Modellierung von Edge-Computing-Clustern zur Ermittlung ressourceneffizienter Softwareverteilungen in der Produktion

Vom Fachbereich Maschinenbau
an der Technischen Universität Darmstadt
zur
Erlangung des Grades eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte

DISSERTATION

vorgelegt von

Benjamin Brockhaus, M. Sc.

aus Marl

Berichterstatter:	Prof. Dr.-Ing. Matthias Weigold
Mitberichterstatter:	Ao. Univ. Prof. Dr. Manfred Grafinger
Tag der Einreichung:	5. Dezember 2023
Tag der mündlichen Prüfung:	7. Februar 2024

Darmstadt 2024
D17

Brockhaus, Benjamin: Modellierung von Edge-Computing-Clustern zur Ermittlung ressourceneffizienter Softwareverteilungen in der Produktion.
Darmstadt, Technische Universität Darmstadt.

Jahr der Veröffentlichung auf TUprints: 2024
URN: urn:nbn:de:tuda-tuprints-267392
Tag der mündlichen Prüfung: 7. Februar 2024

Abstract

Immer mehr Unternehmen erkennen die Potenziale der Industrie 4.0 und erhoffen sich davon Vorteile in ihren Produktionsprozessen. Allerdings sehen sich viele – insbesondere kleine und mittlere – Unternehmen bei der Implementierung mit Schwierigkeiten konfrontiert, sodass heutige Industrie 4.0-Applikationen eine ineffiziente Ressourcennutzung aufweisen. In Disziplinen wie Cloud-Computing sind Lösungen für eine effizientere Ressourcennutzung vorhanden, die für den Einsatz im Produktionsumfeld allerdings an dessen Bedarfe angepasst werden müssen. Diese Arbeit leistet einen Beitrag dazu, indem sie Softwareverteilungen (als Teil der Software-Orchestrierung) um Echtzeitanforderungen und Netzwerkeinflüsse im *Shopfloor* erweitert. Hierzu werden zwei Modelle entwickelt, die zum einen die zu verteilende Software repräsentieren und zum anderen Edge-Devices zu einem Cluster zusammenfassen. Das sich durch die Kombination der beiden Modelle ergebende Optimierungsproblem wird mittels Metaheuristiken gelöst. Der präsentierte Ansatz benötigt nicht nur wenig Rechenzeit, sondern erlaubt auch das Finden von Softwareverteilungen, die nahe am Optimum hinsichtlich des Energieverbrauchs liegen. Die Verwendung von *Cloud-native*-Technologien in Kombination mit diesem Ansatz ermöglicht eine bessere Auslastung von Recheninfrastruktur im gesamten Produktionssystem. Somit liefert diese Arbeit eine Grundlage für Effizienzsteigerungen von Industrie 4.0-Applikationen.

Keywords: Edge-Computing, Ressourceneffizienz, Virtualisierung

Vorwort des Herausgebers

Die digitale Transformation der Produktion bringt mit der Erhöhung der Anzahl von Sensoren und deren kontinuierlicher Datenerfassung die gesteigerte Bedeutung der dezentralen Datensammlung, -verarbeitung und -verwaltung mit sich. Dieses als Edge-Computing bezeichnete Paradigma und die dazu verwendeten Geräte ermöglichen die Verarbeitung von Daten innerhalb der Automationsebene und direkt in den Produktionsanlagen, anstatt alle Informationen an zentrale Server zu senden. Dies führt zu schnelleren Reaktionszeiten und unterstützt Echtzeit-Entscheidungen, was in der Produktion von entscheidender Bedeutung ist. Weiterhin kann durch die lokale Verarbeitung sensibler Daten auf Edge-Geräten die Sicherheit und Integrität der Informationen besser gewährleistet werden. Diese Eigenschaften sind gerade in der aktuellen Diskussion um Datensouveränität in globalen Daten-Ökosystemen von zentraler Bedeutung.

Über diese Vorteile hinaus trägt Edge-Computing dazu bei, die Effizienz von Produktionsanlagen zu steigern, indem es die Übertragung großer Datenmengen an zentrale Server reduziert. Dies bietet die Möglichkeit einer optimierten Ressourcennutzung der Rechnerinfrastrukturen in der Industrie.

An diesem Punkt setzt die vorliegende Dissertation von Herrn Brockhaus auf. Das übergeordnete Ziel seiner Dissertation ist die Optimierung der Ressourcenausnutzung von verteilten und maschinennahen Rechnerverbünden durch Cloud-native-Technologien. Hierbei gilt es besonders die Latenzanforderungen für Fabriksteuerungen und die Regelung von Maschinen, sogenannte harte Echtzeitanforderungen, zu erfüllen. Die von ihm vorgestellte Kombination aus Modellierung und Lösungsalgorithmen erreicht eine nahezu optimale Ausnutzung der Ressourcen in den Rechnerverbünden unter Einhaltung definierter Echtzeitanforderungen.

Darmstadt, im Februar 2024 Prof. Dr.-Ing. Matthias Weigold

Vorwort des Verfassers

Diese Arbeit entstand im Rahmen meiner Arbeit als wissenschaftlicher Mitarbeiter am Institut für Produktionsmanagement, Technologie und Werkzeugmaschinen (PTW) der TU Darmstadt.

Meinem Doktorvater, Herrn Prof. Dr.-Ing. Matthias Weigold, danke ich für die wissenschaftliche Betreuung und die Diskussion der Arbeit. Ebenso möchte ich Herrn Ao. Univ. Prof. Dr. techn. Manfred Grafinger für die Übernahme des Korreferats und die fachlichen Anregungen danken.

Weiter möchte ich mich bei meinen Kolleginnen und Kollegen, zunächst in den Forschungsgruppen WK und WIR und zuletzt der Gruppe TEC für die hilfreiche Zusammenarbeit bedanken.

Mit dem ersten agilen Team haben wir am Institut die Grundlage für den Schwerpunkt Daten- und Serviceökosysteme gelegt, bei den daran beteiligten Kollegen Dr.-Ing. Alexander Fertig und Oliver Kohn sowie im Schwerpunkt Markus Weber, Viktor Berchtenbreiter, Willi Wünschel, Fabian Gast und Augustino Doan möchte ich mich für die vielen fachlichen Diskussionen, Anregungen und gemeinsamen Entwicklungen sowie kurzzeitigen Betätigungen in der Logistik bedanken. Außerdem möchte ich mich bei Cornelia Tepper und Frederik Birk für die regelmäßige Motivation in der Abschlussphase dieser Arbeit bedanken.

Besonderer Dank gilt meiner Frau Anna: Ohne deine Liebe, Motivation und deinen Rückhalt hätte ich dieses Projekt nicht durchführen können. Vielen Dank auch für die regelmäßigen Diskussionen und die Unterstützung bei der Durchsicht und Korrektur der Arbeit.

Abschließend gilt mein größter Dank Jesus Christus, der aus Liebe zu mir den Tod am Kreuz auf sich genommen hat, damit ich in vollständiger Freiheit leben kann. Seine Einladung zu einem solchen Leben steht auch meinen Leserinnen und Lesern offen [Joh. 3, 16].

Darmstadt, im März 2024 Benjamin Brockhaus

Inhaltsverzeichnis

Verzeichnisse

Formelzeichen

Variable	Einheit	Bezeichnung
i, j		Zählvariablen
$C(N, L)$		Cluster
D		Menge von Abhängigkeiten
L		Menge von Verbindungen
N		Menge von Knoten
$P(a)$		Wahrscheinlichkeit von a
P_{el}	W	Elektrische Leistung
R		Menge von Computern
S		Route bzw. Weg
T		Menge von Tasks
V		Verteilungsfunktion
$W(T, D)$		Workflow
$\vec{a}$		Softwareverteilung
a_b		Amplitude von b
$d_{i\,j}$		Abhängigkeit zwischen Task t_i und t_j
f		Güte einer Lösung
h_i	bit	Headergröße eines Protokolls i
k		Relativer Leerlaufverbrauch
$l_{\alpha\,\beta}$		Verbindung zwischen den Knoten n_α und n_β
m_I		Anzahl der Elemente einer Menge I
m_i		Anzahl von i
n_α		Knoten α im Cluster
p_b	s	Periode von b
s		Schrittweite für *Simulated Annealing*
t_i		Task i eines Workflow
u_i		Auslastung von Ressource i
$\hat{u}_i$		Sollwert der Auslastung u_i
v	bit	Datenmenge
w		Gewichtungsfaktor
$\dot{x}$		(Partikel-)Geschwindigkeit

Griechische Formelzeichen

Variable	Einheit	Bezeichnung
α, β, γ		Zählvariablen
μ_x		Mittelwert der Größe x
σ_x		Standardabweichung der Größe x
ϑ_0		Starttemperatur *Simulated Annealing*
τ	s	Zeit
$\tau_{\max}$	s	Latenz
$\hat{\tau}_{\max\ i}$	s	Latenzanforderung für Subtask t_i

Abkürzungen

Bezeichnung	Beschreibung
ADU	Analog-Digital-Umsetzung.
AWS	*Amazon Web Services.*
CNN	*Convolutional Neural Network.*
CNT	*Cloud-native*-Technologien.
CPS	Cyber-Physisches System.
CPU	Prozessor – *Central Processing Unit.*
ERP	*Enterprise Resource Planning.*
FaaS	Function-as-a-Service.
FLOPS	*Floating Point Operations Per Second.*
GA	*Genetic Algorithm.*
GU	Großunternehmen.
HTTP	*Hypertext Transfer Protocol.*
IaaS	Infrastructure-as-a-Service.
IoP	*Internet of Production.*
IoT	*Internet of Things.*
IPC	Industrie-PC.
IT	Informationstechnologie.

Bezeichnung	Beschreibung
JSON	*JavaScript Object Notation.*
KI	Künstliche Intelligenz.
KMU	Kleine und Mittlere Unternehmen.
MEC	*Multi-Access Edge Computing.*
MEL	*MicroElement.*
MES	*Manufacturing Execution System.*
ML	Maschinelles Lernen.
MQTT	*Message Queue Telemetry Transport.*
NCU	*Numerical Control Unit.*
NIST	National Institute of Standards and Technology.
OPC UA	*Open Platform Communications Unified Architecture.*
PaaS	Platform-as-a-Service.
PM	*Predictive Maintenance.*
PSO	*Particle Swarm Optimization.*
PTW	Institut für Produktionsmanagement, Technologie und Werkzeugmaschinen.
RAM	Arbeitsspeicher – *Random Access Memory.*
REST	*Representational State Transfer.*
RPC	*Remote Procedure Call.*
SA	*Simulated Annealing.*
SaaS	Software-as-a-Service.
SDN	*Software-defined Networking.*
SoA	Serviceorientierte Architektur.
SSH	*Secure Shell.*
USB	*Universal Serial Bus.*
VM	Virtuelle Maschine.
WZM	Werkzeugmaschine.

Abbildungen

Tabellen

Code-Ausschnitte

1 Einleitung

Die Welt, und insbesondere die produzierende Industrie, ist – noch – geprägt durch die Globalisierung, Personalisierung von Produkten und immer kürzeren Produktlebenszyklen. Ein hoher Kostendruck durch gesättigte Märkte und weltweite Konkurrenz auch aus Niedriglohnländern sind weitere Einflußfaktoren auf die Industrie. In den letzten Jahren kamen Lieferkettenschwierigkeiten u. a. durch die COVID-19-Pandemie hinzu.

Diesen Herausforderungen soll durch die Technologien der vierten Industriellen Revolution, auch Industrie 4.0 genannt, begegnet werden. Allerdings, trotz starker Öffentlichkeitsarbeit vieler Beteiligter zeigt sich, dass in den 10 Jahren seit Verkündung dieser industriellen Revolution noch nicht viel Überzeugendes erreicht wurde, so die Zusammenfassung von Michael Finkler, Vorstand VDMA Software und Digitalisierung, in [Marx22]. Neben wenigen Konzernen und Großunternehmen (GU) mit bereits umgesetzten Lösungen stehen viele Kleine und Mittlere Unternehmen (KMU), die sich mit der Einführung entsprechender Technologien und Umgestaltung ihrer Geschäftsmodelle schwer tun. Dies ist trotz einer Fülle von Vorgehensmodellen [Dill22, S. 707] zur Einführung »der« Industrie 4.0 im Unternehmen auch heute in KMU weiterhin der Fall [Powe21, S. 31–32].

Um auch KMU an Industrie 4.0 teilhaben zu lassen, ist es, neben einem grundsätzlichen Umdenken bei den Unternehmen, notwendig, die beteiligten Technologien in ihrer Anwendung zu vereinfachen und Implementierungshürden abzubauen. Diese Arbeit soll dies für den Teilbereich der Recheninfrastruktur im *Shopfloor* erreichen. Damit werden von den 9 Industrie 4.0-Themenfeldern (siehe Abbildung 1.1) nach [Dill22] die Bereiche »*Horizontal and Vertical System Integration*«, sowie »*Internet of Things* (IoT)« und »Cloud« adressiert.

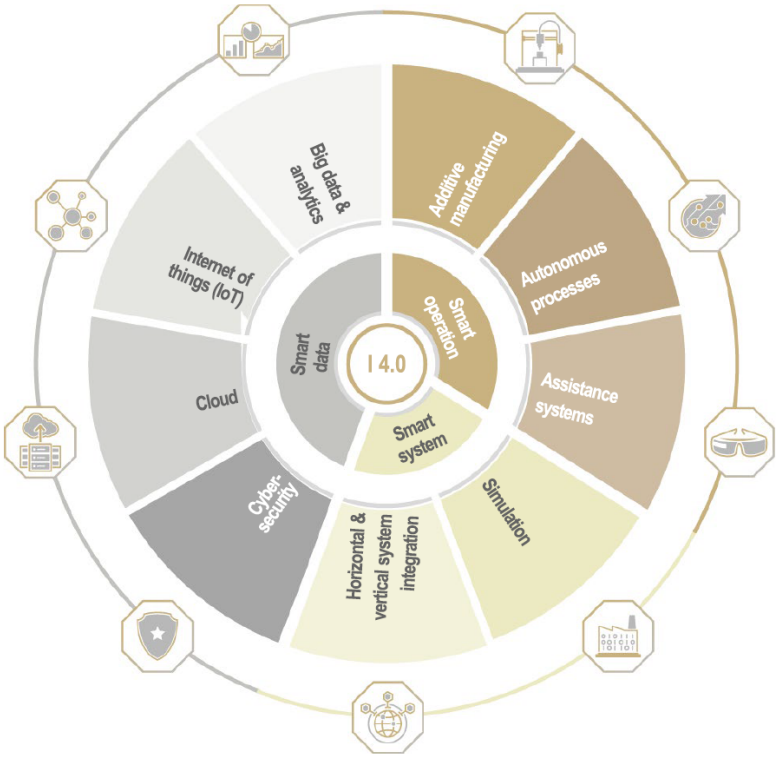


Abbildung 1.1: Die 9 Hauptthemenfelder von Industrie 4.0 [Dill22]

Ausgangslage Industrie 4.0, das *Internet of Production* (IoP) und das IoT verbreiten sich weiter und finden trotz der Schwierigkeiten wegen ihrer Potenziale in immer mehr Unternehmen der Produktionsindustrie Anwendung. Traditionell werden Applikationen im Maschinenbau allerdings mit einer anderen Perspektive entwickelt als dies in der Informatik üblich ist. Im Maschinenbau ist es üblich, dass ein Steuerungssystem einmalig eingerichtet wird und dann über einen großen Zeitraum – 20 Jahre und mehr – funktionsfähig bleiben muss.

Dieses Paradigma überträgt sich auch auf die Industrie 4.0-Anwendungen, die durch Maschinenbauunternehmen entwickelt werden. Die Anwendung wird häufig auf einem Industrie-PC (IPC) installiert und danach wird an diesem nichts mehr verändert, damit die Anwendung weiter lauffähig bleibt. Dies führt u. a. dazu, dass für Nachrüstungen – insbesondere wenn diese durch Drittfirmen durchgeführt werden – weitere Computer eingebaut werden.

Eine Rolle spielen hier auch Sicherheitsbedenken, denn es wird fälschlicherweise angenommen, dass der passwortgesicherte IPC das Know-how der Applikation wirkungsvoll vor Dritten schützt. Weiter erhöht der eingeschränkte Zugriff auf den IPC die Kontrolle über die Laufzeitumgebung der Applikation, wie beispielsweise installierte Treiber, Abhängigkeiten von bestimmten Bibliotheksversionen und Ähnliches. Und zuletzt ist so auch sichergestellt, dass die Applikation keine Konkurrenz um die Systemressourcen erhält. Allerdings führt dieses Vorgehen bereits zu Verschwendung, wenn die Rechnerhardware überdimensioniert für die Applikation ist. Dies ist sehr häufig der Fall, da die Applikationen selten für eingebettete Systeme entwickelt werden, sondern in der Regel der Komfort einer PC-basierten Umgebung vorgezogen wird.

Die oben beschriebenen Probleme sollen am Beispiel einer Werkzeugmaschine eines namhaften deutschen Werkzeugmaschinenherstellers verdeutlicht werden: Die Steuerung (Sinumerik 840D sl) besteht aus zwei IPCs, auf dem einen läuft der Echtzeitregelungsteil, auf dem zweiten die Oberfläche zur Bedienung der Maschine. Um Service und Wartungsarbeiten aus der Ferne anbieten zu können, wird zusätzlich ein IoT-Gateway – ein weiterer IPC – eingebaut, mit dem sich Wartungstechniker des Herstellers remote mit der Steuerung verbinden können. Zusätzlich bietet der Hersteller noch Applikationen zum *Condition Monitoring* der Werkzeugmaschine an, wofür ebenfalls ein weiterer IPC mit der entsprechenden Applikation verbaut wird. Für die zusätzliche Integration von Sensorik in die Werkzeugmaschine werden für zwei verschiedene Sensortypen zwei weitere IPCs mit entsprechenden Hardwareeingängen verbaut. In Summe sind in dieser Werkzeugmaschine sechs IPCs vorhanden,

die jeweils nur einzelne Aufgaben erfüllen. Wie in Kapitel 6 nachgewiesen wird, sind die Rechenkapazitäten der meisten dieser Computer bei weitem nicht ausgelastet.

Vor dem Hintergrund allgemein begrenzter Ressourcen wie Silizium erscheint dieser verschwenderische Umgang mit Rechnerhardware als nicht mehr zeitgemäß. Im Rahmen der COVID-19-Pandemie zusammentreffende Umstände führen seit 2020 zur Chip-Krise [Klei21] und machen den sparsamen Umgang mit den wenigen verfügbaren Chips dringend notwendig, um die Lieferfähigkeit des eigenen Unternehmens sicherzustellen. Die Klima- sowie Energiekrise der letzten Jahre – durch den Ukraine-Krieg in Europa 2022 weiter verschlimmert – erfordern auf der anderen Seite einen ebenso besonnenen Umgang mit Energieverbräuchen im gesamten gesellschaftlichen Spektrum, was auch die Industrie betrifft [Stur22]. Für beide Anforderungen sind nicht ausgelastete Computer kontraproduktiv, da diese einen relativ hohen elektrischen Grundenergieverbrauch haben, der unabhängig von der Auslastung ist, während gleichzeitig mehr Chips als nötig verbaut werden.

Zielrichtung *Cloud-native*-Technologien (CNT)[1] bieten eine mögliche Lösung für die Auslastungsdefizite und die oben genannten Ursachen wie isolierte Laufzeitumgebung oder Konkurrenz um Systemressourcen. Allerdings hat sich der Einsatz von Cloud-Computing im *Shopfloor* bisher nicht umfangreich etablieren können. Ansätze wie Steuerung aus der Cloud [Lech17] scheitern an technischen Einschränkungen, die mit Cloud-Computing einhergehen (siehe Abschnitt 2.4.1).

Mit Fog-Computing[2] lassen sich Aspekte des Cloud-Computings in den *Shopfloor* übertragen. Durch Kooperieren mehrerer IPCs als Fog-Cluster kann die Auslastung dieser effizienter gestaltet werden. Hierfür muss eine Verteilung der Software auf die Einzelrechner festgelegt werden.

Der Ansatz von Fog-Clustern ist in der Produktionstechnik allerdings noch nicht etabliert, diese Arbeit soll einen Beitrag dazu liefern diese zu etablieren. Dazu sollen angenommene Defizite bisher genutzter Verteilungsansätze, die einem Einsatz entgegenstehen, untersucht und ein Lösungsansatz entwickelt werden.

[1] Zum Begriff *Cloud-native* siehe Abschnitt 2.4.1 auf Seite 13.
[2] Zum Begriff Fog-Computing siehe Abschnitt 2.4.3 auf Seite 21.

Forschungsansatz Da im Bereich der Produktionstechnik nur wenig Forschung zu Fog-Clustern vorhanden ist, fehlt hierzu eine etablierte Methodik. Es wird daher ein exploratives Vorgehen entwickelt, um systematisch das Forschungsziel zu erreichen, ohne im Vorfeld alle Teilschritte detailliert zu kennen:

Für die Konkretisierung des Forschungsbedarfs wird über ein semi-systematisches Literaturreview der bestehende Forschungstand analysiert. Hierzu wird die Behandlung verschiedener a priori formulierter Besonderheiten der Produktionstechnik überprüft, sowie bestehende Ansätze zum Umgang damit ermittelt.

Im Anschluss wird aus der identifizierten Forschungslücke das Forschungsziel spezifiziert und ein Vorgehen mit den gefundenen Teilansätzen festgelegt. Über eine experimentelle Voruntersuchung werden getroffene Annahmen bei der Problembeschreibung überprüft.

Über eine Explorative Datenanalyse werden die aufgezeichneten Daten untersucht und Klassen ähnlicher Prozesse zusammengestellt. Diese Klassen lassen sich mittels deskriptiver Statistik auf einige Kenngrößen reduzieren.

Weiter wird aus den gefundenen Teilansätzen ein Gesamtansatz synthetisiert. Hierfür werden die Teilansätze zusammengeführt und um ggf. noch fehlende Aspekte erweitert. Der gewählte Gesamtansatz wird als Software umgesetzt und die Funktionsfähigkeit anhand eines Praxisbeispiels sichergestellt.

Abschließend wird der umgesetzte Gesamtansatz durch geeignete Prüfungen validiert. Hiermit soll sichergestellt werden, dass die verschiedenen Aspekte des Forschungsziels erfüllt werden.

Struktur der Arbeit Die weitere Arbeit gliedert sich entsprechend des Forschungsansatzes wie folgt: In Kapitel 2 werden für das Verständnis notwendige Grundlagen erläutert und Begriffe definiert. Daran schließt sich in Kapitel 3 das Literaturreview an, mithilfe dessen die Defizite der bisherigen Forschung ermittelt werden. Auf Basis des Reviews wird in Kapitel 4 die Zielsetzung formuliert.

In Kapitel 5 wird das Vorgehen mit den gefundenen Lösungsansätzen vorgestellt. Die Kapitel 6 und 7 dienen der Umsetzung des Vorgehens, bevor in Kapitel 8 die Validierung des Ansatzes beschrieben wird.

Kapitel 9 fasst abschließend die vorgestellte Arbeit zusammen und gibt einen Ausblick auf weitere Forschungsbedarfe.

2 Grundlagen

Im folgenden Kapitel werden für diese Arbeit notwendige Grundlagenthemen vorgestellt sowie Definitionen uneindeutiger Begriffe vorgenommen. Damit soll eine Grundlage für die darauffolgende Auseinandersetzung mit dem aktuellen Stand der Wissenschaft und Technik sowie der Gesamtarbeit geschaffen werden.

Zunächst werden die Begriffe Echtzeit und Ressourceneffizienz definiert, anschließend werden Datenverarbeitungsketten erläutert sowie ausgewählte Beispiele aus der aktuellen Produktionsforschung erwähnt. Daraufhin werden Cloud- und Edge-Computing – inklusive verwandter Themen wie bspw. Mist-Computing – beschrieben. Abschließend wird auf Service-Kompositionen – also Software aus mehreren (un-)abhängigen Bestandteilen – eingegangen.

2.1 Echtzeit

Applikationen in der Produktionstechnik – insbesondere Maschinenregelungen – zeichnen sich häufig durch Echtzeit-Anforderungen aus. Die zurückgezogene Norm DIN 44300 empfiehlt statt Echtzeit den Begriff Realzeitverarbeitung und definiert diese als:

> Ein Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen. [DIN 88]

Die Nachfolgenorm ISO/IEC 2382 definiert Echtzeit-Verarbeitung als Verarbeitung von Daten unter Einhaltung von Zeit-Anforderungen eines externen Prozesses [ISO 15]. Der Kernaspekt von Echtzeit ist also ein deterministisches Zeitverhalten beim Reagieren auf externe (oder auch interne) Ereignisse [Hüni19, S. 92].

Dabei unterscheidet man zwischen weicher und harter Echtzeit. Weiche Echtzeit toleriert eine gelegentliche und geringfügige Überschreitung der Zeitanforderung, was ein in der Informatik häufiges Verständnis darstellt. Maschinenregelungen benötigen harte Echtzeit, das bedeutet, dass die maximale

Abarbeitungszeit einen bestimmten Schwellwert nicht überschreiten darf. [Zöbe20, S. 3]

In dieser Arbeit wird der Begriff im Sinne harter Echtzeit verwendet und bedeutet, dass Tasks innerhalb einer vorgegebenen Zeitspanne ihre Berechnung abgeschlossen haben müssen. Für die Positionsregelung einer Vorschubachse bedeutet dies beispielsweise, dass die Erfassung der aktuellen Ist-Position sowie die Berechnung einer Stellgröße für den unterlagerten Geschwindigkeitsregler nach maximal 2 ms abgeschlossen sein muss. Der Regelungstask läuft dabei zyklisch alle 2 ms ohne dass er jeweils neu gestartet werden muss.

2.2 Ressourceneffizienz

In der VDI 4800-1 ist Ressourceneffizienz allgemein definiert als »das Verhältnis eines bestimmten Nutzens oder Ergebnisses zum dafür nötigen Ressourceneinsatz« [VDI 16].

Dieser Nutzen bzw. die funktionale Einheit ist bei dieser Arbeit die Ausführung einer Software. Die Herstellung der Software – z. B. Kompilierung, Transport oder Installation – liegt außerhalb des Betrachtungsrahmens, somit ist auch nur der Energieaufwand während der Nutzung maßgeblich [VDI 12].

Die Normen VDI 4800-1 und VDI 4800-2 [VDI 18] sehen für den Ressourceneinsatz vor, u. a. Primärrohstoffe und Energieressourcen zu erfassen. Für diese Arbeit wird statt der Energieressourcen – wie bspw. Steinkohle – vereinfachend direkt die Endenergie in Form von elektrischem Strom beim Endanwender genutzt. Primärrohstoffe werden durch die Nutzung eines Computers – wieder ohne Herstellung – nicht verbraucht. Eine Ausnahme wäre der Einbau von Ersatzteilen, der aber als Sonderfall in dieser Arbeit nicht berücksichtigt wird.

Somit ist für diese Arbeit unter Ressourceneffizienz das Verhältnis zwischen der Menge ausgeführter Software und dem dafür aufgewendeten elektrischen Energieverbrauch zu verstehen. Praktisch bedeutet dies: Wenn mehr Software bei gleichem Energieverbrauch ausgeführt wird, ist die Ressourceneffizienz höher. Genauso wenn eine konstante Softwaremenge mit geringerem Energieverbrauch ausgeführt wird.

2.3 Datenverarbeitungsketten

In diesem Kapitel soll ein kurzer Einstieg in das Thema Datenverarbeitungsketten gegeben werden. Anschließend werden die im Rahmen dieser Arbeit relevanten Ketten vorgestellt.

Während in früheren Zeiten primär zentrale Systeme wie ein *Enterprise Resource Planning* (ERP) oder *Manufacturing Execution System* (MES) Daten aus dem *Shopfloor* erfasst und verarbeitet haben, so findet diese Verarbeitung heute in Teilen bereits nahe an den Maschinensteuerungen statt. Diese Verarbeitung nahe an der Datenquelle hat verschiedene Vorteile, unter anderem müssen große Datenmengen nicht weit durch das Netzwerk transportiert werden, sondern können direkt am Rande des Netzwerks der sogenannten *Edge* verarbeitet werden. Diese führt zu geringer Latenz und verringert den Bedarf an zentral vorgehaltener Rechenkapazität.

Die erfassten Rohdaten werden zur Analyse und teilweise auch Prozessregelung durch mehrstufige Datenverarbeitungsketten zunächst vorverarbeitet und dann ihrem eigentlichen Zweck zugeführt. Typische Datenverarbeitungsketten enthalten die folgenden Stufen:

1. analoge Vorverarbeitung,
2. Analog-Digital-Umsetzung (ADU), (engl. ADC),
3. digitale Vorverarbeitung,
4. Merkmalsextraktion,
5. Analyse/Überwachung,
6. Reaktion,
7. Visualisierung & Interaktion mit Personen.

Diese Teilschritte sehen bei verschiedenen Anwendungsfällen unterschiedlich aus. Insbesondere bei Einsatz von Künstlicher Intelligenz (KI) oder Maschinellem Lernen (ML) in der Produktion finden die Bereiche Merkmalsextraktion und Überwachung besondere Beachtung.

Die Betrachtung einer einzelnen Datenverarbeitungskette zeigt noch keine Einsparpotenziale auf, da jeder der Schritte notwendig ist, um zu einem qualitativen Ergebnis zu kommen. Kommen allerdings parallel mehrere Ketten zum Einsatz können einzelne Schritte gemeinsam verwendet werden. Insbesondere analoge Vorverarbeitung und ADU müssen in der Regel nur einmal erfolgen, aber auch in der digitalen Domäne können Schritte der digitalen Verarbeitung für mehrere Anwendungsfälle genutzt werden, sodass die Da-

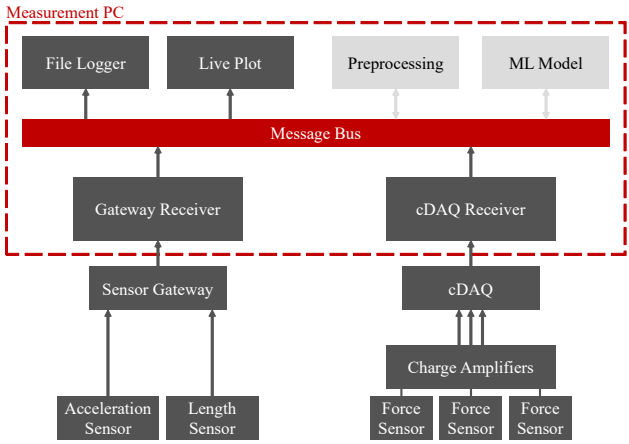


Abbildung 2.1: Systemarchitektur für eine Zustandsüberwachung von Werkzeugmaschinenkomponenten [Broc22]

tenverarbeitungsketten bspw. erst ab dem Schritt der Merkmalsextraktion divergieren.

Die einzelnen Schritte einer Datenverarbeitungskette in der digitalen Domäne – also ab Schritt 3 – können auch als Teilservices – im Sinne des Architekturmusters Serviceorientierte Architektur (SoA) – einer Service-Komposition[3] aufgefasst werden, was sie für eine verteilte Ausführung prädestiniert. Dies ist beispielsweise für den Anwendungsfall der Zustandsüberwachung (engl. »Condition Monitoring«) aus [Broc22] ersichtlich. Die entworfene Datenverarbeitungskette führt zu der in Abbildung 2.1 dargestellten Systemarchitektur. Einzelne Schritte der Datenverarbeitungskette korrespondieren hier mit den Komponenten der Architekturdarstellung, was die Interpretation als Servicekomposition erleichtert und eine Verteilung der einzelnen Software-Komponenten vorbereitet. Dieses Beispiel wird in Abschnitt 2.5 wieder aufgegriffen.

In [Fert22a] wird eine Methode und zugehörige Datenverarbeitungskette vorgestellt, mit der maschineninterne Messdaten einer Werkzeugmaschine (WZM) mit Kontextinformationen ergänzt werden. Dies erhöht die Nutzbarkeit der aufgezeichneten Daten für weitere Analyseschritte. Darauf aufbauend wird von Fertig, Weigold und Chen vorgestellt, wie die Daten automatisiert zugeschnitten werden können, um Abschnitte einzelnen geometrischen Elementen der gefertigten Bauteile zuzuweisen [Fert22b]. Das Ziel der Datenverarbeitungsket-

[3] Zum Begriff Service-Komposition siehe Abschnitt 2.5 auf Seite 30

te ist die Prognose der Qualität der gefertigten Bauteile. Auch hier finden sich Schritte in der Vorverarbeitung, die idealerweise für mehrere Anwendungszwecke verwendet werden können. Eine ähnliche Datenverarbeitungskette wird in [Stan21] und [Kohn22] genutzt, um belastungsorientierte Bezahlmodelle für geleaste WZMs zu entwickeln. Beide Datenverarbeitungsketten werden bei der – in Kapitel 6 vorgestellten – Messung neben anderen Prozessen vermessen.

2.4 Computing-Modelle

In diesem Kapitel werden zunächst die Grundlagen von Cloud- & Edge-Computing erläutert. Aus diesen beiden gegensätzlichen und doch voneinander beeinflussten Ansätzen wird das Konzept des Fog-Computings abgeleitet. Diese drei Ansätze werden häufig zu einer Schichtenarchitektur kombiniert – Cloud-Fog-Edge –, wobei die Bezeichnungen der einzelnen Ebenen teilweise unterschiedlich gewählt werden (beispielsweise [Aral20; Gogo20; Yous19]).

Anschließend werden in diesem Kapitel weitere Computing-Arten – wie Osmotic-Computing – angeführt, die inhaltliche Überschneidungen mit den Zielen dieser Arbeit haben.

2.4.1 Cloud-Computing

Unter Cloud-Computing versteht man im Allgemeinen die Nutzung von IT-Infrastruktur, die nicht vor Ort betrieben wird [Arke18]. Unter einer Cloud kann man ein Rechenzentrum verstehen, das in der Regel von einem externen Unternehmen betrieben wird, und in dem man sich nach Bedarf Ressourcen anmietet. Darauf aufbauend ergeben sich neue Geschäftsmodelle auch für die Nutzung von Software, diese werden im Abschnitt Dienstleistungsmodelle erläutert. Der Einsatz und die Menge solcher Angebote steigen seit ca. 2010 stark an.

Definition von Cloud-Computing

Charakteristiken Mell und Grance sowie viele weitere Autoren nutzen 5 Charakteristiken zur Definition von Cloud-Computing für das National Institute of Standards and Technology (NIST) der Vereinigten Staaten von Amerika [Mell11; Sehg18]:

1. On-Demand Self-Service: Anwender:innen können bei verändertem Bedarf selbstständig weitere Angebote über automatisierte Prozesse

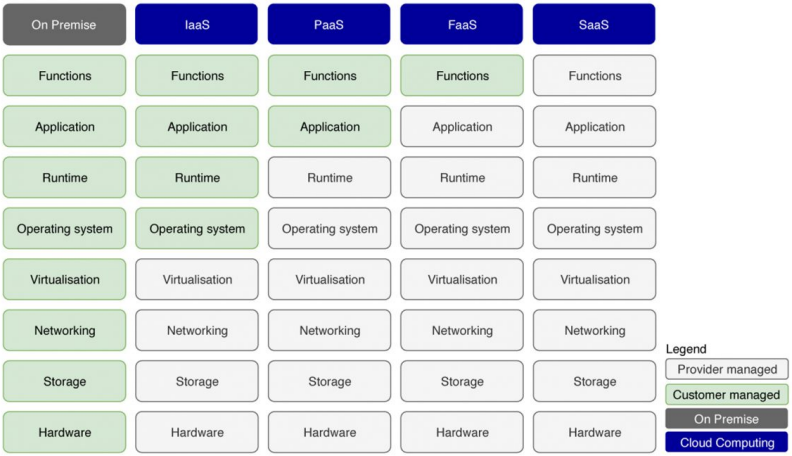


Abbildung 2.2: Zuordnung der Zuständigkeiten zu den Dienstleistungsmodellen bei Cloud-Computing [o V19]

buchen. Hierzu ist keine Interaktion mit Mitarbeitern des Anbieters notwendig.

2. Broad Network Access: Die angebotenen Ressourcen sind über Netzwerk erreichbar.
3. Resource Pooling: Mehrere Kunden des Anbieters werden auf der gleichen Hardware zusammengefasst.
4. Rapid Elasticity: Die gebuchten Ressourcen können »elastisch« skaliert werden. D. h. sie können dem Bedarf angepasst werden, indem weniger oder mehr Ressourcen verwendet – und damit auch abgerechnet – werden.
5. Measured Service: Die genutzten Ressourcen werden gemessen. Dies dient anbieterseitig zur Optimierung der Hardware-Nutzung und für den Kunden zur Abrechnung der in Anspruch genommenen Angebote (*Pay per Use* bzw. im Cloud-Kontext häufig auch *Pay as you go*).

Dienstleistungsmodelle Cloud-Computing kann nach 3–4 Dienstleistungsmodellen der Angebote unterschieden werden. Diese Modelle bauen hierarchisch aufeinander auf und sind durch abnehmenden Aufwand seitens der Anwen-

der:innen gekennzeichnet. In Abbildung 2.2 ist zu sehen, welche Aufgaben je Modell der Anbieter übernimmt, und welche Bausteine seitens der Anwender:innen betreut werden müssen. Als Referenz ist die Spalte On-Premise zu sehen, bei der sämtliche Funktionalitäten von der unternehmenseigenen IT-Abteilung bereitgestellt werden.

Als Basis-Ebene kann Infrastructure-as-a-Service (IaaS) bezeichnet werden, hierbei werden Hardware-Ressourcen – wie Speicher, Server, virtuelle Maschinen ... – als Dienstleistung bereitgestellt [Mell11]. Beispielsweise können die Elastic-Compute-Instanzen des Anbieters *Amazon Web Services* (AWS) zu diesem Bereich gezählt werden.

Darauf aufbauend gibt es das Dienstleistungsangebot Platform-as-a-Service (PaaS), hier wird neben der reinen Hardware-Nutzung bereits weitergehende Funktionalität bereitgestellt und Anwender:innen müssen sich nicht um die zugrundeliegende Hardware oder Basis-Software kümmern [Gil21]. So können bereits Betriebssystem, Entwicklungsumgebungen u. ä. enthalten sein.

Als dritte Ebene wird Software-as-a-Service (SaaS) bezeichnet, bei dem vollständige Softwareanwendungen – also inklusive aller Funktionalitäten – dem Kunden angeboten werden [Mell11]. Ein bekanntes Beispiel ist Microsoft Office 365.

Aufbauend auf den 3 Grund-Ebenen aus der NIST Definition von Cloud-Computing ergänzen neuere Quellen [Asla21; Eyk18; Gill21; Rait22; Robe17] noch eine vierte Ebene zwischen PaaS und SaaS: Function-as-a-Service (FaaS) auch *Serverless Computing* genannt. Hierbei übernimmt der Cloud-Anbieter sämtliches Management der notwendigen Ressourcen, deren Lebenszyklen und die eventgesteuerte Ausführung einer Funktion. Lediglich die Funktion – also die Geschäftslogik – wird vom Kunden bereitgestellt oder aus Bausteinen zusammengestellt. [Eyk18]

Liefermodelle Neben den Charakteristiken und Dienstleistungsumfängen unterscheidet das NIST auch nach der Frage wo bzw. für wen die Cloud eingesetzt wird [Mell11]. Diese Unterscheidung findet primär nach öffentlichem oder privatem Nutzerkreis statt und ist für diese Arbeit nicht relevant.

Ausgewählte Cloud-native Technologien

Die Cloud Native Computing Foundation beschreibt den Nutzen von CNT wie folgt:

> Cloud native Technologien [sic] ermöglichen es Unternehmen, skalierbare Anwendungen in modernen, dynamischen Umgebungen zu implementieren und zu betreiben. Dies können öffentliche, private und Hybrid-Clouds sein. [...]
>
> Die zugrundeliegenden Techniken ermöglichen die Umsetzung von entkoppelten Systemen, die belastbar, handhabbar und beobachtbar sind. Kombiniert mit einer robusten Automatisierung können Softwareentwickler mit geringem Aufwand flexibel und schnell auf Änderungen reagieren. [Clou18]

Im Folgenden werden einige CNT vorgestellt, die für das Verständnis dieser Arbeit relevant sind.

Microservices Das Architekturmuster der Microservices ist abgeleitet von der SoA. Die Anwendung wird hierbei in sehr kleine funktionale Einheiten, die Microservices, zerlegt. Diese Dienste kommunizieren miteinander bei Bedarf in der Regel über *Remote Procedure Call*s (RPCs). Anwendungen können so aus wenigen bis tausenden Instanzen dieser Microservices komponiert sein. Vorteilhaft ist, dass jeder Microservice durch neuere Versionen im Betrieb ausgetauscht werden kann, ohne den Betrieb des Gesamtsystems zu unterbrechen. [Gann17, S. 21]

Jeder Microservice sollte – soweit wie möglich – zustandslos (»stateless«) sein [Gann17, S. 18], womit sich *Representational State Transfer* (REST) als Schnittstellenparadigma anbietet[4].

Virtualisierung & Container Die Virtualisierung von IT-Infrastruktur ermöglicht erst das Bündeln vieler Kunden auf der gleichen physischen Hardware-Infrastruktur [Chen21, S. 2]. Klassisch bedeutete dies Virtuelle Maschines (VMs) auf einem Hypervisor, was für Microservices allerdings zu viel Overhead erzeugt, da die VMs ein vollständiges Gast-Betriebssystem enthalten, wie auch in Abbildung 2.3a zu sehen ist. Im Vergleich dazu sind Container (Abbildung 2.3b) eine leichtgewichtigere Lösung, die durch Funktionalitäten des Linux-Kernels ermöglicht wurden [Gann17, S. 18]. Prozesse können hier in einem eigenen Namespace ablaufen und zusätzlich – über cgroups – mit Ressourcenbeschränkungen belegt werden. Hieraus entwickelten sich defacto standardisierte Container-Runtimes wie Docker [Gann17, S. 18]. Für viele IoT-Geräte ist auch ein Container noch zu groß, bzw. viele dieser Geräte besit-

[4] Zu »Representational State Transfer« (REST), siehe [Fiel00, Kapitel 5]

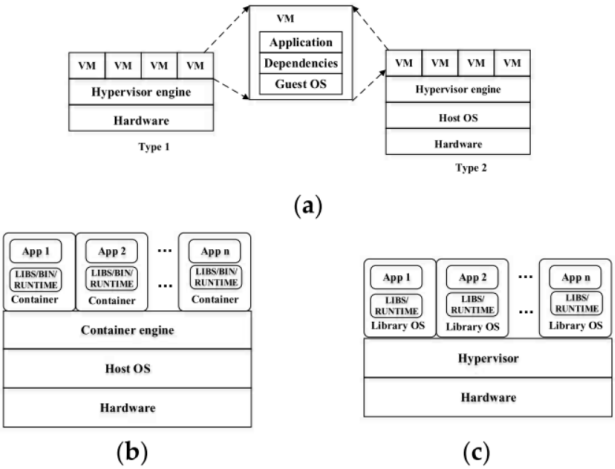


Abbildung 2.3: Unterschiede verschiedener Virtualisierungstechnologien. (a) Virtuelle Maschinen mit Hypervisor Typ 1 & Typ 2; (b) Container; (c) Unikernel [Chen21]

zen kein eigenes Betriebssystem auf dem die Container-Engine laufen könnte, daher werden aktuell Unikernel (Abbildung 2.3c) als noch leichtgewichtigere Alternative entwickelt [Chen21, S. 4]

Container-Orchestrierung Der Einsatz virtualisierter Microservices benötigt eine Management-Ebene, welche die Services zur Laufzeit überwacht und bei Fehlern oder Skalierungsbedarf neue Instanzen startet. Ein Beispiel, bzw. der Inbegriff dieser Orchestrierungslösungen wurde von Google entwickelt und später unter dem Namen »Kubernetes« der Allgemeinheit zur Verfügung gestellt. Kubernetes setzt dabei auf Pods, die wiederum Container (s. o.) bündeln. Alternativen zu Kubernetes stehen u. a. mit Apache Mesos, Docker Swarm oder Microsoft Azure Service Fabric bereit. [Gann17, S. 18]

Service-Discovery Um auf einzelne Dienste zuzugreifen, wird deren Netzwerkadresse (Kombination aus IP-Adresse und Port) benötigt. Bei SoA und insbesondere Microservice-Anwendungen stellt dies eine Herausforderung dar, da durch das dynamische Starten und Beenden von Instanzen – die sich auch

auf anderer physischer Hardware befinden können – die Verbindung zwischen Service und Netzwerkadresse zur Laufzeit veränderlich ist. Die Funktionalität der Service-Discovery besteht darin, die Netzwerkadressen und die Zustände der laufenden Services zu überwachen und ein Auflösen von Servicenamen zu konkreten Netzwerkadressen anzubieten. Die Umsetzung erfolgt üblicherweise über eine Datenbank, die sogenannte *Service Registry*, bei der sich Dienste anmelden. [Gark21, S. 37–38]

Defizite von Cloud-Computing für die Produktion

Cloud-Computing bietet auch für Produktionsbetriebe einige Vorteile, wie z. B. dass nur die in Anspruch genommenen Rechenkapazitäten abgerechnet werden. Auch die einfache und kurzfristige Skalierung der Rechenkapazitäten bei Bedarf ist ein weiterer Vorteil. Den Vorteilen stehen aber auch einige Defizite gegenüber, die einen Einsatz von Cloud-Computing in diesen Unternehmen behindern. Diese Defizite sollen im folgenden Abschnitt erwähnt werden.

Ein großer Nachteil ist der notwendige Transport der Daten vom Unternehmen ins Cloud-Rechenzentrum und zurück. Eingehender und ausgehender Datenverkehr in bzw. aus dem Cloud-Rechenzentrum heraus ist mit Kosten verbunden [Sehg18, S. 171]. Zusätzlich benötigt der Transport einige Zeit, was bei typischen Webanwendungen kein Problem darstellt, im Bereich der Produktionstechnik allerdings Schwierigkeiten – man denke bspw. an Regelungsaufgaben im Maschinenumfeld – bereiten kann. Mohan, Corneo u. a. zeigten in ihrer Studie, dass selbst in Europa und Nordamerika, nur ca. 75 % der verwendeten Messstellen eine Round-Trip-Time zur nächsten Cloud von unter 25 ms erreichen konnten [Moha20, S. 4]. Dies bedeutet das immerhin 25 % auf eine Antwort aus der Cloud länger als 25 ms warten mussten. Für Ansätze wie einer Steuerung aus der Cloud [Vgl. Lech17] ist dies nicht vertretbar, insbesondere die schwankende Latenz der Übertragung stellt die Regelungstechnik vor große Herausforderungen.

Aus der Sicht von Energie- und Ressourcenverbräuchen stellt sich der Transport der Daten zwischen Cloud-Rechenzentrum und Unternehmen ebenfalls als nicht ideal dar. Ob durch die geteilte Hardware in der Cloud im Vergleich zu den gestiegenen Energieverbräuchen beim Datentransport am Ende ein Effizienzgewinn herauskommt, hängt stark vom Anwendungsfall und den zu transportierenden Daten ab. Ebenfalls relevant ist die Strategie, nach welcher der Cloud-Anbieter Kunden auf seine Maschinen verteilt, hier zeigt sich, dass die verwendeten Server häufig nicht gut ausgelastet sind was ebenfalls die Energieeffizienz drückt [Vasa10, S. 6]. Ein eingeschalteter, aber unbelasteter,

Server verbraucht ca. 70 % der elektrischen Energie eines voll ausgelasteten Servers [Hoss20, S. 272; Belo12, S. 759].

Produktionsunternehmen sehen sich mit einem weiteren Problem konfrontiert bei der Übergabe von Daten an einen externen Cloud-Computing-Anbieter: Das Unternehmen gibt seine Datensouveränität auf. Datensouveränität wird von Eggers, Fondermann u. a. definiert als die Ausübung voller Kontrolle über die Lokalisierung der eigenen Daten, sowie über sämtliche Aspekte der Nutzung der Daten [Egge20, S. 3]. Beispielsweise enthalten Prozessdaten aus der Fertigung viele Hinweise auf Know-how des Fertigungsunternehmens, die andere – insbesondere konkurrierende – Unternehmen nicht sehen sollen. Beim Einsatz von Cloud-Computing muss das Unternehmen darauf vertrauen, dass der Cloud-Anbieter die Daten ausreichend schützt, um versehentliches Einsehen durch Dritte auszuschließen. Weiterhin muss darauf vertraut werden, dass der Anbieter die Daten nicht unmittelbar oder auch indirekt an konkurrierende Unternehmen weitergibt oder selbst als Konkurrent auftritt.

Die Bindung an den ausgewählten Cloud-Anbieter ist in der Regel technisch oder wirtschaftlich stark ausgeprägt. Man spricht hierbei vom *Vendor-Lock-In*-Effekt. Durch hohe direkte Kosten für bspw. den Transfer von Daten heraus aus dem Rechenzentrum oder technische Aufwände wird indirekt erzwungen, dass der Kunde den Anbieter nicht wechselt. Damit ist die Wahl des Cloud-Anbieters mit Risiken behaftet, denn ob der Anbieter auf Dauer sein Preis- oder auch Lizenzmodell beibehält oder nicht irgendwann beginnt seine dominante Stellung auszunutzen, kann nicht garantiert werden. [Sehg18, S. 6]

Wie beschrieben basiert das Geschäftsmodell für Cloud-Anbieter darauf, dass viele Kunden auf einer Hardware aggregiert werden. Trotz Virtualisierung und damit getrennten Subsystemen müssen die Kunden sich gewisse zentrale Komponenten teilen. Beispielsweise sind das die Netzwerkzugänge des Rechenzentrums aber auch pro Server die Speicherverwaltung. Hierbei kommt es zu einem Flaschenhals, durch den die Performance der Kunden-Anwendung sinken kann, das Problem wird häufig als *Noisy Neighbours* bezeichnet. [Sehg18, S. 172]

2.4.2 Edge-Computing

Der Begriff Edge-Computing kommt aus dem Bereich der Mobilfunknetze und bezeichnet dort Rechenknoten am Rande (engl. *Edge*) des Netzwerks, bspw. direkt in einer Funkzelle, im Gegensatz zum zentralen Cloud-Rechenzentrum. Im Bereich des IoT und der Industrie 4.0 findet man zunächst eine damit

kohärente Verwendung des Begriffs. Hier wird die Anwesenheit von Rechenressourcen (sogenannte »*Edge-Devices*«) in der Nähe der Datenquellen darunter verstanden [Al-D20, S. 4]. Allerdings gibt es branchenspezifisch deutlich unterschiedliche Vorstellungen vom Rande des Netzwerks, weswegen der Begriff des Edge-Kontinuums vorgeschlagen wurde. Dieser Begriff wird daher in einem folgenden Abschnitt noch näher erläutert. Zunächst soll jedoch die Motivation hinter dem Trend zum Edge-Computing dargestellt werden.

Entwicklung von Edge-Computing

Der Trend zum Edge-Computing ist motiviert durch die bereits dargestellten Defizite von Cloud-Computing. Insbesondere die Latenz durch den Datentransport zum zentralen Cloud-Rechenzentrum kann umgangen werden, indem bei Edge-Computing Rechenressourcen in direkter Nähe zur Datenquelle platziert werden [Strn21, S. 180]. Durch die zunehmende Anzahl von IoT-Geräten steigen auch die zu verarbeitenden Datenmengen rasant an, sodass ein vollständiger Transport zum Rechenzentrum von der Netzwerkinfrastruktur gar nicht geleistet werden kann [Pan18, S. 439], man denke beispielsweise an die aufgezeichneten Datenmengen im Kontext des autonomen Fahrens.

Auch bei der Energieeffizienz kann Edge-Computing eine Verbesserung bedeuten. Varghese, Wang u. a. [Varg16, S. 21] postulieren, dass viele kleine, einfache Rechenaufgaben wie bspw. Aggregation von Daten auf Edge-Geräten mit geringen Leistungen effizienter ausgeführt werden können als auf einem Cloud-Server im Rechenzentrum. Als Zusatzvorteil reduziert sich der Datenstrom zur Cloud, was ebenfalls den Energieverbrauch senkt.

Dem Problem der fehlenden Datensouveränität beim Cloud-Computing kann mit Edge-Computing ebenfalls begegnet werden [Strn21, S. 180]. Denn die Edge-Devices müssen in der Regel von den Produktionsunternehmen selbst betrieben werden, womit volle Kontrolle über den Umgang mit den erhobenen Daten gewährleistet ist. Dass andere Kunden des Cloud-Anbieters Ressourcen blockieren ist so ebenfalls ausgeschlossen.

Sofern die Edge-Devices vom Unternehmen selbst entwickelt und betrieben werden, ist ein *Vendor Lock-in* ebenfalls nicht gegeben. Allerdings kommen häufig auch Edge-Computing-Plattformen zum Einsatz, da die Unternehmen i. d. R. keine ausreichenden Kompetenzen oder Kapazitäten für die Entwicklung von Edge-Devices haben. Hier kann dann wiederum ein *Vendor-Lock-In*-Effekt entstehen, wenn die Interoperabilität mit anderen Edge-Plattformen nicht gegeben ist.

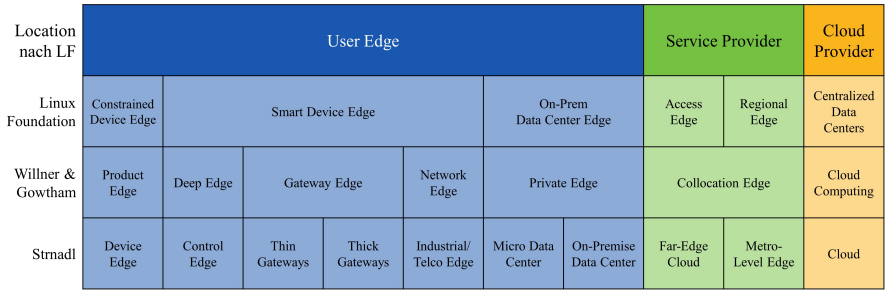


Abbildung 2.4: Gegenüberstellung der verschiedenen Detaillierung des Edge-Kontinuums (Eigene Darstellung, Inhalte gemäß [Linu20], [Will20], [Strn21])

Edge-Computing ersetzt oder ergänzt daher Cloud-Computing in Bereichen mit hohen Datenmengen oder besonderen Latenzanforderungen. Damit ist es besonders geeignet für die Anwendungen auf dem *Shopfloor* der produzierenden Industrie. Allerdings folgen Edge-Computing-Anwendungen höchstens teilweise Cloud-native-Ansätzen oder nutzen entsprechende Technologien [Strn21, S. 181].

Edge-Kontinuum

In der täglichen Kommunikation zeigt sich, dass unter dem Begriff Edge-Device durchaus sehr unterschiedliche Geräte verstanden werden. Dies führt zum Begriff des Edge-Kontinuums, den u. a. Strnadl oder die Linux Foundation um ihre Gruppe »LF Edge« nutzen und definieren [Linu20; Strn21]. Willner und Gowtham sprechen 2020 noch von »edge hierachy levels«, bezeichnen damit aber das gleiche Konzept [Will20]. Wenn Cloud-Computing mit einbezogen wird, kann auch von Edge-Cloud-Kontinuum bzw. Cloud-Edge-Kontinuum gesprochen werden.

Das Edge-Kontinuum setzt sich aus den verschiedenen Geräten zusammen, die von den Daten ausgehend vom (IoT)-Gerät bis zu einer Cloud-Infrastruktur (typ. eines Hyperscalers) durchlaufen werden können [Strn21]. Hierbei unterscheiden sich die Detaillierungsgrade der erwähnten Darstellungen. In Abbildung 2.4 sind die genutzten Edge-Klassen der verschiedenen Quellen gemeinsam dargestellt.

Zunächst kann nach dem Betreiber der Edge-Infrastruktur unterteilt werden. Dies können die Anwender (»*User Edge*«) oder Telekommunikationsanbie-

ter, wie die Internet Service Provider oder Betreiber des Mobilfunknetzes sein (»*Service Provider*«). Dieser Edge-Bereich wird häufig auch als *Multi-Access Edge Computing* (MEC) bzw. historisch als *Mobile Edge Computing* bezeichnet [Hu15, S. 4]. Hier finden sich bereits Cloud-ähnliche Cluster, die lediglich in kleinerem Maßstab aufgebaut sind – im Bereich von zehn bis einigen hundert Servern [Zhao20, S. 226]. Durch die Platzierung bspw. an Funkmasten des Mobilfunknetzes stehen zusätzlich Positionsinformationen der Endanwender bereit. Am rechten Ende der Darstellung des Edge-Kontinuums in Abbildung 2.4 ist zusätzlich Cloud-Computing dargestellt.

Die Linux Foundation hat die gröbste Unterteilung des Kontinuums und trennt entsprechend der zugrunde liegenden Hardware. So basieren »*Constrained Device Edges*« auf typischen Microcontrollern mit eingebetteter Software, wohingegen ab »*Smart Devices*« bereits x86- oder ARM-Architekturen zum Einsatz kommen, diese aber noch nicht durch typische Server gebildet werden. Dies beginnt ab dem Bereich »*On-Premise Data Center Edge*«. [Linu20]

Strnadl weist die größte Detaillierung auf, insbesondere im Bereich »*Smart Device Edge*« bei den Anwendern werden viele verschiedene Klassen unterschieden. Die Benennung und Beschreibung wie »*Control Edge*« deutet auf den Ursprung im Maschinenbau und insb. der Produktionstechnik hin, da die klassische Automatisierungspyramide recht gut abgedeckt wird. Für diese Arbeit wird im Folgenden i. d. R. die Benennung der Linux Foundation genutzt.

Nachteile von Edge-Computing

Wie bereits im Abschnitt Edge-Kontinuum gezeigt, sind Edge-Devices sehr heterogen. Sie können auf Microcontrollern über Einplatinenrechnern wie dem Raspberry Pi mit ARM-Architektur bis hin zu Servern auf x86- oder amd64-Architektur basieren. Durch die hohe Heterogenität der eingesetzten Edge-Devices müssen Softwarestacks häufig angepasst werden, wenn sie von einem Edge-Device auf ein anderes übertragen werden sollen. Eine generisch einsetzbare Software wird dabei selten alle Leistungs-Potenziale der Hardware ausschöpfen können. [Berm18, S. 5; Yu18, S. 6910]

Zusätzlich zeigt sich bei der Implementierung des Edge-Computings ein weiteres Problem, dass allerdings nicht prinzipbedingt ist: Anders als beim Cloud-Computing, das durch wenige große Anbieter dominiert ist, gibt es beim Edge-Computing bisher keine etablierten Standards oder Vorgehensmodelle [Gark21, S. 11].

Auch nimmt den Entwickler:innen bzw. Betreiber:innen einer Edge-Applikation niemand das Management der Applikation ab, was beim Cloud-Computing geschieht. Während der Cloud-Anbieter entscheidet, wo und wie die Applikation genau betrieben wird (und damit für den Nutzer die Infrastruktur transparent macht), muss das für Edge-Computing durch das Anwendungsunternehmen selber gelöst werden. [Shi16, S. 641]

Ein weiteres Defizit von Edge-Devices ist die physische Sicherheit. Die Geräte sind physisch relativ offen zugänglich, wenn keine Schutzmaßnahmen getroffen werden [Berm18, S. 6]. Cloud-Rechenzentren sind im Allgemeinen zutrittsbeschränkt, und durch diverse Sicherheitsmaßnahmen geschützt. Dies ist bei Edge-Devices nur bedingt umsetzbar, was Angriffe wie *Node Capture Attacks* oder auch *Node Replication Attacks* möglich macht [Lin17, S. 1132]. Dabei ist es möglich auf den Knoten vorgehaltene Informationen wie Daten oder sogar private Schlüssel oder Zugangsdaten auszulesen und zu missbrauchen. Selbst im Bereich der Service Provider Edge ist eine Absicherung mit ähnlichem Niveau wie im Cloud-Rechenzentrum nicht machbar [Pan18, S. 441]. Da in Produktionsunternehmen neben dem eigenen Personal häufig auch Dienstleistungsfirmen agieren, sind entsprechende Sicherheitsmaßnahmen vorzusehen.

Taxonomie von Edge-Computing

Ahmed, Ahmed u. a. [Ahme17] stellen eine Taxonomie für Edge-Computing vor (siehe Abbildung 2.5), die in der Klasse Objectives u. a. die Ziele dieser Dissertation enthält. Weiter empfehlen die Autoren eine Unterteilung von Edge-Computing nach den Netzwerktechnologien, der Geräteklasse, Computing-Modellen[5], den genutzten Enabler-Technologien, der Position im Edge-Kontinuum und den Anwendungsbereichen vor (wobei Industrie und Produktion leider außen vor gelassen werden).

2.4.3 Fog-Computing

Fog-Computing bezeichnet vereinfacht gesagt Cloud-Computing aber dezentral(er) am Rande des Netzwerks. Der Name ist eine Anspielung auf Wolken (engl. *clouds*), die nahe am Boden als Nebel (engl. *fog*) bezeichnet werden:

> Fog is cloud closer to the ground. [Mour18, S. 416]

Die Konzepte Edge-Computing und Fog-Computing sind sich inhaltlich recht ähnlich, und die Grenzen zwischen den beiden Ansätzen verschwimmen. Je

[5] Hier zeigt sich bereits, dass Edge-Computing und Fog-Computing inhaltliche Überschneidungen haben.

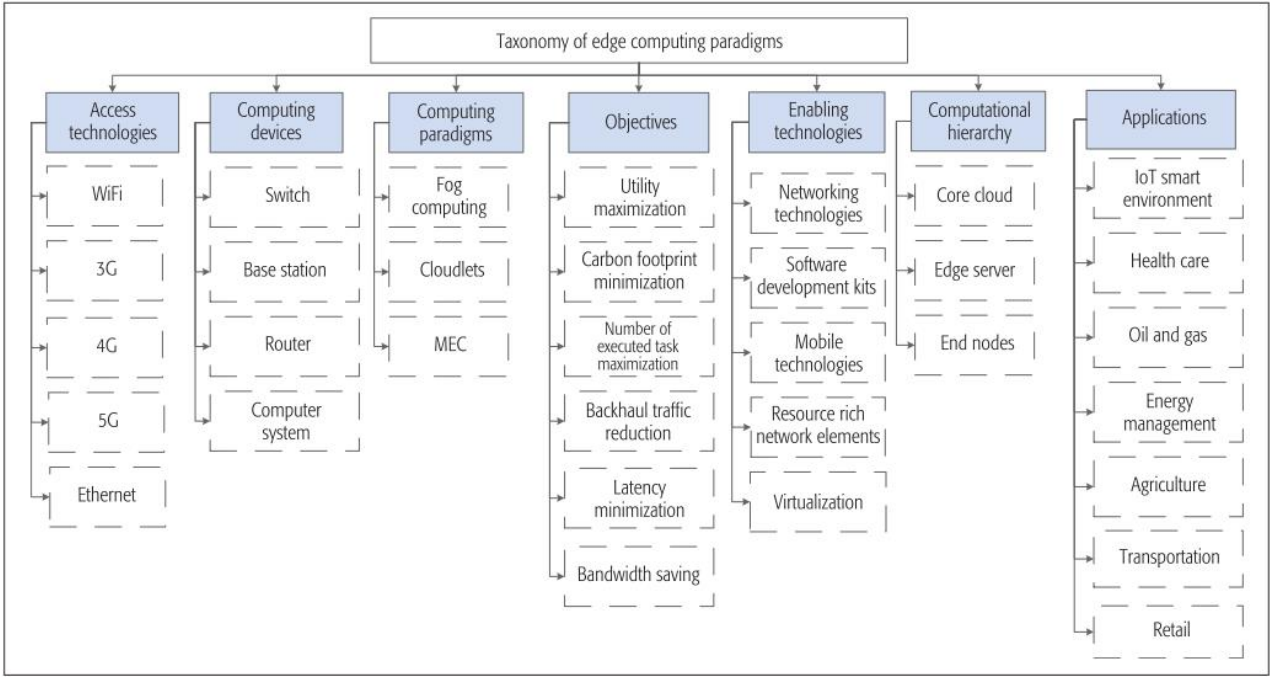


Abbildung 2.5: Taxonomie der Edge-Computing-Paradigmen [Ahme17]

nach Autor werden die Begriffe teilweise sogar synonym verwendet. Im folgenden Abschnitt sollen daher die beiden Haupt-Sichtweisen dargestellt werden, sowie die Begriffe Edge- und Fog-Computing voneinander abgegrenzt werden.

Anschließend werden Charakteristiken von Fog-Computing vorgestellt, die über die Charakteristiken des Edge-Computings hinausgehen. Hierzu zählt auch die typische Schichtenarchitektur für Fog-Computing. Abschließend werden heute noch offene Forschungslücken aufgezeigt, die einen Einsatz in der Produktionstechnik bisher verhindern.

Synonyme und ähnlich verwendete Begriffe

Edge-Computing Der Begriff Fog-Computing wird häufig synonym mit Edge-Computing verwendet, da die Konzepte sehr ähnlich sind. Der Verfasser teilt die Ansicht von Strnadl, Bermbach und weiteren, dass mit Fog-Computing allerdings nur ein Teil von Edge-Computing bezeichnet werden sollte. Notwendige Voraussetzung ist die Nutzung von cloud-nativen Ansätzen und Technologien [Strn21, S. 181]. Im, in Abbildung 2.4 dargestellten, Edge-Kontinuum ist Fog-Computing daher ab dem Bereich *Smart Device Edge* denkbar und ab *On-Prem Data Center* in Richtung Cloud-Computing sehr wahrscheinlich anzutreffen.

Während Edge-Computing häufig als das Ausführen von spezifischen Anwendungen an festgelegten Orten verstanden wird, kann Fog-Computing mehrere Anwendungen – auch unterschiedlicher Herkünfte – ausführen. Dazu kann es dynamisch umkonfiguriert werden, um Anforderungen neuer Anwendungen gerecht zu werden. [Iorg18; Al-D20, S. 10]

Manche Autoren differenzieren zwischen Edge- und Fog-Computing auch durch die Isolation von Edge-Computing während Fog-Computing Anwendungen auch auf Cloud-Ressourcen auslagern kann, sofern dies günstiger ist. [Chia17, S. 18; Berm18, S. 2; Al-D20, S. 10]

Edge-Cloud In den Bereich Fog-Computing fällt ebenfalls der – teilweise synonym verwendete – Begriff Edge-Cloud. Unter Edge-Cloud ist infrastrukturell ein *On-Premises Data Center* zu verstehen, was wiederum *cloud-native* genutzt wird. Entsprechend gehört Edge-Cloud eher dem Fog-Computing als dem Edge-Computing an.

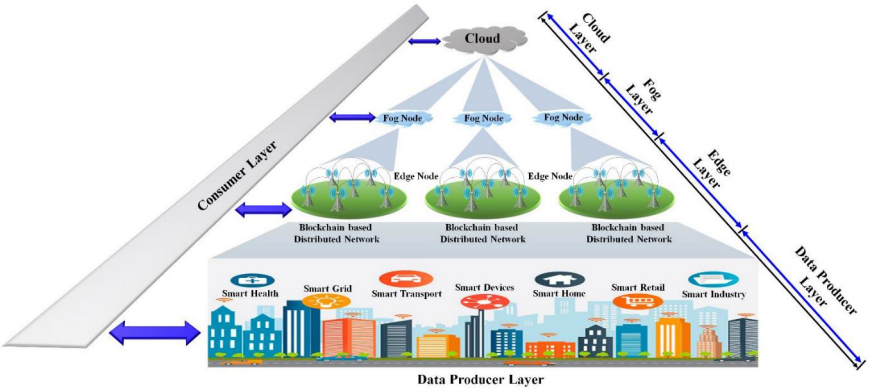


Abbildung 2.6: Beispielhafte Architektur mit 4 Schichten für Fog-Computing [Shar18]

Cloudlet bezeichnet ein regionales Computer-Cluster im Sinne des MEC. Fog-Computing schließt damit Cloudlet mit ein, ist aber nicht synonym zu verstehen. [Saty09, S. 14]

$$\text{Cloudlet} \subset \text{Fog-Computing}$$

Charakteristiken von Fog-Computing

Häufig kommt für Fog-Computing eine Architektur mit 3 bzw. 4 Ebenen zum Einsatz [Al-D20, S. 10], siehe Abbildung 2.6. Auf der untersten Ebene befinden sich in der Regel datenproduzierende (IoT-)Geräte. Auf der darüberliegenden Ebene sind Edge-Devices platziert, diese Ebene wird entsprechend mit Edge-Computing bezeichnet. Auf der obersten Ebene findet sich das Cloud-Computing im externen Rechenzentrum. Die Ebene zwischen Edge und Cloud wird mit Fog-Computing bezeichnet und enthält sogenannte Fog-Server oder Fog-Knoten (engl. *Fog-Nodes*). Das Ziel ist es Latenzen wie auf der Edge-Computing-Ebene zu erreichen und gleichzeitig die aus dem Cloud-Computing bekannte Rechenleistung bereitzustellen. Bei der häufig synonymen Verwendung von Fog- und Edge-Computing werden die Fog- und Edge-Ebenen zusammengefasst, sodass eine Architektur mit 3 Schichten entsteht.

Iorga, Feldman u. a. beschreiben sechs Charakteristiken, die Fog-Computing von weiteren Computing-Paradigmen abgrenzt, wobei diese nicht alle zur Abgrenzung von Edge-Computing geeignet sind [Iorg18, S. 3–4]:

1. Das Bewusstsein der Knoten über ihre netzwerktopologische Position, womit sie niedrige Latenzen bei Entscheidungen als Ziel setzen können.
2. Die geografische Verteilung der Knoten.
3. Mit Fog-Computing werden Daten aus unterschiedlichsten Quellen und unterschiedlicher Arten verarbeitet.
4. Verschiedene Anbieter müssen miteinander kooperieren, um nahtlos Services bereitstellen zu können.
5. Fog-Anwendungen beinhalten Echtzeit-Verarbeitung (Streaming) von Daten im Gegensatz zur Batch-Verarbeitung.
6. Fog-Computing ist adaptiv, d. h. es werden elastische Ressourcen von den Fog-Clustern angeboten.

Die Fog-Knoten sind die Kernkomponenten der Fog-Architektur, sie können sowohl physische als auch virtuelle Komponenten sein. Beispiele sind Gateways, Server, oder auch Netzwerk-Komponenten wie Switche oder Router. Virtuelle Fog-Knoten sind beispielsweise VMs oder virtualisierte Netzwerkfunktionen [Iorg18, S. 3]. Eine weitere Form von virtuellen Fog-Knoten wird von Marín-Tordera, Masip-Bruin u. a. vorgeschlagen: Die Aggregation mehrerer Edge-Devices könne man ebenfalls als einen einzelnen Fog-Knoten betrachten [Marí17, S. 123]. Gemeinsam haben all diese Typen von Fog-Knoten, dass sie Rechen-, Speicher- und Netzwerk-Fähigkeiten aufweisen ([Mora15] zitiert nach [Marí17, S. 118]).

Die Fog-Knoten weisen wiederum einige Charakteristiken auf [Iorg18, S. 4]:

1. Autonomie: Sie können unabhängig voneinander operieren und Entscheidungen treffen.
2. Heterogenität: Sie sind sehr unterschiedlich hinsichtlich ihrer Leistungsklasse als auch ihrer Einsatzumgebung.
3. Hierarchie: Im Edge-Kontinuum können die Fog-Knoten hierarchisch organisiert sein.
4. Handhabbarkeit: Sie sind größtenteils automatisiert in ihrem Management und der Orchestrierung.
5. Programmierbarkeit: Die Knoten sind programmierbar auf (fast) allen Ebenen.

Fog-Knoten können die gleichen Dienstleistungsmodelle wie auch Cloud-Computing aufweisen (IaaS, PaaS, SaaS – siehe Seite 12).

Tabelle 2.1: Technische Unterschiede von Edge-, Fog- und Cloud-Computing (Fog & Cloud basierend auf [Naha18])

	Edge	Fog	Cloud
Teilnehmende Knoten	1	sehr dynamisch	quasi-statisch
Management	verteilt	verteilt/zentral	zentral
Mobilität d. Knoten	gering	hoch	sehr gering
Geräte	divers		Server
Fehlerart	Sehr divers		vorhersehbar
Interne Konnektivität	gemischt, kabel & kabellos		Kabel
Stromversorgung	Batterie, Ökostrom, Stromnetz		Stromnetz
Stromverbrauch	niedrig		hoch
Netzwerk-Hops	wenige		viele
Echtzeit-Anwendungen	möglich		schwierig
Kühl-Aufwand	vernachlässigbar		hoch
Platzbedarf	gering		hoch

Offene Herausforderungen

Die in [Naha18, S. 47985] herausgestellten Unterschiede zwischen Cloud- und Fog-Computing sind – um Edge-Devices ergänzt – in Tabelle 2.1 dargestellt. Durch diese Unterschiede ist es nicht trivial, CNT auf die Ebene Fog-Computing zu übertragen. Sebrechts, Volckaert u. a. nennen 4 Problemfelder bei der Übertragung [Sebr22, S. 2]:

1. Fog ist nicht homogen (→ Geräte, Fehlerarten, Interne Konnektivität)
2. Fog-Knoten können beschränke Ressourcen haben (→ Geräte)
3. Latenz (→ Interne Konnektivität, Echtzeit-Anwendungen)
4. Verteilung von Tasks ist händisch nicht mehr auswählbar (→ Management)

Aral und Maio ergänzen zusätzlich folgende Schwierigkeiten [Aral20, S. 291–292]:

5. Die große Anzahl an Fog-Knoten (→ Teilnehmende Knoten)
6. Unterschiedliche Arten der Datenverarbeitung; Im Cloud-Bereich findet eher Batch-Verarbeitung statt, während bei Fog-Computing Datenströme verarbeitet werden. (→ Echtzeit-Anwendungen)
7. Häufig wechselnde Zusammensetzung der Knoten (→ Mobilität der Knoten)

Für die Anwendung in der Produktionstechnik sind insbesondere die Verwaltung der hohen Zahl an verschiedenen Geräten sowie die Thematik der Latenzen relevant. Die Verwaltung der vielen Edge-Devices stellt anders als beim Cloud-Computing ein Problem dar, denn das Installieren von Updates, Ändern von Konfiguration oder Skalieren von Anwendungen muss auf den vielen teilweise unterschiedlichen Knoten durchgeführt werden [Berm18, S. 5]. Dazu kommt die Schwierigkeit aus der Vielzahl von verfügbaren Knoten den richtigen auszuwählen für eine auszuführende Anwendung, was sich bei Cloud-Computing auf die Auswahl der richtigen Region beschränkt [Sebr22, S. 2].

Während im Cloud-Rechenzentrum die interne Konnektivität vernachlässigbar in Bezug auf die Latenz ist, kann diese zwischen zwei Fog-Knoten durchaus in verschiedenen Größenordnungen liegen. Dies kann zum einen auf unterschiedliche Übertragungsmedien und -techniken zurückgeführt werden, zum anderen hat die Route der Verbindung einen großen Einfluss. Dies liegt an der hierarchischen Struktur der Verbindung, so können zwei Fog-Knoten direkt miteinander oder auf der nächsten Hierarchieebene verbunden sein, möglicherweise aber auch erst im Internet-Backbone zusammenfinden. [Berm18, S. 6]

Ein weiteres Thema, das in der Forschung noch zu bearbeiten ist, stellen Security-Aspekte des Fog-Computings dar. Da diese Aspekte durch diese Arbeit allerdings nicht adressiert werden, sei lediglich der Verweis auf [Yu18] als Einstieg gegeben.

2.4.4 Abgrenzung weiterer Computing-Begriffe

Osmotic-Computing orientiert sich begrifflich an der Osmose, also den Ausgleichsbewegungen in Lösungen mit lokal unterschiedlichen Konzentrationen. Übertragen bedeutet Osmotic-Computing, dass Anwendungen je nach Auslastung dynamisch ihre Ausführungsumgebung ändern können. Dabei wird zwischen Cloud- und Edge-Computing unterschieden, wobei hier Edge-Computing als On-Premise-Datacenter verstanden wird. [Vill16, S. 76–78]

Osmotic-Computing kann als Teilmenge von Fog-Computing verstanden werden, da die Ziele ähnlich sind aber nur ein Teil des Edge-Kontinuums adressiert wird.

Ein verwandter Begriff ist *Service Migration*, der im Rahmen von MEC beschreibt, dass Services auf anderen Edge-Devices gestartet werden müssen, wenn die Konsumenten durch ihre Mobilität sich vom ursprünglichen Edge-Device entfernen und dadurch mit steigenden Latenzen oder sogar Service-Ausfällen konfrontiert werden. [Khos20, S. 141]

Mist-Computing Ausgehend von der Benennung von Fog-Computing wird teilweise der Begriff Mist-Computing verwendet. Mist-Computing bezeichnet dann eine leichtgewichtige Form von Fog-Computing, mit schwächerer Hardware, die direkt am Rande des Netzwerks platziert ist. In dieser Betrachtungsweise sind Fog-Nodes dann ausschließlich potente Server. [Iorg18, S. 6]

Da Fog-Computing in dieser Arbeit auf einen möglichst breiten Teil des Edge-Kontinuums angewendet werden soll, ist Mist-Computing als eine Teilmenge des Fog-Computings zu verstehen.

Distributed Computing oder »Verteilte Verarbeitung« bezeichnet das Ausführen von Anwendungen auf einem verteilten System. Ein verteiltes System definieren Tanenbaum und van Steen als »eine Ansammlung unabhängiger Computer, die den Benutzern wie ein einzelnes kohärentes System erscheinen« [Tane08, S. 19].

Ein wichtiges Ziel verteilter Systeme ist es, die Tatsache zu verbergen, dass ihre Prozesse und Ressourcen physisch über mehrere Computer verteilt sind. Ein solches System, das in der Lage ist, sich Benutzern und Anwendungen so darzustellen, als sei es nur ein einziges Computersystem, wird als transparent bezeichnet. [Tane08, S. 21]

Da die Anwendungen bei einem solchen verteilten System auf keinen gemeinsamen Speicher zugreifen können, müssen Daten über Netzwerkkommunikation ausgetauscht werden. Das kann zum Beispiel über RPC oder Nachrichten geschehen. In Abbildung 2.7 ist ein verteiltes System dargestellt, dass mit einer Middleware ausgestattet ist, die den Anwendungsprogrammen die verschiedenen Computer als ein Gesamtsystem darstellt. [Tane08, S. 140]

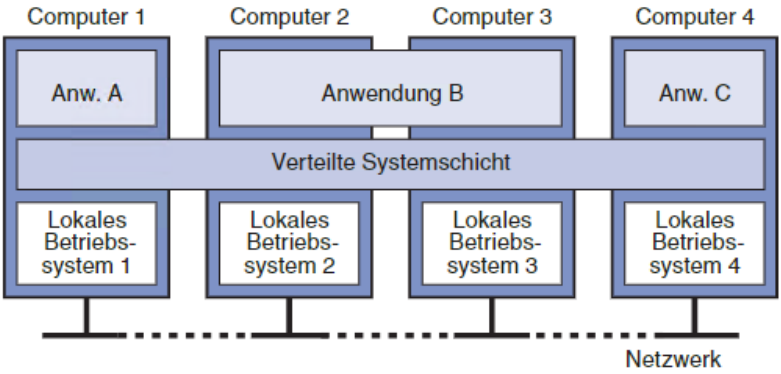


Abbildung 2.7: Ein verteiltes System mit Middleware [Tane08]

Distributed Computing ist ein Teilgebiet der Informatik, dem verschiedene spezielle Ausführungen zugeordnet werden können. Ein für diese Arbeit relevantes ist Grid-Computing:

Grid-Computing leitet sich aus der Analogie zum Stromnetz (engl. »*power grid*«) ab: Wie auch Endnutzer ihre Stromverbraucher einfach an die nächstgelegene Steckdose anschließen können, soll dies mit Grid-Anwendungen ähnlich werden. Der Endanwender kann einfach Zugriff auf Rechenressourcen bekommen, ohne sich darum zu kümmern, wo diese platziert sind, den Betrieb zu organisieren oder sich Gedanken um die technischen Details – wie Betriebssystem oder Hardwarearchitektur – machen zu müssen. [Jaco05, S. 3]

Als Teilgebiet von Distributed Computing ist die Virtualisierung charakteristisch, so kann Grid-Computing als distributed Computing auf virtualisierten Ressourcen bezeichnet werden [Jaco05, S. 6]. Damit werden auch CNT verwendet, was eine notwendige Bedingung für Fog-Computing darstellt.

Notwendige Kriterien für Grid-Computing sind u. a. die gemeinschaftliche Nutzung verteilter Ressourcen. Allerdings dürfen diese Ressourcen nicht von zentraler Stelle aus koordiniert werden und auch nicht durch lediglich eine Entität betrieben werden [Fost02, S. 2]. Damit grenzt es sich deutlich von Cloud-Computing ab und ist auch nicht die passende Bezeichnung für den Inhalt dieser Arbeit, da hier durchaus ein einzelner Betreiber (bspw. das Anwendungsunternehmen) der Infrastruktur in Frage kommt.

2.5 Service-Kompositionen

2.5.1 Tasks

In der Aufgabenplanung (engl. *Task Scheduling*) bezieht sich eine Aufgabe auf eine Arbeitseinheit, die innerhalb eines bestimmten Zeitrahmens abgeschlossen werden muss. Für diese Arbeit wird der Begriff Task allgemein als eine auf einem Computer ausführbare Aufgabe verstanden, und kann synonym mit dem Begriff Dienst (engl. *service*) verstanden werden. Insbesondere können Tasks auch Microservices oder Funktionen im Sinne von FaaS sein.

Jeder Task erhält Daten als Eingabe, verarbeitet diese und liefert Ausgabedaten zurück. Sonderformen können auch ohne die Eingabe- oder Ausgabedaten auskommen und ihre Funktionalität behalten.

Viele Services oder Funktionalitäten im Shopfloor laufen kontinuierlich, werden also nicht erst bei Bedarf gestartet wie bei FaaS. Am Beispiel der Achsregelung einer WZM-Vorschubachse soll dies verdeutlicht werden: Der Lageregler nimmt als Eingangsdaten in jedem Durchlauf die Sollposition als Führungsgröße auf, dazu wird eine Istposition der Achse erfasst. Aus den beiden Größen wird über einen Regelalgorithmus eine Stellgröße für den Geschwindigkeitsregler berechnet. Dieser Prozess muss innerhalb von 2 ms abgeschlossen sein. Danach wird der Regler aber nicht beendet, sondern beginnt im 2 ms-Takt die nächste Ausführung. [Brec21, 136ff]

Ein Beispiel für nicht kontinuierliche Tasks ist das Anfragen eines Report in einem MES. Typischerweise wird der Bericht mit einer Verzögerung von einigen Sekunden dargestellt, da der notwendige Service erst gestartet wird.

2.5.2 Kompositionen von Tasks

Im Bereich *Cloud-native* ist das Architekturmuster Microservices (siehe S. 14) weit verbreitet. Die Microservices decken jeweils eine Funktionalität eines Geschäftsprozesses ab und werden durch geeignete Kombination mit anderen Services zu einem Softwareprodukt kombiniert.

Ein ähnliches Vorgehen gibt es im Bereich *Network Function Virtualization*: Hier werden aus einzelnen virtuellen Maschinen, die jeweils eine Netzwerkfunktionalität liefern durch *Service Chaining* ganze Netzwerkgeräte wie Router u. ä. ersetzt [Khos20, S. 135].

Auch beim *Osmotic Computing* findet sich Software, die aus sog. *MicroElements* (MELs) – Microservices oder Microdata – zusammengesetzt ist:

> Each IoT application can be decomposed into cooperating subprograms and services to improve deployability and scalability. In osmotic computing, IoT applications are decomposed into a number of interacting MELs, which are atomic entities providing simple functionalities [...]. [Vill19, S. 15]

Diese Ansätze sollen im Rahmen dieser Arbeit als Service-Kompositionen bezeichnet werden, da ein gewünschtes Verhalten durch Komposition verschiedener Tasks oder Services erreicht wird.

Die einzelnen Schritte der Datenverarbeitungsketten aus Abschnitt 2.3 können als Subtask im Sinne dieser Arbeit aufgefasst werden. Damit lassen sich die Ketten als Service-Kompositionen oder – relativ linearer – Workflow interpretieren, den es zu verteilen gilt. Von einem *Workflow* spricht man, sofern die Services bzw. Tasks nur einmalig durchlaufen werden müssen.

2.5.3 Repräsentation durch gerichtete azyklische Graphen

Solche Service-Kompositionen mit den Abhängigkeiten zwischen den einzelnen Tasks und Services lassen sich gut als Graphen darstellen [Gast22, S. 26; Wang21c, S. 5901]. Die Knoten sind dabei die einzelnen Tasks und die Kanten repräsentieren die Datenflüsse und damit Abhängigkeiten zwischen den Tasks [Sing17, S. 3].

Da die Datenflüsse eine Richtung haben, kommen gerichtete Graphen zum Einsatz. Da die Kompositionen zusätzlich im Allgemeinen einen definierten Anfangs- und Endpunkt – oder auch mehrere – haben, sind die entstehenden Graphen azyklisch. Die gerichteten azyklischen Graphen werden im Englischen als *directed acyclic graph* bezeichnet, die Abkürzung DAG findet sich auch in deutschen Texten.

Die Komposition wird also durch den Graphen $W = (T, D)$ beschrieben, wobei T die Menge der Tasks und D die Menge der Datenflüsse zwischen den Tasks ist. Ein Datenfluss wird als Tripel aus dem Vorgänger-Task t_i, dem Task t_j und den Datenflussparametern d_{ij} dargestellt.

$$T = \{t_1, t_2, \dots t_{m_T}\} \tag{2.1}$$

$$D = \{(t_i, t_j, d_{ij})\} \quad 0 \leq i, j \leq m_T \tag{2.2}$$

Bei einem Workflow bedeutet die Kante (t_i, t_j, d_{ij}) auch, dass der Task t_i abgeschlossen sein muss, bis t_j ausgeführt werden kann.

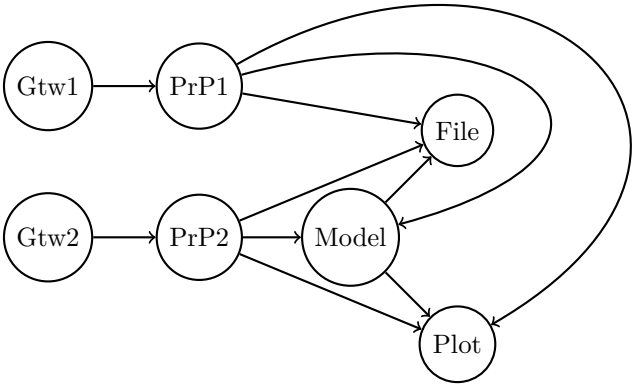


Abbildung 2.8: Abgeleiterer Graph aus der Zustandsüberwachungs-Architektur (Gtw: *Gateway Receiver*, PrP: *Preprocessing*)

Die in Abschnitt 2.3 vorgestellte Systemarchitektur einer Zustandsüberwachung von WZM-Komponenten (s. Abbildung 2.1) enthält voneinander abhängige Softwarekomponenten, die ebenfalls als verteiltes System und damit als Service-Komposition entwickelt werden können. Der gerichtete azyklische Graph der enthaltenen Services ist in Abbildung 2.8 dargestellt.

3 Stand der Wissenschaft und Technik

Nachdem die grundlegenden Begrifflichkeiten eingeführt wurden, wird in diesem Kapitel der aktuelle wissenschaftliche Stand der empirischen Forschung zu Fog-Computing in der Produktion vorgestellt.

Dafür wurden folgende Fragen formuliert:

F1: Wie können die Hardwareeigenschaften von Edge-Devices bei der Softwareverteilung berücksichtigt werden?

F2: Wie wird die dynamische Auslastung von Rechenressourcen durch langlaufende oder kontinuierliche Tasks bei der Verteilungsentscheidung berücksichtigt?

F3: Wie können an die Tasks gestellte Echtzeitanforderungen in die Verteilungsentscheidung einfließen?

F4: Wie können Netzwerkeinflüsse im Verteilungsalgorithmus Berücksichtigung finden?

F5: Wie können spezifische Hardwarebedarfe einzelner Tasks in der Verteilungsentscheidung berücksichtigt werden? (Bspw. wenn der Zugriff auf einen Feldbus notwendig ist.)

Um die Literatur diesbezüglich zu analysieren sowie einzelne Teil-Lösungsansätze in diesem Bereich zu identifizieren, wurde die Recherche in Form eines semi-systematischen Literaturreviews durchgeführt: Zunächst wurde in mehreren Datenbanken nach empirischen Studien gesucht, die entweder die Begriffe Edge- oder Fog-Computing, sowie Produktion, *production* oder *manufacturing* im Titel, Abstract, Keywords oder Volltext enthielt. Die gefundenen Treffer wurden in einem Screening-Prozess selektiert, wobei nur Literatur behalten wurde, bei denen die Begriffe auch im inhaltlichen Zusammenhang standen.

Aus den Fragen wurden die nachfolgenden Kriterien abgeleitet, anhand derer die verbleibenden Texte analysiert wurden:

Hardware

1. Werden Entscheidungen für Verteilung innerhalb der Edge-Ebene getroffen (selbst wenn die Entscheidung von einer übergeordneten Ebene getroffen wird)? (F1)

2. Bezieht sich die Veröffentlichung auf Hardware außerhalb eines Rechenzentrums? (F1)
3. Kommen heterogene Edge-Devices mit unterschiedlichen Ressourcen zum Einsatz? (F1)

Software

4. Wird die Softwareverteilung als dynamisch betrachtet (oder lediglich zur Planungszeit einmalig berechnet)? (F2)
5. Finden unterschiedliche zeitliche Charakteristiken (Lastprofile) der auszuführenden Software Beachtung? (F2)

Anforderungen

6. Werden Echtzeitanforderungen betrachtet? (F3)
7. Findet die Kommunikation zwischen Tasks kontinuierlich (*Streaming*) statt? (F2)
8. Werden Einflüsse des Netzwerks berücksichtigt? (F4)
9. Können einzelne Tasks an bestimmte Hardware gebunden werden? (F5)

Im Verlauf des Reviews wurde noch ein Kriterium im Bereich Software ergänzt, da aufgefallen ist, dass dieses Thema in der Produktionsforschung nicht thematisiert wird:

10. Finden Abhängigkeiten zwischen einzelnen zu verteilenden Tasks Beachtung?

Aufgrund der geringen Zahl an verbleibenden Treffern sowie der Zahl der gefundenen Forschungsdefizite wurde die Literaturanalyse in einer zweiten Stufe auf den allgemeinen Bereich der Informatik erweitert. Die Ergebnisse der beiden Stufen werden in den folgenden Unterkapiteln vorgestellt.

3.1 Computing in der Produktionstechnik

Nur wenige Publikationen aus der Produktionstechnik und -forschung, die Edge- oder Fog-Computing verwenden, beschäftigen sich mit den inneren Mechanismen von Fog-Computing. Der größte Teil der Literatur beschreibt Anwendungsfälle von Edge-Computing im Bereich der Produktion, und verwendet häufig nur einzelne Edge-Devices wie IPCs. Im Folgenden wird zunächst auf die wenigen Publikationen zu Fog-Computing in der Produktion eingegangen, anschließend wird ein Überblick über die Anwendungsfall-Literatur gegeben.

3.1.1 Fog-Computing mit Fokus auf Produktionstechnik

Wang, Li und Hu [Wang21b] präsentieren einen Ansatz, um Fog-Server im *Shopfloor* zu platzieren. Dabei wählen sie die besten Positionen an einem Produktionssystem aus, um eine möglichst gleichmäßige Auslastung der Server zu erreichen und die Antwortzeit der Services zu minimieren. Der präsentierte Ansatz vernachlässigt explizit Einflüsse unterschiedlicher Latenzen bei der Kommunikation [Wang21b, S. 3536].

Jiang, Wan und Abbas [Jian21] präsentieren in ihrer Veröffentlichung ein Konzept, um die Anzahl von Fog-Servern im *Shopfloor* zu optimieren. Als Optimierungsziele werden die Netzwerklatenz sowie nicht spezifizierte Kostenfunktionen für die eingesetzten Fog-Server verwendet. Der Fokus liegt auf dem Einsatz von *Software-defined Networking* (SDN), wobei dem SDN-Controller zusätzlich die Fähigkeit postuliert wird, Tasks auf Fog-Knoten im *Shopfloor* zu platzieren [Jian21, S. 2232]. Sowohl die technologischen Lösungen für SDN und das Service-Placement als auch die Kostenfunktionen sind nur erwähnt, bleiben aber leider unbeschrieben.

Gezer und Wagner [Geze21] stellen ein Framework für Echtzeit-Anwendungen bei Fog-Computing vor. Benutzer oder Endgeräte können die Ausführung eines Tasks bei einem Fog-Server beantragen. Die Gruppe der Fog-Server entscheidet daraufhin welcher Server den Task ausführt. Dies geschieht unter Einhaltung einer vorgegebenen maximalen Ausführungszeit, womit die Echtzeitfähigkeit erreicht wird. Die Autoren vereinfachen die Aufgabe durch Ignorieren sämtlicher Einflüsse der Netzwerkverbindungen. Weiter vernachlässigen sie Speicherbedarfe der Tasks [Geze21, S. 2310].

Zietsch, Vogt u. a. [Ziet20] schlagen ein Vorgehensmodell vor, mit dem die Wahl einer geeigneten Infrastruktur für Industrie 4.0-Anwendungen systematisch getroffen werden kann. Zur Auswahl stehen Edge-, Fog- oder Cloud-Computing und es werden in den verschiedenen Phasen unterschiedliche Anforderungen erhoben. In die Entscheidung fließen Anforderungen der Datenquellen, der Verarbeitungslogik sowie qualitative und quantitative Anforderungen der Gesamtanwendung mit ein. Die Entscheidung kann mit verschiedenen Optimierungszielen getroffen werden, z. B. minimale Betriebsausgaben, Investitionsausgaben oder Stromverbrauch. Die Autoren demonstrieren das Vorgehen an der technischen Gebäudeausstattung – mit Fokus auf die Klimatisierung – einer Produktionshalle.

Wang, Zhao u. a. [Wang21c] nutzen Maschinelles Lernen, um die Latenzen in einem kombinierten Edge-Cloud-Verbund abzuschätzen. Als Eingabewerte

werden u. a. die Ressourcenbedarfe der Services, die verfügbaren Ressourcen der Knoten und einige Kennzahlen der zu transportierenden Nachrichten verwendet. Als Transportprotokoll wird *Message Queue Telemetry Transport* (MQTT) genutzt. Diese Latenzen nutzen sie, um die Platzierung von komplexen Micro-Service-Kompositionen auf den verfügbaren Knoten vorzunehmen. Dabei wird als Optimierungsziel eine minimale Latenz der gesamten Service-Komposition gesetzt und die Kapazitäten der Knoten lediglich als Nebenbedingungen genutzt. Umgesetzt wird der Ansatz an einer Fertigungsanlage für die Halbleiterproduktion.

Zhang und Wei [Zhan21] widmen sich dem Thema Funkübertragung in einem Smart Manufacturing Szenario. Zu der übergeordneten Fragestellung ob Services in einem Edge- oder Cloud-Cluster platziert werden sollen, werden Fragen der Latenz bei WiFi-Verbindungen betrachtet.

Bayer, Moedel und Reich [Baye19] präsentieren eine Architektur auf Basis von Kubernetes für Industrie 4.0 und Condition-Monitoring-Anwendungen. Leider entscheidet der vorgestellte Ansatz lediglich zwischen einer Verteilung der Software auf Cloud- oder Fog-Ebene. Hervorzuheben ist, dass die Möglichkeit für Hardwarebindung vorgesehen ist, d. h. dass bestimmte Anwendungen zwingend eine bestimmte Hardware auf dem Knoten voraussetzen. Dies kann zum Beispiel die Verbindung zu einem Aktuator sein.

Yin, Luo und Luo [Yin18] beschäftigen sich ebenfalls mit der Frage, ob Tasks im Fog-Layer oder in der Cloud platziert werden. Sie berechnen die Ausführungszeit sowohl im Fog-Layer als auch in einer unendlich leistungsfähigen Cloud und entscheiden daran und anhand der Auslastung des Fog-Servers, ob der Task in die Cloud ausgelagert werden muss. Die Berechnung der Ressourcenauslastung des Fog-Servers findet dynamisch in Perioden von 100 ms statt. Der präsentierte Ansatz zielt explizit auf containerisierte Tasks ab.

Zhou, Xiang u. a. [Zhou20] präsentieren einen *Task-Offloading*-Algorithmus für kombiniertes Edge-Fog-Cloud-Computing. Dabei nutzen sie die Metaheuristik *Particle Swarm Optimization* (PSO) um folgende Einflussgrößen zu optimieren:

- Latenz der Ausführung aller Tasks, inkl. des Transports durch das Netzwerk
- Energieverbrauch der lokalen Edge-Devices
- Ein willkürlich gewählter Sicherheitswert für jedes Gerät

Leider wird der Energieverbrauch nur sehr rudimentär integriert, sodass lediglich die anteiligen Verbräuche der Edge-Devices auf der untersten Ebene relevant sind. Der Energieverbrauch findet wohl Eingang in die Optimierungs-

bedingungen, dies wird allerdings nicht näher beschrieben. Auch die Latenzen durch den Netzwerktransport sind nur sehr vereinfacht – unter Ignorieren sämtlicher Strukturen und beteiligten Komponenten des Netzwerks – als 1:1-Übertragungsblackbox modelliert.

3.1.2 Anwendungsfälle in der Produktionstechnik

Kubiak, Dec und Stadnicka [Kubi22] stellen die Ergebnisse eines systematischen Literaturreviews vor, bei dem sie u. a. der Frage nachgingen, welche Anwendungen in der Forschung für Edge-Computing in Bereich der Produktion gesehen werden. Ihre Ergebnisse gruppieren sie u. a. in folgende Kategorien:

- Intelligente Produktionssysteme als Cyber-Physisches Systems (CPSs): Dies deckt sich weitestgehend mit dem Abschnitt »Autonome oder Intelligente Produktionssysteme«.
- Entscheidungsfindung autonomer Systeme und Roboter in (unbekannten) Situationen.
- Datensicherheit bei -erfassung und -transport.
- Echtzeit-Datenverarbeitung: Ziel ist die Überwachung des Produktionsprozesses, der -maschinen und die Qualität der gefertigten Produkte.
- Beschleunigen und Automatisieren von Qualitätskontrollen.
- Simulation des Fertigungsprozesses.
- Erhöhen der Resilienz gegen Kommunikationsfehler oder -ausfälle.
- Wissensmanagement, mit dem Ziel Wissen, überall wo es benötigt wird, bereitzustellen und neues Wissen aus Bekanntem abzuleiten.
- Energieverbräuche, sowohl der Fertigung selber als auch die der Datenverarbeitung.

Kubiak, Dec und Stadnicka fassen zusammen, dass Edge-Computing primär wegen der Möglichkeit zur Verarbeitung großer Datenmengen genutzt wird.

Von den genannten Gruppen kann in folgenden Themenbereichen aus Sicht des Verfassers besonders von kollaborierende Rechnereinheiten im *Shopfloor* profitiert werden, weswegen diese nachfolgend noch etwas detaillierter beschrieben werden:

- Intelligente Produktionssysteme & Entscheidungsfindung autonomer Systeme
- Echtzeit-Datenverarbeitung

Autonome und Intelligente Produktionssysteme

Im Rahmen von Industrie 4.0 findet ein Wandel der Automatisierungspyramide statt. Während früher streng hierarchisch organisierte Automatisierungssysteme die Regel waren, beginnt sich die strenge Auftrennung der Ebenen aufzulösen und durchlässiger zu werden. Wo früher eine zentrale Steuerung Zugriff auf sämtliche Aktoren und Sensoren der Feldebene hatte, etablieren sich heute zunehmend verteilte Steuerungsansätze, bei denen mehrere Steuerungen mit kleineren Zuständigkeitsbereichen miteinander kooperieren. Dies ergibt sich zum einen durch die steigende Verfügbarkeit von kostengünstigen Rechenressourcen, die leicht in Produkte integriert werden können. Zum anderen verlangt die steigende Variantenvielfalt bis hin zu Produktion der Losgröße Eins mit kundenindividuellen Produkten nach flexiblen Produktionssystemen [Guo21]. In Analogie zur SoA in der Softwareentwicklung wird dem mit modularen und flexibel rekonfigurierbaren Produktionssystemen begegnet. [Gore17, S. 557]

Der Trend zu kooperierenden Steuerungen führt zur Konvergenz von Feld- und Steuerungsebene und damit zu »smarten Feldgeräten«. Die Feldgeräte werden zu Cyber-Physischen Systemen (CPS), d. h. mit leistungsstarken Mikrocontrollern und Speichern sowie Netzwerkfähigkeiten ausgestattet [Gore17, S. 563]. Anders als Gorecky, Hennecke u. a. annehmen ist eine Internet-Anbindung nicht zwingend notwendig, wohl aber die Netzwerkfähigkeit der eingebetteten Systeme in solchen CPS. Damit stehen diese smarten Feldgeräte als potenzielle Edge-Devices zur Verfügung und könnten Aufgaben im Rahmen von Fog-Computing im *Shopfloor* übernehmen.

Guo und Martínez-García [Guo21] stellen einen Ansatz vor, der eine Produktionslinie mit Schwarmintelligenz ausstattet, um auf wechselnde Produktionsprozessanforderungen zu reagieren. So werden u. a. neue Komponenten hinzugefügt und durch *group learning* in einer Cloud optimale Anweisungen für alle beteiligten Module ermittelt.

Der vorgestellte Ansatz basiert auf Cloud- und Edge-Computing. Auf Edge-Ebene findet die Datenerfassung aus den verschiedenen beteiligten Produktionsgeräten und -modulen statt. Hierzu werden die jeweils spezifischen Kommunikationsschnittstellen der Geräte bedient und auf *Open Platform Communications Unified Architecture* (OPC UA) als einheitliches Protokoll und Informationsmodell harmonisiert. Dieses Zusammenführen in ein einheitliches Informationsmodell wird von den Autoren als »*cyber-physical fusion mechanism*« bezeichnet, unterscheidet sich aber nicht von Datenfusion.

Eine ähnliche Architektur für ein »*Smart Manufacturing Service System*« präsentieren Qi und Tao [Qi19], sie kombinieren in ihrer Konzeptveröffentlichung die drei Ebenen Edge, Fog und Cloud, um verschiedene Mehrwerte zu postulieren. Dabei weisen sie den einzelnen Ebenen zwar verschiedene Services zu, berichten aber nicht über die internen Funktionsmechanismen der Ebenen.

Im Bereich Entscheidungsfindung autonomer Systeme stellen bspw. Wang, Liu u. a. [Wang21a] einen Ansatz vor, der die Fertigungsplanung und insbesondere die Maschinenbelegung autonom übernimmt. Dabei machen sie sich die Spieltheorie zu Nutze und entscheiden in einer Edge-Cloud für mehrere Fertigungsinseln über die Produktionsplanung. Zielgrößen sind dabei eine möglichst gleichmäßige Auslastung der Maschinen, minimale Fertigungszeiten und gleichzeitig ein minimaler Energieverbrauch.

Echtzeit-Datenverarbeitung: Überwachung von Prozess, Maschinen und Qualität

Die vergleichsweise sehr kurze Latenz von Edge-Computing erlaubt es Anwendungen zu implementieren, die schnelle Reaktionen erfordern. Dadurch ist es möglich aktiv in die Fertigungsprozesse auch auf Maschinenebene einzugreifen, was sehr kurze Latenzen – im Millisekunden-Bereich – und deterministische Zeitdauern unbedingt erfordert – und damit die Definition von Echtzeit erfüllt.

Merino, Bediaga u. a. [Meri19] präsentieren einen erfolgreichen Einsatz von Edge-Computing, um Zerspanungsprozesse adaptiv zu regeln. Am Anwendungsfall der Nachbearbeitung von Güterzug-Radsätzen mittels Drehmaschinen demonstrieren sie die Echtzeit-Datenverarbeitung auf einem Edge-Device. Über Beschleunigungssignale wird Rattern im Bearbeitungsprozess erkannt und vom Edge-Device aktiv auf die CNC-Steuerung eingewirkt, um das Rattern zu unterdrücken. Als Stellsignale kommen hier sowohl die eigentliche Spindeldrehzahl als auch Parameter der kontinuierlichen Spindeldrehzahl-Variation zum Einsatz.

Liang, Li u. a. [Lian21] klassifizieren mittels eines *Convolutional Neural Network* (CNN) anhand von Spindelströmen, ob der Fertigungsprozess normal oder abnormal abläuft. Bei Erkennung eines abnormalen Zustandes wird per Cloud-Computing eine Neuplanung der Fertigung durchgeführt, sodass neue Fertigungspläne an die beteiligten Maschinen übertragen werden. Ebenso wird in der Cloud das (Re-)Training der CNNs durchgeführt.

Ein anderes Beispiel zur Echtzeit-Überwachung und Regelung von Maschinen stellen Liu, Ma u. a. [Liu22] vor. Sie zeigen den Einsatz eines thermischen Modells einer Fräsmaschine, um die Genauigkeit derselben zu erhöhen, in-

dem thermisch bedingte Verlagerungen kompensiert werden. Die thermische Trägheit der Maschinen erlaubt hier den Einsatz eines Überwachungssystems in der Cloud, kombiniert mit lokaler Verarbeitung und Erzeugung der Kompensationssignale.

Ein weiteres Anwendungsfeld für Echtzeit-Datenverarbeitung ist *Predictive Maintenance* (PM) von Maschinen und Anlagen. Teoh, Gill und Parlikad [Teoh21] nutzen Fog-Computing für PM und vergleichen die Energieeffizienz, Ausführungszeit sowie Kosten verschiedener Algorithmen. Bayer, Moedel und Reich [Baye19] präsentieren eine Architektur für generische Condition-Monitoring-Anwendungen in der Industrie.

Li, Ota und Dong [Li18] überwachen die Qualität montierter Baugruppen mittels Bildverarbeitung. Wegen der hohen Datenmengen, die mit hoher Geschwindigkeit verarbeitet werden müssen, kommt auch hier Fog-Computing zum Einsatz.

3.1.3 Zwischenfazit zu bisherigen Forschungsarbeiten

Bisherige Arbeiten im Bereich der Produktionsforschung liefern zwar erste Grundlagen und Anwendungsfälle, beantworten allerdings die Fragen zur Softwareverteilung im Produktionsbetrieb und -umfeld nicht oder nicht ausreichend, wie in der Zusammenstellung der Studien in Tabelle 3.1 ersichtlich wird. Dabei fällt auf, dass die fünfte Frage überhaupt nicht behandelt wird.

Im Bereich Hardware (F1) sind die vorgestellten Publikationen recht treffend, auch wenn heterogene Edge-Devices offensichtlich bisher zu wenig Beachtung finden. Die Annahme, dass die Edge-Devices immer gleiche Hardware enthalten oder zumindest von gleicher Leistungsklasse sind, lässt sich im *Shopfloor* nicht bestätigen, zeugt aber davon, dass die vorgestellten Ansätze von der Cloud-Seite her ins Edge-Kontinuum vordringen oder aber mindestens vereinfachende Annahmen treffen.

Im Bereich Software sind die gefundenen Arbeiten aus den nachfolgenden Gründen nicht zufriedenstellend: Ausschließlich die Autoren von [Geze21] und [Yin18] sehen vor, dass die Softwareverteilung dynamisch betrachtet werden sollte (F2), alle weiteren Veröffentlichungen gehen von einer statischen Verteilung nach einer initialen Analyse der Gegebenheiten vor. Da ein Teil der Arbeiten die Provisionierung – also die Bereitstellung und Platzierung – von Hardware im *Shopfloor* betrachtet, ist es nachvollziehbar, dass diese Positionen nicht dynamisch angepasst werden können. Für die Zukunft, insbesondere durch Trends wie wandelbare Automatisierungssysteme aus Einzelmodulen, ist

Tabelle 3.1: Publikationen zu Fog-Computing-Methoden im Bereich Fertigung
×: zutreffend; (×): eingeschränkt zutreffend; ?: unklar

	Hardware			Software			Anforderungen			
Veröffentlichung	Innerhalb Edge-Ebene	Nicht nur Rechenzentren	Heterogene Edge-Devices	dynamisch	Lastprofile	Task-Abhängigkeiten	Echtzeit(/Low-Latency)	Streaming	Netzwerkeinfluss	Hardwarebindung
Zietsch, Vogt u. a. [Ziet20]		×								
Zhang und Wei [Zhan21]									×	
Bayer, Moedel und Reich [Baye19]		(×)		?			?	(×)	×	×
Jiang, Wan und Abbas [Jian21]	×	×							(×)	
Wang, Zhao u. a. [Wang21c]	×	×	×				(×)		×	
Zhou, Xiang u. a. [Zhou20]	×	×	(×)					?	×	
Wang, Li und Hu [Wang21b]	×	×	(×)		(×)		(×)	?		
Qi und Tao [Qi19]	?	×	×	?					?	
Gezer und Wagner [Geze21]	×			×	×		×	(×)		
Yin, Luo und Luo [Yin18]		×		×					×	

es allerdings schwer vorstellbar, dass eine im Voraus durchgeführte Verteilung auf Dauer die optimale Lösung darstellen kann.

Auch auf Lastprofile oder unterschiedliche Charakteristiken der zu verteilenden und auszuführenden Software (F2) wird ebenfalls nur bei [Geze21] eingegangen. [Wang21b] sieht immerhin vor, dass Aufgaben zyklisch ausgeführt werden könnten, was aber in den Bereichen Maschinensteuerung und Prozessüberwachung den Hauptteil der Anwendungen betrifft. Abhängigkeiten zwischen einzelnen Tasks im Sinne von Service-Kompositionen finden in keiner der analysierten Studien Beachtung.

Auf die besonderen Anforderungen im Bereich der Produktionstechnik wird in den vorgestellten Publikationen nur unzureichend eingegangen. Die besondere Herausforderung, Anwendungen im Echtzeit-Kontext zu betreiben (F3), wird nur von einer Publikation adressiert, zwei weitere enthalten zumindest die Thematik Latenzen und lassen auf einen Einsatz mit notwendigen kurzen Latenzen schließen, wenngleich ein Determinismus fehlt.

Streaming oder Datenströme (F2) werden in keiner der vorgestellten Arbeiten berücksichtigt. Es finden stattdessen immer REST-Aufrufe, RPC oder nachrichtenbasierte Kommunikation wie MQTT Verwendung.

Da der *Shopfloor* keinem Rechenzentrum entspricht, wäre zwar zu erwarten, dass in den Arbeiten die Einflüsse der verschiedenen Netzwerk-Technologien zur Sprache kommen (F4). Allerdings bildet lediglich [Wang21c] Latenzen im Netzwerk durch ein Modell ab, das mit Messdaten durch ML trainiert wurde. Diese Latenzen werden in die Verteilungsentscheidung miteinbezogen.

Der Thematik von kabellosen Kommunikationssystemen wie WiFi, LTE+, 5G u. ä. widmen sich fast keine der Arbeiten aus dem Bereich Produktion. Lediglich [Zhan21] stellt einen Ansatz vor, um zu entscheiden, ob ein Service in der Cloud oder auf Edge-Ebene ausgeführt werden soll. Hierbei wird im Besonderen auf WiFi und die damit zusammenhängenden Risiken und Latenzen eingegangen.

Zusammenfassend lässt sich sagen, dass folgende Aspekte bisher nur unzureichend und nicht in Kombination betrachtet wurden:

- Unterschiedliche Leistungsfähigkeit von Edge-Devices (F1)
- Zeitlich schwankende Ressourcenbedarfe (F2)
- Dynamische Neu-Berechnung der Softwareverteilung (F2)
- Echtzeit-Anforderungen bzw. Priorisierung von einzelnen Softwareteilen (F3)

- Netzwerkeinflüsse, wie bspw. Paketverluste und Überlastungen, aber auch physische und netzwerktechnische Nähe von Knoten (F4)

Darüber hinaus finden folgende Aspekte keine Beachtung bei den bisherigen Arbeiten:

- Abhängige Services bzw. Service-Kompositionen (Kriterium 10)
- Kontinuierliche Datenflüsse zwischen einzelnen Tasks (F2)
- Hardware-Bindung von Softwareteilen (bspw. für Feldbuszugriff) (F5)

3.2 Softwareverteilung

Da, wie bereits erwähnt, im Bereich Produktion keine zufriedenstellenden Arbeiten gefunden werden konnten, wird ein Blick auf den allgemeineren Bereich der Informatik geworfen. In der Literatur wird das Thema der Softwareverteilung uneinheitlich bezeichnet, so kann man entsprechende Forschungsarbeiten unter den Stichwörtern *Task Offloading*, *Task Scheduling*, *Service Placement*, *Resource Allocation* und teilweise sogar *Loadbalancing* finden.

Task Offloading bezeichnet dabei die Fähigkeit eines Knotens bei Überlastung Aufgaben an eine übergeordnete Ebene oder Nachbarn innerhalb derselben Ebene abzugeben. *Task Scheduling* ist eigentlich eher die Planung der Ausführungsreihenfolge von Aufgaben und weniger die Entscheidung, wo ein Task ausgeführt werden soll, dies ist eher *Service Placement* oder *Resource Allocation*, welche die Aspekte dieser Arbeit am besten abdecken. Da in allen Themenbereichen relevante Aspekte zu finden sind, werden im Folgenden ausgewählte Forschungsarbeiten zu den Bereichen vorgestellt.

Zum Abschluss des Kapitels wird ein Überblick über Charakteristiken der zu verteilenden Tasks, sowie der zur Verfügung stehenden Rechenknoten gegeben.

3.2.1 Lastverteilung

Loadbalancing oder Lastverteilung bezeichnet einen Prozess bzw. eine Technologie, um Netzwerkverkehr auf mehrere Geräte oder Server aufzuteilen. Der Endnutzer erkennt die Lastverteilung nicht, z. B. beim Aufruf einer Webseite. [Bour01]

Da im Produktionsbereich hohe Zugriffszahlen auf einzelne Tasks nicht sehr wahrscheinlich sind, fällt klassische Lastverteilung weniger in den Fokus dieser Arbeit. Allerdings sind die Defizite heutiger *Loadbalancing*-Algorithmen u. a.

heterogene Rechenknoten [Kuma19, S. 8], wie sie im *Shopfloor* der Produktionsunternehmen vielfach aufzufinden sind, oder auch, dass die geografische und physische Verteilung der Knoten nicht in die Berechnung miteinbezogen wird [Shah20, S. 130506]. Beides Beanstandungen, die in Abschnitt 3.1.3 bereits an die Produktionstechnik-Literatur gerichtet wurden und offensichtlich ein weiterreichendes Problem für die klassischen Ansätze darstellen.

3.2.2 Task Offloading

Beim *Task Offloading* (deutsch etwa Aufgaben-Auslagerung) werden Tasks bei Überlastung der lokalen Ressourcen weitergereicht. Dies kann entweder an ähnliche Knoten mit freien Ressourcen passieren, oder ein Offloading in die Cloud bedeuten oder auch umgekehrt von der Cloud zurück zur Edge:

> Computation offloading is the offloading of calculations by moving the location of execution from the cloud to the edge of the network in order to accelerate them and reduce delays [...] [Kubi22, S. 6].

Yang, Liu u. a. [Yang19] teilen den Offloading-Prozess in drei Phasen auf: (1) Das eigentliche Offloading, (2) die Berechnung auf dem entfernten Knoten und (3) die Kommunikation des Ergebnisses über den Ursprungsknoten zurück. Das Verschieben der Tasks findet hier zu einem leistungsfähigen MEC-Server am Sendemast einer Funkzelle statt. Das Hauptmerkmal des vorgestellten Ansatzes ist die Einbeziehung der Mobilität von Fahrzeugen, die zwischen Funkzellen wechseln in die Offloading-Entscheidung, da hier nur eine kurze Zeit zur Verfügung steht, um die drei genannten Phasen abzuschließen. In einer erweiterten Betrachtung beziehen die Autoren weitere MEC-Server mit ein, die Offloading-Entscheidung zwischen diesen kollaborierenden Servern wird anhand der Fahrtrichtung der Fahrzeuge getroffen.

Beraldi, Mtibaa und Alnuweiri [Bera17] präsentieren zwei kollaborierende Rechenzentren: Sofern die Anfrage-Warteschlange eines Rechenzentrums voll ist, werden Anfragen an das zweite durchgereicht. Das Ziel ist die Zeit im blockierten Zustand – wenn keine Anfragen mehr entgegengenommen werden können – zu minimieren und gleichzeitig die Gesamtausführungszeit der Tasks zu reduzieren. Der Ansatz basiert auf Warteschlangen und vernachlässigt sämtliche Transportzeiten zwischen den beiden Rechenzentren. Eine weitere Einschränkung ist, dass nur Tasks betrachtet werden, deren Speicherverwendung vernachlässigbar im Vergleich zur Prozessorzeit ist, womit ausschließlich eine Betrachtung der Prozessorzeit stattfindet.

Zhu, Si u. a. [Zhu17] präsentieren einen Algorithmus zur Entscheidung, ob eine Berechnung in den Fog-Layer oder in die Cloud ausgelagert werden soll. Als Ziel dient dabei die Ausführungszeit, die minimiert werden soll. Nebenbei soll auch der Energieverbrauch des mobilen Endgerätes reduziert werden. Die Autoren ergänzen die oben vorgestellten drei Phasen um eine initiale Phase, in der die Entscheidung über das Auslagern getroffen wird.

Lin, Yu u. a. [Lin17] beschreiben einen Anwendungsfall, bei dem mittels *Task Offloading* beispielhaft die Latenz minimiert wird, auch wenn das Vorgehen als *Resource Allocation* bezeichnet wird. Dazu stehen benachbarte Edge-Knoten bereit, deren Ressourcen nicht ausgelastet sind, was aber nicht weiter erläutert wird.

3.2.3 Task Scheduling

Bei der Aufgabenplanung (engl. *Task Scheduling*) werden die Aufgaben den verfügbaren Ressourcen, z. B. Prozessoren oder WZM, so zugewiesen, dass bestimmte Kriterien wie Energieverbrauch, durchschnittliche Verzögerung der Aufgabe oder Fertigstellungszeit optimiert werden [Beze22; Mtsh19; Sabo21]. Die Planung von Aufgaben ist in der klassischen Fertigungstechnik ein bekanntes Optimierungsproblem – das »*job-shop scheduling problem*« – und auch in anderen Disziplinen wie bspw. der Weltraumkommunikation relevant [Sabo21].

Grundsätzlich gibt es 2 Arten von Aufgabenplanung: *Dependent Task Scheduling* oder *Independent Task Scheduling*. Dependent bedeutet, dass die Subtasks abhängig von Vorgängertasks sind, und damit eine bestimmte Reihenfolge einzuhalten ist. Das nennt man auch *Workflow Scheduling*. Im Gegensatz dazu sind beim *Independent Task Scheduling* die Subtasks unabhängig voneinander und können beliebig verteilt werden. [Sing17, S. 2]

Laut Liu, Tan u. a. [Liu19, S. 2] behandelt allerdings ein Großteil der Literatur lediglich Tasks, die keinerlei Abhängigkeiten voneinander haben.

Bansal, Aggarwal und Aggarwal [Bans22] präsentieren Ergebnisse eines systematischen Literaturreviews zu *Task Scheduling* in Fog-Computing. Sie verstehen darunter allerdings das Zuweisen von Tasks zu Fog-Knoten, nicht die Reihenfolge der Bearbeitung. Die untersuchte Literatur klassifizieren sie in 4 Gruppen: (1) Statisch, (2) Dynamisch, (3) Heuristisch und (4) Hybrid. Aktan und Bulut [Akta22] stellen ergänzend zu Gruppe (3) verschiedene Metaheuristiken für Aufgabenplanung bei Cloud-Computing vor.

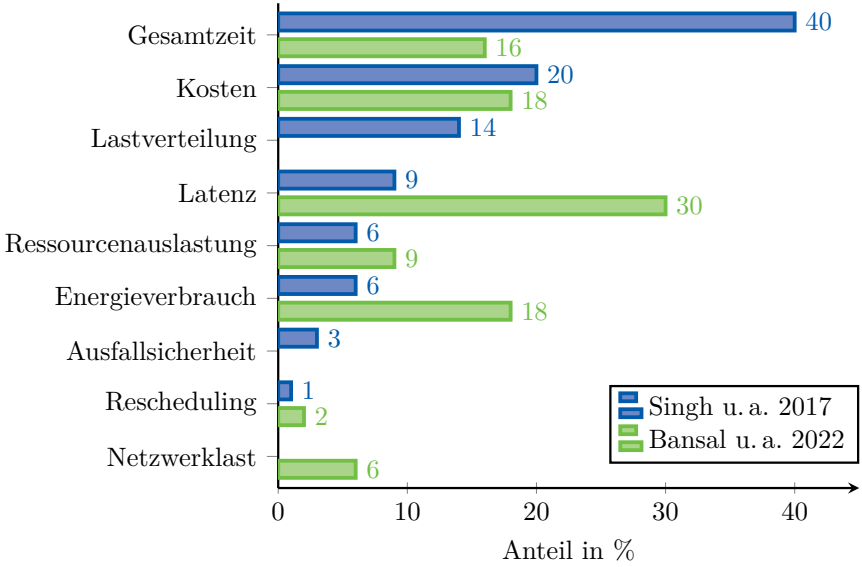


Abbildung 3.1: Hauptoptimierungsziele in der Literatur beim Task Scheduling (basierend auf [Sing17] und [Bans22])

Die beiden Reviews [Sing17, S. 14] und [Bans22, S. 30] gehen u. a. der Frage nach, welche Optimierungsziele in der Literatur bei der Aufgabenplanung verfolgt werden. Die Ergebnisse sind in Abbildung 3.1 dargestellt. Das Review aus dem Jahr 2017 zeigt einen deutlichen Fokus auf die Gesamtausführungszeiten inkl. VM-Hochfahren bzw. Container-Instanziierung bis zur -löschung, was eher die Sicht des Cloud-Anbieters darstellt. Fünf Jahre später zeigt sich, dass der Fokus sich eher auf Latenzen, Energieverbräuche, Kosten und damit die Anwenderperspektive verschoben hat.

Kubernetes-Scheduler

Die weit verbreitete Container-Orchestrierungssoftware Kubernetes nutzt ebenfalls einen *Scheduler*. Der Kubernetes-Scheduler platziert neu erstellte *Pods*[6] auf passende Rechenknoten. Hierzu werden im ersten Schritt alle Knoten herausgefiltert, die, aufgrund von Anforderungen des zu verteilenden Pods, nicht in Frage kommen. Die verbleibenden Knoten werden mit einem *Score* bewertet

[6] Unter *Pods* versteht man im Kubernetes-Kontext Gruppen von mind. 1 Container, die sich einen gemeinsamen Namensraum teilen.

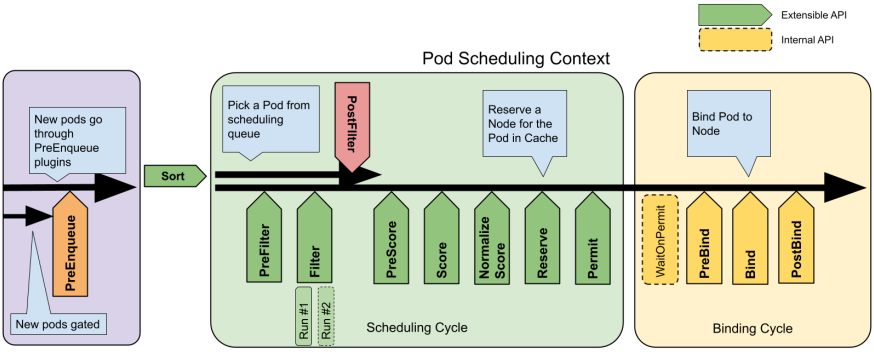


Abbildung 3.2: Scheduling-Prozess und Anpassungsmöglichkeiten des *Kubernetes Scheduling Frameworks* [Kube22b]

und abschließend wählt der *Scheduler* einen der Knoten mit dem höchsten *Score*. Dieser *Score* wie auch der gesamte *Scheduling*-Prozess kann in hohem Maße angepasst werden. Der Prozess sowie alle möglichen Einhängepunkte für Erweiterungen sind in Abbildung 3.2 dargestellt. [Kube22a]

3.2.4 Service Placement und Resource Allocation

Sowohl *Service Placement* als auch *Resource Allocation* beschäftigen sich mit der Frage, auf welchen Ressourcen welche Tasks ausgeführt werden sollten. Die Ziele unterscheiden sich dabei lediglich minimal: *Resource Allocation* hat als Ziel eher die effiziente Auslastung von Ressourcen, während *Service Placement* das Ziel optimaler Performance des Tasks verfolgt. Manche Autoren verwenden die Begriffe auch synonym – z. B. [Gowr21] und [He21].

Klassische Ansätze für die Verteilung von neuen VMs sind *Round Robin*, *First Fit* oder *Best Fit*. Bei *Round Robin* werden die physischen Server der Reihe nach ausgewählt, um die nächste zu platzierende VM aufzunehmen. *Round Robin* führt zu einer sehr unterschiedlichen Auslastung der einzelnen Server. Bei *First Fit* wird der erste Server in einer Liste gewählt, der noch ausreichend freie Ressourcen hat [Shel17, S. 6]. *First Fit* führt zu stark ausgelasteten Servern zu Beginn der Liste und zu wenig bis nicht ausgelasteten Servern am Ende der Liste [Shel17, S. 6]. *Best Fit* hingegen platziert die VM auf dem Server, der die minimalen Restkapazitäten aufweist und gleichzeitig die Anforderungen noch erfüllen kann [Urga07, S. 13].

He, Jin und Dai [He21] erforschen das *Service Placement* für kollaboratives Edge-Computing von Drohnen, die jeweils ihren Onboard-Prozessor für Berechnungen bereitstellen. Ausgangsbasis ist ein mobiles Ad-Hoc-Netzwerk, das von Drohnen aufgebaut wird. Eine Anfrage eines Benutzers kann entweder auf dem Prozessor der nächsten Drohne ausgeführt werden, oder an die benachbarte Drohne weitergereicht werden, womit es sich eigentlich um *Task Offloading* handelt. Durch die Annahme sehr großer Services – in der Größenordnung mehrerer Gigabyte – müssen die Services bereits im Vorfeld auf die Drohnen transferiert werden. Der vorgestellte Algorithmus balanciert die Energie für lokale Berechnungen mit der Energie für den Transfer über das Drohnennetzwerk, um eine maximale Leistungsfähigkeit zu erreichen.

Gowri, Bala und Zion [Gowr21] präsentieren, als Ergebnis eines Literaturreviews, einen Überblick über *Resource Allocation* und *Service Placement* im Bereich Fog- und Cloud-Computing. Sie stellen fest, dass der größte Teil der Arbeiten reaktive Ansätze verfolgt, also erst bei Auftreten eines Problems – bspw. Überlastung einer Ressource – aktiv wird. Auch wenn dies die kostengünstigere Lösung ist, kann sie nicht für zeitkritische Anwendungen wie im IoT genutzt werden [Gowr21, S. 74]. Insbesondere „multi-tier and parallel applications“ identifizieren sie als offene Forschungsaufgabe [Gowr21, S. 77]. Zu Anwendungen im Bereich industrieller Fertigung konstatieren die Autoren Probleme durch fehlende Interoperabilität zwischen den Fertigungsgeräten und der Computing-Infrastruktur, was zu intolerablen Verzögerungen führe. Durch die wenigen gefundenen Forschungsarbeiten stellen sie hier eine weitere Forschungslücke fest [Gowr21, S. 77].

Duan, Chen u. a. [Duan17] präsentieren eine Kombination aus Auslastungs-Vorhersage und Metaheuristik-Scheduler, um den Energieverbrauch in einem Rechenzentrum zu senken. Der Energieverbrauch wird gesenkt, indem VMs auf möglichst wenige Server verteilt werden (*Server Consolidation*). Die nicht benötigten Server können dann ausgeschaltet werden. In Abbildung 3.3 ist der präsentierte Systemaufbau dargestellt. Das Vorhersage-Modell prognostiziert die Auslastung der einzelnen Server im Cluster und löst bei Bedarf einen neuen Durchlauf des Schedulers aus. Der *Scheduler* wiederum basiert auf einem verbesserten *Ant-Colony-Optimization*-Algorithmus. Durch Simulation wird der Ansatz validiert und zeigt u. a. eine Verbesserung von 75 % im Energieverbrauch ggü. dem klassischen *Round-Robin*-Verfahren.

Beloglazov, Abawajy und Buyya [Belo12] stellen einen *Resource-Allocation*-Ansatz für VMs vor. Nach ihrer Ansicht wird der Energieverbrauch eines Servers größtenteils durch Prozessor, Arbeitsspeicher (RAM), Festplatten

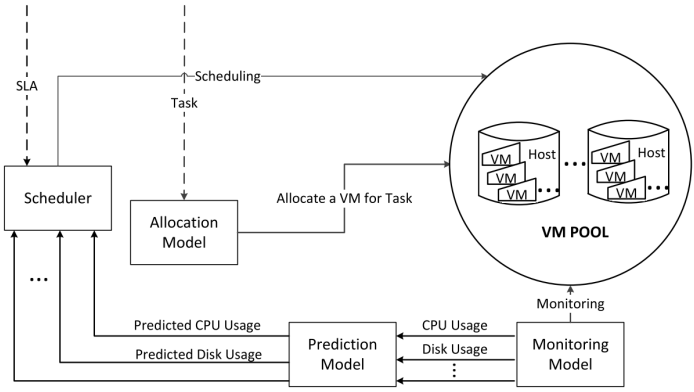


Abbildung 3.3: Systemaufbau zur Auslastungsvorhersage und Scheduling von VM [Duan17]

und Netzwerkschnittstelle bedingt, von denen der Prozessor den größten Anteil hat [Belo12, S. 758]. Für den eigentlichen Algorithmus werden für die Server zwei Schwellwerte für die Prozessorauslastung gesetzt. Wenn der untere unterschritten wird, soll der Server abgeschaltet werden. Wird der obere überschritten, müssen einzelne VMs zu einem anderen Server migriert werden. Im ersten Fall müssen alle VMs migriert werden, im zweiten Fall wird die Anzahl der zu migrierenden VMs minimiert.

Shelar, Sane u. a. [Shel17] präsentieren ebenfalls einen *Resource-Allocation*-Ansatz der das Ziel verfolgt, die physischen Server möglichst auszulasten. Durch die maximale Auslastung wird die Anzahl an gleichzeitig benötigten Servern minimiert, was wiederum den Energieverbrauch des Rechenzentrums minimiert, indem die unbenutzten Server in einen Energiesparmodus geschaltet werden. Die Autoren setzen 70 % Auslastung als oberen Grenzwert, ab dem keine weiteren VMs mehr auf einem Server platziert werden. Sie begründen die verbleibenden 30 % als Puffer für zukünftige Skalierung. Um die Startzeit bei der Neuplatzierung einer VM gering zu halten, wird eine Mindestanzahl von unbenutzten Servern bereitgehalten und nicht in den Energiesparmodus geschickt.

Gill, Garraghan und Buyya [Gill19] bemängeln, dass für Fog-Computing bisher kein *Resource-Allocation*-Algorithmus alle relevanten Ziele gleichzeitig einbeziehe. Allerdings ziehen sie für diesen Ansatz lediglich sieben Publikationen aus den Jahren 2015 bis 2017 heran. Aus Sicht der Autoren sind die relevanten

Ziele Antwortzeit, Netzwerkbandbreite, Energieverbrauch und Latenz, weswegen sie einen Ansatz präsentieren, der die Metaheuristik PSO nutzt, um die am besten geeignete Ressource nach diesen vier Kriterien zu finden.

Lee, Chung und Kim [Lee17] stellen einen Ansatz vor, bei dem SDN-Switches als Fog-Knoten in der Lage sind containerisierte Services auszuführen. Die Auswahl des besten Fog-Knotens erfolgt durch Minimierung der notwendigen Netzwerk-Hops[7] zwischen Datenquelle, Fog-Knoten und Konsument. Die Auslastung oder Energieverbräuche der Knoten wird also nicht in die Entscheidung mit einbezogen, im Gegensatz zu vielen anderen Publikationen aber die Netzwerktopologie des Fog-Clusters.

3.2.5 Charakterisierung von Tasks

Neben den Informationen über die zu übertragenden Daten und Abhängigkeiten in der Ausführungsreihenfolge innerhalb einer Service-Komposition können für die Verteilung der Tasks auf Rechenressourcen noch Informationen zum Laufzeitverhalten der Tasks genutzt werden.

Insbesondere sind die belegten Ressourcen relevant für die Platzierungsentscheidung. In den meisten Publikationen werden diese Ressourcen als statisch oder konstant angenommen. Dies ist dem Umstand geschuldet, dass die meisten Ansätze Tasks als einmalige Aufgaben mit endlicher Laufzeit betrachten, deren Bearbeitungsdauer sich umgekehrt proportional zur Rechenleistung verhält. Für den Bereich Produktion gilt dies allerdings nur für manche Typen zu verteilender Software, während andere kontinuierlich laufen müssen, oder regelmäßig aktiv sind:

Dynamisches Verhalten von Tasks

Mazumdar und Pranzo [Mazu17, S. 6] nennen für ihren *Resource-Allocation*-Algorithmus das Problem, dass dem Cloud-Anbieter nicht bekannt ist, welches zeitliche Verhalten die Ressourcenanforderungen einer VM aufweisen, weswegen die Autoren konservativ annehmen, dass die Auslastung immer maximal ist. Dadurch sinkt die tatsächliche Auslastung der genutzten Server allerdings signifikant, was dieses Vorgehen hinsichtlich der Energieeffizienz nicht ideal macht.

Brecher und Weck nennen 4 Klassen von Überwachungsaufgaben im Bereich der WZM [Brec21, S. 198]:

[7] *Hop* bezeichnet in der Netzwerktechnik eine Zwischenstation auf dem Weg der Datenübertragung.

1. Kontinuierliche Überwachung: Hier wird dauerhaft – während der Betriebszeit – einer oder mehrere Parameter überwacht. Beispielsweise kann dies die Überwachung der Temperatur empfindlicher Baugruppen sein.
2. Kurzperiodische Überwachung: Ein Beispiel ist die Schwingungsüberwachung des Motorspindelsystems, des bereits in Abschnitt 1 vorgestellten Bearbeitungszentrums. Hier soll schnellstmöglich eine Kollision erkannt und Schutzmaßnahmen eingeleitet werden. Allerdings ist diese Überwachung nur im Betrieb der WZM notwendig, in Pausen- oder Rüstzeiten, wenn die Maschine stillsteht, können keine Kollisionen auftreten.
3. Langperiodische Überwachung: Diese Klasse unterscheidet sich von der kurzperiodischen Überwachung lediglich dadurch, dass die Aufgaben seltener durchgeführt werden. Ein Beispiel ist das Fahren einer Referenzfahrt und gleichzeitiges Aufzeichnen und Auswerten eines sog. Maschinen-Fingerprints (so z. B. [Fogl21]).
4. Überwachung auf Anforderung: Die letzte Klasse der Überwachungsaufgaben wird nicht periodisch oder kontinuierlich ausgeführt, sondern lediglich auf konkrete Anforderung einmalig ausgeführt. Dies können Analysemethoden sein, die bei Auftreten oder Verdacht einer Störung ausgelöst werden.

Gezer und Wagner [Geze21, S. 2305] unterteilen abhängig vom Auftreten der Anfragen in drei Kategorien:

1. Periodische Tasks mit konstanter Rate und immer gleichen Aktivitäten pro Ausführung.
2. Aperiodische Tasks, die typischerweise ereignisgesteuert auftreten, mit unlimitierter Zeit zwischen dem Auftreten.
3. Sporadische Tasks, die ebenfalls aperiodisch sind, aber mit begrenzten Zeitintervallen zwischen dem Auftreten.

Kombiniert man beide Quellen, lässt sich folgende Unterteilung für Tasks im Produktionsbereich annehmen – mit abnehmender Häufigkeit des Auftretens pro Zeitintervall:

1. Kontinuierlich
2. Periodisch
3. Sporadisch
4. Aperiodisch

Klassifizierung von *Cloud-Workloads*

Im Cloudbereich ist die Kategorisierung von Belastung der verschiedenen Ressourcen vielfältig untersucht worden [Sehg18, S. 62]. Um hier eine möglichst gleichmäßige Auslastung aller Ressourcen eines Servers – bspw. Prozessor (CPU), Netzwerkschnittstelle, Caches ... – zu erreichen, können Aufgaben verschiedener Kategorien miteinander kombiniert werden, ohne die Leistung der einzelnen Anwendungen wesentlich zu beinträchtigen.

Sehgal und Bhatt [Sehg18, S. 67–70] definieren 18 Klassen von *Cloud-Workloads*:

1. *Big Streaming Data*
2. *Big Database Creation and Calculation*
3. *Big Database Search and Access*
4. *Big Data Storage*
5. *In-Memory Database*
6. *Many Tiny Tasks (Ants)*
7. *Tightly Coupled-Intensive Calculation (HPC)*
8. *Separable Calculation-intensive HPC*
9. *Highly Interactive Single Person Tasks*
10. *Highly Interactive Multi-Person Jobs*
11. *Single-Computer Intensive Jobs*
12. *Private Local Tasks*
13. *Slow Communication*
14. *Real-Time Local Tasks*
15. *Location-Aware*
16. *Real-Time Geographically Dispersed Tasks*
17. *Access Control*
18. *Voice or Video over IP*

Die Anwendungsfälle im Produktionsnetzwerk gehören größtenteils zu den Klassen *Real-Time Local Tasks* bzw. *Real-Time Geographically Dispersed Tasks*. Als die Hauptressourcen für diese Klassen werden Prozessoren und Netzwerkverbindung genannt. [Sehg18, S. 69–70]

Neben der Frage welche Ressourcen die Tasks belegen, ist auch die Frage nach der Migrierbarkeit eines Tasks relevant. Liu [Liu13, S. 677] klassifiziert für das *Task Offloading* Tasks in zwei Gruppen:

1. *Preemptive Applications* – Dies sind Tasks die unterbrochen bzw. pausiert werden können. Werden diese Tasks wieder aufgenommen – ob

auf der Original-Ressource, oder nach Verschieben auf einen anderen Rechenknoten – läuft die Berechnung einfach weiter, als hätte keine Pause stattgefunden.

2. *Non-preemptive Applications* – Diese Tasks können eventuell unterbrochen werden, erfordern allerdings einen kompletten Neustart der Abarbeitung, was zu Verschwendung von Rechenkapazitäten führt.

Hierbei ist noch eine weitere Gruppe zu ergänzen:

3. Tasks, die überhaupt nicht unterbrochen werden dürfen, da es während der Migration zu Datenverlusten kommt. Dies ist insb. ein Problem bei kontinuierlichen Tasks, die per *Streaming* kontinuierlich Daten gesendet bekommen.

Quantitative Beschreibung von Tasks

Neben der allgemeinen Zugehörigkeit in eine der genannten Klassen sind auch konkrete, quantitative Beschreibungen der Ressourcenanforderungen einzelner Tasks notwendig. Varshney, Sandhu und Gupta [Vars20] nutzen für die Charakterisierung der zu verteilenden Tasks:

1. *Storage*: Speicherplatz, den die Anwendung benötigt.
2. *CPU-Cycle*: Anzahl der benötigten Prozessortakte.
3. *Maximum Latency*: Maximale Latenz, bis die Anwendung starten muss.
4. *Network Bandwidth*: Notwendige Bandbreite, um Daten zu transferieren.
5. *Processing Speed*: Gesamtdauer, bis die Bearbeitung abgeschlossen sein muss.

Zhu, Si u. a. [Zhu17, S. 62] beschreiben die zu verschiebenden Tasks mittels vier Größen, die leider nicht weiter detailliert werden: (1) Größe der Anfrage, (2) Menge der Aufgaben die verarbeitet werden müssen, (3) Größe der Antwort und (4) die Deadline des Tasks.

3.2.6 Charakterisierung von Rechenknoten und Netzwerk

Analog zur quantitativen Beschreibung der Ressourcenanforderungen der Tasks müssen auch die Rechenknoten und Netzwerkverbindungen beschrieben werden, damit ein Abgleich dazwischen stattfinden kann. Dies geschieht meist mit wenigen Kennzahlen, um die Komplexität des Optimierungsproblems nicht unnötig zu steigern.

Liu [Liu13, S. 676] nutzt hingegen viele Kennzahlen. Er charakterisiert Rechenknoten mit folgenden Parametern:

- Anzahl der verfügbaren Prozessoren(& -kerne)
- Prozessorgeschwindigkeit
- Kapazität des Arbeitsspeichers

Für die Kommunikations-Knoten kommen dazu noch folgende Parameter:

- Verfügbare Netzwerkbandbreite
- Kapazität der Kommunikationsverbindung
- Kommunikationsverzögerung zwischen zwei Standorten
- Netzwerklatenz

Leider fehlt eine qualifizierte Aussage, wie sich diese teilweise sehr ähnlich klingenden Netzwerkparameter unterscheiden.

Duan, Chen u. a. [Duan17, S. 144] nutzen für ihre Lastvorhersage lediglich die Auslastung des Prozessors und der Lese-Schreib-Bandbreite der Festplatten. Ergänzend fügen sie hinzu, dass ihr Ansatz allgemein verwendbar sei, und zukünftig auch weitere Parameter miteinbezogen werden könnten.

Mazumdar und Pranzo [Mazu17, S. 6] lassen die Anzahl verfügbarer Prozessorkerne, die Arbeitsspeichergröße und die Netzwerk-Bandbreite in ihren Algorithmus einfließen. Sie begründen diese Auswahl damit, dass Anwender:innen bei einem typischen IaaS-Anbieter auch nur diese Auswahlmöglichkeiten haben. Sollte es nötig sein, könnten aber weitere Ressourcen, analog zum vorgestellten Vorgehen, ergänzt werden. Da ihr Ansatz aber aus der Sicht eines Infrastruktur-Anbieter formuliert ist, erscheint diese Einschränkung inkonsistent.

Sehgal und Bhatt [Sehg18, S. 76–77] schlagen vor, statt der *CPU-Utilization* indirekt die Metrik *Instructions per Cycle* zu überwachen, da diese ein besseres Bild abgäbe, ob der Prozessor ausgelastet sei. Ebenso empfehlen sie, *Cache Misses* zu überwachen, da hiermit eine Aussage über die Überlastung der (Arbeits-)Speicher möglich sei. Allerdings sind diese Metriken eher für die Überwachung im Betrieb geeignet, und nicht zur vorhergehenden Berechnung ob ein Server für einen Task geeignet ist.

3.2.7 Zusammenfassung der Informatikliteratur

Nachdem aus der Produktionstechnikliteratur noch alle Fragen offen waren, konnte der Blick in die allgemeine Informatikliteratur einige davon beantworten. Die von den beiden Disziplinen jeweils behandelten Kriterien sind in Tabelle 3.2 dargestellt.

Insbesondere unterschiedliche Leistungsfähigkeit der Edge-Devices (F1) und die Verteilung von Service-Kompositionen scheinen außerhalb der Produktionstechnik bereits adressiert zu sein. Für dynamische Neuberechnungen der Softwareverteilung (F2) existieren Grundlagen, genauso wie auch manche Klassifizierungen für Tasks (F2), allerdings sollten diese weiter untersucht werden. Echtzeit-Anforderungen (F3) werden aber weiterhin nur unzureichend beschrieben. Da jedoch Latenzen bzw. Antwortzeiten als Optimierungsziele berichtet werden, sind Grundlagen vorhanden, die in Richtung Echtzeit weiterentwickelt werden können.

Einflüsse der Netzwerk-Topologie (F4) und insbesondere von Ausfallwahrscheinlichkeiten der Netzwerkverbindungen sind auch in der Informatikliteratur nur unzureichend enthalten, vermutlich da dieses Thema eher im Technologiebereich Mobilfunk (3G, 4G, 5G) thematisiert wird. Auch physische Nähe von Edge-Devices wird insbesondere im Bereich Ad-Hoc-(Mobilfunk-)Netzwerke bereits adressiert. Die gefundenen Ansätze müssen allerdings noch geeignet kombiniert und auf die Besonderheiten der Produktionsumgebung angepasst werden.

Tabelle 3.2: Ableitung von Forschungslücken aus der Abdeckung der Fragen in den beiden Disziplinen Produktionstechnik und Informatik
×: vollständig; (×): eingeschränkt

Nr.	Kriterium	Frage	Produktion	Informatik	Forschungslücke
1	Innerhalb der Edge-Ebene	1	×	×	
2	Nicht nur Rechenzentrum	1	×	×	
3	Heterogene Edge-Devices	1	(×)	×	
4	Dynamisch	2	(×)	(×)	(×)
5	Lastprofile	2		(×)	×
6	Echtzeit	3	(×)	(×)	×
7	Streaming	2			×
8	Netzwerkeinfluss	4	(×)	(×)	×
9	Hardwarebindung	5			×
10	Task-Abhängigkeiten	–		×	

3.3 Forschungslücken

Die bereits beschrieben Tabelle 3.2 enthält in der letzten Spalte auch die verbleibenden Forschungslücken. Vier der zehn formulierten Kriterien (1, 2, 3, 10) sind nach der Literaturanalyse als ausreichend beantwortet zu bewerten. Für die Beantwortung von Frage F1 sind damit ausreichend Ansätze vorhanden (z. B. in [Wang21c] oder [Qi19]).

Die weiteren Kriterien und damit auch Fragen sind nicht ausreichend abgedeckt, sodass sich folgende Defizite der untersuchten Literatur zusammenfassen lassen:

- Dynamische Neu-Berechnungen werden selten vorgestellt. Dies liegt vermutlich daran, dass die meisten Ansätze keine kontinuierliche Ausführung annehmen, sondern Einmal-Ausführungen mit begrenzter Laufzeit postulieren. (F2)
- Lastprofile mit jeweils charakteristischen Größen des zeitlichen Verhaltens finden fast keine Nutzung. (F2)
- Kontinuierliche Datenströme, also *Streaming*, sind in den vorgestellten Studien nicht vorgesehen. (F2)
- Echtzeit-Anwendungen mit garantierter Einhaltung von Latenzanforderungen fehlen. Grundlagen existieren aber durch die Latenzminimierung in einigen vorgestellten Studien. (F3)
- Netzwerkeinflüsse werden nur in Teilaspekten insgesamt aber unzureichend betrachtet. (F4)
- Hardwarebindung von Tasks findet keine Beachtung. (F5)

Die identifizierten Forschungslücken werden im nächsten Kapitel zur Formulierung der Zielsetzung der weiteren Arbeit herangezogen.

4 Zielsetzung der Arbeit

Um die Ressourceneffizienz der *Shopfloor*-IT-Infrastruktur zu erhöhen, wird in dieser Arbeit das Ziel der *Server Consolidation* verfolgt, d. h. dass die Geräte möglichst vollständig ausgelastet werden sollen, da Computer im Leerlauf bereits einen hohen Anteil ihres Maximalenergieverbrauchs benötigen (siehe Abschnitt 2.4.1). Um dies mittels Einsatz von CNT zu erreichen, müssen Hindernisse für deren Einsatz in der Produktionsindustrie behoben werden.

Für die Kooperation von Edge-Devices in einem Fog-Cluster zeigt sich, dass u. a. im Bereich der Container-Orchestrierung noch einzelne Aspekte dem Einsatz im Shopfloor entgegenstehen. Für diese Arbeit wird der Bereich Softwareverteilung – als Teil der Orchestrierung – betrachtet, da insbesondere hier Defizite vorhanden sind. Zu diesen Defiziten gehören in der Produktion für Echtzeitanwendungen geforderte Latenzobergrenzen, welche die typische Größenordnung aus der IT unterschreiten[8] und die von den verfügbaren Ansätzen nicht gewährleistet werden können.

In der Informatik gibt es zwar bereits viele Ansätze zur Softwareverteilung, diese betrachten aber in der Regel entweder die Auslastung der einzelnen Rechner oder das Netzwerk und die Datentransportwege zwischen den Rechnern. Bisher gibt es nur wenige kombinierte Ansätze, die für den Einsatz in der Maschinenbauindustrie aber von großer Bedeutung wären: Für das übergeordnete Ziel der Ressourceneffizienz ist es notwendig, die einzelnen Auslastungen der Computer zu betrachten. Um hingegen den Anforderungen an Latenzen – die sich aus Bearbeitungs- als auch Transportzeiten zusammensetzen –, gerecht zu werden, muss auch das Netzwerk betrachtet werden. Hierfür müssen Topologie, Verbindungsqualität sowie Auslastung der Netzwerkgeräte miteinbezogen werden.

Das **Ziel** dieser Arbeit ist es daher, einen Ansatz zu finden, der Platzierungsentscheidungen für Tasks treffen kann, die geeignet sind, die Ressourceneffizienz des Gesamtsystems zu erhöhen. Für die Tasks, bestehend aus voneinander abhängigen Subtasks, sind jeweils *a priori* Ressourcenanforderungen bekannt. Zusätzlich gelten für sie strenge Anforderungen hinsichtlich Latenzen – in

[8] Maschinenregelungen arbeiten beispielsweise mit Zykluszeiten von 1 ms bis 3 ms, typische Prozessregelungen z. B. Streaming-Analytics in der Lackiererei [Dürr23] um 10 ms bis 12 ms. Dagegen sind in der untersuchten Informatikliteratur eher 100 ms und mehr üblich.

der Größenordnung von wenigen Millisekunden – die der gewählte Ansatz berücksichtigen muss, indem Einflüsse des Netzwerks im Fog-Cluster in die Softwareverteilung einbezogen werden.

Der Ansatz soll also vier der identifizierten Forschungslücken adressieren:

- Betrieb von Echtzeitanwendungen durch Einhalten von Latenzanforderungen ermöglichen.
- Einbeziehen von Netzwerkeinflüssen in die Platzierungsentscheidung.
- Betrachten von kontinuierlichen Datenflüssen zwischen einzelnen Subtasks.
- Die konkrete Bindung einzelner Subtasks an bestimmte Edge-Devices erlauben.

Die dynamische Betrachtung der Ressourcenauslastung in der Softwareverteilung, was gegenwärtig ebenfalls eine Forschungslücke darstellt, wird in dieser Arbeit zwar nicht gelöst, über die Datenanalyse werden aber Grundlagen für die weitere Auseinandersetzung mit diesem Thema geschaffen, indem charakteristische Zeitverläufe ermittelt werden.

In der in [Ahme17] vorgestellten Edge-Taxonomie kann diese Arbeit in die in Abbildung 4.1 farblich hervorgehobenen Bereiche eingruppiert werden. Lediglich die Einordnung in die Kategorie »*Latency minimization*« trifft nur teilweise zu, da die Latenz nicht minimiert werden soll, wohl aber Latenzanforderungen eingehalten werden müssen.

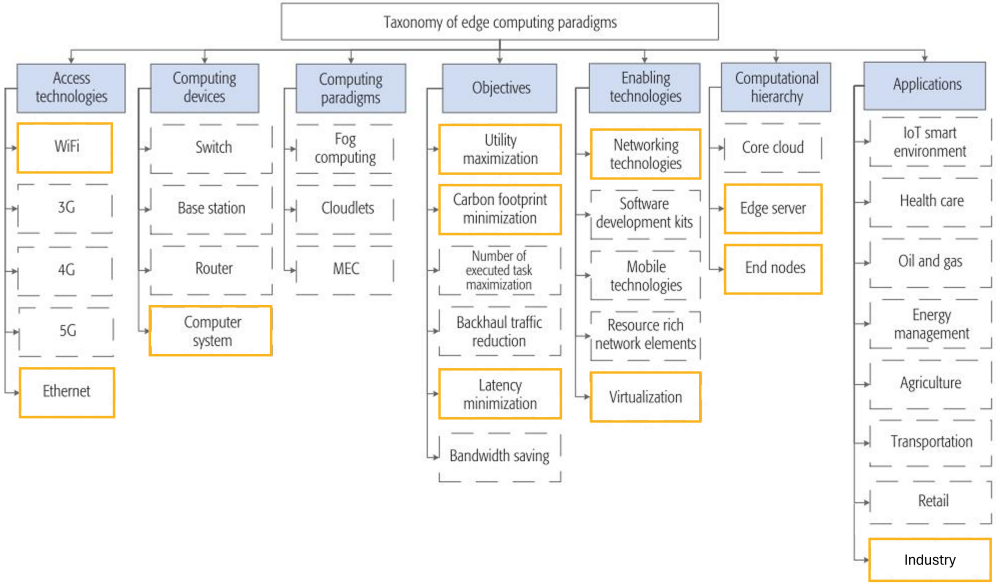


Abbildung 4.1: Einordnung (orange) der Arbeit in die Edge-Taxonomie (basierend auf [Ahme17] erweitert)

5 Vorgehen

Nachdem aus den Forschungslücken ein Ziel formuliert wurde, wird im folgenden Kapitel das weitere Vorgehen konkretisiert. Die gewählten Schritte ergeben sich aus den bereits gefundenen Ansätzen aus der Literatur und notwendigen Ergänzungen. Insgesamt handelt es sich um sechs aufeinander aufbauende Schritte: Begonnen wird mit einer einleitenden Vorstudie sowie der Charakterisierung von einzelnen Tasks. Daraufhin folgt die Task- sowie Cluster-Modellierung, das Optimierungsproblem und anschließend die Validierung des Ansatzes. Diese sechs Schritte werden kurz vorgestellt und verweisen auf die entsprechenden Kapitel, in denen eine detaillierte Beschreibung zu finden ist.

Vorstudie Um sich dem Entscheidungsansatz für die Softwareverteilung im *Shopfloor*-Fog-Cluster zu nähern, wird zunächst eine Vorstudie durchgeführt. Das Design und die Ergebnisse der Studie werden in Kapitel 6 vorgestellt. Das Ziel dieser Vorstudie ist es, die getroffenen Annahmen, nämlich die Verschwendung in Form von wenig ausgelasteten Rechnern in einem Produktionsnetz zu überprüfen. Als Produktionsnetz wird ein Teil des TEC-Labs des Instituts für Produktionsmanagement, Technologie und Werkzeugmaschinen (PTW) der TU Darmstadt gewählt. Hier sind neben den traditionellen Prozessen der Maschinensteuerungen mehrerer WZMs auch Analytik-Anwendungen, Fernwartung und ein MES auf diversen IPCs im Einsatz. Diese Rechner werden mit einem Monitoring-System ausgestattet, dass viele Messwerte des Betriebssystems, Prozessor, Arbeitsspeicher, Festplatten, Netzwerk bereitstellen kann. Über eine zentrale Messwerterfassung werden ausgewählte Größen aller Rechner im Produktionsbetrieb aufgezeichnet und für Auswertungen bereitgestellt.

Charakterisierung von Tasks im Produktionsumfeld Auf Basis der Daten der Vorstudie werden die einzelnen Prozesse – als ausgeführte Instanzen der Subtasks – analysiert. Hierzu wird eine explorative Datenanalyse durchgeführt und ebenfalls in Kapitel 6 vorgestellt. Das Ziel ist es dabei, Gruppen ähnlicher Prozesse zu finden, die jeweils über die gleichen – möglichst wenigen – charakteristischen Größen beschrieben werden können. Erwartet werden gemäß der Literatur in Abschnitt 3.2.5 verschiedene Zeitabhängigkeiten der Prozesse.

Task-Modellierung Nachdem die Charakterisierung von einzelnen Subtasks ermöglicht ist, sollen die Abhängigkeiten zwischen den Subtasks modelliert werden, was in Abschnitt 7.1 beschrieben wird. Gemäß den gefundenen Literaturstellen, wird die Modellierung der Tasks mit den abhängigen Subtasks über Graphen angegangen. Die charakteristischen Größen der statischen Prozessgruppen finden Eingang in die Teilmodelle der Subtasks. Damit wird vereinfachend angenommen, dass die Größen keine zeitliche Abhängigkeit haben. Außerdem werden die Datenflüsse zwischen den Subtasks als Kanten des Graphen aufgenommen. Es ergibt sich das Task-Modell.

Cluster-Modellierung Für die Edge-Devices die als Fog-Cluster kooperieren sollen und das sie verbindende Netzwerk wird ebenfalls eine Modellierung als Graph gewählt. Hierfür werden die Netzwerkverbindungen mit Kabeln, Funk, Switches, Accesspoints, etc. und die Computer selber dargestellt und mit – zu denen der Tasks korrespondierenden – Größen charakterisiert. Dies wird im Folgenden als das Cluster-Modell bezeichnet und wird in Abschnitt 7.2 erläutert.

Optimierungsproblem Als Ergebnis der Modellierung stehen zwei Modelle – Cluster- und Task- – bereit, die es im nächsten Schritt zu kombinieren gilt. Aus dieser Kombination und dem Ziel einer ressourceneffizienten Softwareverteilung ergibt sich ein diskretes Optimierungsproblem mit Nebenbedingungen. Das Optimierungsproblem wird im Rahmen dieser Arbeit mit verschiedenen Metaheuristiken gelöst. Das Optimierungsproblem sowie die Lösungssuche werden in Abschnitt 7.3 vorgestellt.

Diese Metaheuristiken werden in einem Benchmark verglichen, um diejenigen mit einer kurzen Bearbeitungszeit und geringen belegten Rechenressourcen zu ermitteln, die trotzdem zu guten – wenn auch vielleicht nicht optimalen – Lösungen führen. Hier gilt es abzuwägen, ob der Aufwand zum Finden der optimalen Lösung durch die eingesparten Ressourcen gerechtfertigt ist, oder auch eine nicht-optimale Lösung akzeptiert werden kann.

Validierung Da viele der untersuchten Applikationen noch nach dem alten Maschinenbau-Paradigma der statischen Monolithen programmiert sind – die keine Umverteilung ermöglichen – muss eine Alternative zur Validierung am Produktiv-System genutzt werden. Die Gültigkeit des Suchalgorithmus wird daher anhand eines Brute-Force-Verfahrens überprüft, das in Kapitel 8 vorgestellt wird.

Hierfür wird ein reduzierter Task mit wenigen Subtasks und Abhängigkeiten modelliert, der auf dem Cluster-Modell verteilt wird. Auf einem leistungsfähigen Computer werden für alle möglichen Verteilungen jeweils die Güte berechnet, sowie die Nebenbedingungen überprüft. Als Ergebnis ist die global optimale Lösung bekannt, mit der die Lösungen der Metaheuristiken verglichen werden können.

6 Experimentelle Bestimmung von typischen Lastprofilen

6.1 Messumgebung

Zur Vermessung der zu untersuchenden Rechner kommt Prometheus – eine freie *Service-Monitoring*-Software – zum Einsatz. Auf jedem zu vermessenden Rechner wird dazu ein sogenannter *Exporter* eingerichtet. Diese *Exporter* sammeln vom Betriebssystem – oder der Virtualisierungsumgebung im Falle von Docker – Auslastungskenngrößen ein und stellen diese über eine *Hypertext Transfer Protocol* (HTTP)-Schnittstelle zur Verfügung. Da einige Rechner nur eingeschränkte Zugriffe auf das Betriebssystem erlauben – hier seien insbesondere die *Numerical Control Unit*s (NCUs)[9] sowie die Sinumerik-Edge[10] genannt – wurde für diese Linux-basierten Rechner ein anderes Vorgehen zur Messung gewählt.

Da auf allen Linux-Rechnern mindestens das Systemwerkzeug »top« zur Verfügung stand, wird per *Secure Shell* (SSH) eine Verbindung zu einer Kommandozeile auf dem Rechner aufgebaut. Auf der entfernten Kommandozeile wird dann das top-Werkzeug ausgeführt, um Informationen über die Systemauslastung sowie die Prozesse zu erhalten, die das System am meisten belasten. In Code 6.1 ist eine beispielhafte Ausgabe des »top«-Werkzeugs auf einer Sinumerik NCU dargestellt. Ein Programm extrahiert aus der Ausgabe auf dem entfernten Rechner automatisch die relevanten Messwerte und stellt diese anschließend im Prometheus-Metrik-Format bereit.

Unabhängig davon ob die Messwerte direkt auf dem Rechner bereitgestellt werden oder von einem entfernten Rechner über einen SSH-Tunnel extrahiert werden müssen, werden diese abschließend vom *Prometheus Collector* eingesammelt, indem dieser die HTTP-Endpunkte regelmäßig abfragt und die übertragenen Messwerte in die Datenbank (PrometheusDB) einträgt. Anschließend stehen die Werte zur automatisierten Abfrage bspw. über Skriptsprachen oder zur Visualisierung über Grafana bereit.

[9] Die Sinumerik NCU basiert auf einem BusyBox-embeddedLinux, das im Vergleich mit GNU/Linux nur einen sehr eingeschränktem Funktionsumfang hat.

[10] Das Betriebssystem der Sinumerik-Edge erlaubt keine permanenten Änderungen sondern setzt sich nach jedem Neustart auf einen Grundzustand zurück.

Code 6.1: Beispielausgabe von »top«

```
top - 06:33:15 up 5 days, 14:28,  0 users,  load average: 3.13, 2.92, 2.84
Tasks: 210 total,   3 running, 207 sleeping,   0 stopped,   0 zombie
Cpu(s): 24.5% us,  7.0% sy,  0.0% ni, 67.9% id,  0.2% wa,  0.0% hi,  0.5% si
Mem:   1678468k total,  1662764k used,    15704k free,   270492k buffers
Swap:        0k total,        0k used,        0k free,   991124k cached

PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
1444 root      20   0  702m 439m 314m S   72 26.8   9492:20 loadnck
4108 root      20   0  102m  34m  18m R   44  2.1   2508:36 uaserverc
2438 root      20   0 12516 5456 4732 R   40  0.3   0:00.35 opcuastatusclie
4016 root      20   0 1203m 193m 114m S   11 11.8   1778:26 slsmhmihost
2442 manufact  20   0  2408 1052  756 R    6  0.1   0:00.05 top
```

In Abbildung 6.1 ist ein Verteilungsdiagramm des gesamten Messsystems dargestellt. Wie zu erkennen ist, finden 3 Gerätegruppen («device», gelb) im Messaufbau Verwendung. Dies sind zunächst die beiden Zielsysteme, die WZM »Heller CNC-ProfiTrainer PT16« (dargestellt in Abbildung 6.2a) und die WZM »DMG MORI DMC 850 V« (zu sehen in Abbildung 6.2b). Weitere WZMs werden nicht vermessen, da der dafür notwendige (Administrator-)Zugriff auf die Computer fehlt. Zusätzlich kommt für die Messung zusätzliche Hardware zum Einsatz, die in einem Messturm untergebracht ist.

In den Gerätegruppen sind in blau einzelne Rechner dargestellt. Die beiden WZM enthalten jeweils mindestens eine NCU auf der die eigentliche Steuerung der Maschine untergebracht ist, diese werden über einen SSH-Tunnel – wie oben beschrieben – ausgelesen.

Auf dem Edge-Device »EuPG-Edge«, sowie dem Mess-PC ist jeweils Docker als Container-Ausführungsumgebung vorhanden. Diese Umgebung wird mittels 3 *Exporter* vollständig erfasst: *Windows-* bzw. *Node Exporter* liefern Messwerte über das zugrundeliegende Betriebssystem, der *Process Exporter* stellt Messwerte zu den ausgeführten Prozessen auf dem Betriebssystem bereit und der *Docker Exporter* informiert über die Prozesse, die als Container isoliert betrieben werden.

Bei der DMG MORI DMC 850 V tritt ein Problem beim bisher skizzierten Messaufbau auf. Die meisten Rechner sind lediglich an das maschineninterne Anlagennetz angeschlossen und somit von extern nicht erreichbar. Der CELOS-IPC ist als Schnittstelle in das Fabrik-Netzwerk vorgesehen, daher muss hier ein *Reverse Proxy* aktiv sein, der die Verbindung zu den maschineninternen Rechnern erlaubt, indem Anfragen weitergeleitet werden.

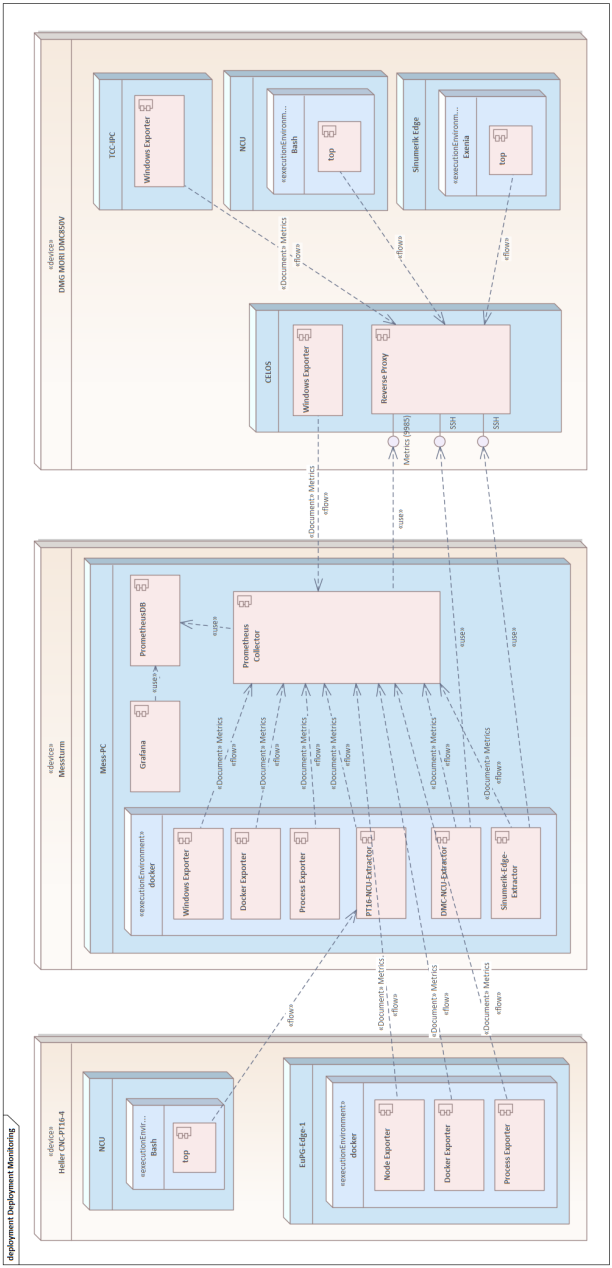


Abbildung 6.1: Deployment-Ansicht des Messsystems

(a) (b)

Abbildung 6.2: Die untersuchten Werkzeugmaschinen:
(a) Heller – CNC-ProfiTrainer [Gebr22];
(b) DMG MORI – DMC 850 V [DMG 23]

6.2 Auslastungsdefizit

Alle vorgestellten Rechner wurden im Normalbetrieb der Werkzeugmaschinen hinsichtlich ihrer Auslastung vermessen. Normalbetrieb bezeichnet hier die Ausführung eines NC-Programms und damit einen Mix aus Haupt- und Nebenzeiten. Nicht ausgewertet sind daher z. B. Rüstvorgänge, da hier noch geringere Auslastungen auftraten. Auf den Rechnern läuft, neben den für den normalen Betrieb benötigten Prozessen, weitere Software aus den Datenverarbeitungsketten, die in Abschnitt 2.3 vorgestellt wurden. Konkret sind dies auf dem Rechner »Edge« Prozesse von [Fert22a] und [Kohn22].

In Abbildung 6.3 ist die Prozessorauslastung der vermessenen Rechnereinheiten der DMC 850 V dargestellt, in Abbildung 6.4 die Auslastung der Arbeitsspeicher im gleichen Zeitraum. Es fällt auf, dass der Prozessor der NCU nur sehr wenig (ca. 5 %) ausgelastet ist, während der Arbeitsspeicher fast vollständig belegt ist. Dies lässt sich mit dem Ziel der Regelung der Maschine im Automatik-Modus erklären. Um hier die notwendige Echtzeitfähigkeit sicherzustellen, wird der Prozessor deutlich überdimensioniert, damit die Berechnungen nicht verzögert werden können, während der Arbeitsspeicher

durch Puffer und ähnliches vollständig ausgenutzt wird, um Verzögerungen durch ein Nachladen von Daten zu verhindern.

Ebenfalls wenig ist der Prozessor des Rechners »CELOS« (< 20 %) belegt, der für die Mensch-Maschine-Schnittstelle und weitere Analytik-Applikationen verwendet wird. Der Arbeitsspeicher ist hier ebenfalls deutlich mehr ausgelastet als der Prozessor (ca. 60 %), weist aber noch Reserven auf. Bei diesem Rechner ist eine Überdimensionierung seitens des Maschinenherstellers anzunehmen, da hier auch nach dem Maschinenkauf weitere Applikationen nachgekauft werden können. Diese würden dann ebenfalls auf dem IPC installiert.

Die beiden Rechnereinheiten »Edge« und »TCC« weisen beide ebenfalls noch freie Prozessorkapazitäten auf, wenn auch weniger als die ersten beiden IPCs, dafür sind die Arbeitsspeicher der beiden IPCs wenig ausgelastet. Auf der Edge läuft eine hochfrequente Datenaufzeichnung sowie weitere -auswertungslogiken. Der TCC-Rechner zeichnet Daten eines in die Motorspindel integrieren Messsystems[11] auf und visualisiert diese. Da dieser Rechner nur eine einzige Aufgabe hat, ist hier die Auslegung auch am besten gelungen, was sich an einer durchschnittlichen Prozessorauslastung bis ca. 80 % und Arbeitsspeicherauslastung um 40 % zeigt.

Aus den Messergebnissen an der ersten WZM lässt sich vermuten, dass bei geeigneter Verteilung der Rechenlasten mindestens einer der Rechner eingespart werden könnte. Diese Aussage beruht auf der Annahme, dass die Rechner alle ungefähr die gleichen Fähigkeiten und Rechenkapazitäten aufweisen. Diese Annahme ist in der Realität nicht haltbar und daher werden in Kapitel 7 weitere Informationen zur Modellierung des Rechnernetzwerkes herangezogen.

6.3 Typische Lastprofile von Tasks

Nachdem im vorherigen Unterkapitel gezeigt wurde, dass die meisten Rechner an und um die WZMs nicht vollständig ausgelastet sind, und damit ungenutzte Rechenressourcen vorliegen, sollen im Folgenden die Prozesse auf diesen Rechnern analysiert werden.

Ein typisches Verhalten der CPU zeigt der Prozess »Tool-Control-Center« auf dem Rechner »TCC«, das in Abbildung 6.5 dargestellt ist. Der Prozess hat eine Grundlast (ca. 30 %), aber periodisch steigt die Auslastung kurzzeitig auf 50 % an.

[11] spike_inspindle der Firma pro-micron GmbH

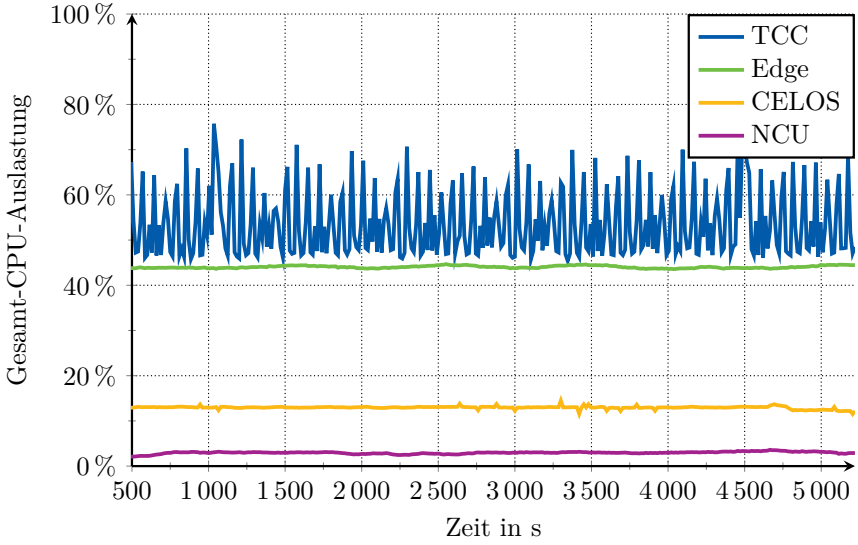


Abbildung 6.3: Prozessorauslastung der Rechner der DMC 850 V

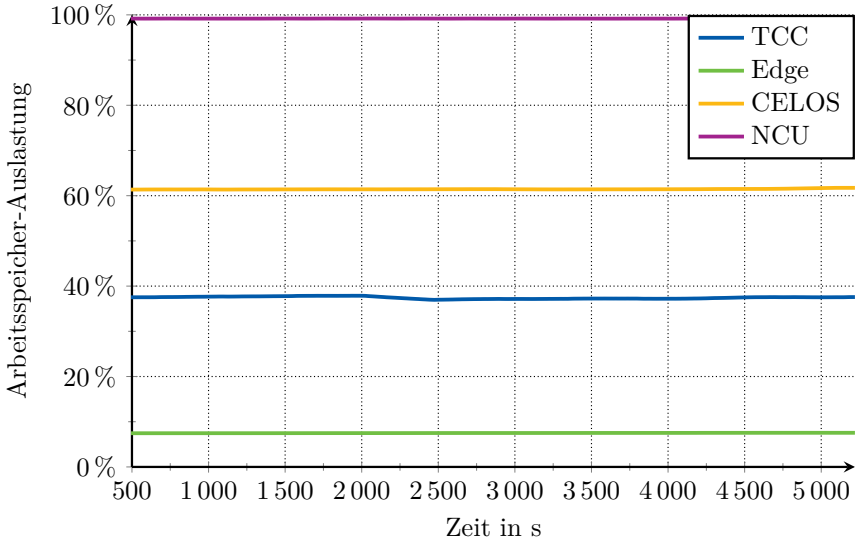


Abbildung 6.4: Arbeitsspeicherauslastung der Rechner der DMC 850 V

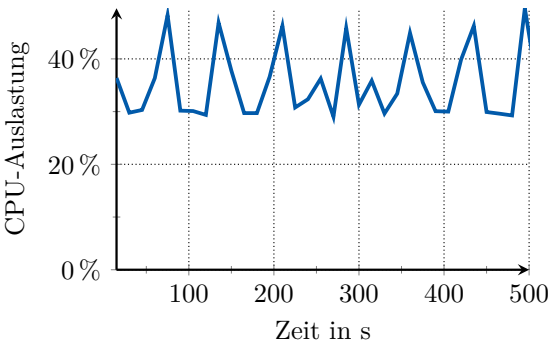


Abbildung 6.5: Prozessorauslastung durch den Tool-Control-Center-Prozess

Für die folgende Auswertung wurden einzelne Prozesse auf den Rechnern ausgewählt, die entweder im Bereich Prozessor oder Arbeitsspeicher nennenswerte Ressourcen benötigten. Die Prozesse werden im Folgenden mit Buchstaben von A bis S bezeichnet, die Zuordnung ist in Tabelle 6.1 ersichtlich. Die Tabelle enthält auch bereits die abschließende Zuordnung zu den Klassen, die noch im Verlauf des Kapitels definiert werden.

Um die Prozesse gut miteinander vergleichen zu können, wurden zunächst die aufgenommenen Zeitreihen der Prozesse normiert auf den Wertebereich von $[0, 1]$:

$$x_{\mathrm{norm}\,i\,j} = x_{i\,j} \cdot \max_i(x_{i\,j})^{-1} \;\forall i \in \,]0, m_X] \in \mathbb{N};\; j \in \{\mathrm{A}, \mathrm{B}, \ldots \mathrm{S}\} \tag{6.1}$$

Die normierten Zeitreihen wurden hinsichtlich ihrer Häufigkeitsverteilungen analysiert und dazu als Boxplots dargestellt. Für diese Darstellung werden die 25 %-, 50 %- und 75 %-Quantile berechnet, wobei das 50 %-Quantil dem Median entspricht. Zu Berechnung der Quantile werden die Werte x_i in der Zeitreihe nach ihrer Größe sortiert, womit Bedingung 6.2 erfüllt ist. Mittels Gleichung 6.3 werden dann die gewünschten Quantile q_p berechnet, wobei m_X die Gesamtzahl der Werte in der Zeitreihe X darstellt.

$$x_1 < \ldots < x_{m_X} \tag{6.2}$$

$$q_p = \begin{cases} x_{m_X \cdot p} & \text{wenn } m_X \cdot p \text{ ganzzahlig ist,} \\ \frac{1}{2}\left(x_{\lfloor m_X \cdot p \rfloor} + x_{\lceil m_X \cdot p \rceil}\right) & \text{wenn } m_X \cdot p \text{ nicht ganzzahlig ist.} \end{cases} \tag{6.3}$$

$$d_Q = q_{0,75} - q_{0,25} \tag{6.4}$$

Die Boxplot-Abbildungen zeigen:

Tabelle 6.1: Analysierte Prozesse und deren Klassifizierung

Nr.	Rechner	Prozess	Klasse	
			CPU	RAM
A	CELOS	HMI-Erweiterung	I	i
B		HMI Host	III	i
C		OPC UA Server	I	i
D		Datenbank	IIa	ii
E		Program Plugin Container	IIb	i
F		Dienst-Host	I	i
G	Edge	Maschinendatenverarbeitung	III	i
H		Maschinendatenquelle	I	i
I		Reverse Proxy	I	i
J		Log Uploader	I	i
K		HMI Host	I	i
L	TCC	SpikeDataHub	I	i
M		Tool-Control-Center	IIa	iii
N		Dienst-Host	IIb	iii
O		VNC Host	I	i
P	NCU	cp_710	IIa	i
Q		NC-Kern	III	i
R		Echtzeit-Netzwerk-Paketempfang	IIa	—
S		Datenbank-Client	I	i

- das 25 %- und 75 %-Quantil als untere bzw. obere Grenze des Kastens,
- den Median als horizontale Linie innerhalb des Kastens,
- Antennen, die maximal bis zum 1,5-fachen des Interquartilsabstandes von der oberen bzw. unteren Grenze des Kastens reichen ($1{,}5 \cdot d_Q$),
- Ausreißer als Kreise.

Anhand der Boxplots können unterschiedliche Gruppen von Prozessen identifiziert werden, die sich jeweils durch ähnliche Boxplots darstellen lassen. Boxplots, bei denen das 25 %-Quantil oberhalb von 0,8 liegt, sind in Abbildung 6.6 zusammengefasst. Analog zeigt Abbildung 6.7 Verteilungen mit dem 75 %-Quantil unter 0,4.

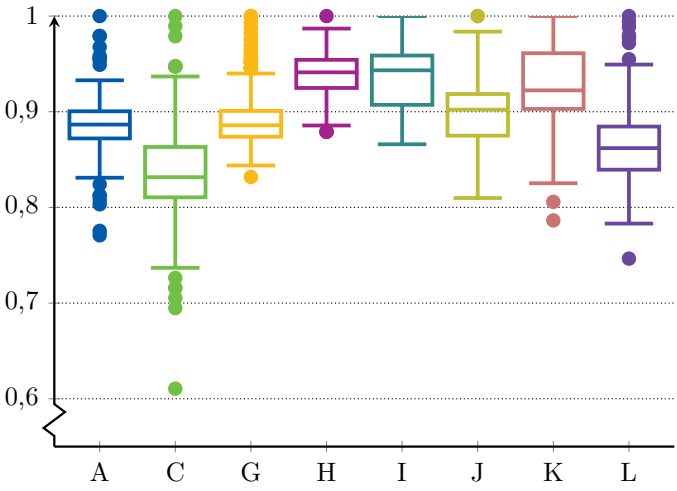


Abbildung 6.6: Normierte Verteilungen von CPU-Auslastungen mit Häufung nahe dem Maximum

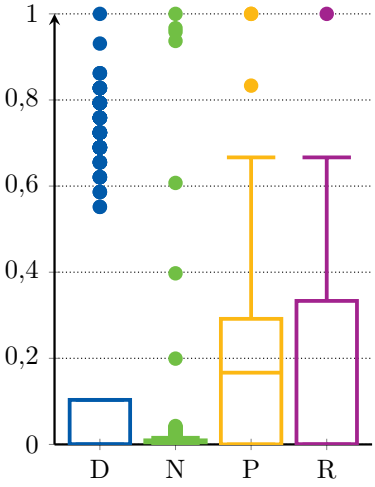


Abbildung 6.7: Verteilung von CPU-Auslastungen mit Häufung nahe Null und Ausreißern

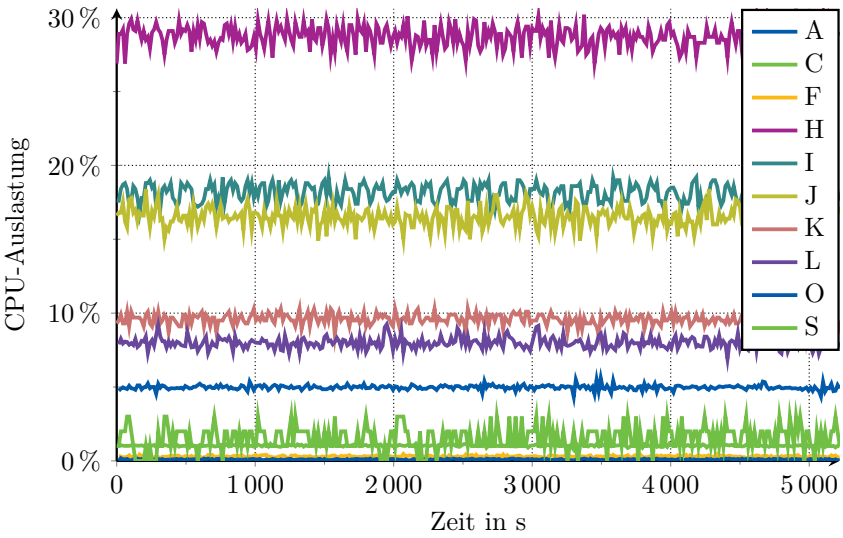


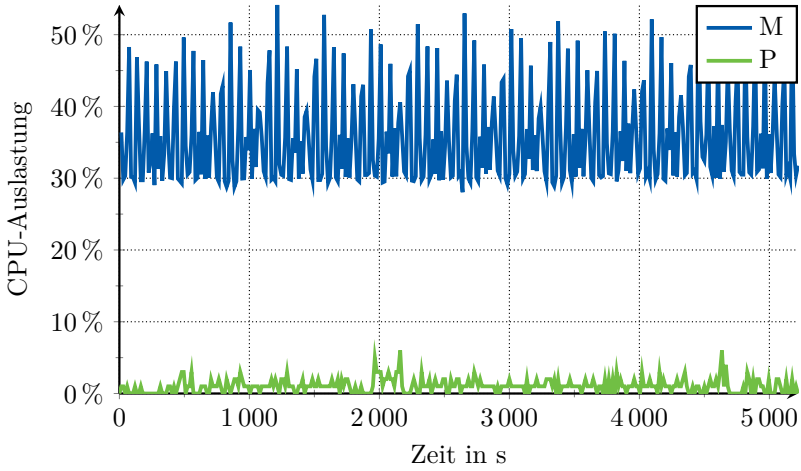
Abbildung 6.8: Prozessorauslastung von Klasse-I-Prozessen

6.3.1 Prozessor-Klassen

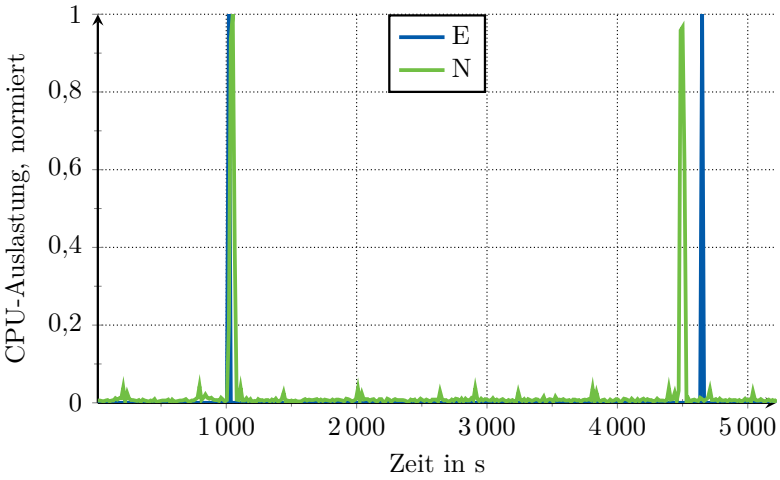
Im Folgenden werden die erkannten Klassen beschrieben.

Klasse I – Konstante Last enthält Prozesse, deren Prozessorauslastung sich im zeitlichen Verlauf nur wenig ändert. Diese Klasse lässt sich durch zwei Größen beschreiben. Zunächst ist dies der Mittelwert μ_u, sowie die Varianz oder Standardabweichung σ_u. Für die Modellierung bzw. die Taskzuweisung lässt sich dies auf einen Wert reduzieren, nämlich die obere Grenze der Auslastung. Viele der untersuchten Prozesse sind in Klasse I einzuordnen und in Abbildung 6.8 dargestellt.

Klasse II – Lastspitzen enthält Prozesse die basierend auf dem typischen Prozess aus Klasse I zusätzlich kurzzeitig erhöhte Auslastungen (»Lastspitzen«) aufweisen. Diese Prozesse fallen im Boxplot dadurch auf, dass die meisten Werte im unteren – normierten – Bereich liegen, darüber allerdings viele Ausreißer auftreten. Dies sind die Boxplots in Abbildung 6.7. Zeitserien von Prozessen der Klasse II sind in Abbildung 6.9 zu sehen.



(a) Klasse IIa



(b) Klasse IIb

Abbildung 6.9: Prozessorauslastungen von Klasse-IIa- und Klasse-IIb-Prozessen

Tabelle 6.2: Charakteristische Größen der CPU-Klassen

I	IIa	IIb	III
μ_u	μ_u	$\mu_u = 0$	$\tau_1, \tau_2, \ldots \tau_n$
σ_u	σ_u	σ_u	$u_1, u_2, \ldots u_n$
	a_u	a_u	
	p_u	p_u	

Zur Beschreibung dieser Auslastungsprofile sind mehrere Größen notwendig. Zunächst kann die Grundlast beschrieben werden wie auch bei Klasse I. Zusätzlich sind hier allerdings Informationen über die Lastspitzen notwendig. Mindestens muss die Spitzenlast sowie die zeitliche Häufigkeit charakterisiert werden. Bei der zeitlichen Häufigkeit stellt sich wiederum die Frage, ob die Lastspitzen eher periodisch oder stochastisch auftreten.

Von der Klasse IIa wurde die Klasse IIb abgeteilt, bei der Lastspitzen auftreten, die Grundlast allerdings nicht vorhanden ist (Mittelwert Grundlast ist Null).

Klasse III – Mehrplateaulasten sind Prozesse, die keine Lastspitzen aufweisen, trotzdem aber nicht konstant in ihrer Auslastung sind. Die Auslastung lässt sich vielmehr als abschnittsweise konstant beschreiben. Zur Charakterisierung dieser Prozesse sind die Zeitverhältnisse und die konstanten Höhen der Abschnitte notwendig.

$$u(\tau) = u_i \ \forall \tau \in \,]\tau_{i-1}, \tau_i] \tag{6.5}$$

$$u(\tau) = \begin{cases} u_1, \ \text{für } \tau \in \,]0, \tau_1] \\ u_2, \ \text{für } \tau \in \,]\tau_1, \tau_2] \\ \quad \ldots \end{cases} \tag{6.6}$$

In Tabelle 6.2 sind die notwendigen Größen zur Beschreibung der verschiedenen CPU-Klassen zusammengefasst. μ_u und σ_u bezeichnen dabei den Mittelwert und die Standardabweichung der Auslastung. Für die Klasse II werden zusätzlich die Amplitude a_u der Lastspitzen sowie deren Periode p_u benötigt. τ_n und u_n beschreiben die verschiedenen Plateaus der Klasse III.

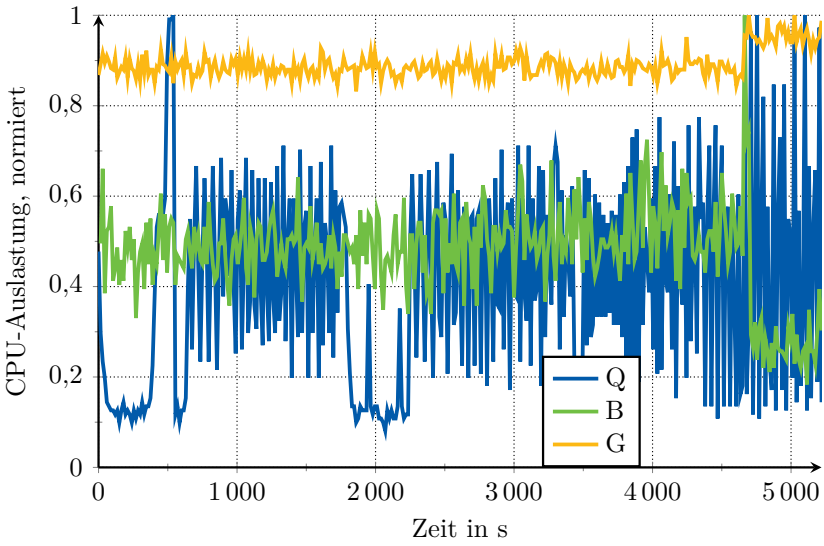


Abbildung 6.10: Normalisierte Prozessorauslastung von Klasse-III-Prozessen

6.3.2 Arbeitsspeicher-Klassen

Analog zum Vorgehen bei der Klassifizierung der Prozessorauslastungen, kommt auch bei den Arbeitsspeichermessreihen das Instrument Boxplot als Darstellung der Häufigkeitsverteilungen zum Einsatz. Allerdings wird hier die Normierung zwischen Minimum und Maximum der jeweiligen Zeitreihe ebenfalls auf den Wertebereich $[0, 1]$ durchgeführt:

$$x_{\text{norm}\,ij} = \left[x_{ij} - \min_i(x_{ij})\right] \cdot \max_i(x_{ij})^{-1} \qquad \forall i \in \,]0, m_X] \in \mathbb{N};\ j \in \{\text{A}, \text{B}, \ldots \text{S}\} \tag{6.7}$$

Die Normierung unter Einbezug des Minimums kommt zum Einsatz, da – anders als beim Prozessor – der Wertebereich beim Arbeitsspeicher deutlich größer ist, und der Wert Null – also keine Arbeitsspeichernutzung – technisch nicht vorkommen kann.

Auch hier wurden anhand der Boxplots ähnliche Prozessverhalten zusammengestellt und daraus Klassen abgeleitet. In Abbildung 6.11 sind ausgewählte Boxplots der normierten Zeitreihen dargestellt. Die Prozesse A, B und C haben gemein, dass die Arbeitsspeicherverwendung größtenteils einen konstanten

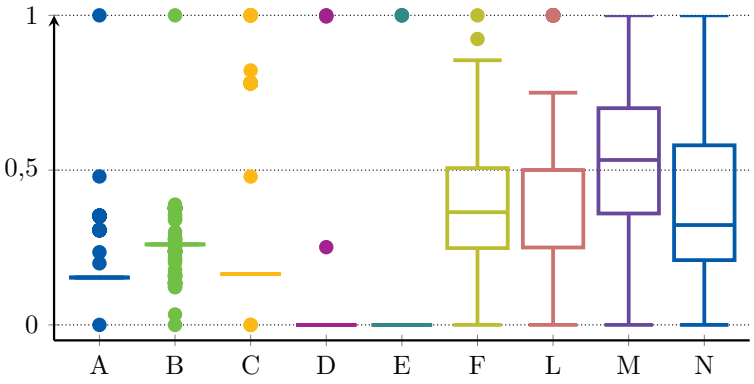
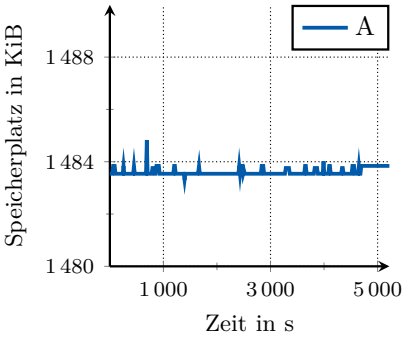


Abbildung 6.11: Normalisierte Verteilungen der Arbeitsspeicherverwendung

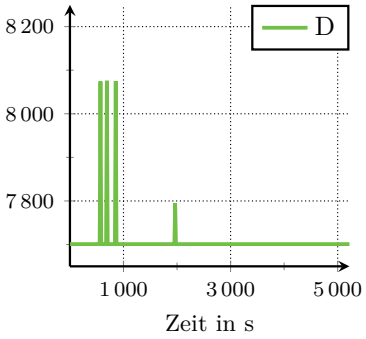
Wert hat, da die Kästen lediglich als Strich dargestellt werden. Entsprechend sind mind. 50 % der Zeitpunkte mit diesem mittleren Wert zu erwarten. Da ebenfalls die Antennen nicht dargestellt sind, ist der Anteil der Zeitpunkte mit diesem Wert noch größer. Zusätzlich zeigen sich bei allen drei Prozessen allerdings einige Ausreißer unter- und oberhalb des Kastens. Die Prozesse mit diesem Erscheinungsbild werden in **Klasse i – Konstant** zusammengefasst, eine beispielhafte Zeitreihe von Prozess A ist in Abbildung 6.12(a) zu sehen. Der größte Teil der untersuchten Prozesse fällt in diese Klasse, d. h. der belegte Arbeitsspeicher ist konstant oder nur mit etwas Rauschen überlagert.

Ein anderes Erscheinungsbild haben – allerdings ebenfalls mit Ausprägung des Kastens als Strich – die Prozesse D und E. Hier lässt die Häufigkeitsverteilung vermuten, dass die Werte typischerweise an einem Minimum sind und lediglich wenige Ausreißer nach oben hin existieren. Dies deutet auf ein Verhalten wie in der CPU-Klasse II – also eine Grundlast mit wenigen Lastspitzen – hin. Dies bestätigt sich bei Betrachtung der Zeitreihen für den Prozess D, dargestellt in Abbildung 6.12(b). Es handelt sich ebenfalls um Lastspitzen, die allerdings anders als beim Prozessor nicht mit Grundlast Null auftreten. Hieraus wird die **Klasse ii – Lastspitzen** gebildet. Der Prozess E ist der Klasse i zuzuordnen.

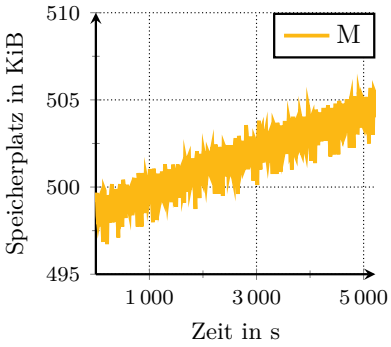
Als dritte Gruppe lassen sich die Boxplots der Prozesse F, L, M und N identifizieren. Diese scheinen – innerhalb ihres Wertebereichs – eine relativ breite Verteilung ohne oder mit nur wenigen Ausreißern zu haben. F und L weisen allerdings ein konstantes Verhalten auf, das anders als bei den bisherigen



(a) Klasse i – Konstant



(b) Klasse ii – Lastspitzen



(c) Klasse iii – Linear

Abbildung 6.12: Arbeitsspeicherverwendung der Prozesse

Prozessen allerdings verrauscht ist. Bei Betrachtung der Zeitreihen der Prozesse M und N zeigt sich ein lineares Verhalten – siehe Abbildung 6.12(c) – was sich bspw. durch kontinuierlich akkumulierte Daten erklären lässt. Diese Prozesse bilden die **Klasse iii – Linear**.

Analog zu den CPU-Klassen sind in Tabelle 6.3 die charakteristischen Größen der Arbeitsspeicher-Klassen dargestellt. Die Klassen i und ii sind analog zu den CPU-Klassen, bei der Klasse iii wird der initiale Wert u_0 und die Steigung der Auslastung über die Zeit $\frac{du}{d\tau}$ genutzt.

Tabelle 6.3: Charakteristische Größen der Arbeitsspeicher-Klassen

i	ii	iii
μ_u	μ_u	u_0
σ_u	σ_u	$\frac{du}{d\tau}$
	a_u	
	p_u	

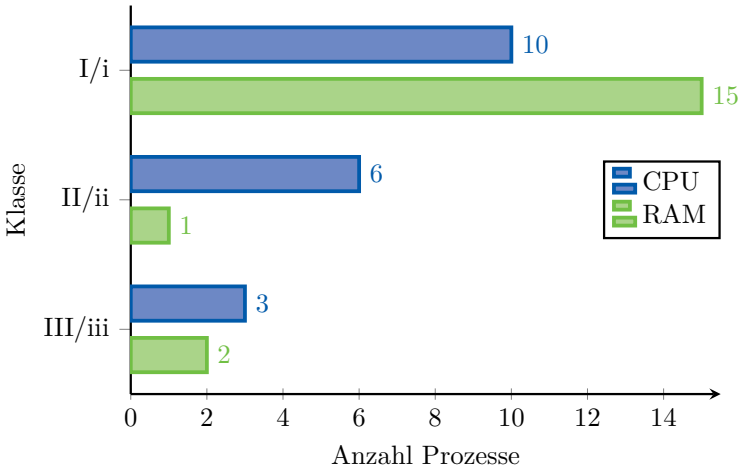


Abbildung 6.13: Anzahl der Prozesse in den zugewiesenen Klassen

6.4 Fazit

Mit dem in diesem Kapitel vorgestellten Messsystem konnte gezeigt werden, dass die Rechner in einem exemplarischen Produktionsumfeld im Betrieb nicht annähernd ausgelastet sind, und damit kein ressourceneffizienter Zustand besteht. Zusätzlich konnten aus der Analyse der aufgezeichneten Prozesse Klassen ähnlicher Prozesse abgeleitet werden. Für den Prozess R konnte allerdings durch die indirekte Messung über das »top«-Systemwerkzeug keine Arbeitsspeicherverwendung gemessen werden. Abgesehen von diesem Prozess lassen sich aber alle Prozesse einer der CPU- und RAM-Klassen zuordnen.

In Abbildung 6.13 sind die abgeleiteten Klassen mit der Anzahl der jeweils zugeordneten Prozesse dargestellt. Für beide Rechnerressourcen sind die

meisten Prozesse in der Klasse i einzuordnen. Beim Arbeitsspeicher sind dies 76 %, während 53 % der Prozesse den Prozessor konstant belasten.

Allerdings bestehen die Arbeitsspeicherklassen ii und iii jeweils nur aus einem bzw. zwei Prozessen. Hier ist eine Allgemeingültigkeit also nicht anzunehmen, auch wenn die Zeitverläufe plausibel erscheinen. Weitere Untersuchungen mit einer größeren Datenbasis sind hier zukünftig notwendig.

Die Notwendigkeit weiterer Forschung wird durch die Beschränkung auf zwei vermessene WZMs zusätzlich gesteigert. Da einige der untersuchten Prozesse repräsentativ für Maschinensteuerungen sind, wird die Verwendbarkeit der Ergebnisse für diese Arbeit trotzdem angenommen: Prozesse wie NC-Kern oder Mensch-Maschine-Schnittstelle sind bei jeder Maschinensteuerung anzutreffen und dürften ein ähnliches Verhalten hinsichtlich der Rechenressourcen aufweisen.

7 Modellierung von Rechnerverbünden

Im folgenden Kapitel wird zu Beginn beschrieben, wie die verschiedenen Modelle für die Tasks sowie die Rechnercluster aufgebaut werden. Die beiden Modelle werden für das Forschungslabor TEC-Lab des PTW konkretisiert und anschließend für die Softwareverteilung ein Optimierungsproblem formuliert. Abschließend wird der gewählte Ansatz zur Lösung des Optimierungsproblems vorgestellt.

7.1 Modellierung von abhängigen Tasks

Sowohl abhängige als auch unabhängige Tasks lassen sich als Graphen darstellen, wie in Abschnitt 2.5.3 bereits vorgestellt. Im Falle von unabhängigen Tasks kommen im Graphen mehrere unabhängige Knoten vor, die jeweils einzeln im Cluster verteilt werden sollen. Damit sind unabhängige Tasks ein Spezialfall, der durch die Modellierung abhängiger Tasks mitbetrachtet wird.

Angelehnt an die Notationen von [Sing17] und [Zhou20] wird das Task-Modell als Graph W definiert, wobei T als Menge der Knoten und D als Menge der Kanten im Graph aufgefasst wird:

$$W(T, D) \tag{7.1}$$

T ist die Menge der zu verteilenden Subtasks t_i. Als m_T wird die Anzahl der Subtasks bezeichnet.

$$T = \{t_1, t_2, \dots t_{m_T}\} \tag{7.2}$$

$$D = \{(t_i, t_j, d_{i\,j})\} \quad i, j \in (x \in \mathbb{N} \mid x \leq m_T) \tag{7.3}$$

Da jeweils mehr als 50 % der analysierten Tasks der konstanten Klasse zugeordnet werden konnten, wird für diese Arbeit vereinfachend angenommen, dass nur solche Tasks zu verteilen sind. Dies bedeutet, dass sämtliche Funktionen von der Zeit unabhängig sind:

$$f(\cdot) \neq f(\tau) \tag{7.4}$$

Zukünftige Arbeiten sollten sich der Aufgabe widmen, die zeitlich veränderlichen Ressourcenbedarfe der weiteren Klassen zu integrieren.

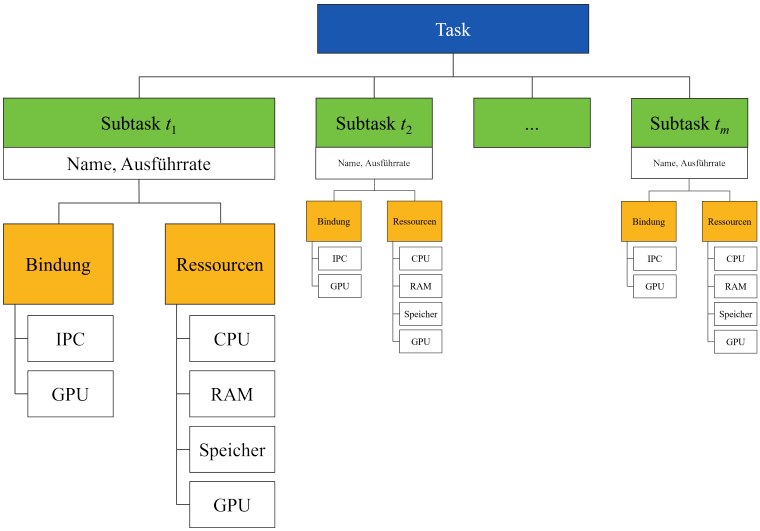


Abbildung 7.1: Modellierung der Subtasks

7.1.1 Tasks

Jeder Subtask t_i wird als Zuordnungstabelle[12] mit Attributen modelliert, wie in Abbildung 7.1 dargestellt ist. Der Name dient der Zuordnung für die Auswertungen und wird als alphanumerischer Index für den Knoten im Graphen genutzt. Die Ausführrate beschreibt die Anzahl an Ausführungen des Subtasks pro Sekunde.

Zunächst werden zwei Hardwareanforderungen kodiert (»Bindung« in Abbildung 7.1). Die erste ermöglicht es, für jeden Subtask eine Bindung an einen spezifischen IPC im Cluster vorzugeben. Dies kann notwendig sein, um Zugriff auf angeschlossene Hardware zu erhalten die nicht netzwerkfähig ist. Typische Beispiele hierfür sind der Zugriff auf Geräte über einen klassischen Feldbus oder *Universal Serial Bus* (USB). Die zweite Hardwareanforderung ist die Notwendigkeit einer Grafikkarte, was bspw. für Anwendungen der industriellen Bildverarbeitung notwendig sein kann. Eine Bindung an einen konkreten IPC ist hier nicht notwendig, das Vorhandensein eines Grafikprozessors ist ausreichend. Diese Anforderung kann als Vorlage für zukünftige Erweiterungen, wie bspw. die Notwendigkeit von ML-Coprozessoren, genutzt werden.

[12] Zuordnungstabellen sind häufig auch unter den Namen *Map* oder *Dictionary* bekannt.

Als weitere Gruppe werden dann die Ressourcenbedarfe (»Ressourcen« in Abbildung 7.1) des Subtasks notiert. Beginnend mit dem Prozessor, wird die Anzahl der Gleitkommazahlrechnungen pro Ausführung in das Modell aufgenommen. Dieser Wert kann entweder über *Profiling* – also eine Code-Analyse – im Vorfeld oder durch Beobachtung zur Laufzeit ermittelt werden. Die Gleitkommazahlrechnungen wurden gewählt, da diese in den typischen Datenverarbeitungsketten häufig vorkommen. Darüber hinaus sind hardwarenähere Größen wie Instruktionen oder Taktzyklen schwieriger zu ermitteln und bieten zudem nur wenig Genauigkeitsvorteil.

Nach dem Prozessor ist der nächste Wert der belegte Arbeitsspeicher während der Laufzeit.

Mit Speicher ist der belegte Massenspeicher gemeint, hier insbesondere der notwendige Platz, um die Anwendung vor der Ausführung dort zu platzieren. Abschließend ist noch ein Kennwert für die Grafikkarte vorgesehen, wobei diese vereinfachend wie ein weiterer Prozessor modelliert wird (siehe Abschnitt 7.2.1).

Neben den beschriebenen Subtasks gibt es noch zwei Spezialformen: Den Quellen- und den Senken-Knoten. Beide zeichnen sich dadurch aus, dass sie eine Hardwarebindung haben und keine Ressourcen auf diesem IPC haben. Diese beiden Konstruktionen sind notwendig, um Datenflüsse von (Quelle) oder nach (Senke) außerhalb des betrachteten Clusters zu modellieren.

7.1.2 Abhängigkeiten

Innerhalb der Tasks gibt es zwei Arten von Abhängigkeiten. Zunächst und hauptsächlich sind dies die Datenflüsse zwischen den Subtasks. Die zweite Art sind Latenzanforderungen und es gilt:

$$D_{\text{Daten}} = \{x \in D \mid \text{type}(x) = \text{Data}\} \tag{7.5}$$

$$D_{\text{Latenzen}} = \{x \in D \mid \text{type}(x) = \text{Latency}\} \tag{7.6}$$

$$D = D_{\text{Daten}} \cup D_{\text{Latenzen}} \tag{7.7}$$

$$|D_{\text{Latenzen}}| \leq |D_{\text{Daten}}| \tag{7.8}$$

Datenflüsse

Wie auch bereits bei den Subtasks, werden den Kanten Informationen in Form einer Zuordnungstabelle angehängt. Damit können die Datenflüsse mit Attributen versehen werden. Hier sind diese allerdings deutlich weniger umfangreich als bei den Subtasks:

1. Transportprotokoll
2. Datenvolumen

Zusätzlich sind in der Repräsentation als Tripel $(t_i, t_j, d_{i\,j})$ noch der Ausgangs- und Zielknoten enthalten. Zur Unterscheidung wird der Kante zusätzlich die Typinformation »Data« in die Zuordnungstabelle geschrieben.

Als Transportprotokollstapel sind aktuell nur »TCP/IP« und »UDP/IP« vorgesehen. Zukünftige Erweiterungen z. B. für »QUIC« sind einfach umsetzbar, allerdings für die bisher zu verteilenden Anwendungen nicht notwendig. Das Datenvolumen beschreibt die Menge an zu übertragenden Nutzdaten pro Ausführung des Subtasks.

Latenzen

Neben den Datenflüssen werden auch die Latenzanforderungen im Graphen als Kanten hinterlegt. Die Kante erhält die Typinformation »Latency« sowie die Höhe der Latenzanforderung in Millisekunden und verknüpft einen Ausgangs-Subtask mit einem Ziel-Subtask. Die Latenzanforderung gilt vom Start der Ausführung des Ausgangs-Subtask über Datentransport bis zum Abschluss der Ausführung am Ziel-Subtask.

7.1.3 Serialisierung

Die gesamte Struktur der Tasks wird in *JavaScript Object Notation* (JSON) angelegt und kann von der entwickelten Software eingelesen sowie in die Graph-Repräsentation umgewandelt werden. Dieses Vorgehen wurde gewählt, da so eine Modifikation mittels eines einfachen Texteditors möglich ist. Ein beispielhafter Task mit zwei Subtasks, die wiederum von einem Quellen-Knoten außerhalb des Tasks abhängen, ist in Code 7.1 dargestellt.

Beim Einlesen wird für die Latenzanforderungen berechnet, welche die Ausgangs- und Ziel-Subtasks sind. Außerdem ist noch die ursprüngliche Platzierung auf einem IPC als »originalHardwareBind« aufgeführt.

7.2 Modellierung des Rechnerclusters

Auch das Modell der Rechnercluster wird als Graph aufgebaut. Diese Form bietet sich hier an, da die Netzwerk-Topologie zwischen den einzelnen IPCs direkt in einen Graphen übersetzt werden kann.

Code 7.1: JSON-serialisierter Task mit zwei Subtasks

```
{
    "LatencyRequirement": 200,
    "SubTasks": [
        {
            "TaskName": "DMC850V-OPCUA",
            "TransportLayer": "TCP",
            "originalHardwareBind": "CELOS",
            "GPU": false,
            "CPU": 192.5,
            "RAM": 23,
            "DiskSpace": 100,
            "Predecessors": [],
            "SourceNodes": [
                {
                    "Name": "NCK-DMC850",
                    "Data": 1.7,
                }
            ],
            "SinkNodes": []
        },
        {
            "TaskName": "CELOS-Erweiterung",
            "TransportLayer": "TCP",
            "originalHardwareBind": "CELOS",
            "GPU": false,
            "CPU": 673.75,
            "RAM": 240,
            "DiskSpace": 850,
            "Predecessors": [],
            "SourceNodes": [
                {
                    "Name": "NCK-DMC850",
                    "Data": 0.65,
                }
            ],
            "SinkNodes": []
        }
    ]
}
```

Für das Cluster wird der Graph C definiert.

$$C(N, L)$$

Die Menge der Knoten N enthält zunächst alle IPCs, aber auch Netzwerkgeräte wie Switches oder Accesspoints. Die Anzahl an Geräten in N wird mit m_N notiert. Die Menge der Kanten L beschreibt die Netzwerkverbindungen zwischen den Geräten, unabhängig ob es sich um kabelgebundene oder kabellose Verbindungen handelt.

$$N = \{n_1, n_2, \dots n_{m_N}\} \tag{7.9}$$

$$L = \{(n_\alpha, n_\beta, l_{\alpha\,\beta})\} \quad \alpha, \beta \in (x \in \mathbb{N} \mid x \leq m_N) \tag{7.10}$$

Zur Verbesserung der Übersichtlichkeit werden für das Cluster-Modell im Folgenden nur griechische Zählvariablen genutzt, während die lateinischen dem Task-Modell vorbehalten bleiben.

7.2.1 Knoten

Die Knoten bekommen in der jeweiligen Zuordnungstabelle n_α als erstes eine Typinformation gesetzt, um die im Folgenden beschriebenen Arten von Geräten im Cluster zu unterscheiden. Die Arten mit ihren Attributen sind in Abbildung 7.2 dargestellt.

Computer

Die wichtigste Klasse im Cluster sind die IPCs bzw. Computing-Ressourcen, die im Folgenden in der Menge R zusammengefasst werden.

$$R \subseteq N \tag{7.11}$$

Die Computer erhalten neben einem Namen zur Identifizierung Attribute, die ihre verfügbaren Ressourcen beschreiben:

1. Prozessor: Leistungsfähigkeit, (Anzahl der Kerne,) Auslastung im Grundzustand
2. Grafikkarte: Leistungsfähigkeit, Auslastung im Grundzustand
3. Arbeitsspeicher: Größe, Belegung im Grundzustand
4. Massenspeicher: Größe, Belegung im Grundzustand

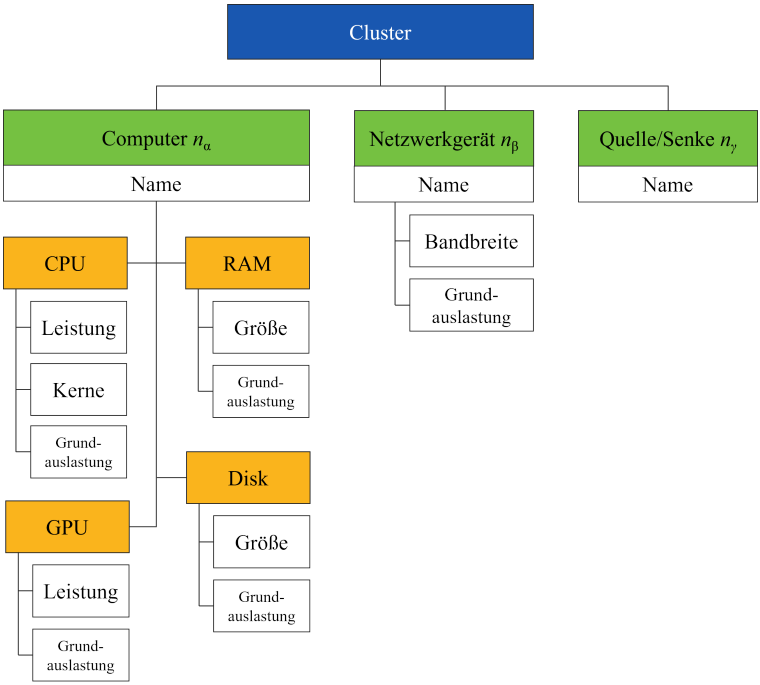


Abbildung 7.2: Modellierung des Clusters

Für die Leistungsfähigkeit des Prozessors werden öffentlich verfügbare Benchmarks herangezogen. Da in den Datenverarbeitungsketten häufig Gleitkommazahlberechnungen vorkommen, wird die Leistungsfähigkeit vereinfachend als der Benchmark-Wert für Gleitkommazahlberechnungen pro Sekunde (engl. *Floating Point Operations Per Second (FLOPS)*) verstanden. Die Anzahl der Kerne ist zwar im Modell kodiert, fließt aber bereits in den FLOPS-Wert ein, sodass diese nicht weiter in den Berechnungen verwendet wird. Die Auslastung im Grundzustand ist der Anteil der Prozessorleistungsfähigkeit, der bei einem System ohne Tasks im Leerlauf in Benutzung ist. Diese Grundlast kommt bspw. vom Betriebssystem oder auch von der Virtualisierungssoftware sowie der Orchestrierungssoftware für die Softwareverteilung.

Die Grafikkarte wird analog zum Prozessor modelliert. Auch sie erhält eine Kennzahl für die Leistungsfähigkeit und die Auslastung im Grundzustand.

Code 7.2: JSON-serialisierter Switch

```
{
    "Name": "Switch_2",
    "Bandwidth": 16000,
    "usage": 0.0
}
```

Der Arbeitsspeicher und der Massenspeicher werden gleichartig ins Modell aufgenommen. Beide erhalten als primäre Kennzahl die verbaute Speichergröße. Zusätzlich wird auch hier die Belegung im Grundzustand (s. o.) aufgezeichnet. Weitere Größen werden für beide Speicherarten aus Gründen der Einfachheit nicht aufgenommen, allerdings könnten beispielsweise die Schreib- und Lesedatenraten mit in das Modell aufgenommen werden, um in die Zeitberechnungen mit einzufließen.

Zu jeder der Leistungsfähigkeiten und Speichergrößen wird jeweils eine aktuelle Auslastung ins Modell aufgenommen. Diese werden initialisiert mit den Werten für den Grundzustand.

Netzwerkgeräte

Die Netzwerkgeräte in der Menge der Knoten N haben nur wenige Attribute. Code 7.2 zeigt die JSON-serialisierte Repräsentation eines Switches. Dabei ist zu beachten, dass bspw. bei einem Switch – analog aber auch Accesspoints u. ä. – die Bandbreite nicht mit der anzuschließenden Ethernet-Leitungen (bspw. $100\,\mathrm{Mbit\,s^{-1}}$) verwechselt wird. Vielmehr ist die interne Verarbeitungsgeschwindigkeit – häufig als *switching capacity* bezeichnet – gemeint.

Quellen & Senken

Um Datenverkehr von oder nach außerhalb des Clusters zu modellieren, werden zwei Spezialtypen eines Computers eingeführt: Quelle und Senke. Diese enthalten nur einen Namen und über die Verbindungen eine Position in der Netzwerktopologie, ansonsten aber keine weiteren Attribute. Wenn man im Cluster alle Geräte eines Unternehmens modelliert, korrespondiert die Quelle und Senke mit dem Internetanschluss (*uplink*) des Unternehmens.

7.2.2 Verbindungen

Die Verbindungen bekommen in der jeweiligen Zuordnungstabelle $l_{\alpha\,\beta}$ analog zu den Abhängigkeiten im Task-Modell Attribute, welche die Art der Verbindung beschreiben. Das Verbindungstripel $(n_\alpha, n_\beta, l_{\alpha\,\beta})$ beschreibt dann zusätzlich noch von welchem Knoten zu welchem Knoten die Verbindung aufgebaut ist.

Die Verbindungen bekommen als erstes Attribut die Bezeichnung der Netzzugangsschicht z. B. »802.3« für Ethernet oder »802.11« für WLAN. Neben den von den Netzwerkgeräten bereits bekannten Attributen Bandbreite und Auslastung kommt für die Verbindungen noch eine Paketverlustrate hinzu:

1. Netzzugang
2. Bandbreite
3. Auslastung
4. Paketverlustrate

7.2.3 Funktionen

Das Cluster-Modell enthält eine Funktion, um die Netzwerk-Route von einem Knoten n_α zu einem anderen Knoten n_β zu berechnen.

Hierfür ist bei den betrachteten Netzwerkgrößen der Dijkstra-Algorithmus geeignet. Bei größeren Clustern sind effizientere Ansätze erforderlich, sodass der Algorithmus gegen einen anderen ausgetauscht werden muss. [Noto00]

Für den Suchalgorithmus wird im Cluster-Modell der Reziprokwert der verbleibenden Bandbreite als jeweiliges Kantengewicht genutzt:

$$w_{\alpha\,\beta} = \frac{1}{(1 - l_{\alpha\,\beta\,\text{Auslastung}}) \cdot l_{\alpha\,\beta\,\text{Bandbreite}}} \tag{7.12}$$

Die Funktionsweise des Dijkstra-Algorithmus kann der Literatur, bspw. [Noto00, S. 2316], entnommen werden.

7.2.4 Serialisierung

Wie auch das Task-Modell wird das Cluster-Modell in JSON-serialisierter Form gespeichert. Code 7.2, 7.3 und 7.4 enthalten die verschiedenen JSON-Serialisierungen für die Cluster-Modell-Typen.

Code 7.3: Computer-Liste des Cluster-Modells

```
"Computers": [
    {
        "Name": "CELOS",
        "CPU_perf": 9625,
        "CPU_cores": 4,
        "CPU_usage": 0.11,
        "RAM": 8000,
        "RAM_usage": 0.5525,
        "disk": 50000,
        "disk_usage": 0.84,
        "GPU_perf": 260000,
        "GPU_usage": 0.05
    },
    ...
]
```

Code 7.4: Verbindungsliste des Cluster-Modells

```
"Connections": [
    {
        "FromName": "CELOS",
        "ToName": "Switch_1",
        "LinkLayer": "802.3",
        "Bandwidth": 100,
        "usage": 0.0,
        "PacketLossProbability": 0.01
    },
    ...
]
```

7.3 Bestimmung der optimalen Softwareverteilung

7.3.1 Formulierung als Optimierungsproblem

Die Softwareverteilung muss alle modellierten Tasks mit den entsprechenden Subtasks auf dem ebenfalls modellierten Cluster verteilen. Das Ziel der Verteilung ist es die Ressourceneffizienz zu verbessern oder sogar optimal zu gestalten.

Angelehnt an den Ansatz der *Server Consolidation* soll hierzu eine möglichst hohe Auslastung der Rechnerressourcen im Cluster erreicht werden. Die elektrische Leistungsaufnahme P_{el} von Computern lässt sich nach [Belo12, S. 759] gemäß Gleichung 7.13 beschreiben, wobei für typische Serverhardware der relative Leerlaufverbrauch mit $k \approx 0{,}7$ anzunehmen ist. $P_{\max}$ bezeichnet die maximale Leistungsaufnahme und u_{cpu} die Auslastung des Prozessors.

$$P_{\text{el}} = P_{\text{el}}(u_{\text{cpu}}) = k \cdot P_{\max} + (1 - k) \cdot P_{\max} \cdot u_{\text{cpu}} \tag{7.13}$$

Für die Ressourceneffizienz ist also eine möglichst hohe Auslastung des Prozessors erstrebenswert. Es wird der Ansatz verfolgt, Rechner, die in ihrer Auslastung nicht von der Auslastung im Grundzustand abweichen, in einen Energiesparmodus zu versetzen.

> **Optimierungsziel**: Das optimal ausgenutzte Cluster enthält nur Computer deren Ressourcen voll ausgelastet – aber nicht überlastet – sind und Computer, die abgeschaltet sind.

Die Verteilungsfunktionen V bilden die Menge der Tasks T auf die Menge der Computing-Ressourcen R ab.

$$V : T \to R,\ t \mapsto r \tag{7.14}$$

Gesucht wird also eine Abbildung V, die das oben genannte Ziel erreicht.

Für die Optimierung wird eine Kostenfunktion c konstruiert, die für eine beliebige Verteilung $\vec{a}$ berechnet werden kann.

$$\vec{a} = V(T) \tag{7.15}$$

$$c(\vec{a}) = \sum_{\alpha} \sum_{\gamma} w_{\gamma} \begin{cases} |\hat{u} - u_{\alpha\,\gamma}(\vec{a})|\,; & \text{wenn } u_{\alpha\,\gamma}(\vec{a}) > u_{\min\,\alpha\,\gamma} \\ 0; & \text{wenn } u_{\alpha\,\gamma}(\vec{a}) \leq u_{\min\,\alpha\,\gamma} \end{cases} \tag{7.16}$$

$$\forall \alpha \in R$$

$$\forall \gamma \in \{\text{CPU}, \text{GPU}, \text{RAM}, \text{Disk}\}$$

$u_{\alpha\,\gamma}(\vec{a})$ bezeichnet hierbei die Auslastung der Ressource γ des Knotens α mit den zugeteilten Subtasks aus $\vec{a}$, während $u_{\min\,\alpha\,\gamma}$ die Auslastung derselben Ressource im Grundzustand darstellt. $\hat{u}$ ist der Sollwert der Einzelauslastungen und w_γ ein Gewichtungsfaktor für die verschiedenen Ressourcentypen.

Rechner, die abgeschaltet werden können, weil die Auslastung dem Grundzustand entspricht, haben keinen Beitrag zur Kostenfunktion. Genauso Rechner die den Zielwert $\hat{u}$ in allen Ressourcentypen erreichen.

Für die Implementierung werden alle vier Gewichtungsfaktoren gleich gewählt. Sollte in zukünftigen Untersuchungen festgestellt werden, dass einzelne Ressourcen mehr Einfluss auf die Ressourceneffizienz haben, kann dies angepasst werden.

$$w_\gamma = 1; \quad \forall\gamma \in \{\mathrm{CPU}, \mathrm{GPU}, \mathrm{RAM}, \mathrm{Disk}\} \tag{7.17}$$

Der Sollwert für die Einzelauslastungen wird auf 80 % festgelegt. Hierdurch verbleiben 20 % als Puffer, sodass die Ressourcen nicht überlastet werden, auch wenn einzelne Tasks kurzfristig mehr Ressourcen belegen. Dies ist notwendig, da alle Tasks vereinfachend als konstant angenommen wurden, was aber in der Praxis nicht auf alle zutrifft. Wenn das zeitliche Verhalten in zukünftigen Arbeiten integriert wird, kann der Sollwert vergrößert werden.

$$\hat{u} = 0{,}80 \tag{7.18}$$

Um das Optimierungsproblem zu lösen, muss die Kostenfunktion minimiert werden:

$$\min c(V(T)) \tag{7.19}$$

Zusätzlich müssen bei der Minimierung auch zwei Nebenbedingungen eingehalten werden. So darf keine Ressourcenauslastung 100 % erreichen bzw. überschreiten.

$$u_{\alpha\,\gamma} < 100\,\%\ \forall\alpha,\gamma \tag{7.20}$$

Die Latenz $\tau_{\max}$ aller Tasks darf nicht größer als die entsprechende Latenzanforderung sein.

$$\tau_{\max}(t_i) \leq \hat{\tau}_{\max\,i} \quad \forall i \in (x \in \mathbb{N} \mid x \leq m_T) \tag{7.21}$$

Um die Nebenbedingungen mit in die Optimierung zu integrieren, wird der Ansatz einer Straffunktion gewählt. Für Verletzungen der Nebenbedingungen wird auf die Kostenfunktion eine zusätzliche Strafe addiert:

$$p(\vec{a}) = \begin{cases} 1 \cdot 10^6; & \neg(u_{\alpha\,\gamma} < 1{,}0) & \forall\alpha,\gamma \\ 1 \cdot 10^9; & \neg(\tau_{\max}(t_i) \leq \hat{\tau}_{\max\,i}) & \forall i \in (x \in \mathbb{N} \mid x \leq m_T) \\ 0; & \text{sonst} & \end{cases} \tag{7.22}$$

Das endgültige Minimierungsproblem lässt sich damit formulieren als:

$$\min f(V(T)) = \min \left[c(V(T)) + p(V(T)) \right] \tag{7.23}$$

Sollte als Minimum ein Wert in der Größenordnung von 10^6 gefunden werden, lässt sich keine Verteilung finden, die die Ressourcen nicht überlastet. Liegt das Minimum in der Größenordnung von 10^9, kann keine Verteilung alle Latenzanforderungen erfüllen. Für sehr große Cluster müssen die Werte des Strafterms vergrößert werden, damit durch die Strafe keine ähnliche Güte wie bei akzeptablen Lösungen erzeugt werden kann. Da im Shopfloor Cluster mit einer Million Computern nicht vorkommen sollten, ist diese Limitierung aber theoretischer Natur.

7.3.2 Prüfen der Latenzanforderungen

Im Strafterm ist eine Prüfung der Latenzen enthalten:

$$(\tau_{\max}(t_i) \leq \hat{\tau}_{\max\, i})$$

Hierfür werden die im Task-Modell enthaltenen Latenzanforderungskanten D_{Latenzen} genutzt. Für jede Latenzanforderungskante wird überprüft welche Latenz die gewählte Verteilung $\vec{a}$ zur Folge hat.

Hierfür werden zunächst die Datenfluss-Pfade durch den Task-Graphen gesucht, die vom Start der Latenzkante zum Ende derselben führen. Jeder Pfad muss separat ausgerechnet werden, auch wenn sich Abschnitte davon eventuell überschneiden, da sich im Vorfeld – bspw. über die Länge des Pfades – nicht feststellen lässt, welcher die größte Latenz erzeugt.

Entlang des Datenfluss-Pfades i werden die Knoten und Kanten durchiteriert. Jeder Knoten trägt eine Berechnungszeit zur Latenz bei, während die Kanten Transportzeiten beisteuern.

$$\tau_{\max\, i} = \sum_{j=1}^{len(i)+1} \tau_{\text{Comp}\, j} + \sum_{j=1}^{len(i)} \tau_{\text{Trans}\, j \to j+1} \tag{7.24}$$

Die Gesamtlatenz zur Latenzanforderungskante ist dann das Maximum der berechneten Latenzen:

$$\tau_{\max} = \max(\tau_{\max\, i}) \tag{7.25}$$

Berechnungszeit

Die Berechnungszeit des Subtasks j platziert auf IPC α wird relativ einfach durch Division der Anzahl der Gleitkommazahlrechnungen durch die Leistungsfähigkeit (FLOPS) des Prozessors, bzw. der Grafikkarte berechnet:

$$\tau_{\mathrm{Comp}\,j} = \frac{t_{j\,\mathrm{CPU}}}{n_{\alpha\,\mathrm{CPU\text{-}Performance}}} \tag{7.26}$$

$$\tau_{\mathrm{Comp}\,j} = \frac{t_{j\,\mathrm{CPU}}}{n_{\alpha\,\mathrm{GPU\text{-}Performance}}} \tag{7.27}$$

Ob Gleichung 7.26 oder 7.27 verwendet wird, hängt vom Eintrag des Subtasks zur Grafikkartenverwendung – und entsprechend ob die Berechnungen auf einer GPU beschleunigt ausgeführt werden können – ab.

Transportzeit

Um die Transportzeit $\tau_{\mathrm{Trans}\,j\to j+1}$ der Kante von Subtask j nach $j+1$ zu berechnen muss zunächst die Route durch das Netzwerk bestimmt werden. Hierfür enthält das Cluster-Modell – wie in Abschnitt 7.2.3 beschrieben – eine Funktion. Entlang der damit gefundenen Route S durch das Netzwerk werden nun die einzelnen Transportabschnitte berechnet.

$$(n_\alpha, n_\beta, l_{\alpha\,\beta}) \in S \subseteq L \tag{7.28}$$

Für das Transportprotokoll »TCP« muss zunächst noch mit Gleichung 7.30 die Gesamtwahrscheinlichkeit für Paketverluste $P(\mathrm{PL})$ entlang der Route berechnet werden, die das Protokoll beheben muss.

$$P(\neg\mathrm{PL}, s) = 1 - s_{\mathrm{Paketverlustrate}} \tag{7.29}$$

$$P(\mathrm{PL}) = 1 - \prod_{s\in S} P(\neg\mathrm{PL}, s) \tag{7.30}$$

Mit der Paketverlustwahrscheinlichkeit kann dann die Brutto-Datenmenge v_{Brutto} berechnet werden, die übertragen werden muss.

Zunächst müssen hierfür die Protokoll-Overheads für die zu übertragende Datenmenge berechnet werden. Dies wird beispielhaft für den Protokollstapel »TCP/IP/Ethernet« vorgestellt. MSS bezeichnet die *Maximum Segment Size*, IPG die *InterPacket Gap* und h die Headergrößen der einzelnen Protokolle im

Protokollstapel. Für detailliertere Erklärungen sowie weitere Protokolle wird auf [Broc24; Gast22] verwiesen.

$$m_{\text{Pakete}} = \lceil v_{\text{Netto}} / \text{MSS} \rceil \tag{7.31}$$

$$v_{\text{Endpaket}} = v_{\text{Netto}} \bmod \text{MSS} \tag{7.32}$$

$$\begin{aligned} v = {} & (m_{\text{Pakete}} - 1) \cdot \text{MSS} + v_{\text{Endpaket}} \\ & + m_{\text{Pakete}} \cdot (h_{\text{OSI-4}} + h_{\text{OSI-3}} + h_{\text{OSI-2}} + h_{\text{OSI-1}}) \\ & + (m_{\text{Pakete}} - 1) \cdot \text{IPG} \end{aligned} \tag{7.33}$$

$$v_{\text{Brutto}} = v \cdot (1 + P(\text{PL})) \tag{7.34}$$

Abschließend wird über die Paketverlustwahrscheinlichkeit aus Gleichung 7.30 die Datenmenge erhöht (Gleichung 7.34), da bei Paketverlust von »TCP« einzelne Pakete neu versendet werden.

Um die Transportzeit zu berechnen, wird mit dieser Bruttodatenmenge nun entlang der Route für jedes Netzwerkgerät und jede Verbindung der Zeitbedarf über die Bandbreite berechnet:

$$\tau_{\text{Trans}\, j \to j+1} = \sum_{\text{Netzwerkgeräte}} \frac{v_{\text{Brutto}}}{n_{\text{Bandbreite}}} + \sum_{\text{Verbindungen}} \frac{v_{\text{Brutto}}}{l_{\text{Bandbreite}}} \tag{7.35}$$

7.3.3 Lösen des Optimierungsproblems mittels Metaheuristiken

Das Optimierungsproblem ist durch die Heterogenität der Ressourcen sowie die unterschiedlichen Charakteristiken der Subtasks NP-vollständig [Sing17, S. 2]. Damit ist eine effiziente Lösung wahrscheinlich nicht möglich. Stattdessen wird der Ansatz einer Lösung über Metaheuristiken gewählt.

Entsprechend der Literaturanalyse werden für die Lösung drei Metaheuristiken herangezogen: *Genetic Algorithm* (GA) aus der Klasse der Evolutionären Algorithmen, *Simulated Annealing* (SA) als Vertreter der physikalisch inspirierten Algorithmen und PSO als schwarm-basierte Metaheuristik.

Der Suchraum für die Lösung ist entsprechend Gleichung 7.36 definiert, in jeder der Dimensionen kann der Wert ganzzahlig von 1 bis zur Anzahl der Rechner $|R|$ gehen. Die Anzahl der Dimensionen entspricht der Anzahl der zu verteilenden Subtasks $|T|$.

$$\{x \in \mathbb{N} \mid x \leq |R|\}^{|T|} \tag{7.36}$$

Die drei Heuristiken haben jeweils Hyperparameter, die über eine Gittersuche bestimmt werden. Für alle drei Algorithmen wird auf die Implementierung in der Python-Bibliothek »MEALPY« [Thie23] zurückgegriffen.

Genetischer Algorithmus

Genetische Algorithmen basieren auf dem biologischen Prinzip der Evolution. Aus einer Population werden die Individuen mit den besten Gütewerten ausgewählt. Durch Kombination ihrer Gene werden Nachkommen erzeugt, die in der nächsten Generation die Population für den Algorithmus stellen und zusätzlich gibt es eine geringe Wahrscheinlichkeit, dass die Gene mutieren, sodass ein größerer Suchraum erschlossen wird. Für GA muss der Suchraum zunächst auf einen eindimensionalen Bitstring kodiert werden, der als Genom die Lösung enthält. Dies geschieht mittels Gray-Kodierung [Math94]. Die erste Generation wird zufällig erzeugt und über die Anzahl der Generationen entwickelt sich die Population hin zu besseren Lösungen.

Für GA ist der untersuchte Parameterraum in der Gittersuche:

$$\begin{aligned} m_{\text{Epochen}} &= \{5; 10; 15; 20; 25; 30\} \\ m_{\text{Population}} &= \{20; 30\} \\ P_{\text{Kreuzung}} &= \{65\,\%; 80\,\%; 95\,\%\} \\ P_{\text{Mutation}} &= \{1\,\%; 10\,\%; 20\,\%\} \end{aligned}$$

Mit jeder der möglichen Kombinationen wurde der GA dreimal erprobt und die jeweils beste erreichte Güte notiert. Abschließend wurden für die drei Wiederholungen der Mittelwert $\bar{f}$ ausgerechnet und danach aufsteigend sortiert. Es ergeben sich so für jede Anzahl der Epochen jeweils eine Kombination mit den besten erreichten Güten, diese und die zweitbesten Kombinationen sind in Tabelle 7.1 aufgezeichnet.

Es lässt sich erkennen, dass in allen Fällen die Populationsgröße 30 die besseren Ergebnisse erzielt hat. 1 % Mutationswahrscheinlichkeit liefert nie das beste Ergebnis, vermutlich da hier zu wenig Exploration – also Erkundung des noch nicht bekannten Raumes – stattfindet. Die beste Güte wird erreicht nach 20 Epochen mit 80 % Kreuzungswahrscheinlichkeit und 10 % Mutationswahrscheinlichkeit (grau hinterlegt). Kürzere und längere Laufzeiten – also Anzahl der Epochen – bringen keine Verbesserung.

Tabelle 7.1: Ergebnisse der Gittersuche für GA

Epochen	$m_{\text{Population}}$	P_{Kreuzung}	P_{Mutation}	$\bar{f}$
5	30	80 %	20 %	4,786
		65 %	10 %	4,803
10	30	95 %	10 %	2,779
		65 %	20 %	2,990
15	30	65 %	10 %	3,014
		80 %		3,223
20	30	80 %	10 %	1,573
		95 %	20 %	2,455
25	30	95 %	10 %	2,500
		65 %	20 %	2,610

Partikelschwarmoptimierung

Die von Kennedy und Eberhart [Kenn95] vorgestellte PSO bildet das Schwarmverhalten von Vögeln nach. Jedes Partikel stellt eine Lösung x dar, das entsprechend eine Güte f zugewiesen bekommt. Dazu hat es eine Geschwindigkeit $\dot{x}$ welche die Bewegung des Partikels durch den Suchraum bestimmt. Die Partikel kennen die aktuell beste Güte im Schwarm und haben ein Gedächtnis ihrer eigenen Lösungen sowie deren Güten. Zu Beginn wird der Schwarm mit zufällig verteilten Partikeln initialisiert. Für jede neue Generation wird die Geschwindigkeit $\dot{x}_{n+1}$ jedes Partikels dann hin zur eigenen besten Güte $f_{\text{best Partikel}}$ sowie zur besten Güte im Schwarm $f_{\text{best Schwarm}}$ angepasst:

$$\begin{aligned}\dot{x}_{n+1} = \dot{x}_n &+ w_{\text{Partikel}} \cdot \text{rand}() \cdot (x_{\text{best Partikel}} - x_n) \\ &+ w_{\text{Schwarm}} \cdot \text{rand}() \cdot (x_{\text{best Schwarm}} - x_n)\end{aligned} \tag{7.37}$$

Die Funktion rand() ergibt einen Zufallswert zwischen 0 und 1 und führt dazu, dass die Partikel sich nicht zu schnell auf eine lokales Optimum zubewegen.

Die Gittersuche wird mit folgenden Hyperparametern durchgeführt:

$$\begin{aligned} m_{\text{Epochen}} &= \{5; 10; 15; 20; 25\} \\ m_{\text{Population}} &= \{20; 30\} \\ w_{\text{Partikel}} &= \{1; 2; 3\} \\ w_{\text{Schwarm}} &= \{1; 2; 3\} \end{aligned}$$

Tabelle 7.2: Ergebnisse der Gittersuche für PSO

Epochen	$m_{\text{Population}}$	w_{Partikel}	w_{Schwarm}	$\bar{f}$
5	30	2	3	2,503
		3	3	2,741
10	30	3	2	2,283
	20	3	2	2,597
15	30	3	2	2,309
	20	2	2	2,410
20	30	1	3	2,404
	20	2	2	2,536
25	30	1	2	2,742
		2	2	2,765

Auch bei der Gittersuche wurden drei Durchführungen mit jeder möglichen Parameterkombination berechnet. Wie auch für GA sind in Tabelle 7.2 die besten Kombinationen aufgelistet und die beste markiert.

Zunächst ist ersichtlich, dass für die untersuchten Epochen die erzielten Güten wenig von der Anzahl der Epochen – also der Laufzeit – abhängen. Die Ergebnisse sind – mit wenigen Ausnahmen – besser als die GA mit entsprechender Laufzeit, auch wenn die beste GA-Kombination die beste Güte (1,573) erreicht.

Simulierte Abkühlung

Anders als die beiden bisherigen Metaheuristiken ist SA nicht schwarm- bzw. populationsbasiert sondern berechnet nur eine Lösung pro Iteration. SA ist ein Suchalgorithmus aus der Gruppe der Lokalen Suchen, der sich aus der Metallurgie abkühlender Kristalle herleitet. Ausgehend von der aktuellen Lösung – die zu Beginn ebenfalls zufällig gewählt wird – wird die Umgebung mit Schrittweite s untersucht auf eine Lösung mit geringerer Energie – hier durch die Gütefunktion berechnet. Wird eine solche Lösung gefunden, wird diese als Ausgangspunkt für die nächste Iteration gewählt. Sind die Nachbarlösungen schlechter, können sie abhängig von der Temperatur mit einer gewissen Wahrscheinlichkeit, die sich über die Boltzmann-Statistik berechnet, trotzdem als Ausgangspunkt gewählt werden. Die Temperatur startet mit einer initial hohen Temperatur ϑ_0 und kühlt je Iteration ab. Zu Beginn werden

Tabelle 7.3: Ergebnisse der Gittersuche für SA

Epochen	ϑ_0	s	$\bar{f}$
25	100	100	2,409
	10 000		2,771
50	100	10	2,378
	1000	100	2,563
75	10 000	100	2,203
	1000		2,529
100	10 000	100	2,432
		10	2,584
1000	10 000	100	1,542
	1000	100	1,589

also Übergänge in schlechtere Zustände häufiger erfolgen und mit der Zeit ein (lokales) Minimum seltener wieder verlassen. [Arno13, S. 181–183]

SA hat am wenigsten Hyperparameter, entsprechend wird die Gittersuche mit folgenden Hyperparametern durchgeführt:

$$\begin{aligned} m_{\text{Epochen}} &= \{25; 50; 75; 100; 1000\} \\ \vartheta_0 &= \{100; 1000; 10\,000\} \\ s &= \{0{,}1; 1; 10; 100\} \end{aligned}$$

Da die Berechnungszeit einer Epoche deutlich geringer ausfällt als bei den beiden anderen Metaheuristiken, wurden hier größere Werte gewählt. Der Wert 1000 Epochen wurde inkludiert, um zu prüfen, ob bei langer Laufzeit eine Verbesserung eintritt, was sich auch gezeigt hat.

Die Ergebnisse der Gittersuche sind in Tabelle 7.3 aufgezeichnet, und die beste Kombination ist wieder mit grau hinterlegt. Die erreichten Güten bei kurzen Laufzeiten sind in der gleichen Größenordnung wie bei PSO, mit 1000 Epochen wird von allen Heuristiken das beste Ergebnis erreicht.

Vergleich der Metaheuristiken

Die drei Algorithmen werden jeweils mit den Hyperparametern aus der Gittersuche miteinander verglichen (siehe Tabelle 7.4). Anders als in der Gittersuche wird keine feste Anzahl an Epochen vorgegeben, sondern mehrere Abbruchkriterien definiert:

Tabelle 7.4: Für den Vergleich ausgewählte Algorithmen mit ihren Hyperparametern

Algorithmus	Hyperparameter		
GA	$m_{\text{Population}} = 30$	$P_{\text{Kreuzung}} = 80\,\%$	$P_{\text{Mutation}} = 10\,\%$
PSO	$m_{\text{Population}} = 30$	$w_{\text{Partikel}} = 3$	$w_{\text{Schwarm}} = 2$
SA	$\vartheta_0 = 10\,000$	$s = 100$	

1. Güte $f < 1$
2. Keine Verbesserung über 10 Epochen, bzw. 50 Epochen bei SA
3. Laufzeit von 2 min überschritten

Die Algorithmen werden jeweils mit 50 Wiederholungen ausgeführt, wobei jeweils eine zufällige Startpopulation bzw. ein zufälliger Startwert erzeugt wird. Alle Algorithmen werden zu Vergleichszwecken ohne Parallelisierung ausgeführt, sodass allen ein CPU-Kern zur Verfügung steht.

Abbildung 7.3 zeigt die Güte verschiedener Ausführungen der Metaheuristiken über die Zeit. Die einzelnen Durchläufe sind mit dem Startwert des Zufallszahlengenerators gekennzeichnet. Es zeigt sich, dass SA am schnellsten eine Lösung liefert. Die beiden anderen Algorithmen benötigen – durch die Berechnung einer Population von Lösungen – deutlich längere Zeiten pro Epoche und konvergieren entsprechend deutlich langsamer.

Die GA-Instanzen laufen fast immer die vorgegebenen 2 min, während SA und PSO durch das zweite Abbruchkriterium beendet werden, wenn sich die Lösung einige Epochen nicht mehr verbessert.

GA erreicht Lösungen mit einer Güte im Bereich von ca. 2 bis 4, PSO befindet sich darüber im Bereich 3 bis 5. Die beste Güte von SA wird durch beiden anderen Heuristiken nicht erreicht, SA befindet sich im Bereich von 1 bis 3. Manche GA-Ausführungen erreichen ähnliche Güten wie SA, benötigen dafür allerdings zwei- bis dreimal so viel Rechenzeit. Allerdings erreicht nur ein Teil der GA-Ausführungen solch gute Lösungen.

Simulated Annealing mit den obigen Abbruchkriterien scheint folglich am besten geeignet zu sein, um das Verteilungsproblem effizient zu lösen.

$$SA(m_{\text{Epochen}} = 1000, \vartheta_0 = 10\,000, s = 100)$$

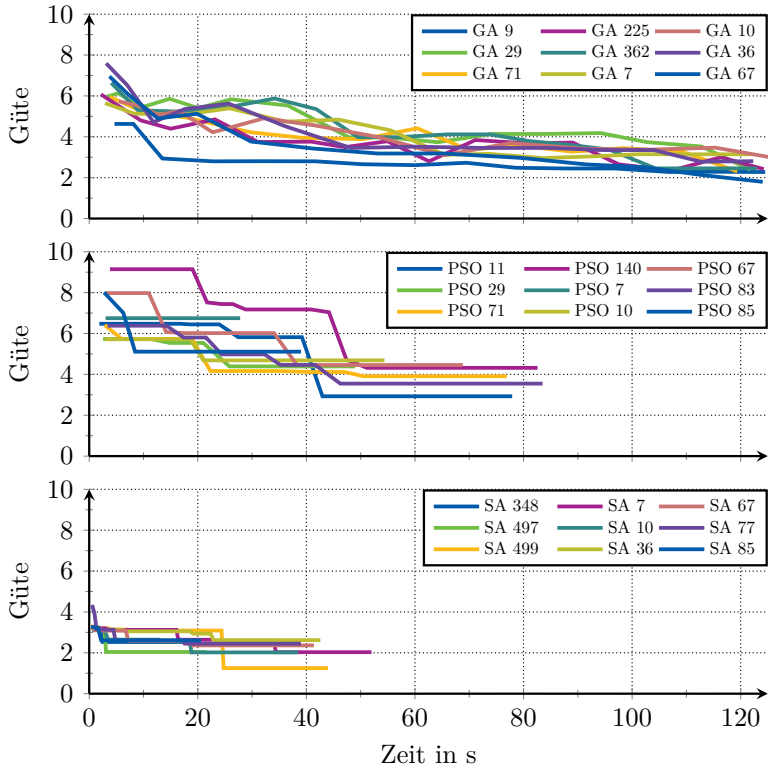


Abbildung 7.3: Entwicklung der Güte der gefundenen Lösungen bei ausgewählten, repräsentativen Wiederholungen. (Angegeben ist jeweils der Startwert des Zufallszahlengenerators.)

8 Validierung

Die Validierung des entwickelten Ansatzes findet anhand der Validierungsplattform aus dem Forschungsprojekt »EuProGigant«[13] – Europäisches Produktionsgiganet zur kalamitätsmindernden Selbstorchestrierung von Wertschöpfungs- und Lernökosystemen – statt. Mit der Validierungsplattform wird die Forschungsfrage untersucht, wie KMU datengetriebene Methoden zur Überwachung von Maschinen und Baugruppen einsetzen können, ohne über eine große Datenbasis zu verfügen [Webe22, S. 94].

Besonders KMU stehen beim Einsatz von Künstlicher Intelligenz (KI) vor großen Herausforderungen, da sie selten über einen großen Maschinenpark verfügen, sodass sie häufig Schwierigkeiten damit haben Datensätze geeigneter Größe zu generieren. Das Ziel der Validierungsplattform ist es diesen Unternehmen trotzdem einen Zugang zu KI- und ähnlichen Technologien zu ermöglichen. Maschineninterne Daten sollen im Edge-Netzwerk des Unternehmens erfasst [Webe21, S. 14–15] und nach außen zum Konsum angeboten werden, sodass sich Unternehmen mit anderen zusammenschließen können, um gemeinsam Analytik-Anwendungen entwickeln zu lassen. Dabei wird ein besonderer Fokus auf die selbstsouveräne Datenhaltung und -verarbeitung gelegt, weswegen der »*Compute-to-Data*«-Ansatz zur Anwendung kommt, sodass die Daten den Einflussbereich des Unternehmens nicht verlassen, sondern die Analytik-Applikation von außen zu den Daten kommt.

Ein Teilanwendungsfall für die Validierungsplattform ist das Anbieten lokal aufgezeichneter Maschinendaten auf einem Gaia-X-Marktplatz. Dafür ist, im Rahmen der Projektarbeit, die in Abbildung 8.1 dargestellte Systemarchitektur entwickelt worden. Die Pfeile zwischen den Knoten symbolisieren Kommunikationsrichtungen. Die Daten werden im Edge-Netzwerk zunächst durch einen *Gateway Service* und den *Data Switch* (1) auf ein einheitliches Datenmodell transformiert und anderen Services zur Verfügung gestellt.

Der *Data Collector* (2) aggregiert Daten, die jeweils zu einem gefertigten Werkstück gehören, und leitet das Datenpaket anschließend an zwei Services weiter. Zum einen ist dies der *File Transfer Service*, der sich um die Speicherung im Edge-Netzwerk oder auf externer Infrastruktur kümmert. Der dargestellte Fall ist die Speicherung auf einer Cloud-Infrastruktur, die durch das Unternehmen

[13] `https://euprogigant.com`

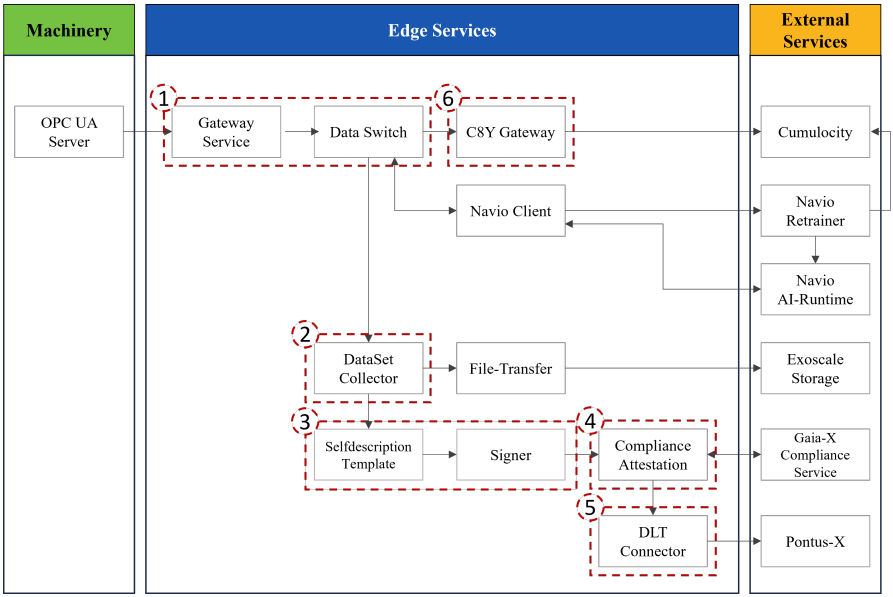


Abbildung 8.1: Architekturausschnitt der beteiligten Services und Kommunikationsrichtungen für die EuProGigant-Validierungsplattform. (Die roten Markierungen zeigen die sechs im Rahmen der Validierung verteilten Services)

administriert wird, sodass die Daten weiterhin im Verfügungsbereich des Unternehmens verbleiben. Der zweite Service ist die *Selfdescription Template Engine*, der eine Gaia-X-Selbstbeschreibung des Datensatzes erstellt, die im Nachgang vom *Signer* kryptographisch signiert wird (3).

Dieser signierten Selbstbeschreibung wird durch den *Gaia-X Compliance Service* die Einhaltung aller im Datenökosystem notwendigen Kriterien und Regularien attestiert (4) und abschließend wird diese über den *DLT Connector* im Ökosystem veröffentlicht (5). Zusätzlich werden die Maschinendaten über den *C8Y Gateway Service* (6) und den *Navio Client* an zwei IoT-Plattformen weitergeleitet, was nicht dem *Compute-to-Data*-Ansatz entspricht, aber für weitere Anwendungsfälle benötigt wird.

Die nummerierten Services sollen im Rahmen der Validierung verteilt werden. Die anderen Services wurden zur Vereinfachung der Validierung ignoriert. Die

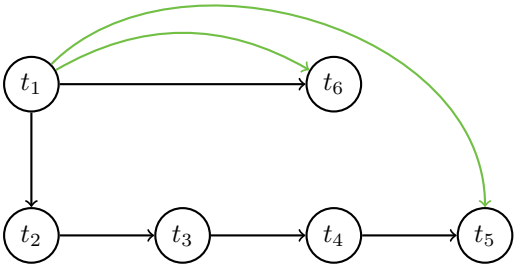


Abbildung 8.2: Graph der zu verteilenden Tasks für die Validierungsplattform (schwarz: Datenflusskanten, grün: Latenzanforderungskanten)

sechs Services lassen sich – wie in Abbildung 8.2 zu sehen – in die modellierte Graphrepräsentation überführen[14].

8.1 Fog-Cluster

Ähnlich wie für die Vorstudie (Kapitel 6) wird für die Validierung ein Teil des PTW-TEC-Labs genutzt. In Abbildung 8.3 ist die Graphrepräsentation des Fog-Clusters ohne Attribute der Knoten dargestellt[15].

Über vier Switche sind acht IPCs miteinander verbunden, sowie der Rest des PTW-Netzwerkes als Senke modelliert. Die Computer sind von unterschiedlicher Leistungsfähigkeit, mit den NCU1 bzw. NCU2 als schwächsten Vertretern und der EuProGigant-Edge als leistungsstärkstem IPC.

8.2 Berechnung aller möglichen Verteilungen

Mit acht Computern und sechs zu verteilenden Services ergeben sich $8^6 = 262\,144$ mögliche Verteilungen. Da eine Güteberechnung bis zu 350 ms benötigt, sind für diesen kleinen Taskgraphen die Rechenzeiten für sämtliche Verteilungen noch in einem handhabbaren Rahmen von ca. 24 h.

[14] Die JSON-serialisierte Beschreibung der Validierungstasks ist im Anhang ab Seite 137 zu finden.

[15] Die vollständige JSON-serialisierte Beschreibung des Clusters mit allen Attributen ist im Anhang ab Seite 140 zu finden.

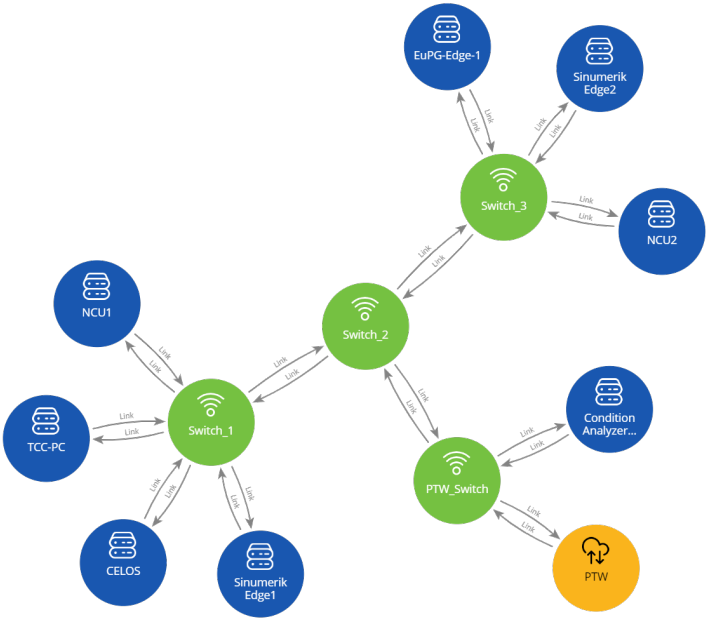


Abbildung 8.3: Graphvisualisierung des Fog-Clusters für die Validierung (blau: IPCs, grün: Netzwerkgeräte, orange: Senke)

Als Referenz für die Validierung werden also sämtliche Verteilungen gebildet und jeweils die Güte der jeweiligen Verteilung bestimmt. Wie in Abbildung 8.4 zu sehen ist, sind nur 1,5 % der 262 144 Verteilungen im akzeptablen Bereich und weisen keine Überlastung von Ressourcen oder Überschreitung von Latenzanforderungen auf. Während der größte Teil mindestens eine Latenzanforderung nicht erfüllen kann, wird beim kleinsten Teil eine oder mehrere Ressourcen überlastet. Dies ist plausibel, da die sechs Tasks in Summe keine sehr großen Ressourcenanforderungen haben.

Eine genauere Häufigkeitsverteilung der akzeptablen Verteilungen ist in Abbildung 8.5 zu sehen. Es ist erkennbar, dass die größte Anzahl der akzeptablen Verteilungen eine Güte zwischen vier und fünf aufweist. Der Median liegt bei 4,572, während die minimale Güte bei 0,740 liegt, welche von genau einer Verteilung erreicht wird. Diese optimale Verteilung ist in Tabelle 8.1 dargestellt.

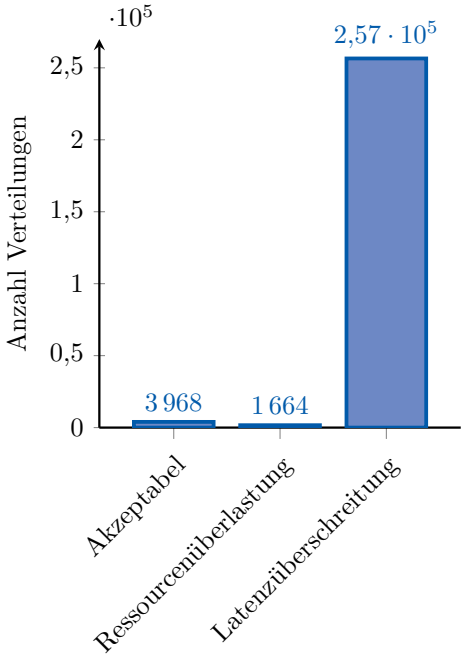


Abbildung 8.4: Klassifizierung aller möglichen Softwareverteilungen

Tabelle 8.1: Optimale Verteilung der Validierungs-Services im Fog-Cluster

Nr.	Service	Computer
1	Data Switch	CELOS
2	DataSet Collector	CELOS
3	Selfdescription Tooling	TCC-PC
4	Compliance Attestation	CELOS
5	DLT Connector	TCC-PC
6	Cumolocity Gateway	TCC-PC

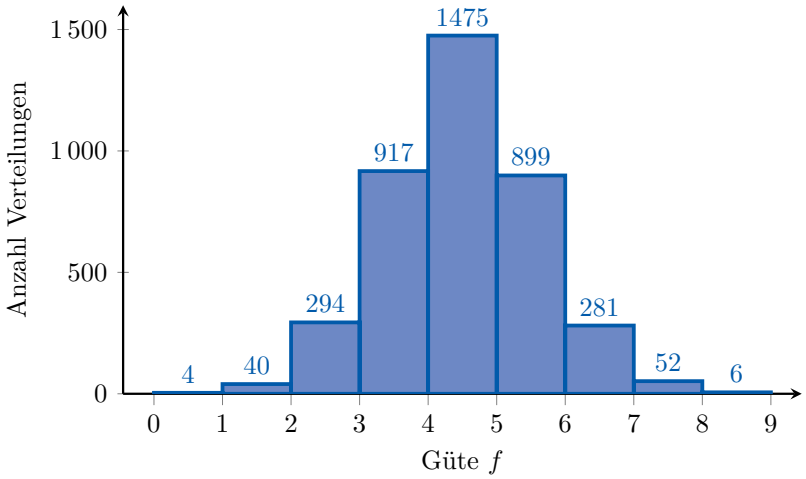


Abbildung 8.5: Häufigkeitsverteilung der Güte der akzeptablen Softwareverteilungen

8.3 Vergleich mit der Metaheuristik

Um einen Vergleich der Metaheuristik SA mit der Gesamtmenge aller Verteilungen zu erlauben, wurde das reduzierte Verteilungsproblem wiederholt durch den SA-Algorithmus gelöst. Bei allen 500 Durchführungen wurde eine Güte $f < 2{,}5$ erreicht, womit sämtliche gefundenen Verteilungen zu den besten 8,5 % der akzeptablen Verteilungen gehören. Im Vergleich mit der Gesamtzahl aller Verteilungen gehören alle gefundenen Verteilungen zu den besten 1,3 ‰.

Die Häufigkeitsverteilung der Güte der jeweils erreichten besten Verteilung ist in Abbildung 8.6 dargestellt. Es ist zu sehen, dass nur in 3 Durchläufen eine Verteilung nahe dem globalen Optimum gefunden wird – in keinem Durchlauf wird das Optimum selber gefunden. Der größte Teil der Verteilungen erreicht aber eine Güte $f < 2$, was bedeutet, dass die gefundene Verteilung in der Regel zum besten Prozent der akzeptablen Verteilungen gehört.

Die Laufzeiten der einzelnen Durchläufe sind in Abbildung 8.7 zusammengefasst. Die Laufzeiten liegen pro Durchlauf zwischen 5,27 s und 15,34 s, was signifikant schneller ist als die Berechnung aller Verteilungen.

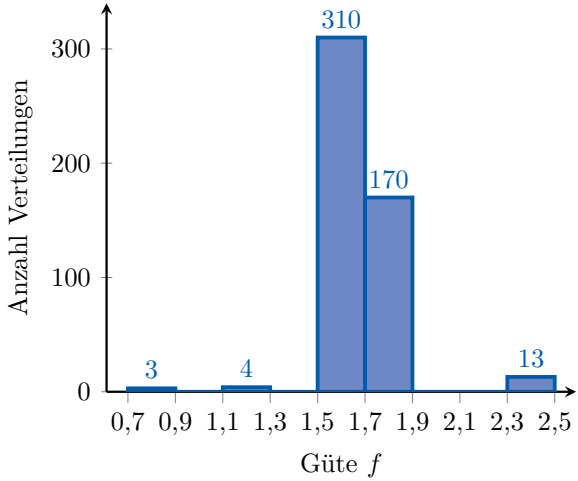


Abbildung 8.6: Häufigkeitsverteilung der Güte der durch SA gefundenen Verteilungen

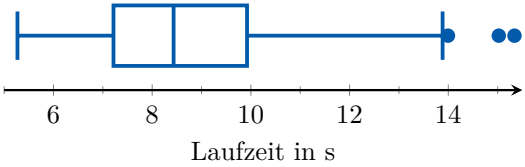


Abbildung 8.7: Boxplot der SA-Laufzeiten

8.4 Fazit

Durch Reduktion auf wenige zu verteilenden Tasks konnte das Verteilungsproblem konventionell gelöst werden. Hierfür wurden als Referenz alle möglichen Verteilungen gebildet und mittels der Gütefunktion bewertet.

Der entwickelte Metaheuristik-Ansatz wurde wiederholt ausgeführt und die Ergebnisse mit der Referenz verglichen. Die Metaheuristik findet zwar nicht das globale Optimum, erreicht aber bei jeder Wiederholung ausreichend gute Ergebnisse. Die dafür benötigten Laufzeiten sind deutlich geringer als bei der Ausrechnung aller möglichen Verteilungen und liegen im ein- bis niedrigen zweistelligen Sekundenbereich.

Mit zunehmender Anzahl an zu verteilenden Tasks steigt der Rechenaufwand für alle Verteilungen exponentiell an, sodass im Rahmen dieser Arbeit die Übertragbarkeit auf komplexere Situationen nicht bewiesen werden kann. Die vielversprechenden Ergebnisse deuten aber darauf hin, dass auch komplexere Verteilungsprobleme durch SA gut und in verhältnismäßig geringer Zeit gelöst werden können. Allerdings sind weitere Untersuchungen notwendig, um eine genaue Aussage treffen zu können.

9 Zusammenfassung & Ausblick

In diesem Kapitel wird die durchgeführte Forschung zusammengefasst, die Neuartigkeit der präsentierten Lösung herausgestellt, sowie die resultierenden wissenschaftlichen Erkenntnisse benannt. Daraufhin werden notwendige Schritte für eine Industrialisierung des vorgestellten Ansatzes genannt. Abschließend werden bestehende Einschränkungen und weitere Forschungsbedarfe erläutert.

9.1 Zusammenfassung

Mit dieser Arbeit sollte ein Ansatz identifiziert werden, mit dem optimale Softwareverteilungen für abhängige Tasks auf einem Fog-Cluster gefunden werden können. Diese Verteilungen sollten die Einhaltung von Latenzanforderungen ermöglichen und dazu geeignet sein, den Gesamtenergieverbrauch des Clusters möglichst minimal zu gestalten. Besondere Herausforderung aus dem Anwendungsgebiet Produktion im *Shopfloor* waren Echtzeit-Anforderungen an die Tasks und die Einflüsse der Netzwerkverbindungen im Fog-Cluster.

Dafür wurde zunächst anhand einer Literaturanalyse gezeigt, dass für den Produktionsbereich die Problemstellungen von Echtzeit und Ressourceneffizienz nicht oder nicht umfassend genug betrachtet werden. Ein Blick in die Literatur über den Maschinenbau hinaus zeigte, dass die Fragen nach Ressourceneffizienz und Latenz durchaus betrachtet werden, allerdings meistens unabhängig voneinander. Die notwendigen geringen Taktzeiten einer Maschinenregelung sind in der Informatik zudem nicht Gegenstand der Forschung, da die dort geforderten Latenzen deutlich größer sind. Zusammenfassend zeigte sich, dass der Einsatz von CNT im Maschinenbau im kleinen Rahmen zwar stattfindet oder zumindest vorgeschlagen wird. Gleichzeitig scheinen aber Hindernisse zu bestehen, die eine aufwandsarme Nutzung der Technologien erschweren, u. a. für KMU. Für den Einsatz im *Shopfloor* zeigte sich, dass die Technologien bisher nicht passend oder ausgereift genug sind.

Der Haupthinderungsgrund für den Einsatz von CNT scheint die fehlende Garantie einer Latenzeinhaltung in maschinebautypischen Größenordnungen zu sein. Erkennbar wurde dies bei der Validierung, die zeigte, dass knapp 98 % der möglichen Softwareverteilungen die Latenzanforderung nicht einhalten

können, obwohl die Anforderung mit 500 ms deutlich weniger anspruchsvoll war als bei einer typischen Maschinenregelungen mit 2 ms.

Entsprechend wurde im Laufe der Arbeit ein Fog-Cluster-Modell entwickelt, das die Netzwerkverbindungen mit ihren Einflüssen miteinbezieht. Sowohl das Cluster-Modell als auch die zu verteilende Software wurden als Graphen modelliert. Eine Gütefunktion bewertet eine gegebene Verteilung auf dem Cluster und prüft Nebenbedingungen wie die Einhaltung aller Latenzanforderungen.

Verschiedene Metaheuristiken wurden erprobt für die Suche einer guten Softwareverteilung auf dem Fog-Cluster-Modell. Als effizientester Algorithmus für das Finden einer nahe-optimalen Lösung in kurzer Zeit wurde eine konkrete Parametrisierung von SA identifiziert.

Durch Reduktion des Suchraumes konnte für die Validierung der Metaheuristik nachgerechnet werden, ob die gefundenen Lösungen wirklich nahe des Optimums liegen. Dieser Nachweis ist gelungen, da SA i. d. R. eine Lösung aus dem besten Prozent aller akzeptablen Lösungen – d. h. unter Einhaltung aller Nebenbedingungen – findet. Somit ist die vorgestellte Kombination aus den beiden Modellen und dem parametrierten SA-Algorithmus geeignet, um eine Softwareverteilung auf dem Fog-Cluster zu finden, die sowohl Latenzanforderungen einhält, als auch ressourceneffizient ist.

Das Ziel dieser Arbeit konnte also erreicht werden, da ein entsprechender Ansatz gefunden und bestätigt wurde. Der Ansatz grenzt sich von bisherigen Arbeiten in diesem Themenbereich ab, indem die folgenden Aspekte in Kombination adressiert werden:

- Der Betrieb von Echtzeitanwendungen wird ermöglicht, da die Netzwerktopologie und Verbindungsqualität in die Latenzberechnung einfließen, welche sich wiederum auf die Softwareverteilung überträgt.
- Dabei wird von langlaufenden Subtasks mit kontinuierlichen Datenflüssen zwischen den Subtasks ausgegangen.
- Der Ansatz erlaubt die Bindung bestimmter Subtasks an konkrete IPC im Fog-Cluster.

In der vorgestellten Arbeit sind drei Beiträge zum Wissenschaftsbereich der Produktionsforschung hervorzuheben:

1. Zunächst wurde eine Forschungsmethodik mit Vorgehen für Forschung zum Thema Software-Verteilungen erstellt und angewandt. Dieses Vorgehen kann für zukünftige Forschungsarbeiten übernommen werden, auch wenn andere Ziele mit der Software-Verteilung angestrebt werden. Die konkreten Modellierungs- und Optimierungsschritte müssen, je nach Ergebnis der Literaturanalyse, ergänzt oder angepasst werden.
2. Zusätzlich wurden Prozesse auf typischen Produktions-IPCs hinsichtlich ihrer Ressourcenbelegung analysiert. Für CPU und RAM wurden jeweils drei typische Klassen gebildet, die mit wenigen charakteristischen Größen beschrieben werden und als Ausgangspunkt für weitere Untersuchungen dienen können.
3. Zuletzt wurde gezeigt, dass die Optimierungsprobleme, die im Rahmen der Software-Verteilungen auftreten, mit Metaheuristiken gelöst werden können. Metaheuristiken allgemein, und die drei konkreten SA, GA, PSO, sind gut geeignet, da sich damit Lösungen nahe des Optimums mit wenig Rechenaufwand finden lassen. Die Lösungen können in wenigen Sekunden gefunden werden, was für den Einsatzzweck bei der Softwareverteilung im *Shopfloor* ausreichend schnell ist.

9.2 Ausblick

Insbesondere für die praktische Anwendung des vorgestellten Ansatzes durch Unternehmen sind noch weitere Schritte durchzuführen. Diese und weitere Forschungsbedarfe werden im Folgenden dargelegt. Abschließend werden einige mögliche Erweiterungen des Ansatzes genannt.

Weiterentwicklung des vorgestellten Ansatzes Die entwickelte Kombination aus Modellen und Metaheuristik sollte in eine Container-Orchestrierungs-Software wie Kubernetes integriert werden. Die entsprechenden Integrationspunkte wurden in Abschnitt 3.2.3 vorgestellt. Das Ergebnis des Algorithmus sollte an den *Scheduler* von Kubernetes übergeben werden, sodass die vorgeschlagene Softwareverteilung von diesem im Cluster umgesetzt wird. Hierzu muss allerdings das *Pod*-Modell von Kubernetes an das vorgestellte Modell angepasst werden, da Kubernetes aktuell keine Abhängigkeiten zwischen *Pods* kennt.

Für den Betrieb eines Fog-Clusters sind weitere Softwarekomponenten – wie ein *Domain Name System*, *Service Discovery*, *Resource Monitoring*, uvm. – notwendig, auf die hier allerdings nicht näher eingegangen wird. Peinl und Holzschuher [Pein15] beschreiben diese und weitere Komponenten und liefern eine gute Übersicht über verschiedene dafür verfügbare Softwarelösungen.

Im Normalbetrieb des Clusters können alle unbenötigten Computer abgeschaltet sein. Sobald ein neuer Task verteilt werden soll, wird das Task-Modell um diesen erweitert und zunächst die Tasks mit Hardwarebindung verteilt. Anschließend sucht die Metaheuristik nach einer neuen Lösung für die ungebundenen Tasks. Setzt die neue Lösung eine andere Menge von aktiven Computern voraus, müssen die neuen Computer geweckt werden. Anschließend können die Tasks auf ihren neuen Zielcomputer migriert werden und abschließend werden die nun unbenutzten Rechner abgeschaltet, sodass insgesamt wieder ein minimaler Energieverbrauch des Clusters erreicht wird.

Sollten häufig neue Tasks zu verteilen sein, kann die Gütefunktion erweitert werden. So könnte ein Term ergänzt werden, der die Anzahl der notwendigen Migrationen bestraft. Dies ist in der Literatur für VMs bereits beschrieben worden, wurde aber für den vorgestellten Anwendungszweck im *Shopfloor* nicht umgesetzt, da hier nicht häufig genug mit neuen Tasks zu rechnen ist.

Weitere Forschungsbedarfe Wie in Kapitel 6 dargestellt wurde, sind bei der Analyse zu typischen Lastprofilen lediglich Prozessor und Arbeitsspeicher als Grundlage herangezogen worden. Diese Einschränkung liegt darin begründet, dass Ressourcen wie Netzwerkschnittstellen oder Massenspeicherschreib- und -leseauslastung nur kumuliert vermessen werden können und somit nicht direkt einzelnen Prozessen zugeordnet werden können. Detailliertere Untersuchungen – bspw. auf isolierten Testsystemen – sind nötig, um die Klassifizierung von Tasks weiter zu verfeinern.

Wie bereits in Kapitel 6 erwähnt wurde, konnten nur zwei WZMs für die Erfassung der Ressourcenauslastung herangezogen werden. Eine Übertragbarkeit auf weitere Maschinen oder andere Steuerungssysteme wurde postuliert, zukünftige Untersuchungen mit einer größeren Datenbasis sollten diese Annahme aber überprüfen.

Für die Modellierung wurde vereinfachend angenommen, dass nur Prozesse der Klassen I und i auftreten, alle Ressourcenbedarfe also von der Zeit unabhängig sind. Zukünftige Arbeiten sollten diese Limitierung beheben, sodass ein dynamisches Modell mit einer Vorausplanung der Ressourcenbedarfe – mindestens über einen begrenzten Zeitraum – entsteht.

Eine weitere Verbesserung der Modelle ist hinsichtlich der Zeitberechnung für die eigentliche Task-Ausführung auf dem Computer möglich. In dieser Arbeit wurde der Allgemeinfall adressiert, für den nur die Prozessor- bzw. Grafikkartenauslastung bestimmend für die Berechnungszeit τ_{Comp} ist. Prozesse mit vielen auf den Massenspeicher zu schreibenden Daten könnten aber zusätzliche Wartezeit benötigen, bis sie schreiben können. Hierbei beeinflussen sich Prozesse auch gegenseitig, was ebenfalls in das oben erwähnte dynamische Modell einfließen sollte. Insgesamt kann es vorkommen, dass ein Prozess blockiert ist, was in die Zeitberechnung zukünftig ebenfalls mit inkludiert werden sollte.

Weiterhin arbeiten moderne Prozessoren nicht mehr mit einer festen Taktfrequenz, sondern diese wird auslastungsabhängig reguliert – sog. *Dynamic Voltage and Frequency Scaling*. Dies verkompliziert die Zeitberechnung auf dem Prozessor zusätzlich, sofern die Funktionalität wegen der Echtzeitfähigkeit des Gesamtsystems nicht deaktiviert wurde. Eine der weiteren Forschungsaufgaben wäre daher, diese Technik sowohl in die Zeitberechnung als auch in das Cluster-Modell zu integrieren.

In dieser Arbeit wurde vereinfachend davon ausgegangen, dass alle Computer die gleiche Rechenleistung pro Energieeinheit erbringen. Die elektrische Effizienz von Prozessoren hat sich mit der Zeit stark entwickelt, sodass die Annahme nur für IPCs der gleichen Architektur und eines ähnlichen Alters

gilt. Zukünftig sollte das Modell um einen Energieeffizienzfaktor jeder Rechnerressource erweitert werden, sodass die Ressourceneffizienz des Clusters genauer in der Softwareverteilung berücksichtigt wird.

Zukünftige Forschung sollte außerdem den Nachweis erbringen, dass eine Softwareverteilung gemäß des vorgestellten Ansatzes tatsächlich zu einer verbesserten Ressourceneffizienz führt. Hierzu könnten Messungen des elektrischen Energieverbrauchs des Fog-Clusters durchgeführt werden. Durch den Vergleich des Ausgangszustands mit dem optimierten Zustand könnte der Nachweis erbracht werden, dass der Zusammenhang von Rechnerauslastung mit der Ressourceneffizienz gerechtfertigt ist.

Mögliche Erweiterungen des vorgestellten Ansatzes Ein wesentlicher Nachteil des präsentierten Ansatzes liegt darin, dass die Ressourcenbedarfe der einzelnen Subtasks im Vorfeld bekannt sein müssen. Dies setzt in der Entwicklung zusätzlichen Aufwand und Kenntnisse voraus, die in KMU nicht zwangsweise vorhanden sein werden. Um die Anwendbarkeit der CNT weiter zu verbessern, könnte hier ein Alternativansatz gewählt werden: Die Ressourcenbedarfe könnten durch Beobachtung im Betrieb ergänzt oder sogar ganz ersetzt werden.

Hierfür könnten KI- oder ML-Modelle eine Vorhersage der Ressourcenbedarfe ermöglichen, sodass nach einer kurzen Beobachtungszeit bereits realistische Kennwerte für die verschiedenen Lastprofile bereitstehen könnten. Praktisch würde dies bedeuten, dass in der Entwicklung nur eine grobe Schätzung der Ressourcenbedarfe abgegeben werden muss und diese im Betrieb relativ bald durch die Modelle präzisiert werden. Diese Modelle könnten für jede Klasse aus Kapitel 6 einzeln vortrainiert werden, sodass nach einer Klassifikation schnell fertig trainierte Modelle bereitstehen könnten.

Alternativ ist auch ein Ablauf denkbar, in dem die Tasks zunächst auf einem freien Computer gestartet werden, ohne dass im Vorfeld eine Charakterisierung erfolgen muss. Auf diesem Testsystem werden die Tasks zunächst beobachtet, bis die Modelle mit einer ausreichenden Genauigkeit die Ressourcenbedarfe – und insbesondere auch deren zeitlichen Verlauf – abschätzen können. Anschließend kann mit den ermittelten Werten eine Softwareverteilung des Clusters ermittelt werden, die per Migration im Anschluss auf das tatsächliche Cluster übertragen wird. Ähnliche Vorgehen – allerdings mit VMs – gibt es in der Literatur bereits, z. B. [Shel17].

Um die Sicherheit der erforderlichen Latenzeinhaltung zu erhöhen, kann anstatt eines statisch aufgebauten Cluster-Modells dieses auch durch Echtzeit-

Überwachung des realen Clusters aktualisiert werden. Hierzu könnten bspw. die Auslastungen der einzelnen Ressourcen zurück in das Modell geführt werden. Besonders relevant ist die Erfassung der Paketverlustrate auf den Netzwerkverbindungen, da diese signifikant für die Transport- und damit Gesamtzeit ist. Insbesondere bei Funkverbindungen kann die Paketverlustrate sich abhängig von den äußeren Verhältnissen deutlich verändern, sodass ein aktueller Messwert bei der Verteilungsentscheidung die Einhaltung der Latenzanforderungen garantieren kann. Durch regelmäßiges Überprüfen, ob eine neue bessere Verteilung gefunden werden kann, kann so auch einer abnehmenden Verbindungsqualität entgegengewirkt werden und das Einhalten aller Latenzanforderungen gewährleistet bleiben.

Wird die Echtzeit-Überwachung angestrebt, muss allerdings ein Kompromiss gefunden werden, denn die Überwachung ist mit Aufwänden – bspw. Prozessorzeit, Netzwerkauslastung – und insbesondere auch Energieverbrauch verbunden. Es gilt abzuwägen, wie oft Messungen erfolgen müssen, um nicht zu viele Ressourcen durch die Überwachung zu belegen aber gleichzeitig auch rechtzeitig reagieren zu können [Sehg18, S. 78].

Um dem Gedanken der Resilienz Rechnung zu tragen, können zukünftige Arbeiten zusätzlich zur Modellierung und Überwachung der Netzwerkverfügbarkeit und entsprechender Softwareverteilung weitere Absicherungsmaßnahmen einführen. So könnte der Verteilungsalgorithmus u. a. dahingehend erweitert werden, dass mindestens eine Ersatzroute durch das Netzwerk mit einer gewissen Mindestausfallsicherheit existiert.

Mit dieser Arbeit wurde ein Grundstein gelegt, um Software-Applikationen in der Produktionsindustrie zukünftig mit CNT entwickeln zu können. Hierdurch sind eine effizientere Nutzung von Ressourcen wie Rechenkapazitäten im *Shopfloor* möglich, sowie weitere Vorteile wie Vereinfachungen in der Applikationsentwicklung nutzbar. Ist eine bessere Genauigkeit und damit zusätzliche Ressourceneinsparungen gewünscht, stellen die Ergebnisse eine gute Grundlage für weitere Forschungsarbeiten dar.

Literaturverzeichnis

Wissenschaftliche Veröffentlichungen

[Ahme17] E. Ahmed, A. Ahmed, I. Yaqoob, J. Shuja, A. Gani u. a. „Bringing Computation Closer toward the User Network: Is Edge Computing the Solution?“ In: *IEEE Communications Magazine* 55.11 (2017), S. 138–144.

[Akta22] M. N. Aktan und H. Bulut. „Metaheuristic task scheduling algorithms for cloud computing environments“. In: *Concurrency and Computation: Practice and Experience* 34.9 (2022), e6513.

[Al-D20] A. Al-Dulaimy, Y. Sharma, M. G. Khan und J. Taheri. „Introduction to edge computing“. In: *Edge Computing*. Hrsg. von J. Taheri und S. Deng. London, United Kingdom: The Institution of Engineering and Technology, 2020, S. 3–25.

[Aral20] A. Aral und V. de Maio. „Simulators and emulators for edge computing“. In: *Edge Computing*. Hrsg. von J. Taheri und S. Deng. London, United Kingdom: The Institution of Engineering and Technology, 2020, S. 291–311.

[Arno13] A. Arnold. *Physik auf dem Computer: Skript zur Vorlesung*. Stuttgart, 2013.

[Asla21] M. S. Aslanpour, A. N. Toosi, C. Cicconetti, B. Javadi, P. Sbarski u. a. „Serverless Edge Computing: Vision and Challenges“. In: *2021 Australasian Computer Science Week Multiconference*. Hrsg. von N. Stanger. ACM Digital Library. New York,NY,United States: Association for Computing Machinery, 2021, S. 1–10.

[Bans22] S. Bansal, H. Aggarwal und M. Aggarwal. „A systematic review of task scheduling approaches in fog computing“. In: *Transactions on Emerging Telecommunications Technologies* (2022), e4523.

[Baye19] T. Bayer, L. Moedel und C. Reich. „A Fog-Cloud Computing Infrastructure for Condition Monitoring and Distributing Industry 4.0 Services“. In: *Proceedings of the 9th International Conference on Cloud Computing and Services Science*. SCITEPRESS - Science and Technology Publications, 2019, S. 233–240.

[Belo12] A. Beloglazov, J. Abawajy und R. Buyya. „Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing“. In: *Future Generation Computer Systems* 28.5 (2012), S. 755–768.

[Bera17] R. Beraldi, A. Mtibaa und H. Alnuweiri. „Cooperative load balancing scheme for edge computing resources“. In: *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*. Piscataway, NJ: IEEE, 2017, S. 94–100.

[Berm18] D. Bermbach, F. Pallas, D. G. Pérez, P. Plebani, M. Anderson u. a. „A Research Perspective on Fog Computing“. In: *Service-Oriented Computing - ICSOC 2017 Workshops*. Hrsg. von L. Braubach, J. M. Murillo, N. Kaviani, M. Lama, L. Burgueño u. a. Bd. 10797. Lecture Notes in Computer Science Ser. Cham: Springer International Publishing AG, 2018, S. 198–210.

[Beze22] R. Bezerra, K. Ohno, S. Kojima, H. A. Aryadi, K. Gunji u. a. „Heterogeneous Multi-Robot Task Scheduling Heuristics for Garment Mass Customization“. In: *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2022, S. 439–446.

[Bour01] T. Bourke. *Server load balancing: Help for network administration*. 1. ed. Help for network administrators. Beijing und Köln: O'Reilly, 2001.

[Brec21] C. Brecher und M. Weck. *Werkzeugmaschinen Fertigungssysteme 3: Mechatronische Systeme, Steuerungstechnik und Automatisierung*. 9. Auflage. VDI-Buch. Berlin: Springer Vieweg, 2021.

[Chen21] S. Chen und M. Zhou. „Evolving Container to Unikernel for Edge Computing and Applications in Process Industry“. In: *Processes* 9.2 (2021), S. 351.

[Chia17] M. Chiang, S. Ha, F. Risso, T. Zhang und I. Chih-Lin. „Clarifying Fog Computing and Networking: 10 Questions and Answers“. In: *IEEE Communications Magazine* 55.4 (2017), S. 18–20.

[Dill22] F. Dillinger, O. Bernhard, M. Kagerer und G. Reinhart. „Industry 4.0 implementation sequence for manufacturing companies“. In: *Production Engineering* 16.5 (2022), S. 705–718.

[Duan17] H. Duan, C. Chen, G. Min und Y. Wu. „Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems“. In: *Future Generation Computer Systems* 74 (2017), S. 142–150.

[Egge20] G. Eggers, B. Fondermann, B. Maier, K. Ottradovetz, J. Pfrommer u. a. *GAIA-X: Technical Architecture: Release – June, 2020.* 2020.

[Eyk18] E. van Eyk, L. Toader, S. Talluri, L. Versluis, A. Uta u. a. „Serverless is More: From PaaS to Present Cloud Computing“. In: *IEEE Internet Computing* 22.5 (2018), S. 8–17.

[Fert22b] A. Fertig, M. Weigold und Y. Chen. „Machine Learning based quality prediction for milling processes using internal machine tool data“. In: *Advances in Industrial and Manufacturing Engineering* 4 (2022), S. 100074.

[Fiel00] R. T. Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. Dissertation. Irvine, CA: University of California, 2000.

[Fogl21] G. Fogliazza, C. Arvedi, C. Spoto, L. Trappa, F. Garghetti u. a. „Fingerprint analysis for machine tool health condition monitoring“. In: *IFAC-PapersOnLine* 54.1 (2021), S. 1212–1217.

[Gann17] D. Gannon, R. Barga und N. Sundaresan. „Cloud-Native Applications“. In: *IEEE Cloud Computing* 4.5 (2017), S. 16–21.

[Geze21] V. Gezer und A. Wagner. „Real-time edge framework (RTEF): task scheduling and realisation“. In: *Journal of Intelligent Manufacturing* 32.8 (2021), S. 2301–2317.

[Gil21] G. Gil, D. Corujo und P. Pedreiras. „Cloud Native Computing for Industry 4.0: Challenges and Opportunities“. In: *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Piscataway, NJ: IEEE, 2021, S. 01–04.

[Gill19] S. S. Gill, P. Garraghan und R. Buyya. „ROUTER: Fog enabled cloud based intelligent resource management approach for smart home IoT devices“. In: *Journal of Systems and Software* 154 (2019), S. 125–138.

[Gill21] S. S. Gill. „Quantum and blockchain based Serverless edge computing: A vision, model, new trends and future directions“. In: *Internet Technology Letters* (2021), e275.

[Gogo20] S. V. Gogouvitis, H. Mueller, S. Premnadh, A. Seitz und B. Bruegge. „Seamless computing in industrial systems using container orchestration“. In: *Future Generation Computer Systems* 109 (2020), S. 678–688.

[Gore17] D. Gorecky, A. Hennecke, M. Schmitt, S. Weyer und D. Zühlke. „Wandelbare modulare Automatisierungssysteme“. In: *Handbuch Industrie 4.0*. Hrsg. von G. Reinhart. München: Hanser, 2017, S. 555–583.

[Gowr21] A. S. Gowri, P. Bala und I. Zion. „Comprehensive Analysis of Resource Allocation and Service Placement in Fog and Cloud Computing“. In: *International Journal of Advanced Computer Science and Applications* 12.3 (2021).

[Guo21] J. Guo und M. Martínez-García. „Key technologies towards smart manufacturing based on swarm intelligence and edge computing“. In: *Computers & Electrical Engineering* 92 (2021), S. 1–6.

[He21] X. He, R. Jin und H. Dai. „Joint Service Placement and Resource Allocation for Multi-UAV Collaborative Edge Computing“. In: *2021 IEEE Wireless Communications and Networking Conference (WCNC)*. Hrsg. von M. Koca. Piscataway, NJ: IEEE, 2021, S. 1–6.

[Hoss20] M. Hosseini Shirvani, A. M. Rahmani und A. Sahafi. „A survey study on virtual machine migration and server consolidation techniques in DVFS-enabled cloud datacenter: Taxonomy and challenges“. In: *Journal of King Saud University - Computer and Information Sciences* 32.3 (2020), S. 267–286.

[Hu15] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher und V. Young. *Mobile Edge Computing: A key technology towards 5G: ETSI White Paper No. 11*. 2015.

[Hüni19] F. Hüning. *Embedded Systems für IoT*. 1. Aufl. 2019. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019.

[Iorg18] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. Goren u. a. *Fog computing conceptual model: NIST Special Publication 500-325*. Gaithersburg, MD, 2018.

[Jaco05] B. Jacob, M. Brown, K. Fukui und N. Trivedi. *Introduction to grid computing*. IBM Corporation, International Technical Support Organization, 2005.

[Jian21] C. Jiang, J. Wan und H. Abbas. „An Edge Computing Node Deployment Method Based on Improved k-Means Clustering Algorithm for Smart Manufacturing“. In: *IEEE Systems Journal* 15.2 (2021), S. 2230–2240.

[Kenn95] J. Kennedy und R. Eberhart. „Particle swarm optimization“. In: *Proceedings / 1995 IEEE International Conference on Neural Networks*. Piscataway, NJ: IEEE, 1995, S. 1942–1948.

[Khos20] M. A. Khoshkholghi, M. G. Khan, Y. Sharma und J. Taheri. „Resource allocation models in/for edge computing“. In: *Edge Computing*. Hrsg. von J. Taheri und S. Deng. London, United Kingdom: The Institution of Engineering and Technology, 2020, S. 125–146.

[Klei21] J.-P. Kleinhans und J. Hess. *Understanding the global chip shortages: Why and how the semiconductor value chain was disrupted*. Nov. 2021.

[Kohn22] O. Kohn, P. Stanula, E. Lang, M. Weigold und J. Metternich. „Development of a Stress Factor as an Indicator for Stress-Based Payment Models for Machine Tools“. In: *Production at the Leading Edge of Technology*. Hrsg. von B.-A. Behrens, A. Brosius, W.-G. Drossel, W. Hintze, S. Ihlenfeldt u. a. Lecture Notes in Production Engineering. Cham: Springer International Publishing, 2022, S. 239–247.

[Kubi22] K. Kubiak, G. Dec und D. Stadnicka. „Possible Applications of Edge Computing in the Manufacturing Industry: Systematic Literature Review“. In: *Sensors* 22.7 (2022), S. 2445.

[Kuma19] P. Kumar und R. Kumar. „Issues and Challenges of Load Balancing Techniques in Cloud Computing“. In: *ACM Computing Surveys* 51.6 (2019), S. 1–35.

[Lech17] A. Lechler und J. Schlechtendahl. „Steuerung aus der Cloud“. In: *Handbuch Industrie 4.0 Bd.1*. Hrsg. von B. Vogel-Heuser, T. Bauernhansl und M. ten Hompel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, S. 61–74.

[Lee17] J.-H. Lee, S.-H. Chung und W.-S. Kim. „Fog server deployment considering network topology and flow state in local area networks“. In: *ICUFN 2017*. Piscataway, NJ: IEEE, 2017, S. 652–657.

[Li18] L. Li, K. Ota und M. Dong. „Deep Learning for Smart Industry: Efficient Manufacture Inspection System With Fog Computing“. In: *IEEE Transactions on Industrial Informatics* 14.10 (2018), S. 4665–4673.

[Lian21] Y. C. Liang, W. D. Li, X. Lu und S. Wang. „Fog Computing and Convolutional Neural Network Enabled Prognosis for Machining Process Optimization“. In: *Data Driven Smart Manufacturing Technologies and Applications*. Hrsg. von W. Li, Y. Liang und S. Wang. Springer, Cham, 2021, S. 13–35.

[Lin17] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang u. a. „A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications“. In: *IEEE Internet of Things Journal* 4.5 (2017), S. 1125–1142.

[Linu20] Linux Foundation. *Sharpening the Edge: Overview of the LF Edge Taxonomy and Framework*. 2020.

[Liu13] Q. S. Liu. „A Novel Model for Multi-Parametric Load Balance in Grid Computing“. In: *Materials Science, Machinery and Energy Engineering* 853 (2013), S. 674–679.

[Liu19] L. Liu, H. Tan, S. H.-C. Jiang, Z. Han, X.-Y. Li u. a. „Dependent task placement and scheduling with function configuration in edge computing“. In: *Proceedings of the International Symposium on Quality of Service*. ACM Digital Library. New York,NY,United States: Association for Computing Machinery, 2019, S. 1–10.

[Liu22] J. Liu, C. Ma, H. Gui und S. Wang. „A four-terminal-architecture cloud-edge-based digital twin system for thermal error control of key machining equipment in production lines“. In: *Mechanical Systems and Signal Processing* 166 (2022), S. 108488.

[Marí17] E. Marín-Tordera, X. Masip-Bruin, J. García-Almiñana, A. Jukan, G.-J. Ren u. a. „Do we all really know what a fog node is? Current trends towards an open definition“. In: *Computer Communications* 109 (2017), S. 117–130.

[Marx22] U. Marx. „Maschinenbaugipfel: »Zehn verlorene Jahre«“. In: *Frankfurter Allgemeine Zeitung* (2022).

[Math94] K. E. Mathias und L. D. Whitley. „Transforming the search space with Gray coding“. In: *Proceedings of the First IEEE Conference on Evolutionary Computation*. Hrsg. von Z. Michalewicz. Piscataway, NJ: IEEE Service Center, 1994, 513–518 vol.1.

[Mazu17] S. Mazumdar und M. Pranzo. „Power efficient server consolidation for Cloud data center“. In: *Future Generation Computer Systems* 70 (2017), S. 4–16.

[Mell11] P. M. Mell und T. Grance. *The NIST definition of cloud computing*. Gaithersburg, MD, 2011.

[Meri19] R. Merino, I. Bediaga, A. Iglesias und J. Munoa. „Hybrid Edge–Cloud-Based Smart System for Chatter Suppression in Train Wheel Repair“. In: *Applied Sciences* 9.20 (2019), S. 4283.

[Moha20] N. Mohan, L. Corneo, A. Zavodovski, S. Bayhan, W. Wong u. a. „Pruning Edge Research with Latency Shears“. In: *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*. Hrsg. von B. Zhao, H. Zheng, H. V. Madhyastha und V. Padmanabhan. New York, NY, USA: ACM, 2020, S. 182–189.

[Mour18] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow u. a. „A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges“. In: *IEEE Communications Surveys & Tutorials* 20.1 (2018), S. 416–464.

[Mtsh19] M. Mtshali, H. Kobo, S. Dlamini, M. Adigun und P. Mudali. „Multi-Objective Optimization Approach for Task Scheduling in Fog Computing“. In: *2019 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD)*. IEEE, 2019, S. 1–6.

[Naha18] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao u. a. „Fog Computing: Survey of Trends, Architectures, Requirements, and Research Directions“. In: *IEEE Access* 6 (2018), S. 47980–48009.

[Noto00] M. Noto und H. Sato. „A method for the shortest path search by extended Dijkstra algorithm“. In: *Cybernetics evolving to systems, humans, organizations, and their complex interactions*. Piscataway, NJ: IEEE Service Center, 2000, S. 2316–2320.

[Pan18] J. Pan und J. McElhannon. „Future Edge Cloud and Edge Computing for Internet of Things Applications“. In: *IEEE Internet of Things Journal* 5.1 (2018), S. 439–449.

[Pein15] R. Peinl und F. Holzschuher. „The Docker Ecosystem Needs Consolidation“. In: *Proceedings of the 5th International Conference on Cloud Computing and Services Science*. SCITEPRESS - Science and and Technology Publications, 2015, S. 535–542.

[Powe21] D. Powell, R. Morgan und G. Howe. „Lean First ... then Digitalize: A Standard Approach for Industry 4.0 Implementation in SMEs“. In: *Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems*. Hrsg. von A. Dolgui, A. Bernard, D. Lemoine, G. von Cieminski und D. Romero. Cham: Springer International Publishing, 2021, S. 31–39.

[Qi19] Q. Qi und F. Tao. „A Smart Manufacturing Service System Based on Edge Computing, Fog Computing, and Cloud Computing“. In: *IEEE Access* 7 (2019), S. 86769–86777.

[Rait22] P. Raith, T. Rausch, S. Dustdar, F. Rossi, V. Cardellini u. a. „Mobility-Aware Serverless Function Adaptations Across the Edge-Cloud Continuum“. In: *Proceedings of the 15th IEEE/ACM International Conference on Utility and Cloud Computing (UCC'22)*. 2022.

[Robe17] M. Roberts und J. Chapin. *What is serverless? Understand the latest advances in cloud and service-based architecture*. First edition. Sebastopol, CA: O'Reilly Media, 2017.

[Sabo21] A. Sabol, R. Alimo, F. Kamangar und R. Madani. „Deep Space Network Scheduling via Mixed-Integer Linear Programming“. In: *IEEE Access* 9 (2021), S. 39985–39994.

[Saty09] M. Satyanarayanan, P. Bahl, R. Caceres und N. Davies. „The Case for VM-Based Cloudlets in Mobile Computing“. In: *IEEE Pervasive Computing* 8.4 (2009), S. 14–23.

[Sebr22] M. Sebrechts, B. Volckaert, F. de Turck, K. Yang und M. F. Al-Naday. „Fog Native Architecture: Intent-Based Workflows to Take Cloud Native Towards the Edge“. In: *IEEE Communications Magazine* 60.8 (2022).

[Sehg18] N. K. Sehgal und P. C. P. Bhatt. *Cloud Computing*. Cham: Springer International Publishing, 2018.

[Shah20] M. A. Shahid, N. Islam, M. M. Alam, M. M. Su'ud und S. Musa. „A Comprehensive Study of Load Balancing Approaches in the Cloud Computing Environment and a Novel Fault Tolerance Approach“. In: *IEEE Access* 8 (2020), S. 130500–130526.

[Shar18] P. K. Sharma, S. Rathore, Y.-S. Jeong und J. H. Park. „SoftEdgeNet: SDN Based Energy-Efficient Distributed Network Architecture for Edge Computing“. In: *IEEE Communications Magazine* 56.12 (2018), S. 104–111.

[Shel17] M. Shelar, S. Sane, V. Kharat und R. Jadhav. „Autonomic and energy-aware resource allocation for efficient management of cloud data centre“. In: *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*. Piscataway, NJ: IEEE, 2017, S. 1–8.

[Shi16] W. Shi, J. Cao, Q. Zhang, Y. Li und L. Xu. „Edge Computing: Vision and Challenges“. In: *IEEE Internet of Things Journal* 3.5 (2016), S. 637–646.

[Sing17] P. Singh, M. Dutta und N. Aggarwal. „A review of task scheduling based on meta-heuristics approach in cloud computing“. In: *Knowledge and Information Systems* 52.1 (2017), S. 1–51.

[Stan21] P. Stanula, O. Kohn, E. Lang, J. Metternich, M. Weigold u. a. „Economic assessment of stress-based payment models“. In: *Procedia CIRP* 103 (2021), S. 182–187.

[Strn21] C. F. Strnadl. „End-to-End-Architekturen zur Datenmonetarisierung im Industrial Internet of Things (IIoT)“. In: *Monetarisierung von technischen Daten*. Hrsg. von D. Trauth, T. Bergs und W. Prinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2021, S. 169–206.

[Stur22] C. Sturm. „Between a rock and a hard place: European energy policy and complexity in the wake of the Ukraine war“. In: *Journal of Industrial and Business Economics* (2022), S. 1–44.

[Tane08] A. S. Tanenbaum und M. van Steen. *Verteilte Systeme: Prinzipien und Paradigmen*. 2., aktualisierte Auflage. Pearson Studium - IT Ser. München u. a.: Pearson, 2008.

[Teoh21] Y. K. Teoh, S. S. Gill und A. K. Parlikad. „IoT and Fog Computing based Predictive Maintenance Model for Effective Asset Management in Industry 4.0 using Machine Learning“. In: *IEEE Internet of Things Journal* (2021), S. 1.

[Thie23] N. van Thieu und S. Mirjalili. „MEALPY: An open-source library for latest meta-heuristic algorithms in Python“. In: *Journal of Systems Architecture* 139 (2023), S. 102871.

[Urga07] B. Urgaonkar, A. L. Rosenberg und P. Shenoy. „Application Placement on a Cluster of Servers“. In: *International Journal of Foundations of Computer Science* 18.05 (2007), S. 1023–1041.

[Varg16] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick und D. S. Nikolopoulos. „Challenges and Opportunities in Edge Computing“. In: *2016 IEEE International Conference on Smart Cloud*. Hrsg. von M. Qiu. Piscataway, NJ: IEEE, 2016, S. 20–26.

[Vars20] S. Varshney, R. Sandhu und P. K. Gupta. „QoE-Based Multi-Criteria Decision Making for Resource Provisioning in Fog Computing Using AHP Technique“. In: *International Journal of Knowledge and Systems Science (IJKSS)* 11.4 (2020), S. 17–30.

[Vasa10] A. Vasan, A. Sivasubramaniam, V. Shimpi, T. Sivabalan und R. Subbiah. „Worth their watts? - an empirical study of datacenter servers“. In: *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, 2010, S. 1–10.

[Vill16] M. Villari, M. Fazio, S. Dustdar, O. Rana und R. Ranjan. „Osmotic Computing: A New Paradigm for Edge/Cloud Integration“. In: *IEEE Cloud Computing* 3.6 (2016), S. 76–83.

[Vill19] M. Villari, M. Fazio, S. Dustdar, O. Rana, D. N. Jha u. a. „Osmosis: The Osmotic Computing Platform for Microelements in the Cloud, Edge, and Internet of Things“. In: *Computer* 52.8 (2019), S. 14–26.

[Wang21a] J. Wang, Y. Liu, S. Ren, C. Wang und W. Wang. „Evolutionary game based real-time scheduling for energy-efficient distributed and flexible job shop“. In: *Journal of Cleaner Production* 293 (2021), S. 126093.

[Wang21b] J. Wang, D. Li und Y. Hu. „Fog Nodes Deployment Based on Space–Time Characteristics in Smart Factory“. In: *IEEE Transactions on Industrial Informatics* 17.5 (2021), S. 3534–3543.

[Wang21c] Y. Wang, C. Zhao, S. Yang, X. Ren, L. Wang u. a. „MPCSM: Microservice Placement for Edge-Cloud Collaborative Smart Manufacturing“. In: *IEEE Transactions on Industrial Informatics* 17.9 (2021), S. 5898–5908.

[Will20] A. Willner und V. Gowtham. „Toward a Reference Architecture Model for Industrial Edge Computing“. In: *IEEE Communications Standards Magazine* 4.4 (2020), S. 42–48.

[Yang19] C. Yang, Y. Liu, X. Chen, W. Zhong und S. Xie. „Efficient Mobility-Aware Task Offloading for Vehicular Edge Computing Networks“. In: *IEEE Access* 7 (2019), S. 26652–26664.

[Yin18] L. Yin, J. Luo und H. Luo. „Tasks Scheduling and Resource Allocation in Fog Computing Based on Containers for Smart Manufacturing“. In: *IEEE Transactions on Industrial Informatics* 14.10 (2018), S. 4712–4721.

[Yous19] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang u. a. „FOGPLAN: A Lightweight QoS-Aware Dynamic Fog Service Provisioning Framework“. In: *IEEE Internet of Things Journal* 6.3 (2019), S. 5080–5096.

[Yu18] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu u. a. „A Survey on the Edge Computing for the Internet of Things“. In: *IEEE Access* 6 (2018), S. 6900–6919.

[Zhan21] Y. Zhang und H.-Y. Wei. „Risk-Aware Cloud-Edge Computing Framework for Delay-Sensitive Industrial IoTs“. In: *IEEE Transactions on Network and Service Management* 18.3 (2021), S. 2659–2671.

[Zhao20] Y. Zhao, W. Wang, Q. Zhu, Y. Li und J. Zhang. „Collaborative platforms and technologies for edge computing“. In: *Edge Computing*. Hrsg. von J. Taheri und S. Deng. London, United Kingdom: The Institution of Engineering and Technology, 2020, S. 223–247.

[Zhou20] H. Zhou, Y. Xiang, H.-F. Li und R. Yuan. „Task Offloading Strategy of 6G Heterogeneous Edge-Cloud Computing Model considering Mass Customization Mode Collaborative Manufacturing Environment“. In: *Mathematical Problems in Engineering* 2020 (2020), S. 1–8.

[Zhu17] Q. Zhu, B. Si, F. Yang und Y. Ma. „Task offloading decision in fog computing system“. In: *China Communications* 14.11 (2017), S. 59–68.

[Ziet20] J. Zietsch, M. Vogt, B. D. Lee, C. Herrmann und S. Thiede. „Enabling smart manufacturing through a systematic planning framework for edge computing“. In: *CIRP Journal of Manufacturing Science and Technology* (2020).

[Zöbe20] D. Zöbel. *Echtzeitsysteme*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020.

Normen und Richtlinien

[DIN 88] DIN Deutsches Institut für Normung. *DIN 44300-9: Informationsverarbeitung – Begriffe – Verarbeitungsabläufe*. 1988.

[ISO 15] ISO International Organization for Standardization. *ISO/IEC 2382: Information technology – Vocabulary*. 2015.

[VDI 12] VDI Verein Deutscher Ingenieure. *VDI 4600: Kumulierter Energieaufwand (KEA) – Begriffe, Berechnungsmethoden*. 2012.

[VDI 16] VDI Verein Deutscher Ingenieure. *VDI 4800 Blatt 1: Ressourceneffizienz - Methodische Grundlagen, Prinzipien und Strategien*. 2016.

[VDI 18] VDI Verein Deutscher Ingenieure. *VDI 4800 Blatt 2: Ressourceneffizienz -Bewertung des Rohstoffaufwands*. 2018.

Internetquellen

[Arke18] M. van Arkel, M. Fiddrich, M. van Mierle, G. Müller-Loeffelholz, F. Thalhofer u. a. *Cloud-Computing*. 2018. URL: `https://www.duden.de/rechtschreibung/Cloud_Computing` (besucht am 13. 12. 2022).

[Clou18] Cloud Native Computing Foundation. *CNCF Cloud Native Definition: v1.0*. 2018. URL: `https://github.com/cncf/toc/blob/main/DEFINITION.md#deutsch` (besucht am 06. 10. 2023).

[DMG 23] DMG MORI Global Marketing GmbH. *DMC 850 V - Vertikalfräsen von DMG MORI*. 2023. URL: `https://de.dmgmori.com/produkte/maschinen/fraesen/vertikal-fraesen/dmc-v/dmc-850-v` (besucht am 05. 06. 2023).

[Dürr23] Dürr Aktiengesellschaft. *DXQanalyze - Dürr*. 2023. URL: `https://www.durr.com/en/products/software-controls/dxq/paint-shop-application-technology/dxqanalyze` (besucht am 02. 08. 2023).

[Fost02] I. Foster. *What is the Grid? A Three Point Checklist*. 2002. URL: `https://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf` (besucht am 29. 12. 2022).

[Gebr22] Gebr. Heller Maschinenfabrik GmbH. *HELLER CNC-ProfiTrainer: Die Bildungsmaschine*. 2022. URL: `https://www.heller.biz/de/maschinen-und-loesungen/cnc-profitrainer/` (besucht am 05. 06. 2023).

[Kube22a] Kubernetes. *Kubernetes Scheduler*. 2022. URL: `https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/` (besucht am 15. 06. 2023).

[Kube22b] Kubernetes. *Scheduling Framework*. 2022. URL: `https://kubernetes.io/docs/concepts/scheduling-eviction/scheduling-framework/` (besucht am 11. 01. 2023).

[Mora15] R. Mora. *Cisco iox: Making fog real for iot*. 2015. URL: `http://blogs.cisco.com/ioe/cisco-iox`.

[o V19] o. V. *On Premise vs. Serverless*. 2019. URL: `https://customers-love-solutions.com/?p=784` (besucht am 13. 12. 2022).

Eigene Veröffentlichungen

[Broc22] B. Brockhaus, F. Hoffmann, J. Metternich und M. Weigold. „Predictive Maintenance for Flexible Protective Covers in Machine Tools“. In: *Production at the Leading Edge of Technology*. Hrsg. von B.-A. Behrens, A. Brosius, W.-G. Drossel, W. Hintze, S. Ihlenfeldt u. a. Lecture Notes in Production Engineering. Cham: Springer International Publishing, 2022, S. 177–185.

[Broc24] B. Brockhaus, F. Gast und M. Weigold. „Modeling of an Edge Computing Cluster for Optimized Distribution of Tasks in Production Environments“. In: *Production at the Leading Edge of Technology*. Hrsg. von T. Bauernhansl, A. Verl, M. Liewald und H.-C. Möhring. Lecture Notes in Production Engineering. Cham: Springer Nature Switzerland, 2024, S. 44–53.

[Fert22a] A. Fertig, O. Kohn, B. Brockhaus und M. Weigold. „Consistent Contextualisation of Process and Quality Information for Machining Processes“. In: *Production at the Leading Edge of Technology*. Hrsg. von B.-A. Behrens, A. Brosius, W.-G. Drossel, W. Hintze, S. Ihlenfeldt u. a. Lecture Notes in Production Engineering. Cham: Springer International Publishing, 2022, S. 195–202.

[Webe21] M. Weber, B. Brockhaus und S. Dumss. *Edge-Computing im Projekt EuProGigant: Vision – Verständnis – Abgrenzung*. Hrsg. von M. Weigold. 2021.

[Webe22] M. Weber, B. Brockhaus, F. Hoffmann, H. Ranzau, M. Weigold u. a. „Anwendungen und Geschäftsmodelle mit Gaia-X: EuProGigant – Von der Anwendung zum Geschäftsmodell mit Gaia-X in der Domäne Industrie 4.0“. In: *wt Werkstattstechnik online* 112.01-02 (2022), S. 91–96.

Studentische Arbeiten

[Gark21] P. Garkusha. „Entwurf und Demonstration einer Systemarchitektur für Gaia-X-kompatibles Edge-Computing im Produktionsumfeld“. Masterthesis. Darmstadt: Technische Universität Darmstadt, 2021.

[Gast22] F. Gast. „Modellierung eines Rechnerverbunds für die optimale Verteilung von Tasks im Produktionsumfeld“. Masterthesis. Darmstadt: Technische Universität Darmstadt, 2022.

A Anhang

A.1 Validierungstasks


```
{
    "LatencyRequirement": 500,
    "SubTasks": [
        {
            "TaskName": "OPCUA-PT16",
            "TransportLayer": "TCP",
            "HardwareBind": "NCU2",
            "GPU": false,
            "CPU": 0,
            "RAM": 0,
            "DiskSpace": 0,
            "Predecessors": [],
            "SourceNodes": [],
            "SinkNodes": []
        },
        {
            "TaskName": "steam",
            "TransportLayer": "TCP",
            "originalHardwareBind": "EuPG-Edge-1",
            "GPU": false,
            "CPU": 1218.58,
            "RAM": 707.3792,
            "DiskSpace": 35.7,
            "Predecessors": [
                {
                    "Name": "OPCUA-PT16",
                    "Data": 0.0,
                    "ReqData_Start": 0.0
                }
            ],
            "SourceNodes": [],
            "SinkNodes": []
        },
        {
            "TaskName": "messe-datenweiche",
            "TransportLayer": "TCP",
            "originalHardwareBind": "EuPG-Edge-1",
            "GPU": false,
            "CPU": 900,
            "RAM": 1000,
```

```
            "DiskSpace": 25,
            "Predecessors": [
                {
                    "Name": "steam",
                    "Data": 1.0,
                    "ReqData_Start": 1.0
                }
            ],
            "SourceNodes": [],
            "SinkNodes": []
        },
        {
            "TaskName": "compliance service connector",
            "TransportLayer": "TCP",
            "originalHardwareBind": "EuPG-Edge-1",
            "GPU": false,
            "CPU": 0,
            "RAM": 8,
            "DiskSpace": 30,
            "Predecessors": [
                {
                    "Name": "messe-datenweiche",
                    "Data": 1.0,
                    "ReqData_Start": 1.0
                }
            ],
            "SourceNodes": [],
            "SinkNodes": []
        },
        {
            "TaskName": "so-publisher",
            "TransportLayer": "TCP",
            "originalHardwareBind": "EuPG-Edge-1",
            "GPU": false,
            "CPU": 788.43,
            "RAM": 70,
            "DiskSpace": 225,
            "Predecessors": [
                {
                    "Name": "messe-datenweiche",
                    "Data": 1.0,
                    "ReqData_Start": 1.0
                }
            ],
            "SourceNodes": [],
            "SinkNodes": []
        },
        {
            "TaskName": "sd-normalization",
            "TransportLayer": "TCP",
```

```
            "originalHardwareBind": "EuPG-Edge-1",
            "GPU": false,
            "CPU": 0,
            "RAM": 30,
            "DiskSpace": 50,
            "Predecessors": [
                {
                    "Name": "messe-datenweiche",
                    "Data": 1.0,
                    "ReqData_Start": 1.0
                }
            ],
            "SourceNodes": [],
            "SinkNodes": []
        },
        {
            "TaskName": "C8Y-OPCUA-Gateway",
            "TransportLayer": "TCP",
            "originalHardwareBind": "EuPG-Edge-1",
            "GPU": false,
            "CPU": 286.72,
            "RAM": 1561,
            "DiskSpace": 80,
            "Predecessors": [
                {
                    "Name": "OPCUA-PT16",
                    "Data": 0.015,
                    "ReqData_Start": 0.015
                }
            ],
            "SourceNodes": [],
            "SinkNodes": []
        }
    ]
}
```

A.2 Validierungscluster


```
{
    "InternetLayer": "IPv4",

    "Computers": [
        {
            "Name": "CELOS",
            "Allocatable": "YES",
            "CPU_perf": 9625,
            "CPU_cores": 4,
            "CPU_usage": 0.11,
            "RAM": 8000,
            "RAM_usage": 0.5525,
            "disk": 50000,
            "disk_usage": 0.84,
            "Machine": "DMG MORI DMC850V",
            "GPU_perf": 260000,
            "GPU_usage": 0.05
        },
        {
            "Name": "NCU1",
            "Allocatable": "YES",
            "CPU_perf": 3629,
            "CPU_cores": 2,
            "CPU_usage": 0.1,
            "RAM": 2000,
            "RAM_usage": 0.27,
            "disk": 8000,
            "disk_usage": 0.11,
            "Machine": "DMG MORI DMC850V",
            "GPU_perf": 0,
            "GPU_usage": 0.0
        },
        {
            "Name": "TCC-PC",
            "Allocatable": "YES",
            "CPU_perf": 1477,
            "CPU_cores": 4,
            "CPU_usage": 0.47,
            "RAM": 4000,
            "RAM_usage": 0.36,
            "disk": 32000,
            "disk_usage": 0.9,
            "Machine": "DMG MORI DMC850V",
            "GPU_perf": 25840,
            "GPU_usage": 0.05
        },
        {
```

```
            "Name": "DMC-MessPC",
            "Allocatable": "NO",
            "CPU_perf": 39191,
            "CPU_cores": 8,
            "CPU_usage": 0.05,
            "RAM": 32000,
            "RAM_usage": 0.28125,
            "disk": 6500000,
            "disk_usage": 0.1,
            "Machine": "DMG MORI DMC850V",
            "GPU_perf": 2560000,
            "GPU_usage": 0.05
        },
        {
            "Name": "Expert Transient",
            "Allocatable": "NO",
            "CPU_perf": 0,
            "CPU_cores": 0,
            "CPU_usage": 0.0,
            "RAM": 0,
            "RAM_usage": 0.0,
            "disk": 0,
            "disk_usage": 0.0,
            "Machine": "DMG MORI DMC850V",
            "GPU_perf": 0,
            "GPU_usage": 0.0
        },
        {
            "Name": "Sinumerik Edge1",
            "Allocatable": "YES",
            "CPU_perf": 1477,
            "CPU_cores": 4,
            "CPU_usage": 0.0,
            "RAM": 4000,
            "RAM_usage": 0.32,
            "disk": 32000,
            "disk_usage": 0.01,
            "Machine": "DMG MORI DMC850V",
            "GPU_perf": 25840,
            "GPU_usage": 0.0
        },
        {
            "Name": "ConditionAnalyzer-PC",
            "Allocatable": "YES",
            "CPU_perf": 1388,
            "CPU_cores": 4,
            "CPU_usage": 0.55,
            "RAM": 4000,
            "RAM_usage": 0.36,
            "disk": 256000,
```

```
        "disk_usage": 0.1,
        "Machine": "DMG MORI DMC850V",
        "GPU_perf": 0,
        "GPU_usage": 0.0
    },
    {
        "Name": "NCU2",
        "Allocatable": "YES",
        "CPU_perf": 3629,
        "CPU_cores": 2,
        "CPU_usage": 0.1,
        "RAM": 2000,
        "RAM_usage": 0.27,
        "disk": 8000,
        "disk_usage": 0.11,
        "Machine": "ProfiTrainer",
        "GPU_perf": 0,
        "GPU_usage": 0.0
    },
    {
        "Name": "Sinumerik Edge2",
        "Allocatable": "YES",
        "CPU_perf": 1477,
        "CPU_cores": 4,
        "CPU_usage": 0.43,
        "RAM": 4000,
        "RAM_usage": 0.32,
        "disk": 32000,
        "disk_usage": 0.01,
        "Machine": "ProfiTrainer",
        "GPU_perf": 25840,
        "GPU_usage": 0.0
    },
    {
        "Name": "EuPG-Edge-1",
        "Allocatable": "YES",
        "CPU_perf": 7168,
        "CPU_cores": 2,
        "CPU_usage": 0.0,
        "RAM": 32000,
        "RAM_usage": 0.15,
        "disk": 500000,
        "disk_usage": 0.15,
        "Machine": "ProfiTrainer",
        "GPU_perf": 0,
        "GPU_usage": 0.0
    },
    {
        "Name": "TCU",
        "Allocatable": "NO",
```

```
            "CPU_perf": 0,
            "CPU_cores": 0,
            "CPU_usage": 0.0,
            "RAM": 0,
            "RAM_usage": 0.0,
            "disk": 0,
            "disk_usage": 0.0,
            "Machine": "ProfiTrainer",
            "GPU_perf": 0,
            "GPU_usage": 0.0
        },
        {
            "Name": "SensoSchu-PC",
            "Allocatable": "NO",
            "CPU_perf": 35822,
            "CPU_cores": 8,
            "CPU_usage": 0.18,
            "RAM": 16000,
            "RAM_usage": 0.29375,
            "disk": 6000000,
            "disk_usage": 0.37,
            "Machine": "",
            "GPU_perf": 643355,
            "GPU_usage": 0.05
        }
    ],
    "Switches": [
        {
            "Name": "Switch_1",
            "Bandwidth": 1000,
            "usage": 0.0
        },
        {
            "Name": "Switch_2",
            "Bandwidth": 16000,
            "usage": 0.0
        },
        {
            "Name": "Switch_3",
            "Bandwidth": 20000,
            "usage": 0.0
        },
        {
            "Name": "Switch_4",
            "Bandwidth": 20,
            "usage": 0.0
        },
        {
            "Name": "PTW_Switch",
            "Bandwidth": 16000,
```

```
            "usage": 0.0
        }
    ],
    "Nodes": [
        {
            "Name": "PTW",
            "Label": "Sink"
        }
    ],
    "Connections": [
        {
            "FromName": "CELOS",
            "ToName": "Switch_1",
            "LinkLayer": "Ethernet802_3",
            "Bandwidth": 100,
            "usage": 0.0,
            "PacketLossProbability": 0.01
        },
        {
            "FromName": "NCU1",
            "ToName": "Switch_1",
            "LinkLayer": "Ethernet802_3",
            "Bandwidth": 100,
            "usage": 0.0,
            "PacketLossProbability": 0.01
        },
        {
            "FromName": "TCC-PC",
            "ToName": "Switch_1",
            "LinkLayer": "Ethernet802_3",
            "Bandwidth": 100,
            "usage": 0.0,
            "PacketLossProbability": 0.01
        },
        {
            "FromName": "Sinumerik Edge1",
            "ToName": "Switch_1",
            "LinkLayer": "Ethernet802_3",
            "Bandwidth": 100,
            "usage": 0.0,
            "PacketLossProbability": 0.01
        },
        {
            "FromName": "Switch_1",
            "ToName": "Switch_2",
            "LinkLayer": "Ethernet802_3",
            "Bandwidth": 100,
            "usage": 0.0,
            "PacketLossProbability": 0.01
        },
```

```
    {
        "FromName": "DMC-MessPC",
        "ToName": "Switch_2",
        "LinkLayer": "Ethernet802_3",
        "Bandwidth": 1000,
        "usage": 0.0,
        "PacketLossProbability": 0.01
    },
    {
        "FromName": "Expert Transient",
        "ToName": "Switch_2",
        "LinkLayer": "Ethernet802_3",
        "Bandwidth": 1000,
        "usage": 0.0,
        "PacketLossProbability": 0.01
    },
    {
        "FromName": "Switch_3",
        "ToName": "Switch_2",
        "LinkLayer": "Ethernet802_3",
        "Bandwidth": 1000,
        "usage": 0.0,
        "PacketLossProbability": 0.01
    },
    {
        "FromName": "NCU2",
        "ToName": "Switch_3",
        "LinkLayer": "Ethernet802_3",
        "Bandwidth": 1000,
        "usage": 0.0,
        "PacketLossProbability": 0.01
    },
    {
        "FromName": "NCU2",
        "ToName": "Switch_4",
        "LinkLayer": "Ethernet802_3",
        "Bandwidth": 10,
        "usage": 0.0,
        "PacketLossProbability": 0.01
    },
    {
        "FromName": "TCU",
        "ToName": "Switch_4",
        "LinkLayer": "Ethernet802_3",
        "Bandwidth": 10,
        "usage": 0.0,
        "PacketLossProbability": 0.01
    },
    {
        "FromName": "Sinumerik Edge2",
```

```
            "ToName": "Switch_3",
            "LinkLayer": "Ethernet802_3",
            "Bandwidth": 1000,
            "usage": 0.0,
            "PacketLossProbability": 0.01
        },
        {
            "FromName": "EuPG-Edge-1",
            "ToName": "Switch_3",
            "LinkLayer": "Ethernet802_3",
            "Bandwidth": 1000,
            "usage": 0.0,
            "PacketLossProbability": 0.01
        },
        {
            "FromName": "Switch_2",
            "ToName": "PTW_Switch",
            "LinkLayer": "Ethernet802_3",
            "Bandwidth": 1000,
            "usage": 0.0,
            "PacketLossProbability": 0.01
        },
        {
            "FromName": "ConditionAnalyzer-PC",
            "ToName": "PTW_Switch",
            "LinkLayer": "Ethernet802_3",
            "Bandwidth": 1000,
            "usage": 0.0,
            "PacketLossProbability": 0.01
        },
        {
            "FromName": "SensoSchu-PC",
            "ToName": "Switch_2",
            "LinkLayer": "Ethernet802_3",
            "Bandwidth": 1000,
            "usage": 0.0,
            "PacketLossProbability": 0.01
        },
        {
            "FromName": "PTW_Switch",
            "ToName": "PTW",
            "LinkLayer": "Ethernet802_3",
            "Bandwidth": 1000,
            "usage": 0.0,
            "PacketLossProbability": 0.01
        }
    ]
}
```