Heuristiken für effiziente Operatorplatzierung in verteilten Stream-Verarbeitungssystemen

Cedric Sillaber

June 2024

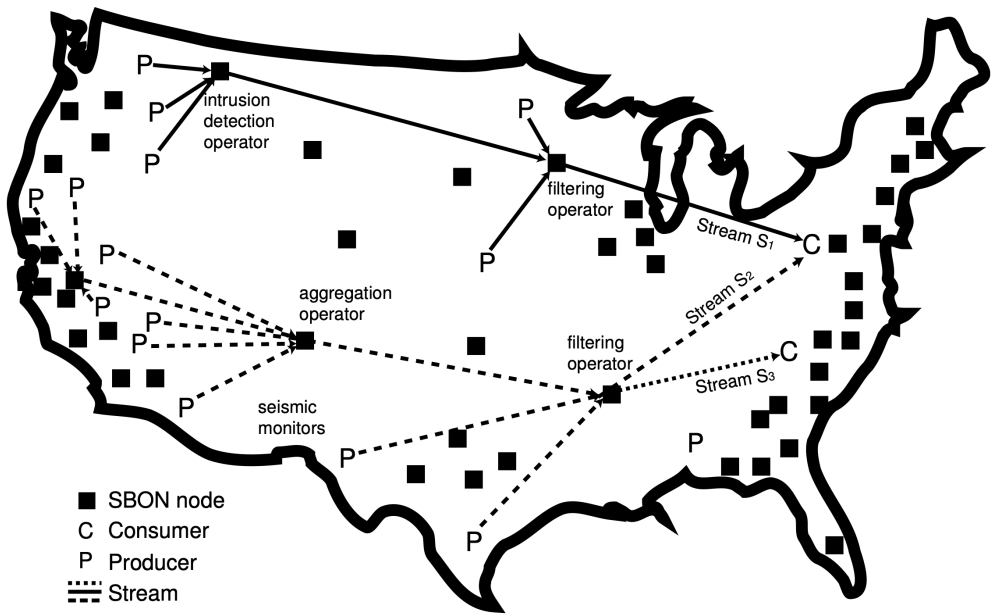
Inhalt

Motivation

- Verteilte Stream-Verarbeitungssysteme bestehen aus etlichen physischen Rechnern.
- Welcher Rechner führt welche Operationen aus?
  - Berechnung optimaler Operatorplatzierung ist NP-schwer!
    $\rightarrow$ Approximierung durch bekannte Heuristiken
- Ziel: Latenz, Netzwerkauslastung, Durchsatz, Verfügbarkeit, etc. minimieren

Was ist ein Operator?

- Verarbeitungseinheit in einem Stream-Verarbeitungssystem
- Beispiele: Filter, Aggregatoren, Joins

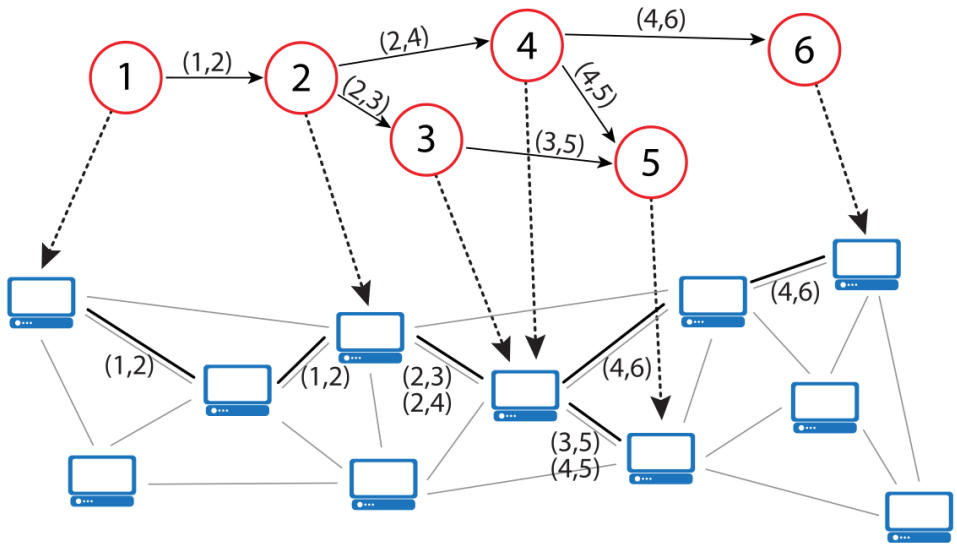


Formale Definition von Streamverarbeitungssystemen

- Zwei Modelle, definiert durch gerichtete gewichtete Graphen $G = (V, E)$
- *Datenstrom-Modell* beschreibt den Fluss von Datenströmen
  - Was sind Quellen, was sind Senken? Wohin fließen Daten?
  - Wo sollte sortiert, gefiltert, gejoint werden?
  - Dargestellt mit $G_{svs} = (V_{svs}, E_{svs})$
- *Ressourcen-Modell* beschreibt physische Rechner und deren Verknüpfungen
  - dargestellt mit $G_{res} = (V_{res}, E_{res})$

Problem der Operatorplatzierung

- Mapping zwischen *Datenstrom-* und *Ressourcen-Modell* (G_{svs} und G_{res}



Formale Definition des OPPs

- kann formal definiert werden als

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

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

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

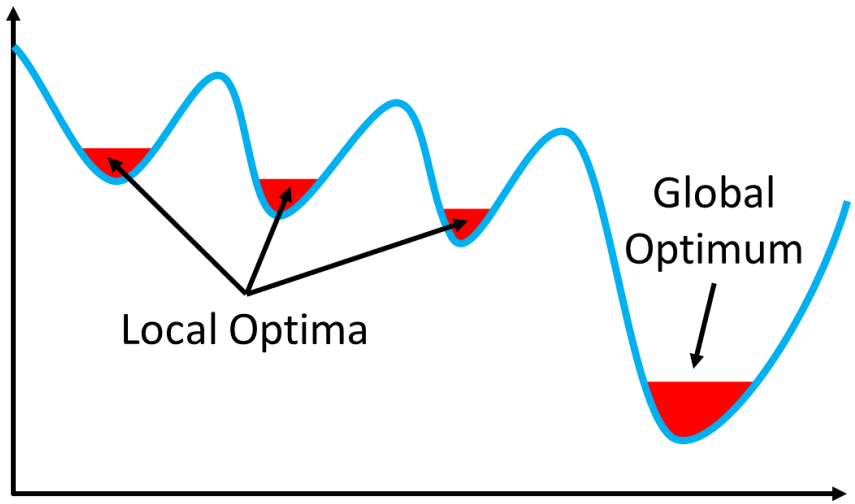
$$x_{i,u} \in \{0,1\} \quad \forall i \in V_{svs}, u \in V_{res}^i$$

zu minimierende Funktion F:

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

Heuristiken

- Näherungsverfahren zur Lösung von Optimierungsproblemen
- **Greedy First-Fit**:
  - Physische Ressourcen basierend auf Straffunktion sortiert (*greedy*)
  - Erster passender Rechner wird ausgewählt und nimmt Operator auf (*first-fit*)
  - Vorteile: Einfachheit, schnelle Berechnungen
  - Nachteile: Kann in lokalen Optima steckenbleiben



Quelle: https://www.allaboutlean.com/polca-pros-and-cons/local-global-optimum/

Lokale Suche

- Erweiterung von Greedy First-Fit
- Ziel: Verbesserung der initialen Lösung
  - Verschiebung von Operatoren zwischen Rechnern zur Optimierung
  - Vermeidung von lokalen Optima
- Vorteile: Bessere Lösung als Greedy First-Fit allein
- Nachteile: Höherer Berechnungsaufwand

Algorithm 4. 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 ← resources hosting the pinned operators of G_dsp
5:    L ← resources of G_res, sorted by the cumulative
6:        link penalty with respect to nodes in P
7:    S ← solve GreedyFirst-fit(G_dsp, L)
8:    do                                          ▷ local search
9:      F ← value of the objective function for S
10:     S ← improve S by co-locating operators
11:     S ← improve S by swapping resources
12:     S ← improve S by relocating a single operator
13:     F' ← value of the objective function for S
14:   while F' < F          ▷ placement solution is improved
15:   return S
16: end function
```

Kombinierte Ansätze

- Aufbauend auf Greedy und Lokaler Suche: Tabu Suche, um Lösungen zu verbessern

Experimentelle Ergebnisse

- Vergleich der Heuristiken Greedy First-Fit, Lokaler Suche und Tabu Suche
- Metriken: Verfügbarkeit, Netzwerklatenz und Verfügbarkeit
- Vergleich von Laufzeit und Beschleunigungsfaktor für drei Topologien

Vergleich der Heuristiken

Methode		DA	SA	RA	Durschnitt
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%
Tabu Suche	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%

- Optimale Lösung benötigt 8h, Greedy First-Fit Bruchteil einer Sekunde.
- Kompromiss zwischen Qualität und Lafzeit

Fazit

- Greedy First-Fit, Lokale Suche und Tabu Suche bieten effiziente Lösungen für das Operatorplatzierungsproblem.
  - Greedy First-Fit: schnelle, aber minderwertige Lösungen.
  - Lokale Suche und Tabu Suche: qualitativ besser, aber zeitintensiver.
- Alle Heuristiken zeigten eine deutliche Verkürzung der Laufzeit im Vergleich zur optimalen Lösung.

Zukunftsaussichten

- Entwicklung komplexerer Heuristiken zur besseren Approximation des OPP-Problems.
- Fokussierung auf zur Laufzeit anpassbare Heuristiken für dynamische Bedingungen.
- Analyse der erneuten Konfiguration während der Laufzeit für Big Data-Anwendungen.