Graphalgorithmen in massiv parallelen Umgebungen

Grundseminar Ausarbeitung

Heinrich Latreider

Hochschule für Angewandte Wissenschaften Hamburg,
heinrich.latreider@haw-hamburg.de

Zusammenfassung. Durch die kontinuierlich ansteigende Menge an Daten unterschiedlicher Dienste, vor allem im Internet, steht die Informatik vor der Herausforderung, diese Datenmenge effizient zu verwalten und zu verarbeiten, da eine Verarbeitung auf einem Rechner nicht mehr effizient möglich ist. Aus diesem Grund haben sich Systeme und Algorithmen zur parallelen Verarbeitung von Daten entwickelt. Ein sehr bekannter Vertreter ist MapReduce, welcher u.a. in Big Data Frameworks wie Hadoop zum Einsatz kommt. Graphen spielen bei der parallelen Verarbeitung eine besondere Rolle, da sie aufgrund ihrer Beschaffenheit zusammenhängende Daten enthalten und somit spezielle Verfahren nötig sind, um diese Daten effizient zu verteilen und zu verarbeiten. Diese Arbeit behandelt die vorhandenen Ansätze zur parallelen Verarbeitung von Graphen und gibt einen Ausblick auf weitere Arbeiten im Rahmen des Masterstudiums.

1 Einleitung

Durch die globale Verbreitung des Internets entstanden in vergangener Zeit eine Vielzahl verschiedener Internetdienste, darunter Suchmaschinen wie Google und Yahoo, Soziale Netzwerke wie Facebook und Twitter oder auch E-Commerce Plattformen wie Amazon. Diese Dienste verfügen über eine hohe Anzahl an Nutzern, so etwa Amazon über 300 Mio. registrierte Nutzer [Sta17a] oder Facebook über etwa 2 Milliarden monatlich aktive Nutzer [Sta17b]. Dies führt unweigerlich zu einem hohen Datenaufkommen, etwa durch Benutzerprofile, Kontakte zwischen Nutzern oder Bestellhistorien, aus denen sich ableiten lässt, welche Nutzer zu welcher Zeit welches Produkt bestellt haben. Aber auch aktuelle Entwicklungen wie Urban Computing tragen zum stetigen Datenwachstum bei. In diesem Fall sammeln eine hohe Anzahl an verschiedenen Sensoren kontinuierlich Daten, z.B. durch die Messung von Temperaturen, Luftverschmutzung oder Verkehrsdichte [ZCWY14].

Die Datenmenge sowie das Datenwachstum sind mittlerweile so groß, dass Ressourcen wie CPU-Leistung und Arbeitsspeicher nicht mehr ausreichen, um diese Daten effizient auf einem Rechner zu verarbeiten. Unter dieser Begebenheit ist

auch der Begriff Big Data entstanden. Um dieses Problem zu lösen, haben sich Technologien entwickelt, mit denen sich Daten parallel auf einem Rechnerverbund (engl. Cluster) verarbeiten lassen. Ein bekanntes Beispiel hierfür ist das Framework Apache Hadoop [Had17], das die verteilte Speicherung von Daten sowie eine verteilte Verarbeitung dieser Daten auf Clustern ermöglicht.

Der wohl wichtigste Algorithmus für die verteilte Verarbeitung großer Datenmengen ist MapReduce [DG08], welcher darauf basiert, dass einzelne Datensätze der Eingabedaten unabhängig voneinander sind und somit eine Verteilung und nebenläufige Verarbeitung stattfinden kann. Es existieren aber auch Datenstrukturen, bei denen Daten miteinander verknüpft sind, z.B. Graphen. Graphen bestehen aus einer Menge von Knoten sowie einer Menge von Kanten, die jeweils zwei Knoten miteinander verbinden. Die Verteilung eines Graphen führt damit unweigerlich zu einem höheren Kommunikationsaufwand bei der Verarbeitung, da benachbarte Knoten eventuell auf verschiedenen Maschinen liegen können. Die meisten Graphalgorithmen basieren zudem auf Iterationen, in denen eine Traversierung des Graphen stattfindet.

In den letzten Jahren haben sich deshalb verschiedene Lösungsansätze entwickelt, die eine verteilte Verarbeitung von Graphen ermöglichen. Diese Ansätze sollen in dieser Arbeit vorgestellt werden. Dazu führt Kapitel 2 in die Grundlagen der Graphentheorie sowie klassische Ansätze der Big Data Verarbeitung ein und setzt diese beiden Bereiche miteinander in Verbindung. In Kapitel 3 werden die beiden Ansätze Pregel [MAB+10] und PowerGraph [GLG+12] erläutert. Kapitel 4 gibt einen Ausblick auf weiterführende Arbeiten im Rahmen des Masterstudiums und fasst diese Arbeit abschließend zusammen.

2 Grundlagen

2.1 Graphentheorie

Die Graphentheorie ist ein Teilgebiet der Mathematik, dessen Anfänge bis in das 18. Jahrhundert zurückreichen, angefangen mit dem "Königsberger Brückenproblem" des Mathematikers Leonhard Euler. Das Königsberger Brückenproblem basiert darauf, dass die Stadt Königsberg vom Fluss Pregel durchquert wird, wodurch einzelne Abschnitte der Stadt durch insgesamt sieben Brücken miteinander verbunden sind. Gesucht ist ein Weg, bei dem alle sieben Brücken exakt ein mal überquert werden. Durch eine graphentheoretische Modellierung konnte Euler die Nichtexistenz eines solchen Weges beweisen.

Aber auch in der nachfolgenden Zeit hat sich die Graphentheorie als Werkzeug zur Lösung verschiedener Probleme erwiesen. Klassische Probleme sind etwa das Finden von kürzesten bzw. günstigsten Pfaden zwischen zwei Lokationen oder das Finden optimaler Matches, bei der eine möglichst optimale Zuordnung zwischen Elementen disjunkter Mengen stattfindet. Aber auch "modernere" Probleme wie Social Analysis (Facebook), Website-Rankings (PageRank) oder Recom-

mendations (Neighborhood Methode) lassen sich graphentheoretisch modellieren und lösen. Im Folgenden werden kurz die mathematischen Grundlagen der Graphentheorie beschrieben.

Definition: Ein Graph $G = (V, E)$ ist ein 2-Tupel bestehend aus einer Menge von Knoten V (Vertices) und einer Menge von Kanten (Edges). Eine Kante $e = (u, v) \in E$ verbindet wiederum zwei Knoten miteinander oder einen Knoten mit sich selbst (Schlaufe). Kanten können dabei gerichtet oder ungerichtet sein. Folglich werden die daraus resultierenden Graphen als gerichtete bzw. ungerichtete Graphen bezeichnet. Der Grad eines Knotens gibt an, wie viele Kanten adjazent zu einem Knoten sind.

Ein Property-Graph ist eine Erweiterung des Graphen, in der jeder Knoten v und jede Kante e mit benutzerdefinierten Eigenschaften angereichert wird. Auf diese Art lassen sich Eigenschaften wie Benutzerprofile, Kantengewichte oder Knotenzustände abbilden.

2.2 Big Data und MapReduce

Unter den Begriff Big Data fallen Daten jeglicher Art, die durch herkömmliche Verfahren zur Datenverarbeitung nicht mehr handhabbar sind. Gründe dafür sind die Datenmenge (Volume), dessen Wachstum (Velocity) und Vielfalt (Variety), welche auch im "3-V-Modell" beschrieben werden [fI17]. Durch die Datenmenge ist die Verarbeitung auf einer Maschine nicht mehr effizient möglich, da Ressourcen wie Speicher und CPU-Leistung nicht ausreichen und eine vertikale Skalierung (Hinzufügen von Ressourcen durch Aufrüsten) nicht mehr rentabel ist. Aus diesem Grund findet eine horizontale Skalierung statt, bei der Ressourcen in Form weiterer Rechner hinzugefügt und diese Rechner zu Clustern zusammengefasst werden. Auf diesen Clustern werden die Daten durch ein verteiltes System verwaltet und verarbeitet. Durch das Hinzufügen und Entfernen von Rechnern in bzw. aus einem Cluster lassen sich Ressourcen somit je nach Bedarf dynamisch skalieren.

Die Datenverarbeitung in den Clustern erfolgt oft durch den MapReduce Algorithmus [DG08], welcher vor allem für große Datenmengen entwickelt wurde. MapReduce basiert auf dem "Teile und herrsche" Prinzip, welches besagt, dass ein Problem dadurch gelöst werden kann, indem es in kleinere Teilprobleme zerlegt wird. Diese einzelnen Teilprobleme werden parallel gelöst und die Teillösungen anschließend zu einer Gesamtlösung zusammengefasst. Beim MapReduce Algorithmus findet zuerst eine Partitionierung der Eingabedaten und die Zuweisung der Partitionen an verschiedene Maschinen im Cluster statt. Anschließend erfolgt die Verarbeitung der Eingabedaten in folgenden Phasen:

- **Map:** In der Map Phase werden die Eingabedaten durch Worker Prozesse lokal zu Schlüssel-Wert-Paaren transformiert. Diese Phase entspricht der parallelen Lösung der Teilprobleme, die Ausgabedaten sind damit die Teillösungen.

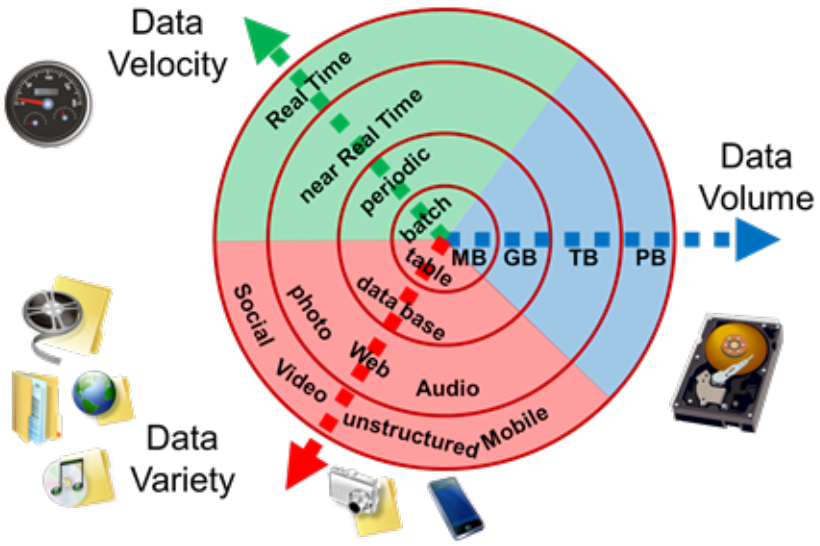

Abb. 1. 3-V-Modell [fI17]

- **Shuffle:** In der Shuffle Phase findet eine Gruppierung der Teillösungen nach Schlüssel statt, um sie für die Reduce Phase vorzubereiten. In dieser Phase findet die Kommunikation über das Netzwerk statt.
- **Reduce:** In der Reduce Phase werden die Teillösungen zu einer entgültigen Lösung zusammengefasst.

Die beiden Phasen Map und Reduce sind von den Eingabe- und Ausgabedaten abhängig und werden durch den Anwender des MapReduce Algorithmus implementiert. Technische Details wie Partitionierung der Daten, Netzwerkkommunikation, Scheduling, Synchronisation und Fehlertoleranz werden dagegen durch das genutzte MapReduce Framework gekapselt.

2.3 Graphen im Bereich Big Data

Die Verarbeitung von Graphen erfolgt in der Regel durch Graphalgorithmen. Diese sind meist iterativer Natur, d.h. es findet eine Traversierung des Graphen statt. In vielen Algorithmen wird dabei der Zustand des Graphen verändert. So enthält z.B. nach Anwendung des Dijkstra Algorithmus jeder Knoten im Graphen die Distanz bzw. Kosten zu einem anfangs angegebenen Startknoten. Diese Kosten werden dabei sukzessive durch die Traversierung des Graphen ermittelt. Das Problem von MapReduce bei der Graphverarbeitung ist die Tatsache, dass die Operationen mengenbasiert sind und die Transformationen einzelner Knoten bzw. Kanten dadurch mit einem hohen Overhead verbunden sind. Dieser Overhead entsteht dadurch, dass die Ausgabedaten des ersten MapReduce-Durchlaufs die Eingabedaten des darauf folgenden Durchlaufs darstellen und

damit eine hohe Netzwerklast und hohe I/O-Last einhergehen [GLG+12]. Aus diesem Grund sind Ansätze nötig, die eine iterative Verarbeitung von Daten in Clustern ermöglichen.

3 Existierende Ansätze zur Graphverarbeitung

Um große Graphen parallel aber gleichzeitig auch effektiv zu verarbeiten, haben sich mehrere Ansätze entwickelt. In dieser Arbeit sollen zwei davon vorgestellt werden, und zwar Pregel und PowerGraph. Zentrale Herausforderungen, die beide Ansätze berücksichtigen, sind zum einen die Verteilung eines Graphen auf mehrere Rechnenknoten und zum anderen die daraus resultierende Parallelität bezüglich der Berechnungen.

3.1 Pregel

Pregel ist ein Lösungsansatz von Google zur parallelen Verarbeitung von Graphen [MAB+10]. Pregel wird dabei sowohl als verteiltes Berechnungsmodell für Graphen nach dem Paradigma "Think like a Vertex" verstanden, aber auch als Plattform, die dieses Modell implementiert. Inspiriert ist der Ansatz vom Berechnungsmodell "Bulk Synchronous Parallel" (BSP) [Val90], welches im Folgenden erläutert wird.

Bulk Synchronous Parallel: Das Bulk Synchronous Parallel Modell entstand bereits im Jahr 1990 als Vorschlag für ein "Bridging Model" zur Abstaktion paralleler Berechnungen vom darunterliegenden verteilten System und zum Entwurf und Implementation paralleler Algorithmen. Der Ablauf eines in BSP implementierten Algorithmus kann Abbildung 2 entnommen werden.

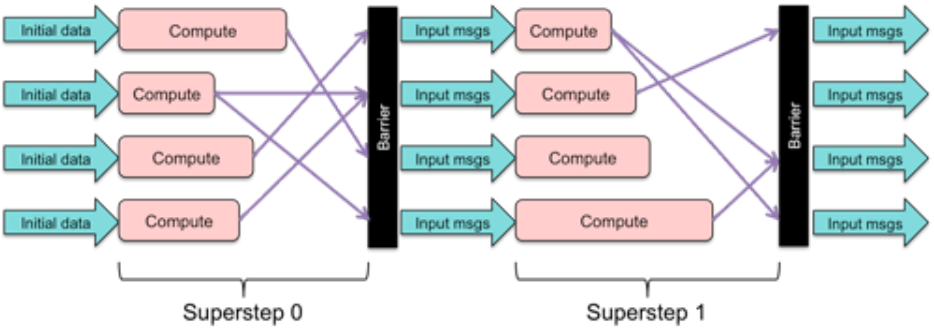

Abb. 2. BSP Berechnungsmodell [Krz12]

Auf der obersten Ebene einer BSP-Berechnung stehen Supersteps, welche einen Berechnungsschritt abbilden. Ein Superstep besteht wiederum aus mehre-

ren Einzelschritten, wobei die einzelnen BSP-Prozesse im ersten Schritt Eingabedaten erhalten. Im nächsten Schritt erfolgt die parallele Verarbeitung der Teildaten durch die einzelnen BSP-Prozesse, indem jeder Prozess lokal Berechnungen auf seinen Daten ausführt. Diese Verarbeitung kann je nach Prozess unterschiedlich lang dauern. Da es sich bei BSP um ein synchrones Berechnungsmodell handelt, findet nach jedem Superstep eine Synchronisation der BSP-Prozesse statt. Nach der lokalen Berechnung ist es jedem Prozess erlaubt, Nachrichten an andere Prozesse zu schicken. Erst nach der Synchronisation ist es den Prozessen erlaubt, in den nächsten Superstep überzugehen, wobei jedem Prozess die im vorherigen Superstep an sie gesendeten Nachrichten zugestellt werden. Diese Nachrichten werden wiederum lokal durch die einzelnen BSP-Prozesse weiterverarbeitet, bis keine Nachrichten zwischen Prozessen mehr ausgetauscht werden.

Pregel macht sich diesen Ansatz zunutze, indem jeder Knoten im Graph als ein Prozess modelliert wird, der in der Lage ist, Nachrichten von anderen Knoten zu empfangen (Gather), diese zu verarbeiten (Apply) und Nachrichten an andere Knoten zu senden (Scatter). Gleichzeitig kann jeder Knoten die beiden Zustände *aktiv* und *inaktiv* einnehmen. Der Zustand hängt wiederum davon ab, ob ein Knoten Berechnungen durchführen muss (*aktiv*) oder nicht (*inaktiv*). Der Wechsel zwischen den Zuständen erfolgt durch das Abschließen der lokalen Berechnung ($aktiv \rightarrow inaktiv$) bzw. durch den Empfang von Nachrichten ($inaktiv \rightarrow aktiv$). Haben alle Knoten den Zustand inaktiv eingenommen, gilt die Berechnung als terminiert.

Da es sich bei Pregel um ein Berechnungsmodell handelt, müssen durch den Anwender der Pregel-Schnittstelle folgende Funktionen implementiert werden, um einen Algorithmus abzubilden.

- **Compute():** Pregel folgt dem Paradigma "Think like a Vertex", weshalb in der Compute-Funktion die Berechnungslogik des Algorithmus aus der Sicht eines Knotens spezifiziert wird. Die Funktion gibt an, wie ein Knoten empfangene Nachrichten verarbeitet, welchen Zustand er daraufhin einnimmt und welche Nachrichten verschickt werden. Die Ausführung dieser Funktion erfolgt parallel durch jeden Knoten.
- **Combine():** Da sich benachbarte Knoten auf verschiedenen Maschinen befinden können, ist eine Netzwerkkommunikation beim Senden von Nachrichten unumgänglich. Aus diesem Grund erfolgt wenn möglich eine Reduktion von Nachrichten mit dem gleichen Zielknoten, um die Netzwerklast möglichst gering zu halten. Diese Reduktion erfolgt durch eine Akkumulation, die in der Combine-Funktion spezifiziert wird. Der Zielknoten erhält somit eine einzelne akkumulierte Nachricht, auf dessen Basis der Knoten weitere Berechnungen vornimmt. Da über die Nachrichtenreihenfolge keine Annahmen getroffen werden, muss die Combine-Funktion zwingend assoziativ und kommutativ sein.

Partitionierung: Damit eine parallele Berechnung stattfinden kann, ist eine Partitionierung des Graphen notwendig. Pregel nutzt dafür standardmäßig eine Hashfunktion, was dazu führt, dass Knoten eines Graphen zufällig auf verschiedenen Maschinen verteilt werden, jedoch Faktoren wie Lokalität nicht berücksichtigt werden. Bei Bedarf kann die Partitionierungsfunktion jedoch durch den Anwender überschrieben werden.

Architektur: Das Pregel Modell folgt der Master/Slave-Architektur. Während der Master-Prozess für die Koordination der Worker Prozesse und für die Fehlertoleranz zuständig ist, führen Worker-Prozesse die Berechnungen durch. Durch Fehlertoleranz wird erreicht, dass eine Berechnung fortgeführt werden kann, auch wenn ein Prozess ausfällt, was in verteilten Systemen nicht ausgeschlossen werden kann.

3.2 PowerGraph

Bei PowerGraph handelt es sich um einen Ansatz zur Verarbeitung natürlicher Graphen. Unter natürlichen Graphen werden Graphen verstanden, die dem "Power Law" unterliegen. Dieses Gesetz bezogen auf Graphen besagt, dass die meisten Knoten im Graphen einen kleinen Grad, aber gleichzeitig wenige Knoten einen hohen Grad besitzen. Ein Beispiel hierfür ist Twitter. Die meisten Benutzer von Twitter verfügen über wenig Follower, während Berühmtheiten mehrere Millionen Follower haben können. Die Follower-Beziehung als Graph zeigt auf, dass ein Prozent der Knoten in diesem Graph adjazent zu 50% der Kanten sind [GLG+12].

Diese Asymmetrie der Knotengrade ist mit diversen Herausforderungen verbunden. So wird die Partitionierung des Graphen erschwert, da ein Knoten mit seinen Kanten eventuell zu groß für eine Maschine sein kann bezüglich Speicher oder Rechenleistung. Auch die Rechenzeit pro Knoten ist ungleichmäßig verteilt, da Vertex-Programme bei Knoten mit einem hohen Grad (High-Degree Vertices) mehr Zeit benötigen als bei Knoten mit geringem Grad. Die Parallelität der Berechnung hängt somit maßgeblich von den High-Degree Vertices ab.

Zur Lösung dieses Problems führt PowerGraph eine neue Abstraktion ein, die eine Parallelisierbarkeit der Vertex-Programme ermöglichen soll und somit der Berechnungsaufwand gleichmäßiger aufgeteilt wird. Inspiriert ist PowerGraph durch das Gather Konzept sowie dem BSP Ansatz von Pregel, aber auch durch die Schnittstelle von GraphLab [LBG+12], die eine Shared-Memory Sicht auf den Graphen ermöglicht. Um die Verteilung von Vertex-Programmen möglich zu machen, führt PowerGraph einen neuen Ansatz zur Graphpartitionierung ein. Diese findet nicht wie bei Pregel durch die zufällige Verteilung von Knoten und dem daraus resultierenden Schnitt der Kanten statt. Der Grund hierfür ist die Wahrscheinlichkeit, mit der sich zwei benachbarte Knoten auf verschiedenen Maschinen befinden. Diese beträgt nach [GLG+12] $1 - \frac{1}{p}$, wobei p die Anzahl der

Maschinen angibt. Bei $p = 20$ beträgt die Wahrscheinlichkeit bereits 95%, was in einem hohen Nachrichtenoverhead resultiert, wenn ein Knoten Informationen über seine Nachbarn benötigt. In realen Clustern ist die Anzahl der Maschinen beliebig nach oben skalierbar, womit auch die Wahrscheinlichkeit für den Schnitt einer Kante immer weiter ansteigt. Statt eines Kantenschnitts findet bei Power-

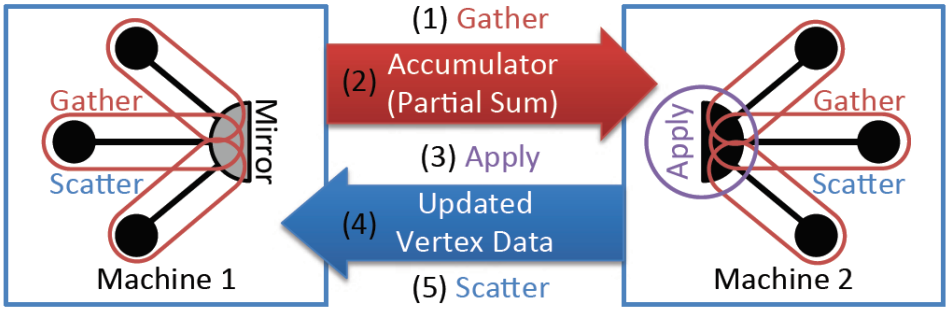

Abb. 3. PowerGraph Vertex Cut [GLG+12]

Graph ein Schnitt über die Knoten statt (Vertex-Cut). In diesem Fall befinden sich die Kanten auf einer Maschine, die Knoten hingegen können auf mehreren Maschinen verteilt sein. Ein Nachrichtenaustausch über das Netzwerk erfolgt somit nicht mehr zwischen benachbarten Knoten, sondern zwischen den Replikaten eines Knotens. Abbildung 3 zeigt den Vertex-Cut eines High-Degree Vertex mit sechs Nachbarn. Von diesem High-Degree Vertex werden zwei Replikate erzeugt und alle benachbarten Kanten gleichmäßig auf beide Maschinen verteilt. Ein Replikat wird dabei zum Master-Knoten ernannt. Dieser Schnitt wird auch als "Balanced p-way Vertex-cut" bezeichnet.

Bei einer Berechnung berechnet sich in der Gather-Phase jedes Replikat nebenläufig aus seiner zugewiesenen Teilmenge benachbarter Kanten und Knoten seine Teilsumme gemäß einer Summierungsfunktion. Im nächsten Schritt übermitteln alle Replikate ihre Teilsumme an einen Master-Knoten, der alle Teilsummen zu einer Gesamtsumme addiert. Abhängig von der Gesamtsumme passt der Master-Knoten in der Apply-Phase dann seinen Zustand an und propagiert die Zustandsänderung wiederum an seine Replikate. In der Scatter Phase wird der neue Zustand dann an die Nachbarn propagiert.

Um den Kommunikationsaufwand bei der Graphverarbeitung möglichst gering zu halten, wird bei der Partitionierung ein möglichst geringer Replikationsfaktor der Knoten angestrebt. Aus diesem Grund existiert neben dem zufälligen Balanced $p-Way$ Vertex-cut ein weiterer Partitionierungsalgorithmus, der sich eine Greedy-Heuristik zunutze macht und somit auch als Greedy Vertex-Cut

bezeichnet wird. Greedy Algorithmen zeichnen sich dadurch aus, dass sie einen Folgezustand auswählen, der zum Zeitpunkt der Auswahl das beste Ergebnis verspricht. Der Greedy Vertex-Cut führt vier Regeln ein, um eine Kante $e = (u, v)$ eines Graphen auf einer Maschine p zu platzieren. Dabei ist zu berücksichtigen, dass ein Knoten u mehreren Maschinen zugewiesen sein kann. Diese Zuweisung wird durch die Menge $A(u)$ angegeben.

1. Wenn $A(u) \cap A(v) \neq \emptyset$, dann platziere e auf einer Maschine $p \in A(u) \cap A(v)$.
2. Wenn $A(u) \cap A(v) = \emptyset$ und $A(u)$ sowie $A(v)$ nicht leer sind, dann wähle aus den Knoten u und v den mit den meisten unplatzierten Kanten aus und platziere e auf einer der Maschinen aus $A(u)$ bzw. $A(v)$.
3. Wenn ein Knoten u bereits platziert wurde, dann platziere e auf einer Maschine $p \in A(u)$.
4. Wenn keiner der beiden Knoten platziert wurde, dann platziere e auf der Maschine mit den wenigsten platzierten Kanten.

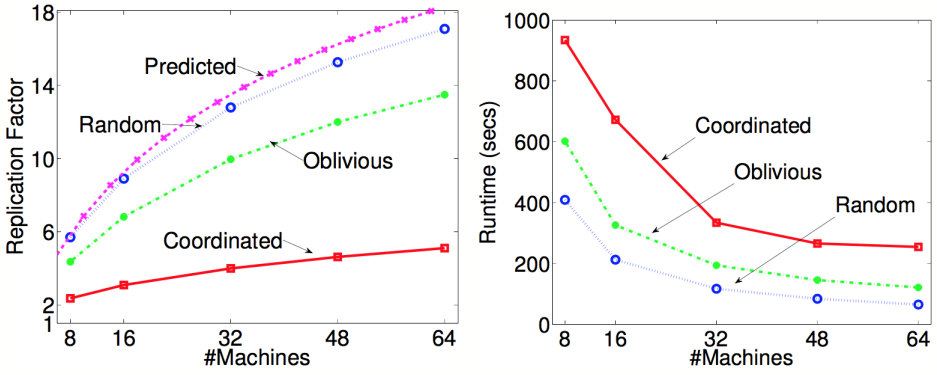

Abb. 4. PowerGraph Replikationsfaktor [GLG+12]

Beim Greedy Vertex-Cut wird zudem zwischen den beiden Implementationen *Coordinated* und *Oblivious* unterschieden. Der Unterschied zwischen beiden Implementationen besteht in der Art, wie einzelne Maschinen die Greedy Heuristik anwenden. Bei *Coordinated* wird die Kantenplatzierung in einer Distributed Table verwaltet, sodass jede Maschine die komplette Verteilung der Knoten kennt. Bei *Oblivious* wird die Greedy Heuristik auf jeder Maschine nebenläufig durchgeführt. Abbildung 4 veranschaulicht Replikationsfaktor sowie Laufzeitverhalten der verschiedenen Implementationen abhängig von der Anzahl der beteiligten Maschinen. Im Vergleich liefert der Coordinated Greedy Vertex-Cut laut [GLG+12] die besten Ergebnisse bezüglich des Replikationsfaktors der Knoten, die Ladezeit des Graphen liegt jedoch höher. Dies ist wohl auf den Kommunikationsaufwand zurückzuführen, der bei der Verwaltung der Distributed Table einhergeht.

4 Ausblick

In dieser Arbeit wurden zwei existierende Ansätze zur Graphverarbeitung vorgestellt. Dabei wurde das Berechnungsmodell Pregel erläutert, welches eine Vertex-basierte Implementation von Algorithmen ermöglicht. Mit PowerGraph wurde ein Ansatz zur Verarbeitung natürlicher Graphen vorgestellt, wobei der Fokus auf dessen Partitionierungsalgorithmen gelegt wurde.

Im Rahmen des Grundprojektes erfolgt eine Einarbeitung in die beiden Graph Processing Frameworks GraphX (Apache Spark) und Gelly (Apache Flink). Beide Frameworks sind Open Source und bieten eine Pregel-Schnittstelle für die Vertex-basierte Implementation von Algorithmen. In [GXD+14] wird außerdem explizit angegeben, dass Spark eine Graphpartitionierung gemäß [GLG+12] vornimmt. Im Grundprojekt sollen Konzepte beider Frameworks nachvollzogen sowie deren Schnittstellen kennengelernt werden. Die HAW Hamburg verfügt durch die Forschungsgruppe Big Data Lab über ein Cluster bestehend aus momentan sechs Rechnern mit insgesamt 48 CPU Kernen und 192 GB RAM. Dadruch bietet sich eine Einrichtung eines Spark- und Flink-Clusters an. Anschließend erfolgt die Implementation eines Graphalgorithmus mithilfe der Schnittstellen GraphX und Gelly. Untersucht werden soll dabei zum einen die Implementierbarkeit des Algorithmus, zum anderen aber auch die Skalierbarkeit bezüglich Größe des Eingabegraphen und Anzahl der Worker-Prozesse.

Weiterführende Arbeiten (Hauptseminar, Hauptprojekt und Masterarbeit) sind zum jetzigen Zeitpunkt noch nicht konkretisiert. Es ist jedoch denkbar, GraphX oder Gelly bezüglich ihrer Funktionalität zu erweitern, da beide Frameworks zwar standardmäßig einige Graphalgorithmen implementieren [Spa17], [Fli17], jedoch wesentlich mehr Graphalgorithmen existieren, die bezüglich Implementierbarkeit und Skalierbarkeit untersucht werden könnten.

Literatur

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[fI17] Gesellschaft für Informatik. Big data, 2017. https://www.gi.de/service/informatiklexikon/detailansicht/article/big-data.html.

[Fli17] Apache Flink. Gelly: Flink graph api, 2017. https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/libs/gelly/index.html.

[GLG+12] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 17–30, Berkeley, CA, USA, 2012. USENIX Association.

[GXD+14] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. Graphx: Graph processing in a distributed dataflow framework. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 599–613, Berkeley, CA, USA, 2014. USENIX Association.

[Had17] Hadoop. Apache hadoop, 2017. http://hadoop.apache.org/.

[Krz12] Paul Krzyzanowski. Exam 3 study guide, bulk synchronous parallel & pregel, 2012. https://www.cs.rutgers.edu/ pxk/417/exam/study-guide-3.html.

[LBG+12] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, April 2012.

[MAB+10] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.

[Spa17] Apache Spark. Graphx programming guide, 2017. http://spark.apache.org/docs/latest/graphx-programming-guide.html.

[Sta17a] Statista. Anzahl aktiver kunden-accounts von amazon weltweit in den jahren 1997 bis 2015 (in millionen), 2017. https://de.statista.com/statistik/daten/studie/297615/umfrage/anzahl-weltweit-aktiver-kunden-accounts-von-amazon/.

[Sta17b] Statista. Anzahl der monatlich aktiven facebook nutzer weltweit vom 3. quartal 2008 bis zum 2. quartal 2017 (in millionen), 2017. https://de.statista.com/statistik/daten/studie/37545/umfrage/anzahl-der-aktiven-nutzer-von-facebook/.

[Val90] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990.

[ZCWY14] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. Urban computing: Concepts, methodologies, and applications. *ACM Trans. Intell. Syst. Technol.*, 5(3):38:1–38:55, September 2014.