universität innsbruck

Institut für Informatik

Heuristische Optimierung der Operatorplatzierung in verteilten Stream-Verarbeitungssystemen

Cedric Immanuel Sillaber
Matrikel-Nr: 12211124

VU Einführung in das wissenschaftliche Arbeiten
Sommersemester 2024

Betreuer:
Prof. Dipl.-Ing. Dr. Thomas Fahringer

22. Juni 2024

Zusammenfassung

In den vergangenen Jahren wurden Big Data Applikationen stets populärer. Da die Anzahl der Daten umfangreicher wird, werden effiziente Ansätze für verteilte Stream-Datenverarbeitung (SVS) benötigt. Ein wesentlicher Performance-Faktor dabei ist das Problem der Operatorplatzierung. Für das Lösen dieses Problems gibt es jedoch keine effiziente Lösung. Diese Arbeit beschäftigt sich mit effizienten heuristischen Methoden, die die optimale Lösung approximieren.

1 Revision der Seminararbeit auf Basis der Gutachten

Reviewer 2 hat mehrere Hinweise zur Kohärenz zwischen deutschen und englischen Begriffen gegeben. Beispielsweise wurde im Originaltext 'Ressourcenmodell' fälschlicherweise als 'Ressourcen Modell' geschrieben. Das gleiche Problem bestand beim "Datenstrommodell". Diese Begriffe wurden nun korrekt zusammen geschrieben.

Es gab auch Inkonsistenzen bei den Namen der Heuristiken. Die Bezeichnungen wurden entsprechend den Hinweisen auf Tabusucheünd "Greedy First-Fitängepasst. Ein weiterer Hinweis betraf die Vereinheitlichung des Separators in den Begriffen NP-schweründ "Greedy First-Fit". Hier stimme ich nicht zu, da für NP-schwertypischerweise ein Bindestrich verwendet wird.

Nach der Revision durch Reviewer 1 wurde ein kurzer Überblick über die Struktur der Arbeit hinzugefügt. Außerdem wurden die Abschnitte 4.1 und 4.2 überarbeitet. Abbildung 2 wurde vergrößert und besser strukturiert. Wie Reviewer 1 angemerkt hat, war die Tabelle 1 etwas unverständlich, daher wurde sie überarbeitet.

2 Einführung

Im Zuge der fortschreitenden Digitalisierung hat sich die Datenverarbeitung zu einem zentralen Aspekt der modernen Welt entwickelt. Der Erfolg vieler Unternehmen beruht auf der Fähigkeit, kontinuierlich anfallende Datenmengen effizient zu verarbeiten. Rund um die Uhr werden Daten gesammelt, die in Echtzeit verarbeitet werden müssen. Verteilte Stream-Verarbeitungssysteme (SVS) sammeln, filtern und verarbeiten Daten. Die Daten stammen von einer Vielzahl an Geräten. Derartige Systeme werden beispielsweise in der Analyse von Finanzmärkten [10], Verarbeitung von sozialen Netzwerk-Interaktionen und der Beobachtung von Network-Traffic [10] eingesetzt.

Ein SVS besteht aus einer Menge von unabhängiger Operatoren, die spezifische Funktionalitäten ausführen. Diese Unabhängigkeit ermöglicht es, dass die Rechner im Netzwerk (Cloud-Edge [4]) lokalisiert werden können, anstatt in der Cloud. In solch einem System stehen zahlreiche Ressourcen zur Verfügung. Als Operatorplatzierung bezeichnet man das Problem, die Operatoren im System optimal auf verfügbare Knoten zu platzieren. Die Lösung dieses Problems ist jedoch NP-hard [2].

Abbildung 1 zeigt ein SVS. Eine Anzahl von Datenquellen P produziert Daten, die zu den Senken C geleitet werden. Die Operatoren werden in der Abbildung als $SBON$ bezeichnet. Das Operatorplatzierungproblem (OPP) beschäftigt sich mit der Frage, welche physischen Knoten die Operatoren aufnehmen sollen [11].

Für die Evaluation solcher Modelle werden diverse Quality-of-Service (QoS) Attribute herangezogen. Dazu gehören Durchsatz, Ende-zu-Ende-Latenz und Verfügbarkeit [9, 2]. Der Ansatz in dieser Arbeit versucht sich auf generalisierte QoS-Attribute zu fokussieren, die einfach angepasst werden können [9].

Diese Arbeit fokussiert sich auf einen Ansatz, der bekannte Heuristiken kombiniert und das Optimum effizient approximiert. Der Ansatz bezieht sich auf keine spezifische SVS Implementierung und ist somit *modell-frei*. Folglich kann diese Lösung für verschiedene Systeme angewendet werden. Die optimale systematische Position in solch einem System stellt einen maßgeblichen Performance-Faktor dar.

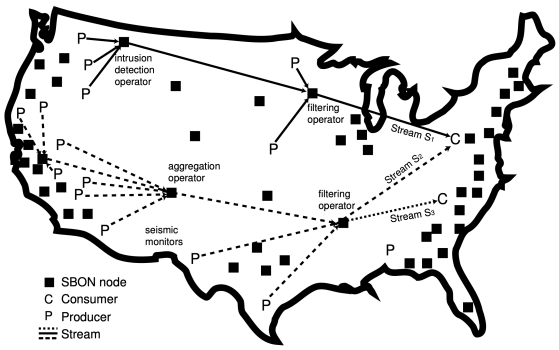


Abbildung 1: Stream Verarbeitungssystem, Quelle: [11].

Die Arbeit ist wie folgt strukturiert: In Abschnitt 3 werden verwandte Arbeiten vorgestellt. Darauf folgt ein Abschnitt mit den wichtigsten Definition für die Formulierung des Optimierungsproblems. Anschließend werden Heursitiken vorgestellt, die das Optimierungsproblem approximieren. In Abschnitt 6 werden die vorgestellten Heuristiken in einem Experiment evaluiert.

3 Verwandte Arbeiten

Das Problem der Operatorplatzierung (OPP) wurde aufgrund seiner Bedeutung in der Forschung umfassend untersucht. In der Literatur werden verschiedene Aspekte der Operatorplatzierung beleuchtet, wie in den Arbeiten von [7, 13] zusammengefasst. Insbesondere konzentriert sich die Forschung auf drei Hauptbereiche: (1) Ziele der optimalen Platzierung, (2) unterschiedliche Optimierungstechniken, und (3) verschiedene Infrastrukturen der verteilten Systeme. Im Hinblick auf Optimierungstechniken (Punkt 2) haben mehrere Studien verschiedene Ansätze untersucht. Greedy Heuristiken, wie in [8] beschrieben, bieten eine vorhersagbare und oft schnelle Lösung, die jedoch nicht immer zu optimalen Ergebnissen führt. Ein weiterer Forschungsansatz befasst sich mit Metaheuristiken zur Optimierung des OPP. Arbeiten wie [12] und [3] präsentieren Methoden, die auf der lokalen Suche basieren. Diese Ansätze zielen darauf ab, durch iterative Verbesserungen und das Durchsuchen des Lösungsraums bessere Platzierungsentscheidungen zu treffen.

4 Definitionen

Um eine konkrete Optimierung des Problems beschreiben zu können, ist es notwendig, die Problematik formal auszudrücken. Im Folgenden wird eine formale Abstraktion des OPP vorgestellt. Zuerst werden das Datenstrom- und das Ressourcenmodell (4.1) definiert. Beide Modelle werden mittels Graphentheorie formuliert und beschreiben ein SVS. Anschließend wird das Problem der Operatorplatzierung mithilfe der zuvor definierten Modelle formuliert (4.2).

4.1 Definition von SVS

SVSs basieren auf einer Anzahl an verteilten Computer-Ressourcen, die zusammen ein komplexes System ergeben. Dieser Sachverhalt kann mittels Graphentheorie beschrieben werden. Grundsätzlich gibt es zwei Abstraktionen solcher Systeme: das *Datenstrommodell* und das *Ressourcenmodell*. Beide Systeme werden mit gerichteten, gewichteten zyklenfreien Graphen $G = (V, E)$ dargestellt.

Datenstrommodelle werden durch $G_{svs} = (V_{svs}, E_{svs})$ beschrieben. Hier wird der Fluss der Daten definiert. In diesem Modell beschreiben Knoten $u \in V_{svs}$ Operatoren , Quellen und Senken im System. Datenquellen stellen Knoten dar, die kontinuierlich Daten produzieren. Produzierte Daten werden über Operatoren verarbeitet und zu den Datenbecken geschickt. Datenbecken stellen einen Endpunkt im System dar. Nachdem Daten von den verteilten Rechnern verarbeitet wurden, werden sie in den Datenbecken gespeichert. Zu den Operatoren gehören auch sogenannte *angepinnte* Operatoren [9], die Datenquellen und -becken beinhalten. Kanten $(u, v) \in E_{svs}$ beschreiben Datenverkehr zwischen den Operatoren u und v. Ein Stream ist ein kontinuierliche Sequenz von Daten.

Ressourcenmodelle werden durch den Graphen $G_{res} = (V_{res}, E_{res})$ dargestellt. Dabei wird der logische Zusammenschluss zwischen verfügbaren Computing Ressourcen beschrieben. Der Knoten $u \in V_{res}$ repräsentiert solch eine Ressource. In diesem Modell beschreiben Kanten $(u, v) \in E_{res}$ eine logische Verknüpfung zwischen den Rechnerressourcen u und v.

4.1.1 Angepinnte Operatoren

Operatoren, die fest an eine bestimmte Stelle gebunden sind, werden als *angepinnte Operatoren* bezeichnet. Datenquellen und Datensenken sind Beispiele für angepinnte Operatoren. *Freie Operatoren* hingegen können an beliebigen Rechnerressourcen platziert werden. Diese Operatoren befinden sich zwischen Datenquellen und Datensenken und führen Funktionen wie *Join*, *Filter* und *Select* aus [11].

In Abbildung 1 sind die mit *SBON* bezeichneten Knoten die freien Operatoren. Diese umfassen Filter-, Aggregations- und Detektionsoperatoren.

4.2 Problemformulierung des OPP

Ein Operator kann aus verschiedenen Gründen nicht auf jedem Knoten im Datenstrommodell platziert werden. Diese Gründe können politischer, sicherheitstechnischer oder topologischer Natur sein [2]. Weitere mögliche Gründe sind beispielsweise technische Restriktionen oder organisatorische Vorschriften. Für jeden Operator $i \in V_{svs}$ gibt es eine Menge an Kandidatressourcen V_{res}^i (Schreibweise wie in [9]).

Das OPP kann als Zuordnungsproblem zwischen den genannten Modellen formuliert werden. Die Zuordnungsfunktion wird eingeschränkt, um die zu minimierenden QoS-Attribute zu berücksichtigen. Folglich wird der optimale Kandidat u in den Kandidatenressourcen V_{res}^i gesucht, damit der Operator i auf Knoten u platziert wird. Hierbei bezieht sich das Problem auf die Inkonsistenz zwischen logisch benachbarten Operatoren im Ressourcenmodell G_{res} und optimalen Entscheidungen der Operatoren im Datenstrom Modell G_{svs}.

Um das Problem formal zu definieren, verwenden wir den binären Ausdruck $x_{i,u}$ $i \in V_{svs}, u \in V_{res} : x_{i,u} = 1$ wenn Operator i auf dem Rechner u platziert wird, andernfalls $x_{i,u} = 0$.

$$\arg\min_x F(x)$$

$$\sum_{i \in V_{svs}} C_i x_{i,u} < C_u \quad \forall u \in V_{res}$$

$$\sum_{u \in V_{res}^i} x_{i,u} = 1 \quad \forall i \in V_{dsp}$$

$$x_{i,u} \in \{0,1\} \quad \forall i \in V_{svs}, u \in V_{res}^i$$

Hierbei wird die Funktion $F(x)$ bezüglich ausgewählter QoS-Attribute minimiert. Dieses Multi-Objective-Optimierungsproblem wird durch die Simple Additive Weight-Technik [15] in ein Single-Objective-Problem umgewandelt. Für typische QoS Attribute wie Antwortzeit, Verfügbarkeit und Netzwerkverwendung [9] wird die Funktion $F(x)$ wie folgt definiert:

$$F(x) = w_r \frac{R(x) - R_{min}}{R_{max} - R_{min}} + w_a \frac{logA_{max} - logA(x)}{logA_{max} - logA_{min}} + w_z \frac{Z(x) - Z_{min}}{Z_max - Z_min}$$

wo $R(x)$, $A(x)$ und $Z(x)$ die QoS-Attribute und $w_r, w_a, w_z \geq 0$ Gewichtungen sind. R_{min}, R_{max}, A_{min}, A_{max}, Z_{min} und Z_{max} sind die minimalen und maximalen Werte der QoS Attribute. Es gilt $w_r + w_a + w_z = 1$.

Mithilfe einer **Straffunktion** werden die Verbindungen zwischen zwei spezifischen Knoten bezüglich der QoS-Attribute bewertet. Dadurch wird ein Vergleich der Rechnerressourcen möglich. Dabei werden die Links zwischen $u \in V_{res}^i$ möglich. Auf die Straffunktion wird im Folgenden eingegangen.

5 Heuristiken

Wie in [2] gezeigt, ist das OPP NP-schwer. Da die initiale Platzierung der Operatoren ein tragender Faktor in der Performance ist, werden effiziente Heuristiken benötigt. Unser Ansatz beinhaltet eine Kombination mehrerer bekannter Heuristiken, die die Funktion F aus 4.2 minimieren. Eine Heuristik ist ein Verfahren, das eine schnelle Lösung für ein Problem liefert, jedoch nicht zwingend optimal ist.

Greedy First-Fit, lokale Suche und Tabusuche sind dabei die gängigen Methoden. Greedy First-Fit Ansätze können schnell eine Lösung approximieren, jedoch sind diese nicht immer optimal. Deshalb wird er mit zur lokalen Suche erweitert. Für Lösungen höherer Qualität wird die lokale Suche zur Tabusuche erweitert.

5.1 Straffunktion

Die Auswahl von verschiedenen Möglichkeiten $u \in V_{res}^i$ bringt Einbußen mit sich. Da die Daten übertragen werden müssen, kommen unvorhersehbare Netzwerklatenzen hinzu. Hierbei müssen Netzwerklatenz, Bandbreite und Netzwerkgeschwindigkeit [9] berücksichtigt werden. Um die Platzierungsentscheidungen zu lenken, benötigen wir eine Metrik, die die Kosten der Nutzung bestimmter Knoten/Link-Ressourcen erfasst. In einem idealen Szenario könnten wir alle Operatoren auf einem einzigen Knoten mit unendlicher Kapazität und Verfügbarkeit platzieren. Da dies in der Praxis nicht möglich ist, müssen die Operatoren auf mehrere Knoten verteilt werden, was zu Netzwerklatenzen und Netzwerkverkehr führt. Wir führen eine Linkstraffunktion $\delta(u, v) \in [0, 1]$ ein, die die Verschlechterung der Performance im Vergleich zur idealen Ressource misst. Diese Funktion ist eine gewichtete Kombination der QoS-Attribute des Links (u, v) und der verbundenen Knoten u und v:

$$\delta(u, v) = w_r\delta_R(u, v) + w_a\delta_A(u, v) + w_z\delta_Z(u, v),$$

wobei $w_r, w_a, w_z \in [0, 1]$ die Gewichte der einzelnen QoS-Metriken sind. Die Terme $\delta_R(u, v)$, $\delta_A(u, v)$ und $\delta_Z(u, v)$ modellieren die Strafe in Bezug auf Anwendungsantwortzeit, Verfügbarkeit und Netzwerknutzung.

Diese Strafen helfen, die Qualität und Effizienz der Ressourcenverteilung in einem verteilten System zu bewerten und zu optimieren.

5.2 Greedy First-Fit

Diese Heuristik wird für das Bin-Packing Problem verwendet, um eine optimale Lösung anzunähern [5]. Andere Arbeiten befassten sich schon mit der Anwendung dieser Methode für das OPP [1, 14]. Wir passen diese Methode an, um die Operatoren auf die verfügbaren Ressourcen zu platzieren. Für jeden Operator werden die verfügbaren Ressourcen $v \in V_{res}$ in einer Liste sortiert. Die Sortierung basiert auf der summierten Straffunktion δ zwischen v und den angepinnten Operatoren P. Folglich:

$$u_i \in P, \text{ wobei P = angepinnte Operatoren}$$
$$\sum_{i=0}^{|P|} \delta(v, u)$$

Ausgewählt wird der erste Rechner in der Liste, den der Operator aufnehmen kann. Mittels Breitensuche werden für alle Operatoren Platzierungen bestimmt. Die daraus resultierenden Operatorplatzierungen werden als eine Konfiguration bezeichnet.

5.3 Lokale Suche

Die lokale Suche ist eine iterative Heuristik zur Optimierung von Lösungen in der Nachbarschaft einer Ausgangskonfiguration. Dabei iteriert diese Methode über die Konfiguration, die aus der Greedy-First-Fit-Heuristik resultiert. Der Algorithmus 1 durchläuft drei Schritte: Zunächst wird eine initiale Konfiguration durch den Greedy First-Fit berechnet. Anschließend wird eine Nachbarschaft von ähnlichen Konfigurationen bestimmt, und schließlich wird die beste Lösung in dieser Nachbarschaft gefunden. Um die Nachbarschaft zu bestimmen, wird zunächst eine sortierte Liste L erstellt, basierend auf der Straffunktion δ wie in 5.2. Mit dieser Liste wird dann eine initiale Konfiguration S durch den Greedy First-Fit Algorithmus erstellt (Zeile 6).

Solange bessere Platzierungen der Operatoren gefunden werden, iteriert die lokale Suche über die Konfigurationen (Zeile (14)). Eine Verbesserung wird durch einen niedrigeren Wert der Zielfunktion F definiert. Die Suche nach besseren Konfigurationen basiert auf den drei Funktionalitäten *co-locate, swap und relocate.*

Algorithm 1 Local Search

1: **function** localSearch(G_{dsp}, G_{res})
2: **Input**: G_{dsp}, DSP application graph
3: **Input**: G_{res}, computing resource graph
4: $P \leftarrow$ resources hosting the pinned operators of G_{dsp}
5: $L \leftarrow$ resources of G_{res}, sorted by the cumulative link penalty with respect to nodes in P
6: link penalty with respect to nodes in P
7: $S \leftarrow$ solve GreedyFirstFit(G_{dsp}, L)
8: **do**
9: $F \leftarrow$ value of the objective function for S
10: $S \leftarrow$ improve S by co-locating operators
11: $S \leftarrow$ improve S by swapping resources
12: $S \leftarrow$ improve S by relocating a single operator
13: $F' \leftarrow$ value of the objective function for S
14: **while** $F' < F$ **do**
15: **return** S
16: **end function**

Zwei Operatoren $i, j \in G_{svs}$, wobei i auf $u \in G_{res}$ und j auf $v \in G_{res}$ platziert ist, werden als *co-located* bezeichnet, wenn i und j zusammen auf einem Rechner platziert sind. Somit befindet sich i und j auf entweder u oder v.
Beim *swap* wird der Operator i, der auf $u \in G_{res}$ platziert ist, auf eine neue Ressource $v \in G_{res}$ aus L verschoben. Für den Fall, dass zuerst n Operatoren $j_1, j_2, ..., j_n$auf u alloziert sind, werden diese auf v verschoben.
Die Funktionalität *relocate* bewegt nur einen einzelnen Operator i von $u \in G_{res}$ zu $v \in G_{res}$, wobei v aus L ausgewählt wird.

Das Zusammenführen von Operatoren auf eine gemeinsame Rechnerressource ermöglicht es, verwandte Aufgaben in einer besseren Lokalität zu platzieren, was die QoS-Attribute verbessern kann. Das Austauschen von Operatoren strebt eine Eliminierung/Minimierung von Ressourcen bedingten Engpässen an. Durch das Bewegen einzelner Operatoren werden somit auch lokale Engpässe minimiert. Dafür werden die Konfigurationen so angepasst, dass Datenströme gleichmäßig verteilt werden. Sobald keine Verbesserungen mehr gefunden werden, terminiert die lokale Suche und gibt die beste Konfiguration zurück. Dabei ist es wichtig zu beachten, dass diese Heuristik, wie 5.2, nur eine Approximation ist und in einer lokal-optimalen Lösung enden kann.

5.4 Tabusuche

Die heuristische lokale Suche ist abhängig von der initialen Konfiguration und terminiert nur bedingt mit einer globalen optimalen Lösung für das OPP. Um dem entgegenzuwirken, wird die Heuristik zur *Tabusuche* [6] erweitert. In dieser Methode wird über mehrere Anfangskonfigurationen iteriert, die anhand der lokalen Suche nicht unmittelbar zu einer Verbesserung führen. Durch diese Erweiterung werden Lösungen evaluiert, die bei einer lokalen Suche nicht betrachtet werden.

Algorithm 2 Tabu Search

```
1:  function tabuSearch(G_dsp, G_res)
2:  Input: G_dsp, DSP application graph
3:  Input: G_res, computing resource graph
4:  S* ← undefiniert
5:  F* ← ∞
6:  S' ← localSearch(G_dsp, G_res) //local optimum
7:  F' ← objective function value for S'
8:  S ← S'
9:  tabuList ← create new tabu list and append S
10: do
11:     improvement ← false
12:     S ← local search for S, excluding solutions in tabuList
13:     if F = F* and S ∉ tabuList then tabuList.append(S)
14:     end if
15:     if F < F* and S ∉ tabuList then
16:         S* ← S; F* ← F
17:         tabuList.append(S)
18:         improvement ← true
19:     end if
20:     limit tabuList to the latest tabuList_max placement configurations
21: while improvement
22: if F' < F* then S* ← S' end if
23: return S*
```

Der Algorithmus startet mit einer Ausgangskonfiguration S', die von der Greedy First-Fit-Heuristik erstellt wurde (Zeile 6).
Von Zeile 10 bis 21 durchläuft er eine Schleife, die in jeder Iteration nach einer besseren Lösung S in der Nachbarschaft sucht. Die Suche wird mittels lokaler Suche durchgeführt. Dabei werden Lösungen aus S ausgeschlossen, die in der Tabuliste enthalten sind (Zeile 12).

Die Tabuliste *tabuList* ist anfangs leer und beinhaltet nach der ersten Iteration die letzten $tabuList_{max}$ Platzierungen. Dadurch wird verhindert, dass die Schleife unendlich iteriert. In jeder Iteration wird die neue Lösung S mit der Zielfunktion F evaluiert (Zeile 13). Wenn die jetzige Lösung S besser als die vorherige Lösung S^* ist, wird S als die beste Lösung S^* gespeichert (Zeile 15, 16).

Sobald die Schleife terminiert ist, wird die neue beste Lösung S^* mit der Ausgangslösung S' verglichen. Die insgesamt beste Lösung wird zurückgegeben.

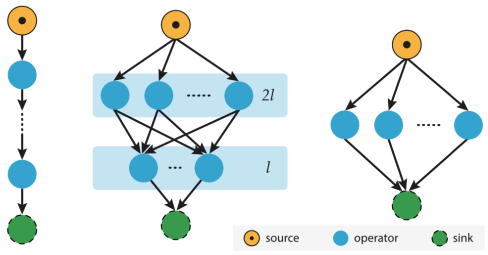
Mit dieser Heuristik wird der Lösungsraum nach besseren Approximationen des Optimums durchsucht. Im Vergleich zur Greedy First-Fit 5.2 und der lokalen Suche 5.3 bietet die Tabusuche eine erweiterte Fähigkeit zur Exploration des Lösungsraums und kann somit zu verbesserten Lösungen führen.

6 Evaluation

Die oben angeführten Heuristiken 5 werden in einem Experiment aus [9] evaluiert. Für das Experiment werden die drei Heuristiken Greedy First-Fit, Greedy First-Fit ohne Straffunktion und die lokale Suche verglichen. Als Bezugsnorm wird die optimale Lösung des Operatorplatzierungsproblems verwendet. Im Experiment von [9] werden mehrere Heuristiken bewertet, die hier der Einfachheit halber nicht aufgeführt werden. Auch die genauen Details der Hardware sind nicht relevant für die Evaluation.

6.1 Infrastruktur

Für die Untersuchung werden drei verschiedene Netzwerktopologien herangezogen. Diese Topologien beschreiben die verteilten Infrastrukturen der SVS. Diesbezüglich werden die Rechnerressourcen und deren Verbindungen definiert, sowie die Inter-Knoten-Latenz [9]. Die drei untersuchten Topologien sind in Abbildung 2 dargestellt. Dabei werden einfache sequentielle, replizierte und Diamant-Topologien betrachtet.



In den dargestellten Topologien hat jede Schicht mindestens einen Operator. Die erste und letzte Schicht verfügen jeweils über einen Operator, die Quelle und das Becken. Alle Ressourcentopologien haben die gleiche Anzahl an Ressourcen.

Abbildung 2: Sequentielle, replizierte und Diamant-Topologie, Quelle: [9].

6.2 Experimentelle Ergebnisse

Für das Experiment werden die Laufzeit (LZ) und die Qualitätseinbuße (QE) betrachtet [9]. Diese Merkmale werden mit der optimalen Lösung verglichen. rt beschreibt die benötigte Zeit, um eine Lösung zu finden, pd beschreibt die Abweichung von der optimalen Lösung. Zusätzlich wird der Beschleunigungsfaktor (bf) definiert als $bf = rt_{optimal}/rt_i$, wobei rt_i die Zeit für die Heuristik i ist. Le wird definiert als $le_i = \frac{F_i - F_{ODP}}{1 - F_{ODP}}$ [9], wobei F_i die Zielfunktion der Heuristik i ist.

Wie in 4.2 definiert, werden im Experiment Verfügbarkeit, Netzwerklatenz und Antwortzeit als QoS selektiert und folglich mit der Straffunktion minimiert.

Die Berechnung der Optimallösung weist eine effiziente Laufzeit für Diamant-Topologien auf. Diese Berechnung ist in unter einer Sekunde verfügbar. Bei replizierten und sequentiellen Topologien benötigt die Berechnung deutlich länger. Für replizierte Topologien dauert die Evaluation mehr als 8 Stunden (=32193 aus Tabelle).

Tabelle 1: Evaluation der Heuristiken

Methode		Diamand	Sequentiell	Repliziert	Durchschnitt
ODP	LZ	0.1	41.4	915.2	
	LZ	0.8	2174.8	32193.9	
Lokale Suche	BF	0.68	150.54	353.07	215.81
	QE	0%	1%	4%	1%
Tabusuche	BF	0.31	65.91	64.93	83.53
	QE	0%	1%	4%	1%
Greedy First-Fit	BF	454.40	$56 \cdot 10^4$	$12 \cdot 10^6$	$11 \cdot 10^6$
	QE	0%	7%	5%	11%
Greedy First-Fit (keine δ)	BF	454.40	$56 \cdot 10^4$	$12 \cdot 10^6$	$11 \cdot 10^6$
	QE	34%	7%	24%	19%

Tabelle aus des Experiments [9], Vergleich für dre Topologien (Diamand, Sequentiell, Repliziert)
BF = Beschleunigungsfaktor, QE = Leistungseinbuße, LZ = Laufzeit

Aus Tabelle 1 ist zu erkennen, dass Greedy First-Fit die schnellste Lösung bietet. Da die übrigen Heuristiken auf Greedy First-Fit basieren, entspricht das auch den Erwartungen. Trotz des hohen Beschleunigungsfaktors bf liefern Greedy First-Fit und Greedy First-Fit (ohne δ) degradierte Lösungen. Somit ergibt sich ein Kompromiss zwischen der Qualität der Lösung und der Laufzeit des Algorithmus. Auffällig ist die Diskrepanz zwischen der Greedy First-Fit Implementierung mit und ohne Straffunktion δ. Bei Beschleunigungsfaktoren ähnlicher Größenordnung resultieren die verschiedenen Implementierungen in unterschiedlichen Qualitäten der Lösung, besonders bei Diamant- und replizierten Topologien.

Lokale Suche ist im Experiment schneller als Tabusuche, jedoch ist die Qualität der Lösung geringer. Somit bietet die Tabusuche die beste Operatorplatzierung, mit der längsten Laufzeit.

Trotzdem stellt sich die Berechnung als viel effizienter heraus als die optimale Lösung. Während die Laufzeit der optimalen Lösung für replizierte Topologien mehr als 8 Stunden beträgt, benötigt die Tabus uche 8,2 Minuten. Die lokale Suche findet in 1,5 Minuten eine Lösung. Dabei beträgt die Qualitätseinbuße bei beiden Methoden 1%. In der Degradation der Performance ist bei der lokalen und der Tabusuche kein klarer Unterschied zu erkennen.

Es gibt in diesem Experiment keinen Gewinner. Zwischen den Heuristiken gibt es Kompromisse zwischen Schnelligkeit und Genauigkeit.

7 Konklusion und Ausblick

In dieser Arbeit wurden mehrere Heuristiken vorgestellt, die das NP-schwere Problem der Operatorplatzierung approximieren. Dabei handelt es sich um bekannte Methoden, die auch für andere NP-schwere Probleme genutzt werden. Mittels einer Straffunktion wurden die Heuristiken auf QoS-Attribute optimiert.
Die vorgestellten Heuristiken basieren auf dem Greedy First-Fit Ansatz und erweitern dessen Funktionalität durch Methoden wie lokale Suche und Tabusuche. Diese Heuristiken wurden in einem Experiment für drei verschiedene SVS-Architekturen evaluiert. Bei der Evaluation wurden sowohl die Laufzeit als auch die Qualität der Lösungen betrachtet. Es zeigte sich, dass keine der Heuristiken eine klare Überlegenheit aufweist. Beispielsweise führte die Greedy First-Fit Heuristik zu schnellen, jedoch qualitativ minderwertigen Lösungen, während die lokale Suche und die Tabusuche qualitativ bessere Platzierungen erreichten, jedoch mehr Zeit benötigten. Dennoch zeigten alle Heuristiken eine deutliche Verkürzung der Laufzeit im Vergleich zur Berechnung der optimalen Lösung.

Zukünftig werden wir komplexere Heuristiken entwickeln, die das OPP-Problem noch besser approximieren können. Die vorgestellten Methoden wurden zur Kompilierzeit verwendet, was bei dynamischen Veränderungen ineffizient ist. Deshalb wollen wir uns auf Heuristiken fokussieren, die zur Laufzeit angepasst werden können. Zudem wollen wir die erneute Konfiguration während der Laufzeit analysieren, damit Big Data-Anwendungen auf verändernde Bedingungen reagieren können.

Literatur

[1] Leonardo Aniello, Roberto Baldoni, and Leonardo Querzoni. Adaptive online scheduling in storm. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems*, DEBS '13, page 207–218, New York, NY, USA, 2013. Association for Computing Machinery.

[2] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti, and Matteo Nardelli. Optimal operator placement for distributed stream processing applications. In *Proceedings of the 10th ACM International Conference on Distributed and Event-Based Systems*, DEBS '16, page 69–80, New York, NY, USA, 2016. Association for Computing Machinery.

[3] Badrish Chandramouli, Jonathan Goldstein, Roger Barga, Mirek Riedewald, and Ivo Santos. Accurate latency estimation in a distributed event processing system. In *2011 IEEE 27th International Conference on Data Engineering*, pages 255–266, 2011.

[4] Marcos Dias de Assunção, Alexandre da Silva Veith, and Rajkumar Buyya. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *Journal of Network and Computer Applications*, 103:1–17, 2018.

[5] M. R. Garey and D. S. Johnson. *Approximation Algorithms for Bin Packing Problems: A Survey*, pages 147–172. Springer Vienna, Vienna, 1981.

[6] Fred Glover. *Future paths for integer programming and links to artificial intelligence*, volume 13, pages 533–549. 1986. Applications of Integer Programming.

[7] Geetika T. Lakshmanan, Ying Li, and Rob Strom. Placement strategies for internet-scale data stream systems. *IEEE Internet Computing*, 12(6):50–60, 2008.

[8] Teng Li, Jian Tang, and Jielong Xu. A predictive scheduling framework for fast and distributed stream data processing. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 333–338, 2015.

[9] Matteo Nardelli, Valeria Cardellini, Vincenzo Grassi, and Francesco Lo Presti. Efficient operator placement for distributed data stream processing applications. *IEEE Transactions on Parallel and Distributed Systems*, 30(8):1753–1767, 2019.

[10] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 49–49, 2006.

[11] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 49–49, 2006.

[12] Ioana Stanoi, George Mihaila, Themis Palpanas, and Christian Lang. Whitewater: Distributed processing of fast streams. *IEEE Transactions on Knowledge and Data Engineering*, 19(9):1214–1226, 2007.

[13] Fabrice Starks, Vera Goebel, Stein Kristiansen, and Thomas Plagemann. *Mobile Distributed Complex Event Processing—Ubi Sumus? Quo Vadimus?*, pages 147–180. Springer International Publishing, Cham, 2018.

[14] Jielong Xu, Zhenhua Chen, Jian Tang, and Sen Su. T-storm: Traffic-aware online scheduling in storm. In *2014 IEEE 34th International Conference on Distributed Computing Systems*, pages 535–544, 2014.

[15] K. Yoon and C.L. Hwang. *Multiple Attribute Decision Making: An Introduction*. Number Nr. 104 in Multiple Attribute Decision Making. SAGE Publications, 1995.