Universität Innsbruck

Institut für Informatik

Heuristische Optimierung der Operatorplatzierung in verteilten Stream-Verarbeitungssystemen

Cedric Immanuel Sillaber
Matrikel-Nr: 12211124

VU Einführung ins Wissenschaftliche Arbeiten

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

13. Mai 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. Das Problem der Operatorplatzierung ist ein entscheidender Performancefaktor. Für das Lösen dieses Problems gibt es jedoch keine effiziente Lösung. Diese Arbeit beschäftigt sich mit einer effizienten heuristischen Methode, die versucht, die optimale Lösung zu approximieren.

1 Einführung

Im Zuge der fortschreitenden Digitalisierung entwickelte sich Datenverarbeitung zu einem zentralen Aspekt der Modernität. Der Erfolg vieler Konzerne beruht auf der Expertise, wie kontinuierliche Datenmengen effizient verarbeitet werden. Rund um die Uhr werden Daten gesammelt, die in Echtzeit verarbeitet werden müssen. SVSs wie (zitat nötig) sammeln, filtern und verarbeiten Daten. Die Daten werden von einer großen Menge an Geräten produziert. Derartige Systeme werden beispielsweise in der Analyse von Finanzmärkten [8], Verabreitung von Sozialen Netzwerk-Interaktionen und der Beobachtung von Network-Traffic [8] eingesetzt. Ein SVS besteht aus einer Menge von unabhängigen Operatoren, die eine spezifische Funktionalität ausführen. Aus der Unabhängigkeit der Operatoren lässt sich die Möglichkeit folgern, dass die Rechner im Netzwerk (Cloud-Edge [3]) lokalisiert sind, 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. Für die Evaluation solcher Modelle werden diverse Quality-of-Service Attribute herangezogen. Dazu gehören Durchsatz, End-zu-End Latenz und Verfügbarkeit [7] [2]. Der Ansatz in dieser Arbeit versucht sich auf generalisierte QoS Attribute zu fokusieren, die einfach angepasst werden können [7]. Diese Arbeit fokusiert sich auf einen Ansatz, der bekannte Heurisiken kombiniert und optimale Lösungen effizient approximiert. Der Ansatz bezieht sich auf keine spezifische Implementierung und ist somit modell-frei. Somit kann diese Lösung für verschiedene Systeme angewendet werden.

Die optimale systematische Position in solch einem System stellt einen maßgeblichen Performancefaktor dar. Die Lösung dieses Problems ist jedoch NP-hard [2].

2 Definitionen

Um eine konkrete Optimierung des Problems beschreiben zu können, ist es notwending die Problematik formal zu formulieren. Im folgenden Kapitel wird eine formale Abstraktion des Operatorplatzierungproblems vorgestellt. Zuerst wird das Datenstrom- und das Ressourcenmodell 2.1 defininiert. Beide Modelle werden mittels Graphentheorie formuliert und beschreiben ein SVS. Folglich wird das Problem der Operatorplatzierung mithilfe der zuvor definierten Modelle formuliert 2.2.

2.1 Definition von verteilten Streamdatenverarbeitungssystemen

SVSs basieren auf einer Menge an verteilten Computer Ressourcen, die zusammen ein komplexes System ergeben. Dieser Sachverhalt kann mittels Graphentheorie beschrieben werden. Grundsätzlich gibt es zwei Abstraktionen solcher Systeme. Erstens das Datenstrom Modell und zweitens das Ressourcen Modell. Beide Systeme werden mit gerichteten, gewichteten zykelfreien Graphen $G = (V, E)$ dargestellt.

Datenstrom Modelle werden durch $G_{svs} = (V_{svs}, E_{svs})$ beschrieben. In diesem Modell beschreiben Knoten $u \in V_{svs}$ Operatoren im System. Zusätzlich sind in V_{svs} Datenquellen und Datenbecken enthalten. Man spricht hierbei von Datenbecken (vgl. ßink"), ein Punkt, an dem die Berechnungen der Operatoren zusammekommen. Zu den Operatoren gehören auch sogenannte *pinned* Operatoren [7], die Datenquellen und -becken beinhalten. Kanten $(u, v) \in E_{svs}$ beschreiben Datenstreams zwischen den Operatoren u und v. Ein Stream ist ein kontinuierliche Sequenz von Daten.

Ressourcen Modelle 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 dem Rechnerressourcen u und v.

2.1.1 Angepinnte Operatoren

2.2 Definition von Operatorplatzierungproblem

Ein Operator kann aus politischen, sicherheitsbezüglichen oder topologichen Gründen [2] nicht auf jedem Knoten $i \in V_{svs}$ im Datenstrommodell platziert werden. Für jeden Operator $i \in V_{svs}$ gibt es eine Menge an Kandidatressourcen V^i_{res} (Schreibweise wie in [7]).

Das Operator Problem bezeichnet eine Abbildung zwischen den genannten Modellen. Die Abbildung wird eingeschränkt, damit die zu minimierenden QuS Attribute eingeschränkt werden. Folglich wird der optimale Kandidat u in den Kandidatenressourcen V^i_{res} gesucht, damit Operator i auf Knoten u platziert wird. Hierbei bezieht sich das Problem auf die Inkonsistenz zwischen logisch benachbarten Operatoren im Ressourcen Modell 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 $x_{i,u} = 0$

$$\arg\min_{\theta} F(x)$$

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

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

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

Hierbei wird die Funktion $F(x)$ bezüglich ausgewählten QoS minimiert. Dieses multi-objective Optimierungsproblem wird durch Simple Additive Weight technique [10] zu einem single-objective Problem umgewandelt. Für typische QoS Attribute wie Antwortzeit, Verfügbarkeit und Netzwerkverwendung [7] wird die Funktion $F(x)$ wie folgt defininiert:

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

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. $w_r + w_a + w_z = 1$

Mithilfe einer **penalty function** werden die Verbindungen zwischen zwei spezifischen Knoten bezüglich der QoS Attribute bewertet. Dadruch wird ein Vergleich der Rechnerressourcen möglich. Dabei werden die Links zwischen $u \in V^i_{res}$ möglich. Auf die penalty function wird im folgenden eingegangen.

3 Heuristiken

Wie in [2] gezeigt, ist das Operatorplatzierungproblem NP-hard. Da die initiale Platzierung der Operatoren eine tragender Faktor in der Performance einnimmt, werden effiziente Heuristiken benötigt. In dieser Abreit wird eine effiziente Methode vorgestellt, die zu einer approximierten Optimalösung führt. Dieser Ansatz beinhaltet eine Kombination mehrere bekannter Heuristiken, die die Funktion F aus 2.2 minimieren. Ein Greedy First-Fit Ansatz in Kombination mit einer lokalen Suche findet meist lokale Optima. Um dem entgegenzuwirken wird dieser Ansatz mit einer Tabu Search verbunden. Somit werden häufiger [7] globale Optima gefunden.

Ein Greedy First-Fit Algorithmus wird für das Bin-packing Packing Problem verwenden, aber auch oft für das Operator Placement Problem [1][9]. Da diese Heuristik meist nur lokale Optima findet, werden andere Ansätze hinzugezogen. Zum einen wird Local-Search verwendet, ein Verfahren, das mit einem Greedy Ansatz über einen Teil der Funktion iteriert. Da auch dieses Verfahren dazu neigt, lokale Optima auszuwählen, wird zusätzlich Tabu Search implementiert. Die drei Heuristiken werden gekonnt kombiniert und führen somit zu einer besseren Approximation.

3.1 Penalty Function

Die Auswahl von verschiedenen Möglichkeiten $u \in V^i_{res}$ bringt Einbussen mit sich. Diverse Ressourcen haben verschiedene Lokalitäten, deren Performance durch Netzwerkdynamiken beeinflusst wird. Da die Daten übertragen werden müssen, kommen nicht vorhersehbare Network Delays dazu. Hierzu müssen Network Delay, Bandbreite und Netzwerkgeschwindigkeit[7] betrachtet werden.

3.2 Greedy First Fit

Diese Heuristik wird für das Bin-Packing Problem verwendet, um eine optimale Lösung anzunähern [4]. Für jeden Operator werden die verfügbaren Ressourcen $v \in V_{res}$ in einer Liste sortiert. Die Sortierung basiert auf der summierte penalty function δ zwischen v und den angepinnten Operatore P. Folglich:

$$u_i \in P, \text{ wobei P = angepinnten Operatoren}$$

$$\sum_{i=0}^{|P|} \delta(v, u)$$

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

3.3 Local Search

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 resultieret. Der Algorithmus dafür wird im Algorithmus 1 3.3 gezeigt. Dieser Algorithmus 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 penalty function δ wie in 3.2. 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 (insert)). Eine Verbesserung wird durch einen niedrigeren Wert der Zielfunktion F definiert. Die Suche nach besseren Konfigurationen basiert auf den drei Funktionalitäten *co-locate operators, swap resources, Bewegen von Operator.*

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.
Bei *swap resources* 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 *move single locator* bewegt nur einen einzelnen Operator i von $u \in G_{res}$ zu $v \in G_{res}$, wobei v aus L ausgewählt wird.

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 colocating 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**

Das Zusammenführen von Operatoren auf eine gemeinsame Rechnerressource ermöglicht es, verwandte Aufgaben in eine bessere Lokalität zu platzieren, was QoS verbessern kann. Das Austauschen von Operatoren strebt eine Eliminierung/Minimierung von ressourcelichen 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 wie 3.2 nur eine Approximation ist und in einer lokal-optimalen Lösung enden kann.

3.4 Tabu Search

Die Heurisitische lokale Suche ist abhängig von der initialen Konfiguration und terminiert nur bedingt mit einer globalen optimalen Lösung für das Operatorplatzierungproblem. Um dem entgegenzuwirken, wird die Heurisik zur *Tabu Suche* [5] erweitert. In dieser Methode wird über mehrere Anfangskonfigurationen iteriert, die anhand der Lokalen Suche nicht unmittelbar zu einer Verbesserung führen. Durch diese Eweiterung werden Lösungen evaluiert, die bei einer lokalen Suche nicht betrachtet werden.

Der Algortihmus kann in die folgenden Schritte unterteilt werden [6]. Zuerst wird eine Ausgangslösung berechnet. Anhand dieser Ausgangslösung wird eine Nachbarschaft bestimmt. Die Nachbarschaft wird um die sogenannten Tabu Züge verkleinert. Aus der resultierenden Nachbarschaft wird die beste Lösung bestimmt und ausgewählt. Am ende der iteration wird die Tabuliste basierend auf einem Tabukriterium aktualisiert.

Gestartet wird mit einer Ausgangskonfiguration S' , die von der Greedy First Fit Heuristik erstellt wurde (Zeile 6).
Von Zeile 10 bis 21 wird startet eine Schleife, die in jeder Iteration nach einer besseren Lösung in der Nachbarschaft sucht. Die Suche wird mittels lokaler Suche durchgeführt. Hierfür werden Lösungen aus S ausgeschlossen, die in der Tabu Liste enthalten sind (Zeile 12).
Die Tabu liste tl ist anfangs leer, beinhaltet nach der ersten Iteration die letzten tl_{max} Platzierungen. So wird verhindert, dass die Schleife unendlich iteriert.

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: $S^* \leftarrow$ undefiniert
5: $F^* \leftarrow \infty$
6: $S' \leftarrow$ localSearch(G_{dsp}, G_{res}) //local optimum
7: $F' \leftarrow$ objective function value for S'
8: $S \leftarrow S'$
9: *tabuList* $\leftarrow$ create new tabu list and append S
10: **do**
11: improvement $\leftarrow false$
12: $S \leftarrow$ local search for S, excluding solutions in *tabuList*
13: **if** $F = F^*$ **and** $S \notin tabuList$ **then** *tabuList*.append(S)
14: **end if**
15: **if** $F < F^*$ **and** $S \notin tabuList$ **then**
16: $S^* \leftarrow S; F^* \leftarrow F$
17: *tabuList*.append(S)
18: improvement $\leftarrow true$
19: **end if**
20: limit *tabuList* to the latest $tabuList_{max}$ placement configurations
21: **while** improvement
22: **if** $F' < F^*$ **then** $S^* \leftarrow S'$ **end if**
23: **return** S^*

Mit dieser Heuristik wird der Lösungsraum nach besseren Approximationen des Optimums durchsucht. Im Vergleich zur Greedy First Fit 3.2 und der lokalen Suche 3.3 bietet die Tabu-Suche eine erweiterte Fähigkeit zur Exploration des Lösungsraums und kann somit zu verbesserten Lösungen führen.

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] 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.

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

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

[6] Fred Glover. Tabu Search: A Tutorial. *Interfaces*, 20(4):74–94, August 1990.

[7] 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.

[8] 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.

[9] 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.

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