   Avg_ISR_Switch          => 0.00,
   Best_ISR_Switch         => 0.00,
   Speed_Factor            => 1.00);
```

Listing 3.2: Beispiel MAST Processing-Resource – Regular-Processor

Zusammengenommen erschien MAST vielversprechend und wurde deshalb im weiteren Verlauf der Arbeit genutzt.

3.2.1. MAST-Modellierung

Um die Echtzeitfähigkeit einer vCPS-Anwendung, beispielhaft siehe Abbildung 2.3, festzustellen, soll eine Auswertung einer Platzierung durch MAST stattfinden. Im späteren Verlauf wird die MAST-Software direkt in den evolutionären Algorithmus integriert, um jedoch eine bessere Nachvollziehbarkeit der Ergebnisse zu ermöglichen, wird im aktuellen Abschnitt die Modellierung in der MAST-eigenen Syntax vorgestellt. Wir benötigen dafür die MAST-Entitäten *Processing_Resource*, *Scheduler*, *Scheduling_Server* und *Operation*, die schließlich innerhalb einer Transaction zusammen einen Ausführungsfluss ergeben.

Beginnend kann man *Processing_Resource*s definieren, also die Entitäten, auf denen Operationen ausgeführt werden können. Im Rahmen unserer Modelle benötigen wir die zwei Typen *Regular_Processor* und *Packet_Based_Network*, die unterschiedliche Parameter erwarten.

Listing 3.2 zeigt die Definition eines Prozessors, in unserem Fall ein Host auf dem vCPS-Anwendungen ausgeführt werden können – entsprechend wird er benannt. Als Parameter erwartet dieses Objekt minimale und maximale Interrupt-Prioritäten, Zeiten für Wechsel zwischen Interrupt-Service-Routinen (Minimum, Maximum und Durchschnitt), und einen Speed-Faktor, der relevant ist, wenn man unterschiedlich schnelle Ausführungsressourcen anlegen möchte. Alle Parameter werden von unserer Abstraktion nicht berücksichtigt, sodass wir für alle Parameter Standard-Werte wählen. Von diesen *Processing-Resources* benötigen wir eine Definition pro Host, der benutzt werden soll.

Außerdem werden zwei Netzwerke, eines für die Kommunikation des CPS und eines für VM-spezifische Kommunikation benutzt, siehe dazu auch Ab-

```
Processing_Resource (
   Type                    => Packet_Based_Network,
   Name                    => network_vmm,
   Transmission            => FULL_DUPLEX,
   Throughput              => 62500000.0,
   Max_Blocking            => 0.00,
   Max_Packet_Size         => 10000.0,
   Min_Packet_Size         => 1.0,
   Speed_Factor            => 1.00);

Processing_Resource (
   Type                    => Packet_Based_Network,
   Name                    => network_data,
   Transmission            => FULL_DUPLEX,
   Throughput              => 62500000.0,
   Max_Blocking            => 0.00,
   Max_Packet_Size         => 10000.0,
   Min_Packet_Size         => 1.0,
   Speed_Factor            => 1.00);
```

Listing 3.3: Beispiel MAST Processing-Resource – Packet-Based-Network

bildung 2.1. Eine entsprechende Modellierung in MAST-Syntax ist in Listing 3.3 ersichtlich. Erwartet werden zusätzlich zu den bereits bekannten Parametern der Durchsatz und die Übertragungsart der Verbindung, eine maximale Block-Zeit und die kleinste und größte Größe von Paketen. Als Übertragungsart wird FULL-DUPLEX gewählt, da es der quasi-Standard für aktuelle Ethernet-Verbindungen ist, und wie in den Annahmen begründet legen wir die Bandbreite auf 62.5 MB/s fest. Für die restlichen Parameter wurden bisher keine Einschränkungen vorgenommen, sodass wir wieder Standard-Werte festlegen.

Jeder *Processing-Resource* muss ein *Scheduler*-Objekt zugewiesen werden, welches steuert, welche Aufgabe (Berechnung oder Übertragung) als nächstes gestartet wird. Auch hier werden zwei Typen genutzt, einmal ein *fixed priority*-Typ für die Hosts und zweitens ein Paket-basierter *fixed priority*-Typ für die beiden Netzwerke. Die Definition findet sich in Listing 3.4. Als Parameter werden vor allem *Overhead*-Werte für Kontextwechsel und Kommunikations-Rahmen erwartet, denen wir wieder die Standard-Werte zuweisen, da wir derartige Annahmen in unserem Modell nicht getroffen haben.

Für jede Aktion, die auf einer *Processing Resource* mithilfe eines *Schedulers* ausgeführt werden soll, muss ein *Scheduling Server* definiert werden. Sie sind der Rahmen, in dem die später definierten Operationen verpackt und

```
Scheduler (
   Type            => Primary_Scheduler,
   Name            => host_0,
   Host            => host_0,
   Policy          =>
      ( Type                => Fixed_Priority,
        Worst_Context_Switch => 0.00,
        Avg_Context_Switch   => 0.00,
        Best_Context_Switch  => 0.00,
        Max_Priority         => 32767,
        Min_Priority         => 1));

Scheduler (
   Type            => Primary_Scheduler,
   Name            => network_vmm,
   Host            => network_vmm,
   Policy          =>
      ( Type                    => FP_Packet_Based,
        Packet_Worst_Overhead    => 0.00,
        Packet_Avg_Overhead      => 0.00,
        Packet_Best_Overhead     => 0.00,
        Max_Priority             => 32767,
        Min_Priority             => 1));

Scheduler (
   Type            => Primary_Scheduler,
   Name            => network_data,
   Host            => network_data,
   Policy          =>
      ( Type                    => FP_Packet_Based,
        Packet_Worst_Overhead    => 0.00,
        Packet_Avg_Overhead      => 0.00,
        Packet_Best_Overhead     => 0.00,
        Max_Priority             => 32767,
        Min_Priority             => 1));
```

Listing 3.4: Beispiel MAST Scheduler

vom Scheduler verarbeitet werden können. Der wichtigste Parameter bei diesen Definitionen ist die Priorität, wobei in MAST eine größere Zahl einer höheren Priorität entspricht. Für die Netzwerk-Aktionen reichen vier Definitionen, siehe Listing 3.5, wovon nur die Initialisierung der HA als weniger wichtig eingestuft wird, als die restlichen Kommunikationen. Da im Rahmen der Hosts RM-Scheduling zum Einsatz kommen soll, benötigen wir dabei pro Host und pro Priorität einen Scheduling-Server. Eine beispielhafte Definition für einen Host mit den verfügbaren Prioritäten 1 und 2 findet sich in Listing 3.6.

Schließlich kommen wir zu den letzten Bausteinen für die Struktur, den Operationen. Jede Box der Abbildung 2.3 ist eine Operation in MAST. Eine Beispieldefinition ist in Listing 3.7 zu sehen. Die Parameter entsprechen unseren Modell-Parametern.

Aus den Grundbausteinen müssen nun Ablauffolgen gebildet werden. Ebenso wie in unserer Abbildung 2.3 zwei Flüsse zu erkennen sind, müssen in MAST zwei Transaktionen definiert werden. Eine Transaktion in MAST besteht aus einem externen Event, mehreren internen Events und sogenannten Event-Handlern, die entweder eine zuvor definierte Operation ausführen können oder Events teilen oder zusammenführen können. Die erste Transaktion jeder vCPS-Anwendung, siehe Listing 3.8, repräsentiert die Initialisierung der HA, die sich durch die Nutzung der entsprechenden Operation auszeichnet und eine Zeitbeschränkung von 5 Sekunden enthält. Eine zweite Transaktion, siehe Listing 3.9, stellt den zweiten Fluss unseres Modells dar, wird deshalb mit einem periodischen externen Event angelegt und enthält einen Multicast-Event-Handler, der dafür sorgt, dass Ausgabe-Kommunikation und reguläre HA-Übertragung parallel erfolgen können. Beide internen Events am Ende enthalten eine Zeitbeschränkung, um sowohl Grenzen für die Echtzeitfähigkeit der vCPS-Anwendung, als auch für ein zeitnahes Checkpointing festzulegen. Diese Zeitbeschränkungen werden von MAST genutzt, um die Einhaltung der Echtzeitbedingungen zu entscheiden.

Wie eingangs erwähnt, lassen sich Multi-Core-Systeme, in denen Prozesse nicht regelmäßig den ausführenden Prozessor wechseln, simulieren. Pro Core wird jeweils ein Host angelegt und bei der Übertragung der VMs in die MAST-Datenstruktur muss für zwei datenabhängige VMs nur geschaut werden, ob die beiden zugewiesenen Hosts tatsächlich nur zwei Prozessoren auf einem realen Host sind. Falls dies der Fall ist, wird der Kommunikationsbedarf auf 0 gesetzt, sodass das Netzwerk nicht für Kommunikation zwischen zwei Cores genutzt wird.

```
Scheduling_Server (
   Type                       => Regular,
   Name                       => input,
   Server_Sched_Parameters    =>
      ( Type         => Fixed_Priority_Policy,
        The_Priority => 1,
        Preassigned  => YES),
   Scheduler                  => network_data);

Scheduling_Server (
   Type                       => Regular,
   Name                       => output,
   Server_Sched_Parameters    =>
      ( Type         => Fixed_Priority_Policy,
        The_Priority => 1,
        Preassigned  => YES),
   Scheduler                  => network_data);

Scheduling_Server (
   Type                       => Regular,
   Name                       => ha_init,
   Server_Sched_Parameters    =>
      ( Type         => Fixed_Priority_Policy,
        The_Priority => 2,
        Preassigned  => YES),
   Scheduler                  => network_vmm);

Scheduling_Server (
   Type                       => Regular,
   Name                       => ha_regular,
   Server_Sched_Parameters    =>
      ( Type         => Fixed_Priority_Policy,
        The_Priority => 1,
        Preassigned  => YES),
   Scheduler                  => network_vmm);
```

Listing 3.5: Beispiel MAST Scheduling-Server für Netzwerk

```
Scheduling_Server (
   Type                       => Regular,
   Name                       => host_0_priority_1,
   Server_Sched_Parameters    =>
      ( Type        => Fixed_Priority_Policy,
        The_Priority => 1,
        Preassigned  => YES),
   Scheduler                  => host_0);

Scheduling_Server (
   Type                       => Regular,
   Name                       => host_0_priority_2,
   Server_Sched_Parameters    =>
      ( Type        => Fixed_Priority_Policy,
        The_Priority => 2,
        Preassigned  => YES),
   Scheduler                  => host_0);
```

Listing 3.6: Beispiel MAST Scheduling-Server für Hosts

3.2.2. MAST Algorithmen

Das Modell aus dem vorigen Abschnitt wird von MAST mittels bestimmter Algorithmen bewertet, welche wir uns hier im Detail anschauen. Die Literatur der Scheduling-Algorithmen enthält eine Vielzahl verschiedener Algorithmen, die jedoch nicht immer auf alle Modell-Typen anwendbar sind. MAST führt deshalb für ein Modell Checks durch, die grundlegende Eigenschaften eines Modells prüfen, und stellt außerdem sicher, dass ein Modell syntaktisch sinnvoll ist. Zu den von MAST geprüften Eigenschaften zählen z. B.:

Monoprocessor Only Es gibt nur eine Ausführungs-Ressource, und die ist ein Prozessor.

Fixed Priority Only Alle Entitäten haben fixe Prioritäten und es werden keine hierarchischen Scheduler genutzt.

Linear Transactions Only Jede Transaktion enthält nur ein externes Event und alle Event-Handler sind Aktivitäten, d. h. es gibt vor allem keine Teilen-/Zusammenführen-Handler.

Abhängig von den erfüllten und nicht erfüllten Eigenschaften, können unterschiedliche Algorithmen von MAST angewandt werden. Vor allem unsere zweite periodische Transaktion ist maßgeblich, denn sie zählt nicht mehr zu den linearen Transaktionen und grenzt die anwendbaren Algorithmen stark

```
Operation (
   Type                        => Simple,
   Name                        => vm_0_computation,
   Worst_Case_Execution_Time   => 4.00,
   Avg_Case_Execution_Time     => 4.00,
   Best_Case_Execution_Time    => 4.00);

Operation (
   Type                        => Message_Transmission,
   Name                        => vm_0_in,
   Max_Message_Size            => 1.00,
   Avg_Message_Size            => 1.00,
   Min_Message_Size            => 1.00);

Operation (
   Type                        => Message_Transmission,
   Name                        => vm_0_out,
   Max_Message_Size            => 1.00,
   Avg_Message_Size            => 1.00,
   Min_Message_Size            => 1.00);

Operation (
   Type                        => Message_Transmission,
   Name                        => vm_0_ha_regular,
   Max_Message_Size            => 500.00,
   Avg_Message_Size            => 500.00,
   Min_Message_Size            => 500.00);

Operation (
   Type                        => Message_Transmission,
   Name                        => vm_0_ha_init,
   Max_Message_Size            => 0.00,
   Avg_Message_Size            => 0.00,
   Min_Message_Size            => 0.00);
```

Listing 3.7: Beispiel MAST Operation

```
Transaction (
   Type            => regular,
   Name            => vm_0_singular,
   External_Events =>
      ( ( Type  => Singular,
          Name  => vm_0_e0,
          Phase => 0.000)),
   Internal_Events =>
      ( ( Type => Regular,
          Name => vm_0_eX,
          Timing_Requirements =>
            ( Type             => Hard_Global_Deadline,
              Deadline         => 5.000,
              Referenced_Event => vm_0_e0))),
   Event_Handlers  =>
      ( (Type               => Activity,
         Input_Event        => vm_0_e0,
         Output_Event       => vm_0_eX,
         Activity_Operation => vm_0_ha_init,
         Activity_Server    => ha_init)));
```

Listing 3.8: Beispiel MAST Transaktionen

ein. Nach Tabelle 1 in [Dra+14, S. 6] bleibt die Holistische Analyse als einziger Algorithmus übrig.

Zurück geht die Analyse von Echtzeitfähigkeit auf einen Aufsatz von Joseph und Pandya, die im Jahr 1986 erste exakte Garantien für Antwortzeiten erforschten [JP86]. Die Methoden wurden ca. 10 Jahre später von Tindell und Clark auf verteilte Echtzeitsysteme ausgeweitet [TC94; GGH97], indem die Methoden für Prozessoren auch auf Kommunikationskanäle angewandt wurden (mit kleinen Anpassungen), und voneinander abhängige Aufgaben derart analysiert werden, dass die schlechtesten Antwortzeiten der vorhergehenden Aufgabe die Periode und den Jitter der davon abhängenden Aufgabe definiert. Im Zuge der Entwicklung von MAST wurde dieses Verfahren von Gutierrez Garcia, Gutierrez und Gonzalez Harbour auf Multiple-Event-Systeme ausgeweitet, also Systeme bei denen eine Aufgabe nicht nur ein eingehendes und ein ausgehendes Event haben kann, sondern auch mehrere, sodass 1-N und N-1 Beziehungen möglich werden [GGG00]. Dazu wandeln sie Multiple-Event-Systeme in äquivalente lineare Systeme um, die dann mit den bestehenden Techniken analysiert werden können. Die Ergebnisse müssen danach für das Ursprungsmodell nur noch geschickt kombiniert werden. Das resultierende Verfahren bietet keine exakte Analyse, sondern berechnet obere Schranken für die

```
Transaction (
   Type            => regular,
   Name            => vm_0_periodic,
   External_Events =>
      ( ( Type       => Periodic,
          Name       => vm_0_e1,    Period     => 12.000,
          Phase  => 0.000,     Max_Jitter => 0.000)),
   Internal_Events =>
      ( ( Type => Regular,   Name => vm_0_e2),
        ( Type => Regular,   Name => vm_0_e3),
        ( Type => Regular,   Name => vm_0_e4),
        ( Type => Regular,   Name => vm_0_e5),
        ( Type => Regular,   Name => vm_0_e6,
          Timing_Requirements =>
            ( Type             => Hard_Global_Deadline,
              Deadline         => 10.000,     Referenced_Event => vm_0_e1)),
        ( Type => Regular,   Name => vm_0_e7,
          Timing_Requirements =>
            ( Type             => Hard_Global_Deadline,
              Deadline         => 11.000,     Referenced_Event => vm_0_e1)),
          )),
   Event_Handlers  =>
      ( (Type               => Activity,
         Input_Event        => vm_0_e1,
         Output_Event       => vm_0_e2,
         Activity_Operation => vm_0_in,
         Activity_Server    => input),
        (Type               => Activity,
         Input_Event        => vm_0_e2,
         Output_Event       => vm_0_e3,
         Activity_Operation => vm_0_computation,
         Activity_Server    => host_0_priority_2),
        (Type              => Multicast,
         Input_Event       => vm_0_e3,
         Output_Events_List => ( vm_0_e4, vm_0_e5)),
        (Type               => Activity,
         Input_Event        => vm_0_e4,
         Output_Event       => vm_0_e6,
         Activity_Operation => vm_0_out,
         Activity_Server    => output),
        (Type               => Activity,
         Input_Event        => vm_0_e5,
         Output_Event       => vm_0_e7,
         Activity_Operation => vm_0_ha_regular,
         Activity_Server    => ha_regular)));
```

Listing 3.9: Beispiel MAST Transaktionen

Aufwände, um so Garantien aussprechen zu können: „Although this analysis technique is slightly pessimistic in some cases, it allows obtaining an upper bound of the system's worst-case response time, thus making it possible to guarantee the fulfillment of the timing requirements of the system.“ [GGG00]

Diese Variante der holistischen Analyse ist die, die nach wie vor in MAST ausgeführt wird, und somit von uns zur Bewertung einer durch den EA generierten Platzierung genutzt werden kann.

3.3. Evolutionäre Algorithmen

Der Begriff „evolutionärer Algorithmus“ deckt ein breites Feld an Methoden ab, in dessen Rahmen sich auch die Begrifflichkeiten im Verlauf der Zeit verändert haben. Das vorliegende Kapitel stützt sich deshalb vor allem auf [Wei07], ein Buch aus dem Jahr 2007, welches nicht nur einen detailreichen, sondern auch zeitgemäßen Einblick in die Thematik gewährt. Für eine kürzere, pragmatischere Darstellung der Thematik sei auf [Kra09] verwiesen.

Evolutionäre Algorithmen sind randomisierte, metaheuristische Lösungsverfahren für Optimierungsprobleme, die sich am Prozess der natürlichen Evolution anlehnen.

3.3.1 Definition. Ein *Optimierungsproblem* ist ein Tupel $(\Omega, f, \succ)$, bestehend aus einem Suchraum Ω, einer Bewertungsfunktion $f : \Omega \to \mathbb{R}$ und einer Vergleichsrelation $\succ \in \{<, >\}$.

Die *Menge der globalen Optima* ist dann definiert als

$$\mathcal{H} = \{x \in \Omega \mid \forall x' \in \Omega : f(x) \succeq f(x')\}. \tag{3.19}$$

■

In allgemeiner Form sind evolutionäre Algorithmen in der Lage jegliche Klasse an Optimierungsproblemen zu approximieren, weshalb sie zu den metaheuristischen Optimierungsverfahren zählen. Durch Spezialisierung auf bestimmte Optimierungsprobleme, lässt sich die Qualität der berechneten Lösungen verbessern.

3.3.1. Analogien zur natürlichen Evolution

Mögliche Lösungskandidaten $x \in \Omega$ eines Optimierungsproblems müssen im Rahmen eines Algorithmus kodiert werden, um mit ihnen operieren zu können. Wie in der Natur unterscheidet man deshalb zwischen dem Suchraum Ω, dem so genannten *Phänotyp*-Raum, und einer Darstellung der Lösungskandidaten, dem so genannten *Genotyp*-Raum $\mathcal{G}$. Phänotyp bezeichnet dabei im

Allgemeinen das äußere, sichtbare Erscheinungsbild, wohingegen Genotyp die Erbfaktoren eines Lebewesens umfasst.

3.3.2 Definition. Sei Ω der Phänotyp-Suchraum eines Optimierungsproblems und $\mathcal{G}$ ein Genotyp-Raum, dann ist $dec : \mathcal{G} \to \Omega$ eine **Dekodierungsfunktion**, um einzelne Genotypen in einen Lösungskandidaten abzubilden. ■

3.3.3 Bemerkung. Es kann sinnvoll sein $\Omega = \mathcal{G}$ zu wählen, sodass die Dekodierungsabbildung als Identität zu wählen ist, $dec \equiv \mathrm{id}$. ■

Ein evolutionärer Algorithmus operiert auf Individuen, die sich aus den bisher genannten Komponenten zusammensetzen. Insbesondere wird die Bewertungsfunktion f des Optimierungsproblems genutzt, um im Algorithmus eine Richtung wählen zu können.

3.3.4 Definition. Ein **Individuum** A ist ein Tupel $(A.G, A.F)$ bestehend aus einem kodierten Lösungskandidaten (Genotyp) $A.G \in \mathcal{G}$ und einem Gütewert $A.F = f(dec(A.G)) \in \mathbb{R}$.

Wir schreiben auch kurz $F(A.G) = f(dec(A.G))$ und nennen $F : \mathcal{G} \to \mathbb{R}$ die (induzierte) Bewertungsfunktion. ■

Damit der EA nicht zu einem Bergsteiger-Algorithmus/Gradientenverfahren[8] degeneriert, bedient er sich des Konzepts der Populationen. Eine Population umfasst eine beliebige Anzahl an Individuen, auf denen der Algorithmus gleichzeitig operiert. Ziel ist es mithilfe der enthaltenen Individuen ein breites Spektrum des Lösungsraums abzudecken, sodass die sich entwickelten Individuen möglichst unterschiedliche Extremwerte der Zielfunktion erreichen und nicht an einem lokalen Extremwert hängen bleiben.

3.3.5 Definition. Eine **Population** ist ein Tupel $(A_1, \ldots, A_n)$ bestehend aus n Individuen A_i mit $1 \leq i \leq n$. ■

Ausgehend von einer initialen Population werden die beiden von Darwin identifizierten Evolutionsfaktoren Selektion und Variation (bestehend aus Rekombination und Mutation) wiederholt simuliert, um immer neue Individuen und damit Populationen zu generieren. Ziel ist es im Laufe der Zeit, in Bezug auf die Bewertungsfunktion f, überlegene Individuen zu identifizieren. Diese überlegenen Individuen sind dann approximative Lösungen für das zugrundeliegende Optimierungsproblem.

[8] Sowohl Bergsteiger-Algorithmus als auch Gradientenverfahren operieren auf meist nur einer zu verbessernden Lösung und bewegen diese Lösung mit unterschiedlich großen Schritten in eine bessere Richtung. Probleme bereiten dabei regelmäßig lokale Extrema, welche ohne weitere Anpassungen die Lösung anziehen und dann eine weitere Konvergenz zu einem globalen Extrema verhindern.

3.3.6 Definition. Sei $\mathcal{G}$ ein Genotyp-Raum für ein Optimierungsproblem und $\xi \in \Xi$ der Zustand eines Zufallszahlengenerators, dann ist ein **Mutationsoperator** eine Abbildung

$$Mut^{\xi} : \mathcal{G} \to \mathcal{G}. \tag{3.20}$$

Ein Genotyp wird hierbei in einen anderen Genotyp mutiert. ■

3.3.7 Definition. Sei $\mathcal{G}$ ein Genotyp-Raum für ein Optimierungsproblem und $\xi \in \Xi$ der Zustand eines Zufallszahlengenerators, dann ist ein **Rekombinationsoperator** eine Abbildung

$$Rek^{\xi} : \mathcal{G}^r \to \mathcal{G}^s, \tag{3.21}$$

wobei $r \geq 2$ die Anzahl der Eltern und $s \geq 1$ die Anzahl der Kinder ist. Aus r Eltern-Individuen werden s Kind-Individuen generiert. ■

3.3.8 Definition. Sei $\mathcal{G}$ ein Genotyp-Raum für ein Optimierungsproblem und $\xi \in \Xi$ der Zustand eines Zufallszahlengenerators, dann ist ein **Selektionsoperator** eine generische Abbildung

$$\begin{aligned} Sel^{\xi} &: (\mathcal{G} \times \mathbb{R})^r \to (\mathcal{G} \times \mathbb{R})^s \\ &\left(A^{(i)}\right)_{1 \leq i \leq r} \mapsto \left(A^{IS^{\xi}(b_1,\ldots,b_r)_k}\right)_{1 \leq k \leq s} \quad \text{mit } A^{(i)} = (a_i, b_i) \end{aligned} \tag{3.22}$$

mit einer spezialisierten Indexselektion

$$IS^{\xi} : \mathbb{R}^r \to \{1, \ldots, r\}^s. \tag{3.23}$$

Da bei der Selektion keine neuen Individuen generiert werden, sondern nur eine Auswahl getroffen wird, nutzt diese Definition das Konzept einer Indexselektion. Während der generische Teil des Selektionsoperators für alle EA identisch ist, findet die eigentliche Auswahl in der Indexselektion statt, welche r Gütewerte erwartet und die Indizes der s gewählten Gütewerte zurückgibt.■

Die Evolutionsfaktoren orientieren sich regelmäßig stark an der Natur. Für den Selektionsoperator werden z. B. häufig die Auswahlkriterien „zufällig“, „beste“, „schlechteste“ genutzt. Im Rahmen des Mutationsoperators werden einzelne Teile eines Individuums zufallsgesteuert verändert, sowie auch im natürlichen Umfeld von Generation zu Generation zufällige Mutationen stattfinden können. Die Rekombinationsoperatoren orientieren sich regelmäßig an der Überkreuzung von Chromosomen, z. B. werden aus zwei Folgen a_1a_2 und b_1b_2 durch Überkreuzung die Folgen a_1b_2 und b_1a_2.

Hervorzuheben ist, dass die Evolutionsfaktoren im Vergleich zur Bewertungsfunktion auf unterschiedlichen Mengen operieren. Während die drei Evolutionsoperatoren nur auf dem Genotyp agieren, muss zur Bewertung eine Abbildung in den ursprünglichen Suchraum vorgenommen werden. Dies erlaubt uns im späteren Verlauf der Arbeit verschiedene Genotypen miteinander zu vergleichen, ohne mehrere Implementierungen der Fitness-Funktion vornehmen zu müssen.

3.3.2. Abstraktion eines evolutionären Algorithmus

Die bisher genannten Operatoren sind regelmäßig die einzigen Komponenten eines EA, somit lässt sich eine generische Definition eines EA aufstellen.

3.3.9 Definition. Ein **generischer evolutionärer Algorithmus** ist ein Tupel

$$(\mathcal{G}, dec, Mut, Rek, IS_{\text{Eltern}}, IS_{\text{Umwelt}}, \mu, \lambda) \tag{3.24}$$

wobei μ die Größe der Elternpopulation und λ die Anzahl der generierten Kinder pro Iteration ist. Dann gilt hier für die oben vorgestellten evolutionären Operatoren:

$$\begin{aligned} Rek &: \mathcal{G}^k \to \mathcal{G}^{k'} \\ IS_{\text{Eltern}} &: \mathbb{R}^\mu \to \{1, \ldots, \mu\}^{\frac{k}{k'} \cdot \lambda} \quad \text{mit } \frac{k}{k'} \cdot \lambda \in \mathbb{N} \\ IS_{\text{Umwelt}} &: \mathbb{R}^{\mu+\lambda} \to \{1, \ldots, \mu + \lambda\}^\mu. \end{aligned} \tag{3.25}$$

■

Abbildung 3.1 zeigt den Kreislauf eines generischen EA und wie die Komponenten zusammenwirken. Wie dort ersichtlich startet ein Algorithmus in der Initialisierungsphase, in der zumeist zufällig Individuen erstellt werden, um eine erste Population zu erschaffen. Diese Population wird bewertet und bildet den Start für den Kreislauf. Der Rest des Algorithmus wird solange wiederholt, bis eine Terminierungsbedingung eingetroffen ist. Diese Bedingung kann beispielhaft abhängig von der Anzahl der Iterationen oder von der Diversität der aktuellen Population sein.

Wiederholt wird erst eine Paarungsselektion durchgeführt. In diesem Schritt werden mithilfe eines Selektionsoperators die Individuen ausgewählt, die im nächsten Schritt mithilfe eines Rekombinationsoperators kombiniert werden sollen. Danach erfolgt die Mutation mithilfe eines Mutationsoperators, entweder auf der gesamten Population oder nur auf den gerade rekombinierten neuen Individuen – abhängig vom aktuellen EA. Die neu erstellten oder mutierten Individuen müssen bewertet werden, um im weiteren Verlauf korrekt mit ihnen

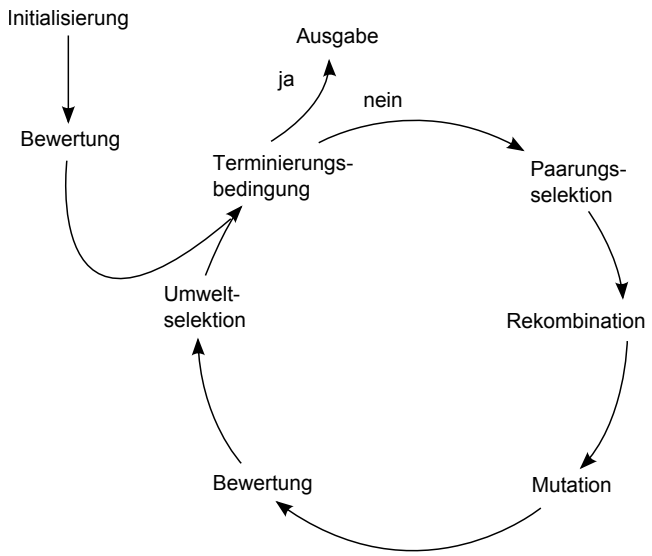

Abbildung 3.1.: Abstrakter Ausführungskreislauf eines evolutionären Algorithmus nach [Wei07].

operieren zu können. Der letzte Schritt ist die sogenannte Umweltselektion, in der ausgewählt wird, welche der soeben neu erstellten Individuen tatsächlich auch Einzug in die Population schaffen sollen. Nicht jedes neu geschaffene Individuum überlebt. Nach der Prüfung der Terminierungsbedingung startet die Ausführung erneut bei der Paarungsselektion.

3.3.3. Einordnung evolutionärer Verfahren

Wie eingangs erwähnt deckt der Begriff „Evolutionäre Algorithmen“ ein breites Feld an Algorithmen ab, was sich vor allem historisch erklären lässt. Die zeitgleiche Entwicklung ähnlicher Verfahren in unterschiedlichen Teilen der Welt, führte zu einer anderen Benennung vergleichbarer Verfahren. Eine strikte Unterteilung ist aufgrund der Ähnlichkeit der Verfahren und der synonymen Benutzung der Begriffe heute kaum sinnvoll und stillt vor allem historische Interessen. Trotzdem werden einige Arten kurz vorgestellt, um den Umfang des Begriffs zu verdeutlichen:

Genetischer Algorithmus In genetischen Algorithmen werden Lösungskandidaten häufig binärcodiert dargestellt. Daraus ergibt sich, dass man eine Genotyp-Phänotyp-Abbildung benötigt, um Lösungskandidaten in eine Repräsentation umzuwandeln, die bewertet werden kann. Ein Rekombi-

nationsoperator steuert den Algorithmus maßgeblich und die Elternselektion erfolgt oft randomisert.

Evolutionsstrategien Evolutionsstrategien kommen fast immer ohne Genotyp-Phänotyp-Abbildung aus, d. h. Problem- und Suchraum sind identisch und entsprechen meist $\mathbb{R}$. Darüber hinaus kommen Rekombinationsoperatoren oft nicht zum Einsatz, da die klassischen n-Punkt-Crossover-Operatoren Lösungen ausschließen [SHK94, S. 70] und spezielle arithmetische Operatoren genutzt werden müssen.

Genetisches Programmieren Im Rahmen der genetischen Programmierung sind Lösungskandidaten Baumstrukturen, die eine transformative Logik enthalten, um Eingaben in Ausgaben umzuwandeln. Beispiele hierfür wären Syntax-Bäume mathematischer Funktionen, die Variablen durch Anwendung von Operatoren in einen Funktionswert überführen.

Evolutionäres Programmieren Die evolutionäre Programmierung ähnelt der genetischen Programmierung, mit dem Hauptunterschied, dass die zugrundeliegenden Strukturen endliche Automaten sind.

Darüber hinaus gibt es noch weitere an die Natur angelehnte Verfahren, z. B. Schwarmintelligenz oder künstliche Immunsysteme, welche einige Autoren bei den Evolutionären Algorithmen einordnen, andere wiederum nur der darüber liegenden *Computational Intelligence*.

Die im Rahmen dieser Arbeit entwickelten Verfahren lassen sich am ehesten den genetischen Algorithmen zuordnen.

3.3.4. Spezielle Techniken für spezifische Problemanforderungen

Über den bisher allgemeinen vorgestellten Algorithmus hinaus gibt es spezielle Methoden, um einen EA an besondere Anforderungen anzupassen. Dazu zählen nach [Wei07] zum Beispiel folgende Problemklassen:

- Optimierung mit Randbedingungen,
- Mehrzieloptimierung,
- zeitabhängige Optimierungsprobleme,
- zeitaufwändige Bewertung und
- Bewertung durch Testfälle unter Nutzung von Koevolution.

Im Folgenden werden nur die Themen angeschnitten, die für die vorliegende Arbeit relevant sind.

3.3.4.1. Optimierung mit Randbedingungen

Ein Suchraum Ω kann durch weitere Randbedingungen eingeschränkt sein. Diese Randbedingungen lassen sich in harte und weiche Bedingungen unterteilen, wobei harte Bedingungen ein Individuen eindeutig als gültig oder ungültig kennzeichnet, wohingegen weiche Bedingungen graduell unerwünschte Individuen ausschließen soll. Weiche Randbedingungen lassen sich regelmäßig in die Bewertungsfunktion integrieren [Wei07, S. 185], sodass vor allem harte Randbedingungen mehr Aufmerksamkeit erfordern.

Eine Möglichkeit harte Randbedingungen zu beachten ist es die Dekodierfunktion so anzupassen, dass sie alle Genotypen in gültige Lösungskandidaten abbildet, was jedoch nicht immer möglich ist. Eine weitere Möglichkeit ist es den EA so anzupassen, dass ungültige Individuen innerhalb des Algorithmus sofort entfernt werden. Hierbei ist zu beachten, dass man bei zerklüfteten Lösungsräumen eventuell den Sprung in einen anderen gültigen Abschnitt ohne ungültige Individuen nicht schafft. Außerdem lassen sich die Operatoren so anpassen, dass sie gültige Individuen bevorzugen, z. B. bei der Selektion. Der wohl beliebteste Ansatz ist das Einsetzen einer Straffunktion (oder eines Strafterms) in die Bewertungsfunktion für die Verletzung von Randbedingungen. Das größte Problem bei diesem Ansatz ist es die Bewertungs- und die Straffunktion so aufeinander abzustimmen, dass der kombinierte Funktionswert sinnvoll bleibt.

3.3.4.2. Mehrzieloptimierung

Wenn Lösungen anhand verschiedener gegenläufiger Kriterien bewertet werden müssen, müssen besondere Vorkehrungen getroffen werden. Im Rahmen der Platzierung von virtuellen Maschinen können wir beispielhaft die Echtzeitfähigkeit regelmäßig dadurch garantieren, dass wir jede VM auf genau einem Host platzieren. Dieses Ziel kollidiert mit dem Ziel möglichst wenige Hosts in Anspruch zu nehmen und umgekehrt gilt diese Aussage ebenso. Fraglich ist nun welcher Lösungskandidat im Zweifel besser ist. Darüber lässt sich oft keine Aussage treffen und man möchte lieber eine Menge von derartig gleichwertigen Lösungen vom Algorithmus erhalten, um die Entscheidung später z. B. selbst zu treffen. Gleichzeitig gibt es aber natürlich auch Lösungskandidaten, die offensichtlich schlechter sind als die in dieser Äquivalenzklasse enthaltenen Lösungen. Dies motiviert die folgende Definition eines neuen Vergleichsoperators.

3.3.10 Definition. Seien F_i mit $1 \leq i \leq k$ Bewertungsfunktionen und A, B Individuen, dann ist der **Dominanzoperator** wie folgt definiert:

$$\begin{aligned} B >_{dom} A := \; & \forall 1 \leq i \leq k : F_i(B.G) \succeq F_i(A.G) \\ & \wedge \exists 1 \leq i \leq k : F_i(B.G) \succ F_i(A.G) \end{aligned} \tag{3.26}$$

Wir schreiben dann auch B *dominiert* A, wenn B in mindestens einer Dimension besser ist als A und ansonsten nicht schlechter.

Für eine Menge P von Lösungskandidaten, nennen wir die Menge der besten gleichwertigen Lösungen **nicht-dominierte Menge**:

$$nichtdom(P) := \{A \in P \mid \forall B \in P : \neg(B >_{dom} A)\} \tag{3.27}$$

Die nicht-dominierte Menge für ein Optimierungsproblem mit Suchraum Ω nennen wir **Pareto-Front**: $nichtdom(\Omega)$. ■

Dieser neue Vergleichsoperator muss natürlich im Rahmen der Operationen eines EA verwendet werden. Ziel eines derartig angepassten Algorithmus ist es eine Teilmenge der Pareto-Front zu berechnen, die möglichst repräsentativ für die Pareto-Front ist, was selbstverständliche weitere Anpassungen notwendig macht. Auf eine Vorstellung möglicher Algorithmen verzichten wir aus Platzgründen und verweisen stattdessen auf [Wei07, S. 201ff]. Der Algorithmus, der im Rahmen dieser Arbeit verwendet wird, wird darüber hinaus weiter hinten in Abschnitt 4.3.6 separat erläutert.

3.3.4.3. Zeitaufwändige Bewertung

Durch die hohe Anzahl an Bewertungen im Laufe eines EA ist es hinderlich, wenn eine Ausführung der Bewertungsfunktion lange dauert. Eine Möglichkeit diesem Problem entgegen zu wirken ist, nur einen Teil der Individuen, mithilfe der richtigen Bewertungsfunktion, zu evaluieren. Die restlichen Individuen können mithilfe einer kostengünstigeren Alternative bewertet werden. Dafür bieten sich „insbesondere neuronale Netze [...], RBF-Netze, polynomielle Regressionsmodelle und Gauß-Prozesse“ an [Wei07, S. 217].

Die Erstellung und Integration eines derartigen Modells in einen EA ist jedoch keine triviale Aufgabe, sodass regelmäßig auf Parallelität ausgewichen wird, um die Ausführungszeiten zu beschleunigen. Hierbei gibt es verschiedene Ansätze, was parallelisiert werden kann. Von einer *globalen Parallelisierung* spricht man bei einer Master-Slave-Anordnung der Prozesse. Während der Master-Prozess mit den Hauptaufgaben des EA betreut ist, delegiert er die Ausführung der Bewertungsfunktion an Slave-Prozesse. Darüber hinaus

gibt es noch die *grobkörnige* und die *feinkörnige Parallelisierung*, in denen die Population entweder in wenige, große oder in viele, sich überlappende, kleine Unterpopulationen zerlegt wird, um sie jeweils in einzelnen Prozessen zu bearbeiten.

3.3.4.4. Bewertung durch Testfälle unter Nutzung von Koevolution

Für den Fall, dass es zur Bewertung einer Lösung nur mehrere Testumgebungen gibt, bei denen die Ausführung aller Tests für jedes Individuum jedoch zu lange dauert, nennt Weicker die Koevolution als mögliche Lösung. Innerhalb einer Iteration werden nur bestimmte Testfälle auf bestimmte Individuen angewandt und die Fitness ergibt sich als Mittelwert über eine feste Anzahl der letzten ausgeführten Testfälle für das Individuum. Durch geschickte Wahl bei der Selektion haben bessere Individuen mehr Nachkommen, werden aber auch öfter getestet. Gleichzeitig werden auch die Testfälle durch geschickte Selektion so gesteuert, dass keine Testumgebung vernachlässigt wird.

Auf diesem Prinzip aufbauend wurden noch weitere Verfahren entwickelt, z. B. die kooperative Koevolution. Sie unterscheidet sich von der normalen Koevolution dahingehend, dass sie parallel mehrere Populationen mit jeweils einer Funktion evaluiert, und regelmäßig Individuen zwischen diesen Populationen austauscht.

Zu beachten ist, dass sich diese Verfahren regelmäßig nicht als allgemeine Verfahren eignen. Besonders relevant sind sie beim Vorhandensein mehrerer aufwändiger Testfälle [Wei07, S. 219f] oder Funktionen, die sich separieren lassen [PJ94; RY09], d. h. Funktionen, die sich so aus Unterfunktionen zusammensetzen, dass bestimmte Parameter nur von bestimmten Unterfunktionen genutzt werden, sodass man die Unterfunktionen isoliert optimieren könnte, um durch spätere Summierung ein Optimum für die Funktion zu ermitteln.

Auf nicht-separierbaren Funktionen arbeiten diese Algorithmen ohne Anpassung nicht unbedingt gut: „The above studies highlight that the basic form of CCEA has difficulties in solving problems, which are nonseparable [...]“ [RY09, S. 986]. Erst durch weitere Anpassungen erreichen die Autoren einen Algorithmus, der auch auf nicht-separierbaren Funktionen ähnliche Qualitäten erreicht, wie ein Standard-EA: „The results of CCEA-AVP for nonseparable problems indicate that it is better than the basic CCEA and has comparable average performance as EAs“ [RY09, S. 988].

Bevor koevolutionäre Verfahren angewandt werden sollen, sollte man sich also das Optimierungsproblem genau anschauen. Da in unserem Fall wenig für eine Separierbarkeit der Optimierungsfunktion spricht, wird im weiteren Verlauf der Arbeit kein koevolutionäres Verfahren angewandt.

In diesem Kapitel haben wir Möglichkeiten kennengelernt das Platzierungsproblem in eingeschränkter Form in ein lineares Programm zu überführen, eine vorliegende Platzierung mithilfe des formalen Leistungsbewertungstools MAST zu bewerten und einen grundlegenden Einblick in evolutionäre Algorithmen gewonnen. Im nächsten Kapitel wird die Integration der formalen Leistungsbewertung in einen evolutionären Algorithmus vorgestellt, um die Vorteile beider Verfahren zu vereinigen.

Kapitel 4

Entwicklung und Implementierung eines evolutionären Algorithmus

Evolutionäre Algorithmen haben viele Eigenschaften, die für das gegebene Problem hilfreich sind: Sie haben kein Problem mit exponentiell großen Suchräumen, sind trotz Randomisierung merklich besser als rein randomisierte Verfahren und durch die Struktur der Algorithmen ist eine Integration vorhandener Bewertungsmethoden, z. B. einer formalen Leistungsbewertung, einfach möglich.

Die grundlegende Struktur dieser Idee zeigt Abbildung 4.1. Die durchzogenen Pfeile zeigen den Ausführungsablauf, der bei einem formalen Modell in Form einer CSV-Datei startet. Dieses Modell wird von einem Programm eingelesen und in ein Objektmodell überführt, welches die Datengrundlage für den evolutionären Algorithmus bildet. Danach wandelt der EA wiederholt eine generierte Platzierung in ein Modell für das Leistungsbewertungsverfahren um und lässt dieses Verfahren alle gewünschten Kennzahlen errechnen. Die Ergebnisse werden zurück an den EA gegeben, der diese nutzen kann, um seine weitere Ausführung zu steuern. Die gestrichelten Pfeile zeigen die Modell-Transformationen, zeigen also den Verlauf der Daten unabhängig von der Ausführung.

Die Bandbreite an Möglichkeiten im Bereich der evolutionären Algorithmen ist groß und die Nutzung bestehender Frameworks bietet sich an, um auf robuste Verfahren zurückgreifen zu können. Das Kapitel beginnt mit einem Vergleich beliebter EA-Frameworks, zeigt dann die für das Platzierungsproblem definierten evolutionären Operatoren, und endet mit der Darstellung der Integration von MAST in den Evaluierungsoperator.

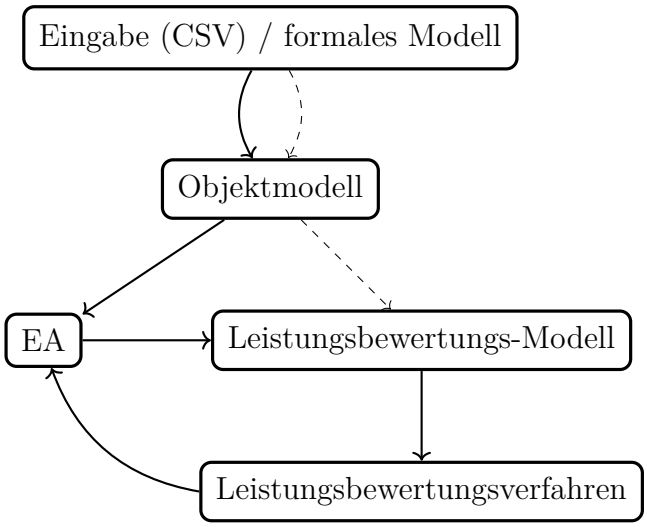

Abbildung 4.1.: Abstrakte Darstellung des Ausführungsablaufs und der Transformationen der zugrundeliegenden Daten-Modelle.

4.1. Wahl eines EA-Frameworks

Im Laufe der Zeit hat sich eine Vielzahl von Programmierern an evolutionären Algorithmen versucht und über die Jahre sind Frameworks entstanden, die das Implementieren von EAs beschleunigen soll. Im Rahmen dieser Arbeit sollte auf eines dieser Frameworks gebaut werden. Folgende in C++ implementierte Frameworks wurden zu diesem Zweck im Detail angeschaut:

Open BEAGLE (OB) ist ein Framework von Mitarbeitern der Universität Laval in Kanada.[1] Es existiert eine stabile Version 3.0.3 von November 2007 und eine Entwickler-Version 4.0.0-alpha2 von Februar 2010.[2] [GP06]

ECF ist eine Entwicklung von Mitarbeitern der Universität Zagreb in Kroatien.[3] Die aktuelle Version ist 1.4.1 von November 2014. [JGČ14]

EO ist eine Kollaboration ursprünglich von Mitarbeitern der Universität Granada in Spanien entwickelt, im Laufe der Zeit jedoch an zwei Forscher

[1] https://github.com/chgagne/beagle

[2] Zumindest ist die Version 4.0.0-alpha2 laut Webseite die aktuelle Entwickler-Version. Der Code im Repository bezeichnet sich schon als Version 4.1, jedoch fehlt eine Auflistung der Änderungen in den Changelogs. Da auch keine Repository-Referenzen für Versionen oder wenigstens einzelne Commits für Versionssprünge vorhanden sind, kann nicht einfach nachverfolgt werden, ob und was diese Versionen unterscheidet.

[3] http://gp.zemris.fer.hr/ecf/

aus den Niederlanden und Frankreich übergeben.[4] Die aktuelle Version ist 1.3.1 vom Juli 2012. [Kei+02]

PISA ist ein weiteres Framework, welches sich noch eine Abstraktionsschicht höher ansiedelt als evolutionäre Algorithmen und diese nur als eine mögliche Ausführung betrachten.[5] Es wurde von Mitarbeitern der ETH Zürich entwickelt und bietet auch Möglichkeiten die Algorithmen statistisch genau zu untersuchen. Eine Problemlösung wird hier in Variatoren und Selektoren unterteilt, wobei erstere Probleme und zweitere Algorithmen beinhalten. Durch ein einheitliches Text-Format zum Austausch zwischen beiden Teilen, lassen sich beliebige Variatoren mit beliebigen Selektoren ausführen. [Ble+03]

Zur Entscheidungsfindung wurde ein funktionaler Vergleich der Frameworks basierend auf den öffentlichen Informationen der Webseiten und Dokumentationen vorgenommen, deren Ergebnis in Tabelle 4.1 zu sehen ist.[6] Die ersten Zeilen (gekennzeichnet mit *A*) nennen Standard-Algorithmen zur Optimierung von Problemen mit nur einem Kriterium. Danach folgen einige Zeilen, die vorhandene Genotyp-Darstellungen untersuchen (gekennzeichnet mit *G*). Folgend wird die Unterstützung von multi-kriteriellen Problemen untersucht. Abschließend noch Parallelität und Checkpointing, dies bezeichnet die Möglichkeit den aktuellen Zustand das Algorithmus in eine Datei zwischenzuspeichern, um später diese Datei wieder laden zu können und von eben diesem Zustand die Ausführung fortzuführen.

Auffallend ist, dass nur Open BEAGLE sowohl Algorithmen für einfache als auch für multi-kriterielle Optimierungsprobleme mitbringt. Die Entwickler von PISA konzentrieren sich gewollt vor allem auf multi-kriterielle Algorithmen, sodass das Framework in diesem Bereich überdurchschnittliche viele Algorithmen mitbringt, es kann aber prinzipiell auch eindimensionale Probleme lösen. Für EO existieren Erweiterungen unter dem Name ParadiesEO, welche multikriterielle Unterstützung nachrüsten sollen. ECF unterstützt zur Zeit keine derartigen Algorithmen.

Die deutlich unterschiedliche Architektur von PISA erschwert die Bewertung von vorhandenen Genotypen und vor allem den Vergleich mit den anderen Frameworks. Da die sogenannten Variatoren problemspezifisch sind, ist eine Anwendung auf andere Probleme nicht einfach möglich, und unterscheidet sich von den Arbeiten, die bei anderen Frameworks nötig ist.

[4] http://eodev.sourceforge.net/

[5] http://www.tik.ethz.ch/sop/pisa/

[6] Die Breite des abgedeckten Bereichs erschwert teilweise eindeutige Antworten, sodass die Tabelle in einigen Zellen besser leer bleibt.

	OB	ECF	EO	PISA
A: Steady-State	Ja	Ja	Ja	
A: Generational Roulette Wheel	Ja	Ja	Ja	
A: Particle Swarm Optimization		Ja	Ja	
G: Binär	Ja	Ja	Ja	Möglich
G: Permutation	Ja	Ja	Ja	Möglich
G: Eigene	Ja	Ja	Ja	Möglich
Multi-Kriterieller Algorithmus	NSGA-II	Nein	Nein	NSGA-II, SPEA2, u. a.
Checkpointing	Ja	Ja	Ja	Nein
Parallelität	Ja	teilweise	Ja	Nein

Tabelle 4.1.: Funktionsübersicht der untersuchten EA Frameworks. Legende: A(lgorithmus), G(enotyp).

Ein wichtiges Entscheidungskriterium für ein Framework für die vorliegende Arbeit ist die Ermöglichung von Parallelität, vor allem des Evaluationsoperators. Wie in der Tabelle ersichtlich, unterstützen nur Open BEAGLE und EO Parallelität, wie es für diese Arbeit notwendig ist.

Für die weitere Arbeit wurde schließlich Open BEAGLE als Framework gewählt, vor allem aus folgenden Gründen:

- Einwandfreie Unterstützung von Parallelität.
- Durchdachte und einheitliche C++ Codebasis, z. B. Nutzung von Smart-Pointern.
- Gemeinschaftliches Projekt vieler Experten.

Im Detail wurde die zum Zeitpunkt der Veröffentlichung dieser Arbeit aktuelle Version 4.1.0[7] verwendet.

4.2. Hinweise zu Open BEAGLE

Open BEAGLE kann als Bibliothek über die Standard-Befehlskette kompiliert und im System installiert werden:

[7]Die Version 4.1.0 existiert als Entwickler-Version unter zwei URLs https://github.com/chgagne/beagle und https://code.google.com/p/beagle/. Es wurde die Github-Variante verwendet, da dieses das Repository bei Google zukünftig ablösen soll.

```
./configure && make && make install
```

Es nutzt zur Unterstützung von parallelen Berechnungen OpenMP[8], welches eine API für Parallelität definiert und auf Direktiven basiert. Die Programmierung paralleler Programme wird vereinfacht und erscheint durch die Nutzung von Direktiven aufgeräumt. OpenMP muss innerhalb des Compilers und auf der Plattform zur Verfügung stehen, und insbesondere muss der Compiler mit den korrekten Parametern aufgerufen werden, damit diese Funktionalität aktiviert wird. Für einen aktuellen GNU C++ Compiler sieht ein Kommandozeilen-Aufruf wie folgt aus:

```
g++ -fopenmp file.cpp
```

Der Build-Prozess von Open BEAGLE setzt auf CMake, welches viele Konfigurationen automatisch vornimmt, unter anderem die notwendigen Einstellungen für OpenMP. Darüber hinaus wurden folgende Einstellungen an CMake übergeben:

```
cmake -DBEAGLE_USE_C0x=ON \
      -DBEAGLE_OMP_MODE=dynamic \
      -G "Unix Makefiles" .
```

Der erste Parameter sorgt dafür, dass ein aktueller C++-Standard beim Kompilieren gewählt wird. Der zweite Parameter steuert das OpenMP-Scheduling. Beim parallelen Ausführen einer *for*-Schleife können die Iterationen statisch gestückelt und zur Ausführung in deterministischer Variante an die Threads übergeben werden, oder die Stückelung erfolgt dynamisch und sobald ein Thread sein aktuelles Stück abgearbeitet hat, bearbeitet er das nächste Stück. Der dynamische Modus bietet sich an, wenn die Ausführungszeiten der Stücke variieren, und ermöglicht dann eine bessere Auslastung der Threads. Dieses Modell arbeitet jedoch nicht deterministisch, was bei manchen Ausführungen hinderlich sein kann. Da dies bei der vorliegenden Arbeit nicht relevant ist, wird der dynamische Modus ausgewählt.

Außerdem wurden an der CMakeLists-Datei zwei Korrekturen vorgenommen:

- Eine C++ Präprozessor-Direktive steuert die Verwendung von parallelen STL-Containern, abhängig von der Compiler-Version. Die Überprüfung der Compiler-Version innerhalb der CMakeLists-Datei war fehlerhaft, sodass in vielen Fällen die parallelen STL-Container deaktiviert wurden, obwohl sie zur Verfügung standen.

[8] http://openmp.org/

- Open BEAGLE hängt von PACC (*Portable Agile C++ Classes*)[9] ab, und auch PACC muss mit parallelen STL-Containern kompiliert werden, damit diese von Open BEAGLE genutzt werden. Ob dies geschehen ist, wird im Rahmen der Open BEAGLE CMakeLists-Datei mithilfe eines Test-Kompilierungsvorgangs überprüft. Der Test-Quellcode nutzt einen parallelen STI-Container mit PACC und der Test wird als erfolgreich gewertet, wenn die Datei kompiliert werden kann, aufgrund einer falschen Reihenfolge der Linker-Parameter, kompilierte diese Datei jedoch nicht.

Für beide Fehler wurden Pull-Requests erstellt, sodass diese vermutlich in einer zukünftigen Version behoben werden.

4.3. Komponenten

Um das vorgestellte Problem mithilfe eines evolutionären Algorithmus lösen zu können, muss sowohl eine Genotyp-Phänotyp-Abbildung, als auch darauf anwendbare evolutionäre Operatoren entwickelt werden. Im Folgenden werden zwei mögliche Genotyp-Phänotyp-Abbildungen vorgestellt, einerseits eine klassische Binärcodierung und andererseits eine Codierung mit variabler Länge.

4.3.1. Binärer Genotyp

Eine klassische binäre Codierung zeichnet sich durch eine fixe Länge und standardisierte evolutionäre Operatoren aus. Um eine fixe Länge zu ermöglichen, muss eine Obergrenze für die zu nutzenden Hosts festgelegt werden. Wie in Kapitel 2 vorgestellt, gehen wir davon aus, dass jede Berechnung einer vCPS-Anwendung innerhalb der Zeitbegrenzung mindestens durch einen dedizierten Host abgearbeitet werden kann. Damit ergibt sich eine gültige Verteilung im schlimmsten Fall dadurch, dass jede vCPS-Anwendung einem dedizierten Host zugewiesen wird, womit die Obergrenze für die zu nutzenden Hosts feststeht.

4.3.1 Definition. Für ein vCPS $\tau = (\{\tau_1, \ldots, \tau_n\}, \phi)$ ist $\mathcal{G} = \{0,1\}^{n^2}$ ein möglicher Genotyp-Raum. Sei $X_{i,j}$ eine binäre Variable, die *wahr* ist, wenn die vCPS-Anwendung i auf Host j verteilt wird, und sonst *falsch*, dann lässt sich ein binärer Genotyp $x \in \mathcal{G}$ wie folgt interpretieren:

$$x = (X_{1,1} \ldots X_{1,n} X_{2,1} \ldots X_{2,n} \ldots X_{n,1} \ldots X_{n,n}) \tag{4.1}$$

[9] http://sourceforge.net/projects/pacc/

Gültig ist ein derartiger Genotyp, wenn folgende Bedingungen eingehalten werden:

$$\sum_{j=1}^{n} X_{i,j} = 1 \quad \forall 1 \leq i \leq n, \tag{4.2}$$

wenn also jede vCPS-Anwendung höchstens einem Host zugewiesen ist. ■

Ein binärer Genotyp kann auch eine ungültige Platzierung repräsentieren, z. B. eine nicht-disjunkte Partition, und die Einhaltung der Gültigkeits-Bedingung muss geprüft werden. Im Rahmen eines EA kann dies durch einen Strafterm in der Bewertungsfunktion erfolgen.

4.3.2. FL-Schedule

Im Laufe der Evaluation, siehe Kapitel 5, hat sich gezeigt, dass die hohe Anzahl an Bitstrings, die keine gültige Codierung für eine Platzierung sind, mit steigender Länge des Bitstrings zu einem Problem bei dieser Codierung wird. Wir nutzen deshalb, zusätzlich zu einer binären Codierung, eine weitere Codierung fixer Länge namens FL-Schedule (*fixed length schedule*), die sich dadurch auszeichnet, dass sie immer eine gültige Platzierung kodiert.

4.3.2 Definition. Für ein vCPS $\tau = (\{\tau_1, \ldots, \tau_n\}, \phi)$ ist

$$\mathcal{G} = \{((1, i_1), \ldots, (n, i_n)) \mid 1 \leq i_l \leq n \quad \forall 1 \leq l \leq n\} \tag{4.3}$$

ein möglicher Genotyp-Raum fixer Länge.

Jedes Tupel repräsentiert eine vCPS-Anwendung, wobei die zweite Komponente den zugewiesenen Host repräsentiert. ■

4.3.3. VLG-Schedule

Natürliche DNA enthält auch sogenannte nichtcodierende Abschnitte, also Abschnitte in der DNA, die vorhanden sind, aber keine direkte Auswirkung auf den Phenotyp haben [Kni+07]. Außerdem sind einzelne Gene in biologischer DNA unabhängig von der Position, was bei informationstechnischen Codierungen im Rahmen von EA zumeist nicht beachtet wird. In der wissenschaftlichen Literatur wurden solche Besonderheiten jedoch durchaus auch für evolutionäre Algorithmen erfolgreich untersucht und die Ergebnisse sprechen dafür, dass Codierungen variabler Länge, mit nichtcodierenden Abschnitten und Positions-unabhängigen Komponenten, beachtenswert sind [Bur+98; WL96; Lee00; Lee00]. Solche Genotypen werden auch VLG (*variable length genome*) genannt. Zusätzlich zur den beiden Codierung fixer Länge betrachten wir auch eine Codierung variabler Länge.

4.3.3 Definition. Für ein vCPS $\tau = (\{\tau_1, \ldots, \tau_n\}, \phi)$ ist

$$\mathcal{G} = \{(g_i)_{i=1,\ldots,l} \mid g_i \in M, l \in \mathbb{N}\} \tag{4.4}$$

mit

$$M = \{(x, y) \mid 1 \leq x, y \leq n\} \tag{4.5}$$

ein möglicher Genotyp-Raum variabler Länge.

Jedes Tupel der Menge M lässt sich als Zuordnung einer vCPS-Anwendung x zu einem Host y interpretieren. Ein Genotyp $z \in \mathcal{G}$ ist also eine endliche Folge von Zuordnungs-Tupeln. Sofern mehrere Tupel in einem Genotyp existieren, die eine vCPS-Anwendung unterschiedlichen Hosts zuweisen, ist das erste Tupel ausschlaggebend. Alle weiteren Vorkommen sind nichtcodierende Abschnitte. ■

4.3.4 Bemerkung. Auch bei diesen Genotypen kann es vorkommen, dass ungültige Lösungskandidaten generiert werden, welche von der Bewertungsfunktion bestraft werden müssen, z. B. wenn eine vCPS-Anwendung keinem Host zugewiesen wird.

In der Praxis ist es sinnvoll die Länge der Genotypen zu beschränken, z. B. indem zu lange Genotypen durch die Fitness-Funktion immer als ungültig bewertet werden, vor allem deshalb, weil es ansonsten vorkommen kann, dass die Genotypen mit steigenden Iterationen immer länger werden anstatt sich bei einer großen Länge einzupendeln. Das ergibt sich daraus, dass zusätzlich mitgetragene Länge nichts kostet, aber potentiell gute Regionen enthalten kann und somit Vorteile bietet [Bur+98]. ■

4.3.4. Paarungsselektion, Rekombination und Mutation

Abhängig von der Codierung verwenden wir unterschiedliche Operatoren, welche im Folgenden vorgestellt werden.

4.3.4.1. Binärer Genotyp

Die vorgeschlagene binäre Codierung gründet auf der Idee nur eine geeignete Codierung des Problems finden zu müssen, um danach Standard-Operatoren anwenden zu können. Dieses Vorgehen nutzt wenig Problemwissen und ist deshalb schnell zu implementieren: Praktisch jedes EA-Framework bringt Basis-Operatoren mit, sodass kaum Implementierungsarbeit notwendig ist.

Wir nutzen die beiden von Open BEAGLE bereitgestellten Operatoren *GA-MutationFlipBitStrOp* und *GA-CrossoverUniformBitStrOp*:

GA-MutationFlipBitStrOp Für ein zu mutierendes Individuum wird für jedes Bit gewürfelt, um zu entscheiden, ob das betroffene Bit gewechselt wird. Innerhalb eines Durchlaufs können also mehrere Bits gewechselt werden.

GA-CrossoverUniformBitStrOp Der Operator erwartet zwei selektierte Individuen zur Paarung, läuft über deren Länge und würfelt für jedes Bit, ob hier ein Crossover stattfinden soll. Falls ja, werden die Bits der beiden Individuen an der aktuellen Position getauscht.

Für die Auswahl der Individuen verwenden wir bei dieser Codierung eine zufällige Selektion, d. h. den *SelectRandomOp*-Operator von Open BEAGLE.

4.3.4.2. FL-Schedule

Für einen FL-Schedule nutzen wir zwei eigene Operatoren, namentlich *GA-ScheduleMutSetout* für die Mutation und *GA-CrossoverOnePointScheduleOp* für die Rekombination:

GA-ScheduleMutSetout Für ein zu mutierendes Individuum wird zuerst berechnet, welche Hosts noch unbelegt sind. Falls es noch freie Hosts gibt, wird für jedes Tupel des Individuums gewürfelt, ob die aktuelle vCPS-Anwendung auf einen der (dann zufällig ausgewürfelten) freien Hosts verschoben werden soll. Gibt es hingegen keine freien Hosts mehr, wird für jedes Tupel gewürfelt, ob die aktuelle vCPS-Anwendung auf irgendeinen anderen Host verschoben werden soll.

GA-CrossoverOnePointScheduleOp Dieser Operator gleicht dem bereits vorgestellten Operator *GA-CrossoverUniformBitStrOp*, mit dem Unterschied, dass er nicht auf Bits sondern auf Tupeln operiert.

4.3.4.3. VLG-Schedule

Ein VLG-Schedule ähnelt einem FL-Schedule, sodass auch hier der Operator *GA-ScheduleMutSetout* für die Mutation zum Einsatz kommt. Für die Rekombination nutzen wir jedoch einen neuen Operator, der Individuen variabler Länge generieren kann, *GA-CrossoverOnePointVLGScheduleOp*:

GA-CrossoverOnePointVLGScheduleOp Der Operator erwartet zwei zu paarende Individuen und würfelt für beide unabhängig eine Rekombinations-Position aus. An diesen Stellen werden die Individuen auseinander geschnitten, und danach die hinteren Teile vertauscht. Sofern nicht zufällig beide Individuen gleich lang sind und bei beiden der gleiche Schnittpunkt ausgewählt wird, verändert sich somit die Länge der Individuen.

4.3.5. Fitness

Die Bewertung eines Lösungskandidaten kann anhand der in Abschnitt 2.5 vorgestellten Kriterien erfolgen. Open BEAGLE bringt eine Klasse *FitnessMultiObjMin* mit, welche einen Vektor beliebig vieler Fitnesswerte enthält und eine Methode beherbergt, um auf Dominanz gegenüber einem anderen FitnessMultiObjMin-Objekt zu testen, mit dem Ziel einen möglichst kleinen Fitness-Vektor zu finden. Um diese Klasse nutzen zu können, sind zwei Probleme zu lösen, einerseits müssen alle Kriterien so berechnet werden, dass eine bessere Güte mit einem kleineren Fitness-Wert einhergeht, andererseits müssen die Kriterien so im Vektor integriert werden, dass der Dominanz-Test die gewünschte Richtung vorgibt.

Die notwendigen Kennzahlen lassen sich in mehrere Schichten einteilen:

- Abstand zu einer gültigen Codierung,
- Abstand zu einer gültigen Platzierung und
- die Optimierungskriterien.

4.3.5.1. Abstand zu einer gültigen Codierung

Bei Vorliegen eines gültigen Genotyps sollen die Fitness-Werte den Wert 0 annehmen, ansonsten sollen die Fitness-Werte besser (also kleiner) sein, je näher der Genotyp einer gültigen Codierung ist.

Für die Abweichung eines Bitstring-Genotyps $x = (X_{1,1} \dots X_{n,n})$ von einem gültigen Bitstring-Genotyp lässt sich folgende Formel verwenden:

$$\sum_{i=1}^{n} \left| 1 - \sum_{j=1}^{n} X_{i,j} \right| \tag{4.6}$$

Pro vCPS-Anwendung soll genau ein Host ausgewählt sein, in diesem Fall nimmt die Gesamtsumme den Wert 0 an. Sind mehr oder weniger Bits auf *wahr* gesetzt, wird die Anzahl dieser summiert und vergrößern (und verschlechtern somit) den Fitness-Wert. Da nicht relevant ist, ob zu viele oder zu wenige Bits auf *wahr* gesetzt sind, wird der Betrag der Abweichung genutzt. Alternativ könnte man eine quadratische Fehlerformel verwenden, die größere Abweichungen stärker berücksichtigt.

Für die Abweichung eines VLG-Schedules $(g_k)_{k=1,\dots,l}$ mit $l \in \mathbb{N}$ von einem gültigen VLG-Schedule kann die folgende Formel verwendet werden:

$$n - \sum_{i=1}^{n} \min \left(1, \sum_{(a,b) \in (g_k)_{k=1,\dots,l}} e(a,i) \right) \text{ mit } e(x,y) = \begin{cases} 1 & \text{falls } x = y \\ 0 & \text{sonst} \end{cases} \tag{4.7}$$

Jede vCPS-Anwendung muss mindestens einem Host zugewiesen sein, weitere Vorkommen sollen den Fitness-Wert jedoch nicht beeinflussen. Für jede vCPS-Anwendung i werden also alle Tupel (a, b) überprüft, ob sie für die vCPS-Anwendung relevant sind (ob also $a = i$ gilt). Falls dies der Fall ist, wird das Tupel mithilfe der Abbildung e gezählt, ansonsten nicht. Da nur maximal ein Tupel gezählt werden soll, wird von der Summe aller Tupel das Minimum mit 1 gebildet.

4.3.5.2. Abstand zu einer gültigen Platzierung

Die notwendigen Kennzahlen, um die Gültigkeit einer Platzierung zu prüfen, werden von MAST berechnet. Einerseits muss die Auslastung von allen Prozessoren und Netzwerken jeweils unter 100% liegen. Andererseits muss für jede gesetzte Zeitbeschränkung die Einhaltung überprüft werden. Insgesamt kommen so in der vorgestellten MAST-Modellierung $2+4n$ Kennzahlen zusammen; jeweils 3 Zeitbeschränkungen müssen pro vCPS-Anwendung überprüft werden, außerdem gibt es n Hosts und 2 Netzwerke, die auf eventuelle Überlastung geprüft werden müssen.

Auch hierbei ist zu beachten, dass die Kennzahlen bei Vorliegen einer gültigen Platzierung 0 betragen sollen. Eine Prozentzahl p der Auslastung wird als $\max(0, p - 100)$ in den Fitness-Vektor eingetragen. Für einen Zeitbedarf t eines MAST-Events mit der Zeitbeschränkung d wird $\max(0, t - d)$ in den Fitness-Vektor eingetragen.

4.3.5.3. Optimierungskriterien

Zu den Optimierungskriterien zählen wir die Kriterien, deren Anwendung nur auf gültige Lösungskandidaten sinnvoll ist. Für das vorliegende Problem ist das die Ende-zu-Ende-Latenz, der Umfang der notwendigen Migrationen und die benötigten Hosts. Diese Kriterien sind graduell und auch hierbei muss ein kleinerer Wert im Fitness-Vektor für eine bessere Lösung stehen. Die notwendigen Daten liefert zum Teil MAST, die Ende-zu-Ende-Latenz als *Worst_Global_Response_Times*, und zum Teil können sie aus dem vCPS und den gegebenen Platzierungen gewonnen werden, nämlich die Anzahl der benötigten Hosts und der Umfang der notwendigen Migrationen.

Die Ende-zu-Ende-Latenz kann direkt als Fitness-Wert genutzt werden, denn ein höherer Wert entspricht einer schlechteren Fitness.

Die gleiche Logik gilt für den Umfang der notwendigen Migrationen, sodass folgende Berechnung für diesen Fitness-Wert herangezogen werden kann:

$$\sum_{i=1}^{n} m(i) \cdot H_i^{\text{init}} \quad \text{mit } m(i) = \begin{cases} 1 \text{ falls vCPS-Anwendung } i \text{ migriert wird} \\ 0 \text{ sonst} \end{cases} \tag{4.8}$$

Die Anzahl der benötigten Hosts entspricht der Anzahl der berechneten Partitionen und kann ebenso direkt als Fitness-Wert benutzt werden.

4.3.5.4. Integration in einen gemeinsamen Vektor

Alle vorgestellten Kennzahlen können in einem Vektor vereint werden, sie müssen jedoch schichtenweise betrachtet werden. Standardmäßig müssen alle Fitness-Werte auf $\mathcal{M}$ stehen, wobei $\mathcal{M}$ eine sehr große Zahl ist, dann werden die Schichten nacheinander betrachtet und die nächste Schicht wird erst betrachtet, wenn alle Fitness-Werte der aktuellen Schicht ihren optimalen Wert 0 erreicht haben. Mit dieser Methodik wird erreicht, dass eine gültige Codierung immer eine ungültige Codierung dominiert, dass ein gültiger echtzeitfähiger Lösungskandidat immer nicht-echtzeitfähige Lösungskandidaten dominiert und dass in der dritten Schicht eine Optimierung der gewünschten Kriterien stattfindet. Außerdem ist hervorzuheben, dass sich durch die Kontinuität alle Kennzahlen in allen Schichten immer eine Suchrichtung ermitteln lässt.

Im Laufe dieses Abschnitts wurden alle Kriterien vorgestellt, welche auf verschiedenen Kennzahlen basieren. Einige der Kennzahlen lassen sich direkt von einem Lösungskandidaten ableiten, für viele andere muss die jeweilige Lösung erst durch MAST verarbeitet werden, um z. B. die Ende-zu-Ende-Latenz zu erhalten. Die genaue Integration von MAST in den EA zeigt der spätere Abschnitt 4.4.

4.3.6. Multi-kriterieller evolutionärer Algorithmus

Wie bereits in Abschnitt 3.3.4.2 erläutert, muss ein evolutionärer Algorithmus, der hinsichtlich mehrerer gegenläufiger Kriterien optimieren soll, nicht nur einfach auf Dominanz testen. Gesucht wird eine möglichst breite Menge an unterschiedlichen gleichwertigen Lösungen. Damit diese Menge auch tatsächlich gefunden wird, sind besondere Algorithmen nötig.

Eine umfangreiche Übersicht über sogenannten MOEA (*multi-objective evolutionary algorithms*) gibt Kunkle [Kun05], namentlich zu nennen sind z. B. NSGA II [Deb+00] (in Open BEAGLE implementiert), SPEA2 [ZLT01] (im PISA-Framework implementiert), SPEA2+ [Kim+04] und PESA-II [Cor+01].

Schon die Namensgebung lässt erahnen, dass die meisten Algorithmen in der Vergangenheit überarbeitet wurden und mittlerweile in einer zweiten Version vorliegen. Regelmäßig lässt sich in den Schlussbemerkungen der referenzierten Aufsätze nachlesen, dass die Algorithmen im Vergleich miteinander oftmals auf bestimmten Problemen leicht besser sind als Vergleichsalgorithmen, aber kein Algorithmus auf allen Problemen besser ist als alle anderen. Dies ist nach dem *No free lunch*-Theorem auch zu erwarten und wird durch einen breiteren Vergleich von APA, NSGA II, PAES und SPEA von Groşan und Dumitrescu unterstützt, deren Ergebnisse kaum Unterschiede zwischen den Algorithmen zeigen [GD02]. Ohne exakte Analyse des Problems, welche gerade bei komplexen Suchräumen kompliziert ist, oder einer Evaluation unterschiedlicher Algorithmen auf dem zu untersuchenden Problem, lässt sich kein besonders passender Algorithmus auswählen.

In Open BEAGLE sind NSGA II und NPGA2 [EMH01] implementiert. Aufgrund der größeren Bekanntheit und der vorigen allgemeinen Argumentation hinsichtlich MOEA wurde entschieden für den weiteren Verlauf der Arbeit NSGA II zu nutzen [Deb+00].

NSGA II (non-dominated sorting genetic algorithm) ist ein elitärer Algorithmus, der auf nicht-dominiertem Sortieren und Nischenbildung basiert. Im einzelnen lassen sich die Eigenschaften wie folgt detailliert erklären:

elitärer Algorithmus Ein elitärer Algorithmus unterscheidet sich von einem nicht-elitären Algorithmus dadurch, dass das beste Individuum einer Eltern-Population in die Kind-Population übernommen wird, wenn es nach wie vor zu den besten Individuen zählt. Demgegenüber wird bei nicht-elitären Algorithmen die Eltern-Population verworfen, und die neue Generation besteht nur aus neuen Individuen.

nicht-dominiertes Sortieren Gesucht ist eine Teilmenge der Pareto-Front, der man sich mit jeder Iteration weiter annähern muss. Ein nicht-dominiertes Sortierverfahren sortiert eine Menge dafür in Schichten $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \ldots)$, wobei die Menge $\mathcal{F}_1$ alle Elemente enthält, die von keinem Element dominiert werden, die Menge $\mathcal{F}_2$ alle Elemente enthält, die von keinem Element (ohne $\mathcal{F}_1$) dominiert werden, usw. Wenn man die neue Population anhand der Schichten aufsteigend generiert, enthält die Population immer die am wenigsten dominierten Individuen.

Nischenbildung Ein MOEA soll nicht nur eine beliebige Teilmenge der Pareto-Front ermitteln, sondern eine möglichst breite Repräsentation davon. Eine Möglichkeit um dies zu erreichen ist es sogenannte Nischen zu bilden. Eine Menge gleichwertiger Lösungen wird nach ihrer absteigenden Dichte

sortiert, d. h. anhand ihrer Nähe zueinander. Wählt man dann die ersten Elemente der sortierten Menge, erhält man eine breite Abdeckung der aktuellen Lösungen. Eine Garantie mit dieser Methode schließlich auch eine breite Repräsentation der Pareto-Front zu erhalten, gibt es allerdings nicht.

Die Laufzeit von NSGA II wird vom nicht-dominierten Sortieren bestimmt und beträgt $\mathcal{O}(mN^2)$ pro Iteration, wobei m die Anzahl der Kriterien und N die Größe der Population ist.

Der NSGA II-Algorithmus ist in Open BEAGLE als *ReplacementOperator* implementiert und entspricht somit der Umwelt-Selektion in Abbildung 3.1 er ist aber auch für die Generation neuer Individuen zuständig. Aus den Individuen einer Eltern-Population der Größe N werden durch festlegbare Selektions-, Rekombinations- und Mutationsoperatoren weitere N Kind-Individuen generiert. Die Gesamtmenge aller Eltern- und Kind-Individuen wird dann durch den NSGA II-Algorithmus verarbeitet, um eine neue Population der Größe N zu selektieren.

4.4. Integration von Ada und C

Um ein Modell mithilfe einer der Leistungsbewertungsmethoden bewerten zu können, müssen die im EA berechneten Daten in ein für das Verfahren verständliches Modell überführt werden. Um diesen Schritt möglichst schnell vollziehen zu können, wurde das MAST-Tool direkt in den EA integriert. Da MAST in der Programmiersprache Ada geschrieben ist, wohingegen der EA in der Programmiersprache C++ entwickelt wurde, musste eine Integration dieser beiden Programmiersprachen stattfinden.

Ada ist eine strukturierte, statisch getypte Programmiersprache, die Programmiersprachen wie Pascal ähnelt. Sie zeichnet sich vor allem dadurch aus, dass der Compiler bereits viele Fehler beim Kompilieren sucht, anstatt Laufzeit-Fehler zu generieren. Außerdem lässt sie sehr feingranulare Entscheidungen zu, sodass der Programmierer wahlweise jeden Aspekt des generierten Programms steuern kann, um die Ausführung vorhersagbar und deterministisch zu machen.

Um Quellcode in der Programmiersprache Ada zu kompilieren bietet sich vor allem der Compiler *GNAT* an. Die ursprüngliche Finanzierung der Entwicklung erfolgt durch die *United States Air Force* unter der Bedingung, dass der Quellcode des Compilers unter der GNU GPL lizenziert wird. In den folgenden Jahren fand der Compiler Einzug in die *GNU Compiler Collection*, und dürfte damit der meistgenutzte Compiler für Ada sein. Kommerzielle Unterstützung bietet das Unternehmen *AdaCore* an. Sie offerieren mit *GNAT*

Pro eine vollständige Entwicklungsumgebung für Ada samt Compiler, IDE, Benchmark-Tools u. a., stellen aber auch eine Version namens GNAT GPL für die Entwicklung freier und akademischer Software zum kostenlosen Download zur Verfügung.

Prinzipiell lässt sich Quellcode von Ada und C im Rahmen einer gemeinsamen Entwicklung integrieren. Entsprechende Grundsteine wurden bereits im *Ada Reference Manual*[10] gelegt und fanden auch in GNU GNAT Einzug[11].

4.4.1. Kompilierung von Ada Quellcode

Für den Fall, dass man nur Ada-Quellcode im Rahmen eines größeren C-Projekts nutzen möchte, bietet die GNU GNAT Dokumentation einen guten Einblick in Möglichkeiten. Im Allgemeinen bietet es sich an den Ada Quellcode in eine Bibliothek zu kompilieren, gegen welche der C-Objektcode dann gelinkt wird. Das Kompilieren eines Ada-Projekts erfolgt z. B. mithilfe des *GNAT Project Managers*[12] und einer Projektdatei, die alle notwendigen Einstellungen enthält. Der eigentliche Compile-Vorgang kann dann mit dem Befehl

```
gnatmake -P <Projektdatei>.gpr
```

gestartet werden. Eine beispielhafte Projektdatei befindet sich in Listing 4.1, deren Schlüsselworter folgende Bedeutungen haben:

Library_Name Der Name der zu kompilierenden Ada-Bibliothek. Er bestimmt auch den Namen der Initialisierungs-/Finalisierungs-Funktionen (siehe *Library_Auto_Init*).

Source_Dirs Der Ordner in dem der Ada-Quellcode zu finden ist.

Library_Auto_Init Die Ada-Bibliothek wird mit Init- und Final-Code kompiliert. Dieser kann automatisch ausgeführt werden (wenn die Plattform dies unterstützt) oder muss manuell vor eigenem Ada-Quellcode aufgerufen werden. Mit der beispielhaften Projektdatei heißen die beiden Funktionen *adainit* und *adafinal*.

Library_Interface Enthält den Ada Paket-Namen des Pakets, der das Interface der Bibliothek darstellt.

Library_Kind Spezifiziert den Typ der Bibliothek, *dynamic* oder *static*.

[10] http://www.adaic.org/resources/add_content/standards/05rm/html/RM-B-3.html
[11] https://gcc.gnu.org/onlinedocs/gnat_ugn/Mixed-Language-Programming.html
[12] http://docs.adacore.com/gprbuild-docs/html/share/gnat_project_manager.html

```
project Ada is
   for Library_Name use "ada";
   for Source_Dirs use ("mast_analysis", "mast_lib_interface", "mast_utils");
   for Library_Auto_Init use "true";
   for Library_Interface use ("simple_cpp_interface");
   for Library_Kind use "dynamic";
   for Library_Dir use "dll";
   -- for Library_Options use ("-Wl,--out-implib,dll\libada.lib");
   for Object_Dir use "build";
   for Library_Ali_Dir use "ali";
end Ada;
```

Listing 4.1: GNAT Project Manager Projektdatei

Library_Dir Ausgabeordner der vom Compiler erstellten Dateien.

Library_Options Hier können weitere Optionen an den Linker übergeben werden. Wenn man eine dynamische Bibliothek unter Windows einbinden möchte, benötigt man dafür regelmäßig eine sogenannte *Import Library* (Dateiendung *.lib*), welche Informationen über die in der DLL-Datei enthaltenen Funktionalitäten enthält. Diese Datei lässt man am besten durch GNAT erstellen, die dafür notwendigen Linker-Parameter lassen sich dem Listing entnehmen.

Object_Dir Dies ist ein weitere Ausgabeordner des Compilers, in dem kompilierte Objektdateien erstellt werden.

Library_Ali_Dir In diesen Ordner werden ALI-Dateien (*Ada Library Information*) erstellt. Sie enthalten Informationen, die der Compiler beim Kompilieren gesammelt hat, z. B. Abhängigkeiten zwischen Paketen.

4.4.2. Integration von Kontrollflüssen

Prozeduren oder Funktionen in Ada müssen zum Exportieren mithilfe des Schlüsselworts *Pragma* markiert werden. GNAT erstellt dann Bytecode, der kompatibel zur angegebenen Programmiersprache ist. Listing 4.2 zeigt eine Ada Paket-Spezifikation mit einer Prozedur *output_result*, die zum Exportieren an C markiert wurde. Die entsprechend markierten Prozeduren/Funktionen müssen in C wie üblich mit dem *extern* Schlüsselwort definiert werden. Eine beispielhafte Definition befindet sich in Listing 4.3, die auch mögliche Definitionen der Init-/Final-Bibliotheks-Funktionen enthält.

```
1 package Simple_Cpp_Interface is
2    procedure output_result;
3    pragma Export (C, output_result);
4 end Simple_Cpp_Interface;
```

Listing 4.2: Beispielhafter Export einer Ada Prozedur

```
1 extern "C" {
2   void adainit(void);
3   void adafinal(void);
4
5   void output_result();
6 }
```

Listing 4.3: Extern C Definition zur Ada Paket-Spezifikation in Listing 4.2

4.4.3. Integration von Daten

Um nicht nur Methoden übergreifend aufrufen zu können, sondern auch Parameter und Rückgabewerte teilen zu können, müssen weitere Anstrengungen unternommen werden. Beiden oben genannten Dokumente zur Integration von Ada und C mangelt es diesbezüglich an größeren praxis-relevanten Beispielen, sodass man sich vor allem auf das *Ada Reference Manual* verlassen muss, welches aus Teilen der Ada-Paket-Spezifikation von *Interfaces.C* besteht, gespickt mit kurzen Erklärungen. Das Paket *Interfaces.C* enthält diverse Utility-Funktionen in Ada zum Umgang mit C-Datentypen.

Ein Großteil der primitiven Datentypen passiert die Schnittstelle zwischen Ada und C problemlos, nur bei Verwendung eines Bool-Typs in Ada weist GNAT daraufhin, dass man in C den Datentyp *char* verwenden soll, denn ein Bool ist in Ada eine Enumeration. Die Umwandlung zwischen beiden Repräsentationen kann jedoch einfach erfolgen, siehe Listing 4.4.

Anders gestaltet sich die Situation bei Strings und Arrays, da diese beliebig lang sein können. C-Strings enden standardmäßig mit *NULL*, was in Ada genutzt werden kann, um das Ende zu erkennen. Für diesen Zweck gibt es bereits die Funktion `Value`, welche einen *chars_ptr* in den Ada-Typ *char_array* konvertiert. Arrays enden nicht mit einem bestimmten Wert, sodass diese Variante dabei nicht anwendbar ist. GNAT listet beim Kompilieren eines betroffenen Falls den Hinweis, dass Array-Grenzen zusätzlich übergeben werden müssen. Eine alternative Lösung ist es ein Array eben doch auch mit *NULL* enden zu lassen und beim Durchlaufen auf diesen Wert zu testen. Diese Variante wurde

```
inline bool mast_boolean_value(char ada_value) {
  if (ada_value == 0) {
    return false;
  } else if (ada_value == 1) {
    return true;
  } else {
    throw std::runtime_error("ada boolean contains unexpected bit representation");
  }
}
```

Listing 4.4: Umwandlung eines Ada Booleans in einen C++ Bool

in der vorliegenden Arbeit genutzt, erfordert zwar mehr Aufmerksamkeit beim Programmieren, funktioniert ansonsten jedoch einwandfrei.

4.4.4. Weitere MAST-spezifische Besonderheiten

Wir kennen nun die regelmäßigen Probleme, die bei der Verbindung von Ada mit C auftreten können, und auch mögliche Lösungen. Darüber hinaus gab es noch MAST-spezifische Probleme, die es im Zuge der Integration zu lösen galt.

Im vorigen Abschnitt haben wir eine funktionale Schnittstelle zwischen Ada und C kennengelernt, welche offensichtlich voraussetzt, dass in Ada Funktionen existieren, die von C aus aufgerufen werden können. Dies war bei MAST nicht der Fall. Die Software-Architektur sieht so aus, dass innerhalb des Parsers für Dateien im MAST-Dateiformat direkt der Quellcode steht, der die eingelesenen Daten in die internen Datenstrukturen schreibt. Für unseren Anwendungszweck wäre es einfacher gewesen, wenn der Parser Funktionen einer Abstraktionsschicht zur Interaktion mit den internen Datenstrukturen aufruft. Auch die Ausführung eines Algorithmus auf die interne Datenstruktur ist nicht isoliert, sondern fest mit der CLI-Anwendung verbunden.

Eine optimale Lösung des Problems zieht die zeitaufwändige Refaktorierung großer Teile der MAST-Anwendung nach sich, weshalb darauf im Rahmen dieser Arbeit verzichtet wurde. Die gewählte Alternative besteht darin die betroffenen Code-Fragmente in einer neuen funktionalen Abstraktionsschicht zu duplizieren.

Ein weiteres Problem besteht darin, dass Ada standardmäßig keine Garbage-Collection (GC) durchführt. Der Ada-Standard verbietet zwar keinen GC, allerdings ist die Nachfrage danach offenbar so klein, dass wenige Compiler einen GC standardmäßig enthalten. Eine mögliche Begründung kann sein, dass die Ausführung eines GC oftmals nicht-deterministisch ist und deren benötigte Rechenzeit schlecht vorhersagbar ist. Dies kollidiert mit der häufigen Intention

von Ada-Programmierern eine deterministische und vorhersagbare Software zu programmieren.

Es existiert darüber hinaus die Möglichkeit händisch Speicher wieder freizugeben, welche jedoch von den MAST-Entwicklern im Analyse-Tool nicht genutzt wurde. Der Lebenszyklus einer MAST-Analyse-Instanz war bisher überschaubar:

1. Einlesen einer MAST-Datei,
2. Ausführen eines Algorithmus,
3. Ausgabe der Ergebnisse und
4. Terminierung.

Da zur Terminierung belegter Speicher wieder freigegeben wird, bestand bisher keine Notwendigkeit ihn händisch freizugeben. Im Rahmen eines EA, bei dem dieser Zyklus mehrere tausend Male durchlaufen werden muss, sind die Anforderungen selbstverständlich anders. Innerhalb einer bestehenden Applikation die Speicherverwaltung zu überarbeiten und, noch schwieriger, dabei kein Speicherleck zu übersehen, ist auch ein sehr zeitaufwändiges Unterfangen. Um nicht zu viel Zeit auf diesen Aspekt aufbringen zu müssen und trotzdem eine akzeptable Lösung zu erhalten, wurde die Ausführung von MAST in einen neuen Prozess ausgelagert. Wenn also ein Modell mithilfe von MAST evaluiert werden soll, wird zuerst *shared memory* für die Ergebnisse angelegt, danach wird der Systemaufruf *fork* ausgeführt, um einen neuen Prozess zu erhalten. Der Kind-Prozess überführt nun das Modell mithilfe der aus Ada importierten Funktionen in die dortige interne Datenstruktur, führt einen Algorithmus aus, und schreibt die Ergebnisse in den *shared memory*. Danach terminiert dieser Prozess und mit ihm wird jeder von Ada belegter Speicher freigegeben, die Ergebnisse stehen dann im Eltern-Prozess zur weiteren Verwendung zur Verfügung. Da der Systemaufruf *fork copy-on-write* nutzt, erschien diese Lösung ein gangbarer Kompromiss zwischen Laufzeit und Arbeitsaufwand.

In diesem Kapitel wurde eine vollständige Implementierung eines evolutionären Algorithmus vorgestellt, welcher das Platzierungsproblem von echtzeitfähigen virtuellen Maschinen löst. Dafür wurde ein Framework ausgewählt und installiert, eine Anwendung entwickelt, die auf dem Framework basiert, extra für das Problem Datentypen und Operatoren entworfen, und MAST als Leistungsbewertungsverfahren zum Sicherstellen von Echtzeitfähigkeit integriert. Eine offene Frage ist wie gut oder schlecht der entwickelte EA ist, weshalb sich das nächste Kapitel der Evaluation des EA widmet.

Kapitel 5

Evaluation des evolutionären Algorithmus

Evolutionäre Algorithmen zählen zu den Approximationen, deshalb ist es interessant zu untersuchen, welche Güte man bei der Benutzung der Approximation zu erwarten hat. Unter Zuhilfenahme der LP-Formulierung, lässt sich der im vorigen Kapitel entwickelte EA bewerten. Darüber hinaus lassen sich weitere Eigenschaften am EA unabhängig von einer optimalen Lösung evaluieren. Beide Aspekte werden im aktuellen Kapitel untersucht und vorgestellt, wofür zuerst Beispiel-Modelle erstellt werden müssen.

5.1. Test-Modelle

Wir nutzen zufällig erstellte Modelle für die Evaluation. Um sinnvolle Modelle zu erstellen, werden für die unterschiedlichen Parameter C_i, T_i, B_i^{in}, B_i^{out}, D_i, H_i^{init}, H_i^{regular} einer vCPS-Anwendung i unterschiedliche Wahrscheinlichkeitsverteilungen zugrunde gelegt.

Die genutzte Wahrscheinlichkeitsdichtefunktion der Perioden T_i ist in Abbildung 5.1 gezeigt. Sie zählt zu den Gruppen der Mischverteilungen, besteht also aus zwei Wahrscheinlichkeitsverteilungen, die mit unterschiedlichem Gewicht berücksichtigt werden. Wir wählen die beiden Normalverteilungen $\mathcal{N}(6, 2)$ mit der Gewichtung 0.8 und $\mathcal{N}(60, 20)$ mit der Gewichtung 0.2, welche mehrere kleinere und wenige große vCPS-Anwendungen repräsentieren. Vorstellen kann man sich ein derartiges vCPS-System als Zusammenarbeit von vielen kleinen Sensorkomponenten und wenigen größeren Komponenten, die Daten aggregieren. Eine Realisierung einer derartig verteilten Zufallsvariable erreicht man, indem zuerst unter Berücksichtigung der Gewichte zufällig eine der beiden Wahrscheinlichkeitsverteilungen ausgewählt wird, und diese dann realisiert wird. Der Erwartungswert ergibt sich als gewichtete Summe der einzelnen Erwartungswerte also $\mathbb{E}[X] = 0.8 \cdot 6 + 0.2 \cdot 60$, wenn X wie in Abbildung 5.1 verteilt ist.

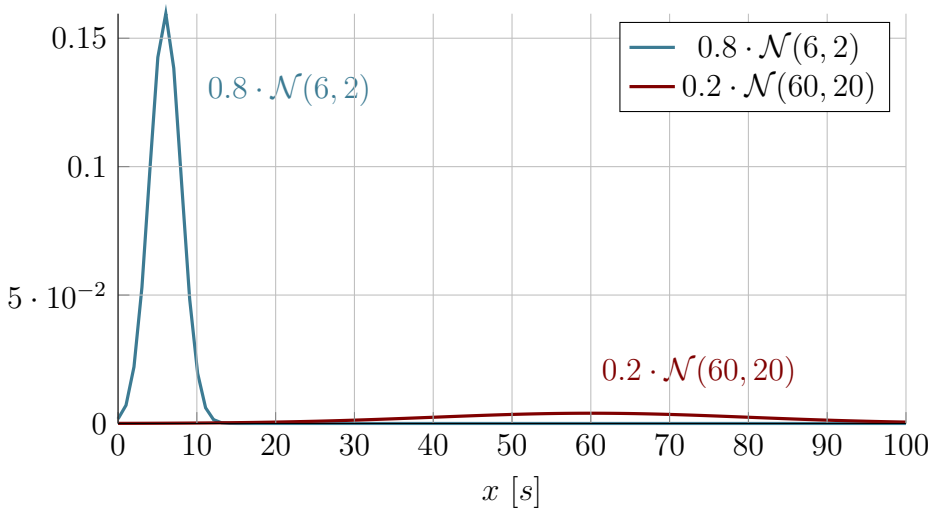

Abbildung 5.1.: Dichtefunktion der Perioden als Mischverteilung.

Als Zeitbeschränkung D_i nutzen wir pro generierter vCPS-Anwendung die jeweilige Periode T_i. Dies ist bei Task-Systemen in der Literatur auch eine gängige Annahme.

Da die maximale Ausführungsdauer C_i die Periodendauer nicht überschreiten soll, und es eine plausible Annahme ist, dass Periode und Ausführungsdauer voneinander abhängen, wird die Dichtefunktion für die maximale Ausführungsdauer in Abhängigkeit von einer Periodendauer P wie in Abbildung 5.2 festgelegt. Sie entspricht einer Gauss-Verteilung $\mathcal{N}\left(\frac{P}{3}, \frac{P}{8}\right)$, die während der Realisierung auf das Interval $]0, P]$ beschränkt wird. Der Erwartungswert einer derartig verteilten Zufallsvariable ist $\mathbb{E}[X] = \frac{P}{3}$.

Bezüglich der Ein- und Ausgabe einer vCPS-Anwendung, also B_i^{in} und B_i^{out}, gehen wir von der moderaten Anforderungen aus, dass eine vCPS-Anwendung regelmäßig ca. 64 KB erwartet und verschickt. Das entspricht ca. einer CAN-Nachricht, wobei im Rahmen der vorliegenden Arbeit davon ausgegangen wird, dass eine Nachricht genau so groß ist wie die darin enthaltenen Daten, insofern sind konkrete Nachrichten-Typen nicht von Bedeutung. Um die gewünschte Verteilung zu erreichen, nutzen wir eine Logarithmische Normalverteilung $\mathcal{LN}(0, 0.125)$, deren x-Werte mit dem Faktor 64 bearbeitet werden. Die Dichtefunktion ist in Abbildung 5.3 zu sehen und der Erwartungswert beträgt $\mathbb{E}(X) = exp\left(\mu + \frac{\sigma^2}{2}\right) = exp\left(0 + \frac{0.125}{2}\right) \sim 1.0644 \rightsquigarrow 1.0644 \cdot 64 \text{ KB} \sim 68 \text{ KB}$. Während der Realisierung einer Zufallsvariable wird das Ergebnis gerundet, um eine gerade Anzahl Bits zu erhalten.

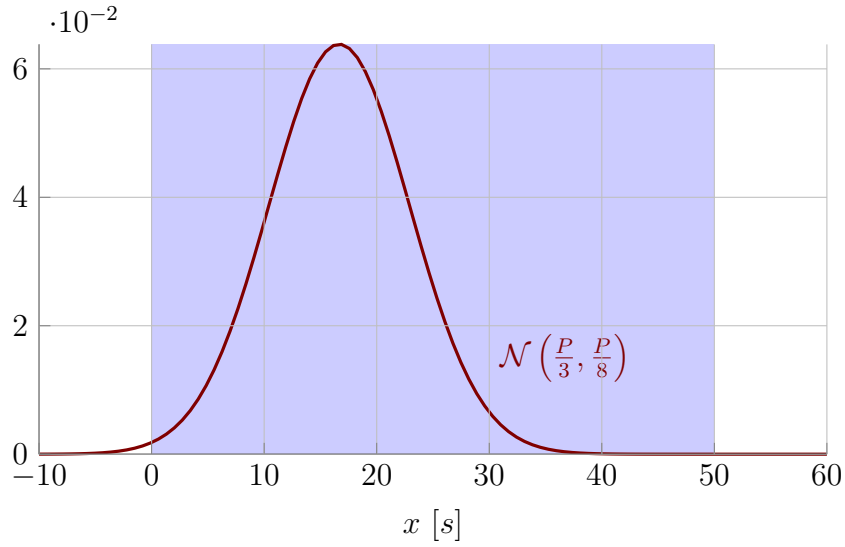

Abbildung 5.2.: Dichtefunktion der max. Ausführungsdauer für eine Periode $P = 50$.

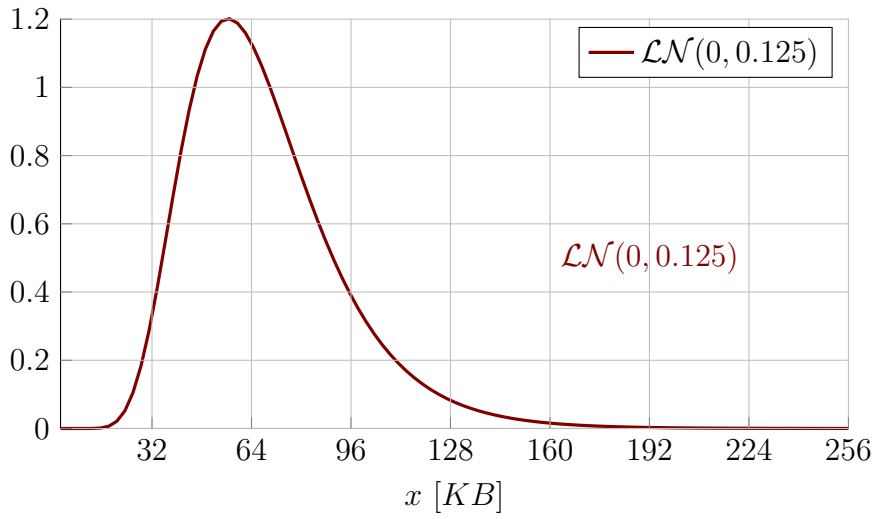

Abbildung 5.3.: Dichtefunktion von B^{in} und B^{out} mit skalierter x-Achse.

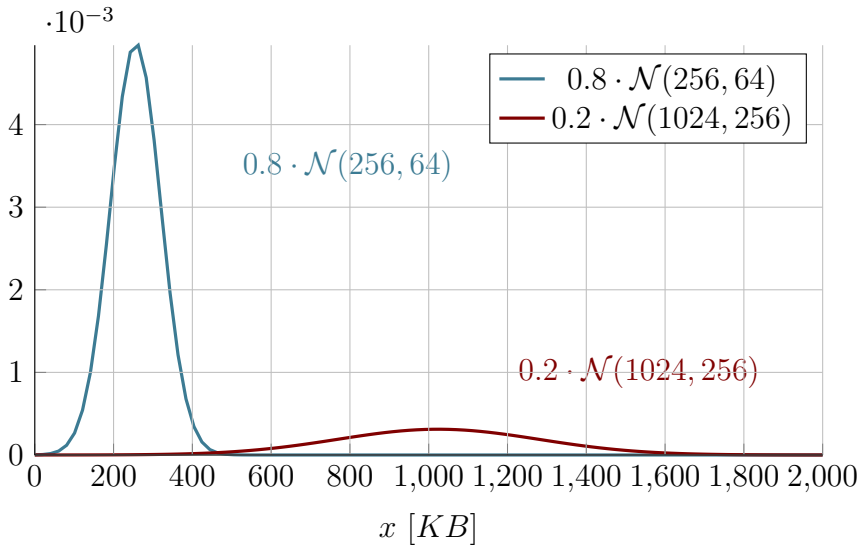

Abbildung 5.4.: Dichtefunktion von H^{init}.

Der initiale Speicherbedarf einer VM ist auch von der Größe einer vCPS-Anwendung abhängig. Da wir für Perioden bereits unterschiedliche Dichtefunktionen mit der zugrundeliegenden Interpretation von unterschiedlich großen vCPS-Anwendungen kombiniert haben, wird für H_i^{init} der gleiche Ansatz erneut genutzt. Als Komponenten kommen hier $\mathcal{N}(256, 64)$ mit dem Faktor 0.8 und $\mathcal{N}(1024, 256)$ mit dem Faktor 0.2 zum Einsatz, sodass wir häufiger von kleinen vCPS-Anwendungen ausgehen, die deshalb weniger initialen Speicher benötigen, aber auch ein paar große vCPS-Anwendungen erhalten, die initial mehr Speicher belegen. Abbildung 5.4 zeigt die Mischverteilung und der Erwartungswert beträgt hier $\mathbb{E}[X] = 0.8 \cdot 256 + 0.2 \cdot 1024 = 409.6$. Wie auch bei der vorigen Dichtefunktion, wird im Rahmen der Realisierung das Ergebnis gerundet, um auf eine gerade Anzahl Bits zu kommen.

Für den Wert H_i^{regular} nutzen wir die Annahme von Clark u. a., die feststellten, dass sich im Rahmen der Ausführung einer VM regelmäßig nur 50% der Seiten verändern [Cla+05]. Wir legen deshalb $H_i^{\text{regular}} = \frac{H_i^{\text{init}}}{2}$ fest.

Weiter muss für die Migration der aktuelle Host einer VM bekannt sein, von dem die VM unter Umständen weg migriert werden muss. Dazu wird eine Gleichverteilung über den Bereich $[-1, n-1]$ genutzt, wobei n die Anzahl von Hosts ist (wobei wir annehmen, dass die Anzahl von Hosts gleich der Anzahl von vCPS-Anwendungen ist). Eine Abbildung der Wahrscheinlichkeitsdichtefunktion ist in Abbildung 5.5 zu sehen, die x-Achse wird im Vergleich zu einer

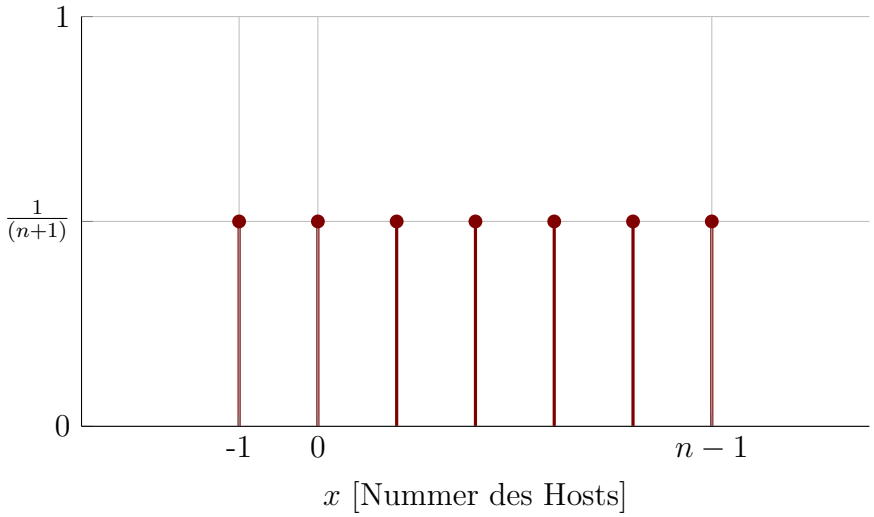

Abbildung 5.5.: Wahrscheinlichkeitsfunktion der Vorbelegung einer VM abhängig von einer Host-Anzahl n, mit Anpassung der x-Achse um den Wert -1 zu ermöglichen.

normalen diskreten Gleichverteilung verschoben, um den Wert -1 für eine neu eingeführt vCPS-Anwendung in das System zu ermöglichen. Der Erwartungswert einer derartig verteilten Zufallsvariable X lässt sich berechnen als $\mathbb{E}[X] = \frac{n+2}{2} - 2$.

Für die Datenabhängigkeit haben wir in den Annahmen festgelegt, dass nur einfach verkettete vCPS-Anwendungen berücksichtigt werden. Wir nutzen zur Ermittlung der Länge einer derartigen Kette eine geometrische Verteilung mit $p = 0.7$, womit 70% der Ketten die Länge 1 erhalten. Der Erwartungswert ergibt sich als $\mathbb{E}[X] = \frac{1}{p} = \frac{1}{0.7} \sim 1.42$. Eine Abbildung der Dichtefunktion befindet sich in Abbildung 5.6.

Außerdem setzen wir innerhalb einer solchen Kette die Periode und die Zeitbeschränkung für alle vCPS-Anwendungen auf den gleichen Wert. Dies kann man so interpretieren, dass wir verhindern wollen, dass Buffer zwischen den vCPS-Anwendungen über- oder leerlaufen.

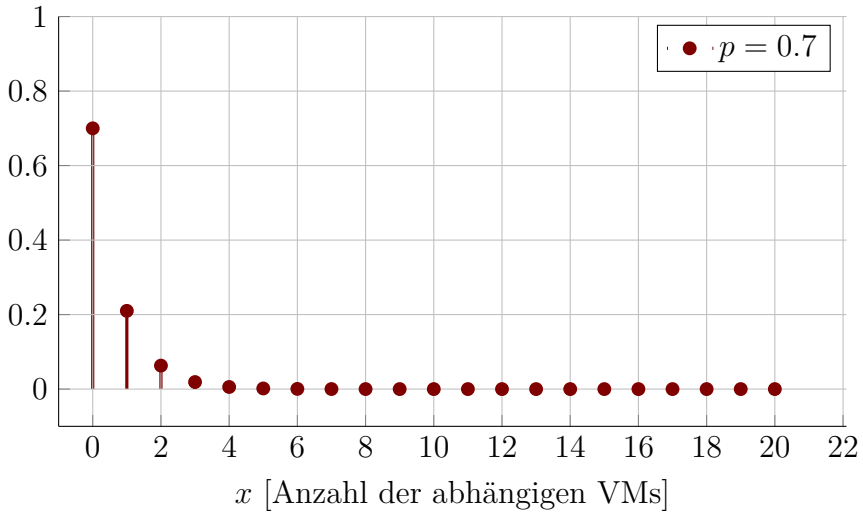

Abbildung 5.6.: Wahrscheinlichkeitsfunktion der Anzahl der abhängigen vCPS-Anwendungen

5.2. EA-Lösungen im Vergleich zur optimalen Lösung

Im ersten Teil der Evaluation sollen vom evolutionären Algorithmus berechnete Lösungen mit optimalen Lösungen verglichen werden, welche durch LP-Formulierungen bestimmt werden. Da die LP-Formulierungen exponentiell groß sind, muss ein verhältnismäßig kleines Modell genutzt werden. Mithilfe der im vorigen Abschnitt vorgestellten Wahrscheinlichkeitsverteilungen wurde das Modell *Modell10* generiert, welches aus 10 vCPS-Anwendungen besteht und ohne Ein-/Ausgabe und Datenabhängigkeiten generiert wurde – denn diese Kennzahlen können wir nicht in unserer LP-Formulierung abbilden. Das *Modell10* liegt dieser Arbeit in Anhang A.1 bei.

Die zu optimierenden Kriterien sind gegenläufig, lassen sich somit nicht einfach in einem Wert vereinen. Der EA berücksichtigt dies zwar im Rahmen des verwendeten MOEAs, aber die LP-Modellierung kann einfacher einzelne Kriterien optimieren. Auch hinsichtlich Vergleichbarkeit werden wir im Folgenden deshalb drei Kriterien jeweils einzeln zwischen LP und EA vergleichen, nämlich „Anzahl der verwendeten Hosts“, „Umfang der Migrationen“ und beispielhaft die „Ende-zu-Ende-Latenz von vCPS-Anwendung 0“.

Als Abbruchkriterium nutzen wir „maximale Anzahl an Generationen“ mit dem Parameter 15. Um außerdem verlässlichere Statistiken zu erhalten, wurde

jeder Durchlauf 15 Mal gestartet, und wir betrachten im Folgenden Boxplots oder Durchschnittswerte dieser 15 Ausführungen.

Die drei weiter oben vorgestellten Codierungen Binärstring, FL-Schedule und VLG-Schedule können prinzipiell unterschiedlich gute Ergebnisse liefern, deshalb wurden die drei Codierungen auf dem gleichen Modell ausgeführt und verglichen. Selbst bei diesem kleinen Modell, schafft es der EA mithilfe der Binärcodierung nicht eine einzige gültige Lösung zu ermitteln. Hauptproblem dieser Codierung ist, dass mit steigender Länge des Bitstrings die Anzahl an möglichen Repräsentationen von ungültigen Platzierungen deutlich schneller steigt, als die Anzahl an möglichen Repräsentationen von gültigen Platzierungen. Erst bei einem Abbruchkriterium von maximal 50 Generationen, schafft es der EA ähnlich gute Lösungen zu produzieren, wie mithilfe der FL- oder der VLG-Codierung. Aus diesem Grund werden wir die Binärcodierung im Folgenden nicht weiter evaluieren, denn die FL-Codierung ist der Binärcodierung sehr ähnlich, mit dem großen Unterschied, dass nur gültige Genotypen vorkommen können.

Die meisten Kennzahlen, die mit den LP-Lösungen verglichen werden sollen, werden von MAST berechnet. MAST nutzt zur Berechnung der Kennzahlen allerdings einen pessimistischen Algorithmus, weshalb davon ausgegangen werden muss, dass selbst im besten Fall noch ein Abstand zwischen einer optimalen Lösung vom LP und einer vom EA berechneten Lösung bleiben wird.

Im Abschnitt 4.3.4 wurden bereits die evolutionären Operatoren vorgestellt, welche im Rahmen von Open BEAGLE mithilfe einer XML-Datei kombiniert werden können und in welcher auch die genannten Wahrscheinlichkeiten für die Operatoren eingestellt werden können. Die im Laufe der Evaluation benutzen Konfigurationen zeigt für die FL-Codierung das Listing 5.1 und für die VLG-Codierung das Listing 5.2. Zusätzlich zu den bereits bekannten Bausteinen kommen noch die beiden Selektionsoperatoren *SelectRandomOp* und *SelectBestOp* hinzu. Der erste Operator wählt zufällig zwei Individuen zur Rekombination aus, der zweite Operator wählt immer das beste Individuum zur Mutation aus. Die Kombination von *bester* Selektion und dem Mutationsoperator *GA-ScheduleMutSetout* sorgt dafür, dass der EA schnell echtzeitfähige Platzierungen berechnen kann. Außerdem wurde festgelegt, dass der NSGA2-Operator pro Generation aus den 100 Individuen 100 neue Individuen generieren soll, aus deren Vereinigung er dann mithilfe des Dominanz-Kriteriums die 100 besten für die nächste Generation auswählt.

Abbildung 5.7 zeigt den Gütevergleich für das Kriterium „Anzahl der Hosts“. Zu sehen ist die vom LP berechnete optimale Lösung 3 und zwei Boxplots für die FL- und die VLG-Codierung, basierend auf 15 Ausführungen des EA. Wie zu erwarten war, wird die optimale Lösung vom EA nicht erreicht, gleichzei-

```
<NSGA2Op ratio_name="1">
  <BitstringEvalOp>
    <GA-CrossoverOnePointScheduleOp matingpb="0.23">
      <SelectRandomOp />
      <SelectRandomOp />
    </GA-CrossoverOnePointScheduleOp>
  </BitstringEvalOp>
  <BitstringEvalOp>
    <GA-ScheduleMutSetout mutationpb="0.76" mutintpb="0.01">
      <SelectBestOp />
    </GA-ScheduleMutSetout>
  </BitstringEvalOp>
</NSGA2Op>
```

Listing 5.1: Open BEAGLE Einstellungen für FL-Schedule.

```
<NSGA2Op ratio_name="1">
  <BitstringEvalOp>
    <GA-CrossoverOnePointVLGScheduleOp matingpb="0.23">
      <SelectRandomOp />
      <SelectRandomOp />
    </GA-CrossoverOnePointVLGScheduleOp>
  </BitstringEvalOp>
  <BitstringEvalOp>
    <GA-ScheduleMutSetout mutationpb="0.76" mutintpb="0.01">
      <SelectBestOp />
    </GA-ScheduleMutSetout>
  </BitstringEvalOp>
</NSGA2Op>
```

Listing 5.2: Open BEAGLE Einstellungen für VLG-Schedule.

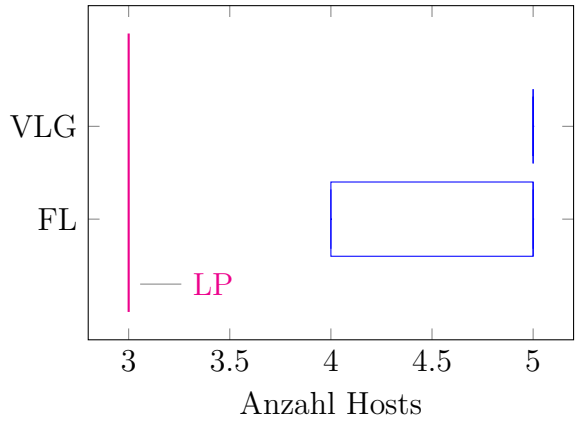

Abbildung 5.7.: Boxplot der besten Lösung des EA im Vergleich zur optimalen LP-Lösung hinsichtlich des Kriteriums „Anzahl der Hosts“ beim Modell *Modell10*.

tig ist jedoch zu sehen, dass die Varianz der berechneten Lösungen vom EA klein ist und die Lösungen nur wenig von der optimalen Lösung abweichen, wenn man berücksichtigt, dass die schlechteste Lösung 10 Hosts nutzen würde. Hervorzuheben ist außerdem, dass alle vom EA gefundenen Platzierungen die Echtzeitfähigkeit garantieren.

Bei dem Kriterium „Anzahl der Migrationen“ sieht die Situation anders aus, wie in Abbildung 5.8 zu sehen ist. Die LP-Modellierung hat als optimale Lösung ca. 500 KB berechnet. Der EA hat über 15 Ausführungen hinweg erneut nicht das Optimum erreicht, was aber auch zu erwarten war, weist aber vor allem bei der FL-Codierung auch eine deutlich breitere Streuung auf. Eine mögliche Erklärung dafür kann sein, dass keiner der aktuellen Operatoren dediziert dafür zuständig ist dieses Kriterium in einer bestehenden Population zu verbessern. Trotzdem ist auch hier hervorzuheben, dass vom EA durchgängig echtzeitfähige Platzierungen berechnet wurden, und die berechneten Lösungen deutlich unterhalb der schlechtesten Lösung von 4100 KB liegen.

Die in Abbildung 5.8 beispielhaft betrachtete Ende-zu-Ende-Latenz von der vCPS-Anwendung 0 fällt deutlich aus dem Rahmen, da das Kriterium bei allen 15 Ausführungen des EA das gleiche Ergebnis ergeben hat. Außerdem mag überraschen, dass die vom EA berechnete Lösung besser ist, als der vom LP bestimmte Wert. Bei genauer Betrachtung der im Abschnitt 3.1.3.2 vorgestellten LP-Modellierung der Ende-zu-Ende-Latenz löst sich diese Diskrepanz jedoch: Die LP-Modellierung berechnet den spätesten Zeitpunkt des frühesten Zeitfensters, in dem die Berechnung einer vCPS-Anwendung abgeschlossen wird.

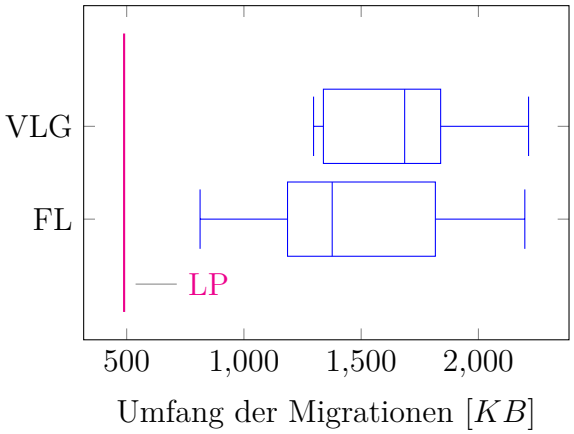

Abbildung 5.8.: Boxplot der besten Lösung des EA im Vergleich zur optimalen LP-Lösung hinsichtlich des Kriteriums „Umfang der Migrationen“ beim Modell *Modell10*.

MAST nutzt offensichtlich ein anderes Verfahren und berechnet den frühesten Zeitpunkt. Die errechneten Ende-zu-Ende-Latenzen erfüllen aber vor allem in allen Fällen die Echtzeitbedingungen.

Der Zeitaufwand, sowohl für die FL-, als auch für die VLG-Codierung, für das Absolvieren von 15 Generationen, beträgt im Durchschnitt ca. 6 Minuten, wie in Abbildung 5.10 zu sehen. Die Zeit wurde auf Basis eines i7-Prozessors[1] gemessen, mit einer durchschnittlichen CPU-Auslastung von über 90% durch 8 Threads. Hier wird offensichtlich, wie hoch der Aufwand für eine Ausführung von MAST ist, weshalb eine parallele Ausführung der Bewertungsfunktion notwendig ist. Außerdem bedingt diese Eigenschaft, dass die Anzahl an Generationen und Individuen pro Iteration niedriger ist, als bei anderen Anwendungen eines evolutionären Algorithmus.

5.3. Weitere Evaluierungen des EA

Der letzte Abschnitt dieses Kapitels soll mithilfe eines größeren Modells den evolutionären Algorithmus ausgiebiger testen. Zu diesem Zweck wurde, basierend auf den vorgestellten Wahrscheinlichkeitsverteilungen, ein größeres Modell *Fullmodel_dep*, bestehend aus 24 vCPS-Anwendungen, generiert, diesmal

[1] Intel Core i7 4790K 4x 4.00GHz

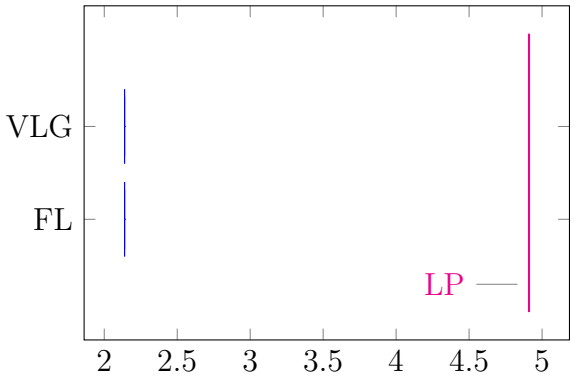

Abbildung 5.9.: Boxplot der besten Lösung des EA im Vergleich zur optimalen LP-Lösung hinsichtlich des Kriteriums „Ende-zu-Ende-Latenz“ von vCPS-Anwendung 0 beim Modell *Modell10*.

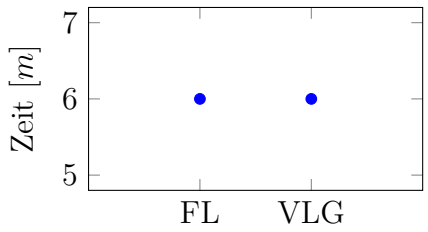

Abbildung 5.10.: Zeitverbrauch des EA für 15 Generationen beim Modell *Modell10*.

sowohl mit Ein-/Ausgabe, als auch mit datenabhängigen vCPS-Anwendungen. Der Arbeit angehängt ist dieses Modell in Anhang A.2.

Aufgrund der größeren Modelle benötigt MAST mehr Zeit, um die angefragten Kennzahlen zu berechnen. Um die Ausführungszeit des EA in einem akzeptablen Rahmen zu halten, wurde das Abbruchkriterium deshalb im Vergleich zum vorigen Abschnitt angepasst. Nach wie vor wird das Kriterium „maximale Anzahl an Generationen“ genutzt, hier jedoch wird bereits nach 10 Generationen gestoppt.

Da der Unterschied zwischen FL- und VLG-Codierung im letzten Abschnitt nicht gravierend ausfiel, wollen wir uns im aktuellen Abschnitt auf die FL-Codierung beschränken. Dafür werden, erneut mit dem Hintergedanken Zeit zu sparen, zwei Varianten miteinander vergleichen, die sich in der Anzahl der pro Generation generierten Individuen unterscheiden. Im vorigen Abschnitt wurden pro Generation aus 100 Individuen 100 neue erstellt. Eine schnellere Variante generiert pro Generation nur noch 50 neue Individuen. Beide Varianten im Vergleich werden zeigen, ob die Güte der Lösungen durch diese Einsparung leidet.

Die Abbildung 5.11 zeigt erneut zuerst das Kriterium „Anzahl der Hosts“, in den beiden Varianten 50 und 100 neue Individuen pro Generation, über jeweils 15 Ausführungen des EA. Wie zu sehen, berechnet der EA innerhalb von 10 Generationen Lösungen, die zwischen 14 und 17 Hosts benötigen. Die Varianz der Lösungen ist relativ klein, und die berechneten Lösungen liegen deutlich unter der schlechtesten Lösung, die 24 Hosts benötigen würde. Alle berechneten Lösungen erfüllen die Echtzeitbedingungen.

Beim Kriterium „Umfang der Migrationen“ zeigt sich das Einsparen von neuen Individuen pro Generation, siehe Abbildung 5.12. Die Streuung der Lösungswerte ist bei nur 50 neuen Individuen pro Generation deutlich breiter als in der Variante mit 100 neuen Individuen pro Generation. Während der Durchschnitt bei 100 neuen Individuen deutlich unterhalb der schlechtesten Lösung von 10519 KB liegt, liegt der Durchschnitt bei 50 neuen Individuen merklich höher. Auch wenn die Ergebnisse für dieses Kriterium nicht besonders gut sind, hat es der EA in allen Fällen geschafft echtzeitfähige Lösungen zu berechnen.

Für die Ende-zu-Ende-Latenz, hier beispielhaft der vCPS-Anwendung 0, zeigt sich in Abbildung 5.13, dass über alle 15 Ausführungen für beide Varianten der gleiche Wert ermittelt wurde. Dies war bereits auch beim kleinen Modell der Fall. Ein möglicher Grund für diese Resultate kann sein, dass der pessimistische Algorithmus von MAST dazu führt, dass unser EA eher eine Lösung generiert, die mehr Hosts benutzt, als eine, mit einer höheren Ende-

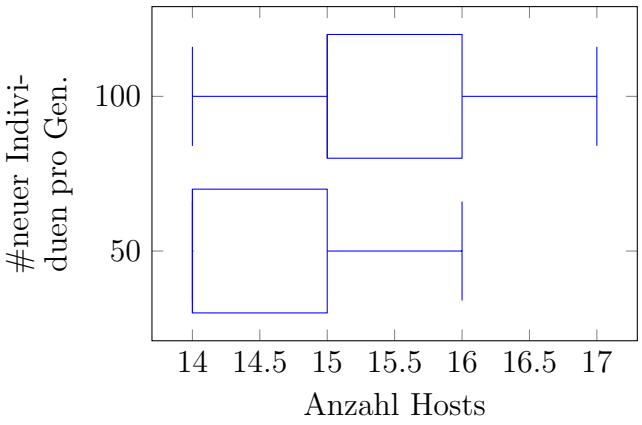

Abbildung 5.11.: Boxplot der besten Lösung des EA im Vergleich zur optimalen LP-Lösung hinsichtlich des Kriteriums „Anzahl der Hosts“ beim Modell *Fullmodel_dep*.

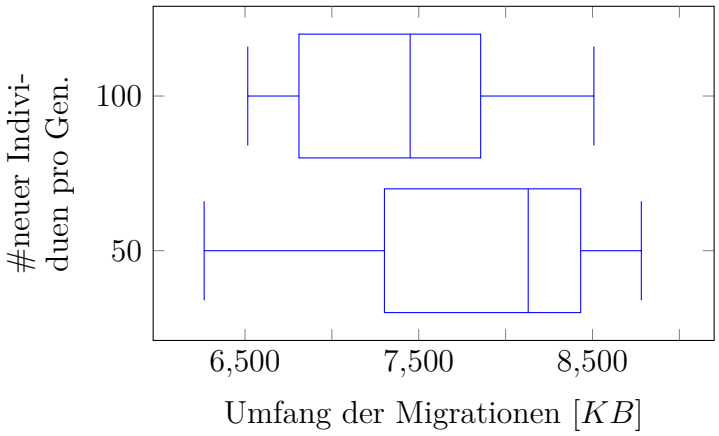

Abbildung 5.12.: Boxplot der besten Lösung des EA im Vergleich zur optimalen LP-Lösung hinsichtlich des Kriteriums „Umfang der Migrationen“ beim Modell *Fullmodel_dep*.

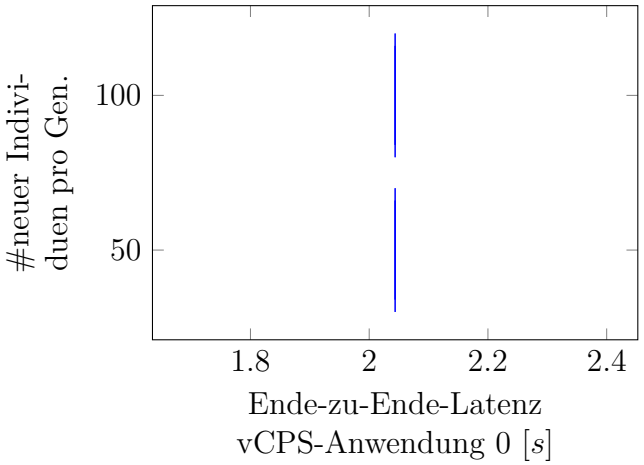

Abbildung 5.13.: Boxplot der besten Lösung des EA im Vergleich zur optimalen LP-Lösung hinsichtlich des Kriteriums „Ende-zu-Ende-Latenz“ von vCPS-Anwendung 0 beim Modell *Fullmodel_dep*.

zu-Ende-Latenz. Auch wenn die generierten Lösungen untypisch aussehen, erfüllen doch alle berechneten Lösungen die Echtzeitanforderungen.

Während sich nur beim Umfang der Migrationen ein merklicher Unterschied in der Qualität der berechneten Lösungen offenbarte, zeigt sich beim durchschnittlich benötigten Zeitaufwand eine deutliche Verbesserung fast um den Faktor 2, siehe Abbildung 5.14. Unter Berücksichtigung davon, dass hier ca. eine Stunde Rechenzeit pro Ausführung gespart werden kann, lohnt sich vermutlich die Operatoren so zu überarbeiten, dass sie auch mit weniger Individuen pro Generation ein besseres Ergebnis liefern.

Zum Abschluss wollen wir uns noch die Entwicklung der drei Kriterien über die während der Ausführung des EA generierten Generationen hinweg anschauen. Die Abbildungen 5.15, 5.16 und 5.17 zeigen jeweils den aktuell besten Gütewert des Kriteriums von der 1. bis zur 10. Generation. Während bei der Anzahl der Hosts ein interessantes Phänomen beobachtbar ist, nämlich, dass der MOEA durchaus auch mal Lösungen mit geringeren Lösungswerten verwirft, um eine insgesamt breitere Pareto-Front zu berechnen, zeigt sich beim Umfang der Migrationen eine kontinuierliche Verbesserung, je länger der EA läuft. Beim Kriterium Ende-zu-Ende-Latenz, wieder beispielhaft an der vCPS-Anwendung 0, zeigt sich nach wie vor das ungewöhnliche Bild, dass nur ein Lösungswert über die gesamte Ausführung des EA hinweg präsent ist.

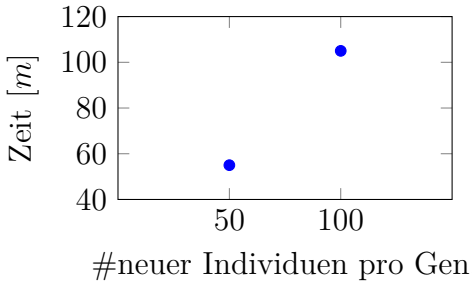

Abbildung 5.14.: Zeitverbrauch des EA für 10 Generationen beim Modell *Fullmodel_dep* in Abhängigkeit von der Anzahl der in jeder Generation neu generierten Individuen.

Vor allem bemerkenswert ist jedoch, dass der EA in allen Fällen bereits nach der Initialisierung mindestens eine Platzierung in der Population vorliegen hat, für die die Echtzeitfähigkeit garantiert werden kann. Die Lösungsgüte wird im weiteren Verlauf dann nur noch verbessert.

Insgesamt zeigt die Evaluation, dass es möglich ist, mithilfe unseres Ansatzes der Kombination eines EAs mit einer Methode der formalen Leistungsbewertung, echtzeitfähige Platzierungen zu berechnen. Dies klappt auch noch verhältnismäßig gut für Modelle, die aufgrund ihrer Größe zumindest nicht mehr mit der in dieser Arbeit verwendeten LP-Modellierung lösbar sind.

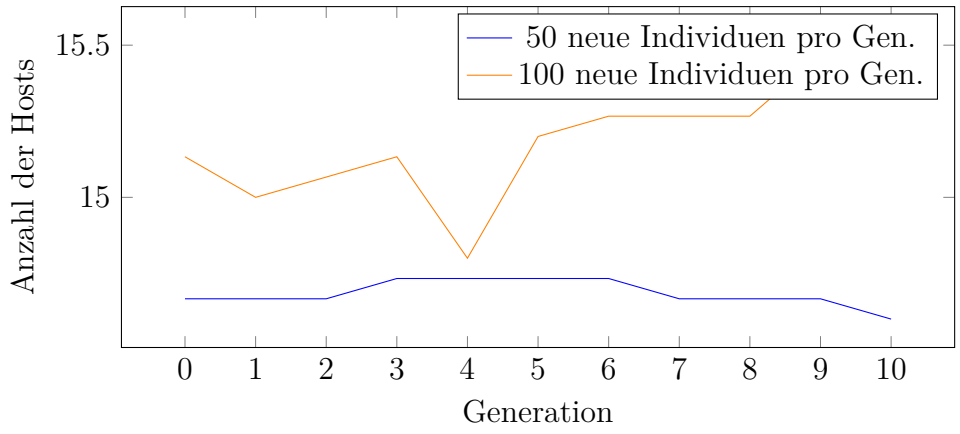

Abbildung 5.15.: Durchschnittliche Entwicklung des Kriteriums „Anzahl der Hosts“ über die Generationen beim Modell *Fullmodel_dep*.

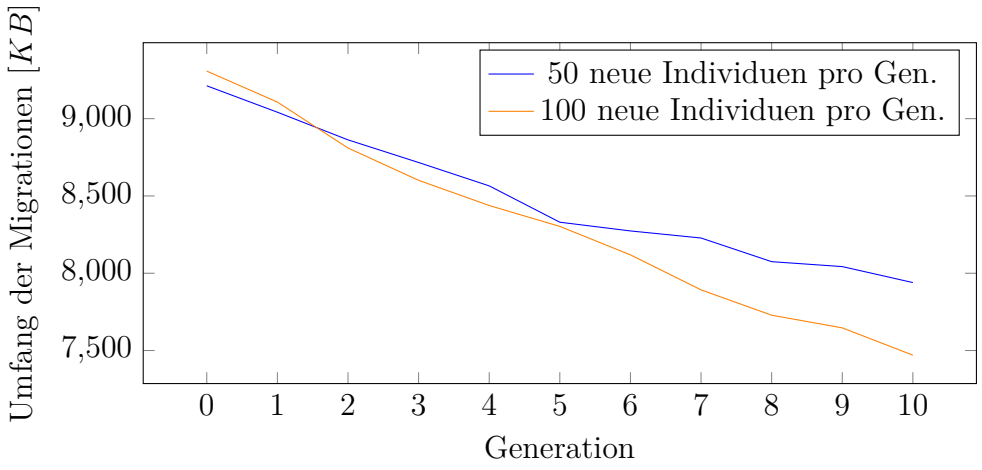

Abbildung 5.16.: Durchschnittliche Entwicklung des Kriteriums „Umfang der Migrationen“ über die Generationen beim Modell *Fullmodel_dep*.

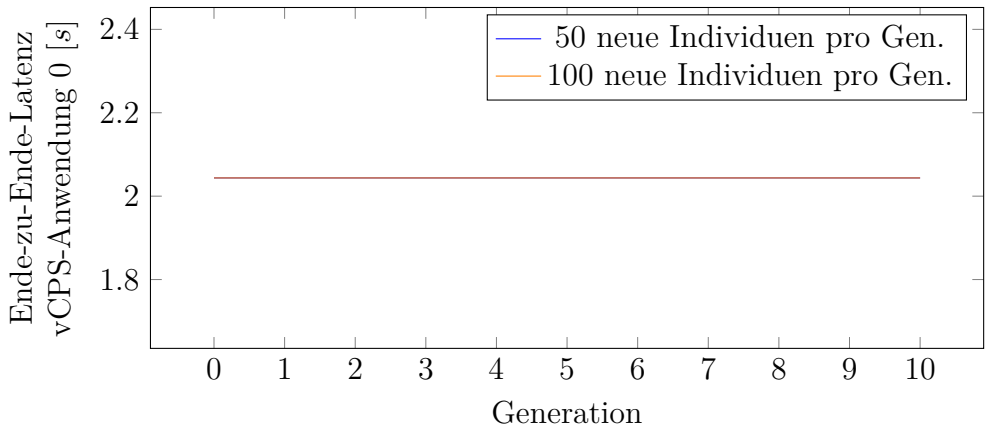

Abbildung 5.17.: Durchschnittliche Entwicklung des Kriteriums „Ende-zu-Ende-Latenz“ von vCPS-Anwendung 0 über die Generationen beim Modell *Fullmodel_dep*.

Kapitel 6

Fazit und Ausblick

Die vorliegende Arbeit widmete sich dem Optimierungsproblem echtzeitfähige virtuelle Maschinen auf Virtualisierungshosts unter Einhaltung nichtfunktionaler Eigenschaften zu platzieren. Dafür wurde das Problem zuerst charakterisiert, um dann unterschiedlicher Lösungsansätze vorzustellen, und ihre Anwendbarkeit auf das Problem zu untersuchen. Im weiteren Verlauf wurde die neuartige Idee, eine Methode der formalen Leistungsbewertung in einen evolutionären Algorithmus zu integrieren, entwickelt und implementiert. Dieser Algorithmus wurde schließlich noch evaluiert, um zu prüfen, ob er die gestellten Anforderungen erfüllt.

6.1. Eigenbeitrag

Folgende Aspekte wurden im Rahmen der Erstellung dieser Ausarbeitung erforscht und stellen somit Eigenbeiträge zum aktuellen Stand der Forschung dar:

- In den Abschnitten 2.3 und 2.4 wurde eine formale Abstraktion entwickelt, welche sowohl viele Aspekte des Problems einfängt, als auch einen Vergleich mit bereits bestehender Literatur ermöglicht.
- In Abschnitt 2.5 wurden mögliche Qualitätskriterien für das untersuchte Problem zusammengetragen.
- In Abschnitt 3.1.3 wurde die von Baruah und Bini vorgestellte LP-Formulierung weiterentwickelt, wobei sowohl die bisherige Formulierung korrigiert, als auch um Modellierungen für weitere Optimierungsfunktionen erweitert wurde. Die erweiterte Modellierung ist in der Lage große Teile des zuvor entwickelten formalen Modells abzubilden.
- In Abschnitt 3.2.1 wurde gezeigt, wie die vormals entwickelte formale Modellierung in die MAST-Syntax überführt werden kann. Dies ermöglicht die Anwendung aller Algorithmen, die MAST anbietet.

- In Abschnitt 4.3 wurden evolutionäre Komponenten entwickelt, um das Platzierungsproblem abbilden zu können, und durch einen EA lösen zu können. Zugrunde liegt die neuartige Idee Methoden der formalen Leistungsbewertung in die Bewertungsfunktion eines evolutionären Algorithmus zu integrieren. Auch wurde ein Konzept für einen geschichteten Fitness-Vektor entwickelt, der es ermöglicht unterschiedliche Entwicklungsstufen von Individuen ohne Anpassungen am restlichen EA handhaben zu können.

- In Abschnitt 4.4 wurde die Integration von Ada-Quellcode in ein C-Programm anhand von Literatur untersucht, und im Zuge dessen der MAST-Quellcode um eine funktionale Abstraktionsschicht erweitert, die ohne den Umweg über das Kommandozeilen-Programm direkt von Drittanwendungen als Bibliothek genutzt werden kann.

  Die Integration von MAST eröffnet prinzipiell die volle Bandbreite an möglichen Modellierungen in MAST, deutlich über die in dieser Arbeit genutzten Funktionalitäten hinaus.

- In Abschnitt 5.1 wurden geplant Wahrscheinlichkeitsdichtefunktionen konstruiert, die eine zufällige Erstellung von ausführbaren vCPS ermöglichen.

- In den anderen Abschnitten von Kapitel 5 wurde der entwickelte evolutionäre Algorithmus noch evaluiert.

6.2. Fazit

Die grundlegende Idee, eine Methode der formalen Leistungsbewertung mit einem evolutionären Algorithmus zu kombinieren, konnte erfolgreich umgesetzt werden. Der entwickelte Algorithmus ist in der Lage „gute“ Lösungen zu finden, auch für Modelle, für die zumindest die von uns entwickelte LP-Formulierung keine Ergebnisse mehr liefern kann.

Die Berechnung notwendiger Kennzahlen durch MAST beansprucht einen Großteil der Laufzeit des evolutionären Algorithmus, sodass im Rahmen der Entwicklung andere Herausforderungen zu bewältigen waren, als bei Anwendungen dieser Methode auf andere Optimierungsprobleme, z. B. hinsichtlich der Tatsache, dass es nicht möglich ist in einem Lauf mehrere zehntausend Individuen zu generieren und zu bewerten. Die lange Laufzeit der Bewertungsfunktion erzwingt auch praktisch die Nutzung von Parallelität, wenn die Ergebnisse einigermaßen schnell berechnet werden sollen.

Die Integration von MAST über eine Schnittstelle zwischen Ada und C erwies sich als schwierig, und ist zur Zeit noch wenig robust. MAST verfügte nicht über die für diese Integration notwendige Abstraktionsschicht, sodass sie für die Zwecke der vorliegenden Arbeit erst implementiert werden musste. Zur Zeit mangelt es noch an Fehlerbehandlungsroutinen, die ein robusteres Arbeiten mit der Integration ermöglichen würden.

Auch der verwendete Algorithmus im Rahmen von Open BEAGLE ist nicht auf parallele Ausführung ausgelegt, und musste entsprechend erweitert werden. Hier mögen weitere Fehler versteckt sein, deren Ausmerzung für eine robuste Software notwendig wären.

6.3. Ausblick

Auf Basis der vorliegenden Arbeit ließen sich weitere Forschungen begründen:

- Die LP-Modellierung nutzt zur Zeit ein partitionsbasiertes Verständnis des Scheduling-Problems. Demgegenüber gibt es in der Literatur auch Ansätze ein globales Scheduling zu betrachten. Dies könnte weiter untersucht werden.

- Die von Zhu u. a. vorgestellte LP-Modellierung kann auf Nachrichten beliebiger Größe erweitert werden und eine Anwendung auf das hier bearbeitete Platzierungsproblem untersucht werden.

- Die Performance des entwickelten evolutionären Algorithmus kann verbessert werden, z. B. indem die evolutionären Operatoren präziser auf bestimmte Stadien der Population angepasst werden. Zur Zeit scheint es dem Algorithmus leicht zu fallen eine Platzierung zu ermitteln, welche die Zeitbeschränkungen einhält, danach scheint es ihm jedoch schwerer zu fallen die gefundenen Lösungen zu verbessern. An dieser Stelle könnte man mit neuen Operatoren vermutlich für eine Verbesserung sorgen.

  Durch schnellere Lösungsfindung ließen sich auch Generationen einsparen, was aufgrund der Laufzeit der Bewertungsfunktion die Laufzeit des gesamten Algorithmus maßgeblich beeinflussen würde.

- Die Bewertungsfunktion könnte um einen LRU-Cache erweitert werden, mit dem Hintergedanken, dass von den zeitaufwändigen Bewertungen einer Platzierung eventuell welche eingespart werden können, wenn diese Platzierung kürzlich schon mal bewertet wurde und das Ergebnis aus dem Cache geladen werden kann.

- Die zur Zeit von MAST angewandte holistische Analyse ist pessimistisch, und auch die Autoren selbst schlagen vor z. B. Offset-basierte Verfahren in dem Tool zu implementieren [Gut+14]. Alternativ könnte man die angefragten Kennzahlen nicht mehr durch MAST berechnen lassen, sondern einen entsprechenden Algorithmus direkt implementieren.

Literatur

[BB04] Enrico Bini und Giorgio C. Buttazzo. „Schedulability Analysis of Periodic Fixed Priority Systems“. In: *IEEE Trans. Computers* 53.11 (2004), S. 1462–1473. DOI: 10.1109/TC.2004.103.

[BB08] Sanjoy Baruah und Enrico Bini. „Partitioned scheduling of sporadic task systems: an ILP-based approach“. In: *Proceedings of the International Conference on Design and Architectures for Signal and Image Processing (DASIP 2008)*. Brussels, Belgium, Nov. 2008.

[BG73] J. P. Buzen und U. O. Gagliardi. „The Evolution of Virtual Machine Architecture“. In: *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition*. AFIPS '73. New York, New York: ACM, 1973, S. 291–299. DOI: 10.1145/1499586.1499667.

[Ble+03] Stefan Bleuler u. a. „PISA — A Platform and Programming Language Independent Interface for Search Algorithms“. In: *Evolutionary Multi-Criterion Optimization (EMO 2003)*. Hrsg. von Carlos M. Fonseca u. a. Lecture Notes in Computer Science. Berlin: Springer, 2003, S. 494–508.

[BMR90] Sanjoy K. Baruah, Aloysius K. Mok und Louis E. Rosier. „Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor“. In: *In Proceedings of the 11th Real-Time Systems Symposium*. IEEE Computer Society Press, 1990, S. 182–190.

[Bur+98] Donald S. Burke u. a. „Putting More Genetics into Genetic Algorithms“. In: *Evolutionary Computation* 6.4 (1998), S. 387–410. DOI: 10.1162/evco.1998.6.4.387.

[CKT03] Samarjit Chakraborty, Simon Kunzli und Lothar Thiele. „A General Framework for Analysing System Properties in Platform-Based Embedded System Designs“. In: *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1*. DATE '03. Washington, DC, USA: IEEE Computer Society, 2003, S. 10190–. ISBN: 0-7695-1870-2.

[Cla+05] Christopher Clark u. a. „Live Migration of Virtual Machines“. In: *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, S. 273–286.

[Cor+01] David W. Corne u. a. „PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization“. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001*. Morgan Kaufmann Publishers, 2001, S. 283–290.

[CSG14] Faruk Caglar, Shashank Shekhar und Aniruddha S. Gokhale. „iPlace: An Intelligent and Tunable Power- and Performance-Aware Virtual Machine Placement Technique for Cloud-Based Real-Time Applications“. In: *17th IEEE International Symposium on Object / Component / Service-Oriented Real-Time Distributed Computing, ISORC 2014, Reno, NV, USA, June 10-12, 2014*. 2014, S. 48–55. DOI: 10.1109/ISORC.2014.35.

[Cul+08] Brendan Cully u. a. „Remus: High availability via asynchronous virtual machine replication“. In: *In Proc. NSDI*. 2008.

[Deb+00] Kalyanmoy Deb u. a. „A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II“. In: Springer, 2000, S. 849–858.

[Dra+14] José Mariá Drake u. a. *Description of the MAST Model*. 2000–2014. URL: http://mast.unican.es/mast_description.pdf.

[EMH01] Mark Erickson, Alex Mayer und Jeffrey Horn. „The Niched Pareto Genetic Algorithm 2 Applied to the Design of Groundwater Remediation Systems“. In: *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*. EMO '01. London, UK, UK: Springer-Verlag, 2001, S. 681–695. ISBN: 3-540-41745-1.

[Fei+15] Frank Feinbube u. a. „Evolving Scheduling Strategies for Multi-Processor Real-Time Systems“. In: *11th annual workshop on Operating Systems Platforms for Embedded Real-Time applications, OSPERT 2015, Lund, Sweden, July 7, 2015*. 2015.

[GD02] Crina Groşan und Dumitru Dumitrescu. „A comparison of multiobjective evolutionary algorithms.“ eng. In: *Acta Universitatis Apulensis. Mathematics - Informatics* 4 (2002), S. 101–110. URL: http://eudml.org/doc/129389.

[GGG00] J.J. Gutierrez Garcia, J.C.P. Gutierrez und M. Gonzalez Harbour. „Schedulability analysis of distributed hard real-time systems with multiple-event synchronization“. In: *Real-Time Systems, 2000. Euromicro RTS 2000. 12th Euromicro Conference on*. 2000, S. 15–24. DOI: 10.1109/EMRTS.2000.853988.

[GGH97] José C. Palencia Gutiérrez, José Javier Gutiérrez García und Michael González Harbour. „On the schedulability analysis for distributed hard real-time systems“. In: *Proceedings of the Ninth Euromicro Workshop on Real-Time Systems, RTS 1997, 11-13 June, 1997, Toledo, Spain*. 1997, S. 136–143. DOI: 10.1109/EMWRTS.1997.613774.

[GKP94] Ronald L. Graham, Donald E. Knuth und Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. 2nd. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1994. ISBN: 0201558025.

[GLS88] Martin Grötschel, László Lovász und Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. English. Bd. 2. Algorithms and Combinatorics. Springer, 1988. ISBN: 3-540-13624-X, 0-387-13624-X (U.S.)

[Gol73] R. P. Goldberg. „Architecture of Virtual Machines“. In: *Proceedings of the Workshop on Virtual Computer Systems*. Cambridge, Massachusetts, USA: ACM, 1973, S. 74–112. DOI: 10.1145/800122.803950.

[GP06] Christian Gagné und Marc Parizeau. „Genericity in Evolutionary Computation Software Tools: Principles and Case Study“. In: *International Journal on Artificial Intelligence Tools* 15.2 (2006), S. 173–194.

[Gut+14] José C. Palencia Gutiérrez u. a. *Analysis Techniques used in MAST*. 2000–2014. URL: http://mast.unican.es/mast_analysis_techniques.pdf.

[JGČ14] Domagoj Jakobović, Marin Golub und Marko Čupić. „Asynchronous and implicitly parallel evolutionary computation models“. English. In: *Soft Computing* 18.6 (2014), S. 1225–1236. ISSN: 1432-7643. DOI: 10.1007/s00500-013-1140-5.

[JP86] M. Joseph und P. Pandya. „Finding Response Times in a Real-Time System“. In: *The Computer Journal* 29.5 (Mai 1986), S. 390–395. DOI: 10.1093/comjnl/29.5.390.

[JS12] Boguslaw Jablkowski und Olaf Spinczyk. „Continuous Performance Analysis of Fault-Tolerant Virtual Machines“. In: *Proceedings of the 1st GI Workshop on Software-Based Methods for Robust Embedded Systems (SOBRES '12)*. (Braunschweig, Germany). Lecture Notes in Informatics. German Society of Informatics, Sep. 2012, S. 494–505.

[JS15] Boguslaw Jablkowski und Olaf Spinczyk. „CPS-Xen: A Virtual Execution Environment for Cyber-Physical Applications“. In: *28th International Conference on Architecture of Computing Systems (ARCS '15)*. Porto, Portugal: Springer-Verlag, März 2015, S. 108–119.

[Kei+02] Maarten Keijzer u. a. „Evolving Objects: A General Purpose Evolutionary Computation Library“. In: *Artificial Evolution* 2310 (2002), S. 829–888.

[Kim+04] Mifa Kim u. a. „SPEA2+: Improving the Performance of the Strength Pareto Evolutionary Algorithm 2“. English. In: *Parallel Problem Solving from Nature - PPSN VIII*. Hrsg. von Xin Yao u. a. Bd. 3242. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, S. 742–751. ISBN: 978-3-540-23092-2. DOI: 10.1007/978-3-540-30217-9_75. URL: http://dx.doi.org/10.1007/978-3-540-30217-9_75.

[Kni+07] Carole Knibbe u. a. „A Long-Term Evolutionary Pressure on the Amount of Noncoding DNA“. en. In: *Molecular Biology and Evolution* 24.10 (Okt. 2007), S. 2344–2353. DOI: 10.1093/molbev/msm165.

[Kra09] Oliver Kramer. *Computational Intelligence*. Informatik im Fokus. Springer, 2009. ISBN: 9783540797395.

[KT15] Jonathan Koomey und Jon Taylor. *New data supports finding that 30 percent of servers are 'Comatose', indicating that nearly a third of capital in enterprise data centers is wasted*. 3. Juni 2015. URL: http://anthesisgroup.com/wp-content/uploads/2015/06/Case-Study_DataSupports30PercentComatoseEstimate-FINAL_06032015.pdf.

[Kun05] Daniel Kunkle. „A Summary and Comparison of MOEA Algorithms“. In: (Mai 2005). URL: http://www.ccs.neu.edu/home/kunkle/papers/techreports/moeaComparison.pdf.

[Lee00] C. -y. Lee. „Variable Length Genomes for Evolutionary Algorithms“. In: *In Proceedings of the Genetic and Evolutionary Computation Conference, 806. Las Vegas*. Morgan Kaufmann, 2000, S. 806.

[LL73] C. L. Liu und James W. Layland. „Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment“. In: *J. ACM* 20.1 (Jan. 1973), S. 46–61. ISSN: 0004-5411. DOI: 10.1145/321738.321743.

[LSD89] John P. Lehoczky, Lui Sha und Ye Ding. „The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior“. In: *IEEE Real-Time Systems Symposium*. 1989, S. 166–171. DOI: 10.1109/REAL.1989.63567.

[Mas+11] Alejandro Masrur u. a. „Designing VM Schedulers for Embedded Real-Time Applications“. Englisch. In: *Proceedings of the 7th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. Taipei, Taiwan, 2011.

[OW04] Jaewon Oh und Chisu Wu. „Genetic-algorithm-based real-time task scheduling with multiple goals.“ In: *Journal of Systems and Software* 71.3 (10. Nov. 2004), S. 245–258. DOI: 10.1016/S0164-1212(02)00147-4.

[PJ94] Mitchell A. Potter und Kenneth A. De Jong. „A Cooperative Coevolutionary Approach to Function Optimization“. In: *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*. PPSN III. London, UK, UK: Springer-Verlag, 1994, S. 249–257. ISBN: 3-540-58484-6. URL: http://dl.acm.org/citation.cfm?id=645822.670374.

[PWT06] Simon Perathoner, Ernesto Wandeler und Lothar Thiele. *Evaluation and Comparison of Performance Analysis Methods for Distributed Embedded Systems*. Techn. Ber. 276. Computer Engineering und Networks Laboratory, ETH Zurich, März 2006.

[RD69] B.C. Rennie und A.J. Dobson. „On stirling numbers of the second kind“. In: *Journal of Combinatorial Theory* 7.2 (1969), S. 116–121. ISSN: 0021-9800. DOI: 10.1016/S0021-9800(69)80045-1.

[RY09] Tapabrata Ray und Xin Yao. „A cooperative coevolutionary algorithm with Correlation based Adaptive Variable Partitioning.“ In: *IEEE Congress on Evolutionary Computation*. IEEE, 2009, S. 983–989. DOI: 10.1109/CEC.2009.4983052.

[SHK94] Joachim Stender, Eva Hillebrand und Jason Kingdon. *Genetic algorithms in optimisation, simulation and modelling*. Bd. 23. IOS Press, 1994.

[TC94] Ken Tindell und John Clark. „Holistic Schedulability Analysis for Distributed Hard Real-time Systems“. In: *Microprocess. Microprogram.* 40.2-3 (Apr. 1994), S. 117–134. ISSN: 0165-6074. DOI: 10.1016/0165-6074(94)90080-9.

[TCN00] L. Thiele, S. Chakraborty und M. Naedele. „Real-time calculus for scheduling hard real-time systems“. In: *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*. Bd. 4. 2000, 101–104 vol.4. DOI: 10.1109/ISCAS.2000.858698.

[Tem93] N. M. Temme. „Asymptotic Estimates of Stirling Numbers“. In: *STUDIES IN APPLIED MATHEMATICS 89*. Elsevier Science Publicshing, 1993, S. 233–243.

[Wei07] K. Weicker. *Evolutionäre Algorithmen*. Leitfäden der Informatik. Vieweg+Teubner Verlag, 2007. ISBN: 9783835102194.

[Wil06] Herbert S. Wilf. *Generatingfunctionology*. Natick, MA, USA: A. K. Peters, Ltd., 2006. ISBN: 1568812795.

[WL96] Annie S. Wu und Robert K. Lindsay. „A Comparison of the Fixed and Floating Building Block Representation in the Genetic Algorithm“. In: *Evolutionary Computation* 4.2 (1996), S. 169–193. DOI: 10.1162/evco.1996.4.2.169.

[Zhu+13] Qi Zhu u. a. „Optimization of Task Allocation and Priority Assignment in Hard Real-time Distributed Systems“. In: *ACM Trans. Embed. Comput. Syst.* 11.4 (Jan. 2013), 85:1–85:30. ISSN: 1539-9087. DOI: 10.1145/2362336.2362352.

[ZLT01] Eckart Zitzler, Marco Laumanns und Lothar Thiele. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. Techn. Ber. 2001.

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Anhang A

Evaluations-Modelle

A.1. Modell_10.csv

number	period	wcib	wcet	wcob	deadline	ha-init	ha-regular	depends-on	preset
0	6.8	0	2.11	0	6.8	324	0	−1	1
1	8.14	0	2.13	0	8.14	423	0	−1	3
2	5.56	0	0.68	0	5.56	489	0	−1	−1
3	7.31	0	0.29	0	7.31	212	0	−1	8
4	4.91	0	2.06	0	4.91	518	0	−1	5
5	68.3	0	25.75	0	68.3	459	0	−1	5
6	76.87	0	31.49	0	76.87	484	0	−1	9
7	7.84	0	3.05	0	7.84	399	0	−1	1
8	8.52	0	2.88	0	8.52	428	0	−1	1
9	8.12	0	3.09	0	8.12	364	0	−1	2

A.2. fullmodel_dep.csv

number	period	wcib	wcet	wcob	deadline	ha-init	ha-regular	depends-on	preset
0	8.2	658	1.87	15	8.2	427	213	−1	14
1	78.43	285	15.03	19	78.43	394	197	−1	18
2	4.89	37	2.31	49	4.89	456	228	−1	3
3	74.72	61	12.85	175	74.72	435	217	−1	19
4	6.73	30	1.93	8	6.73	320	160	−1	17
5	6.73	12	2.72	547	6.73	377	188	4	0
6	6.73	182	2.79	24	6.73	320	160	5	1
7	42.48	18	12.21	118	42.48	466	233	−1	10
8	2.48	148	0.7	36	2.48	426	213	−1	5
9	2.48	61	1.12	10	2.48	451	225	8	19
10	5.16	130	1.87	80	5.16	451	225	−1	17
11	5.23	16	1.22	71	5.23	480	240	−1	13
12	5.23	53	1.41	79	5.23	379	189	11	−1
13	7.27	31	1.15	106	7.27	460	230	−1	18
14	9.89	162	4.62	110	9.89	396	198	−1	13
15	9.89	36	4.03	138	9.89	443	221	14	2
16	5.78	288	1.4	46	5.78	406	203	−1	10
17	11.12	41	5.18	242	11.12	422	211	−1	14
18	5.01	24	1.24	24	5.01	353	176	−1	1
19	3.3	145	0.6	61	3.3	456	228	−1	0
20	4.19	168	2.24	251	4.19	451	225	−1	3
21	5.24	72	1.05	46	5.24	366	183	−1	6
22	8.85	146	3.28	50	8.85	451	225	−1	3
23	8.53	89	3.51	64	8.53	429	214	−1	1

Tabelle A.2 – Fortsetzung der letzten Seite

24	7.26	67	1.64	411	7.26	504	252	−1	8

Digitaler Anhang

Eidesstattliche Versicherung

Gabor, Ulrich Thomas _____ _____

Name, Vorname Matr.-Nr.

Ich versichere hiermit an Eides statt, dass ich die vorliegende ~~Bachelorarbeit~~/Masterarbeit* mit dem Titel

Platzierung von echtzeitfähigen virtuellen Maschinen

als praktisches Optimierungsproblem

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -)

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

_____ _____

Ort, Datum Unterschrift