Hochschule für Technik, Wirtschaft und Kultur Leipzig

Grundlagen der Graphdatenverarbeitung

Zusammenfassung des Vortrags für das Oberseminar
Datenbanksysteme - Aktuelle Trends

von
Carolin Gümpel

Leipzig, 19.6.2018

Inhaltsverzeichnis

1 | Einführung

Graphdatenbanken wurden entwickelt, um stark vernetzte Daten effektiv speichern und analysieren zu können. Dabei stehen die Datenbeziehungen im Fokus. Es gibt bereits viele Unternehmen, die Graphdatenbanken verwenden. Dazu gehört die Adidas Group. Dort wird Neo4j für einen internen Metadatendienst genutzt. Das Ziel ist es dabei die vielen isolierten Datensilos des Unternehmens zu verknüpfen und Zusammenhänge zu erkennen, um beispielsweise speziell auf die einzelnen Kunden zugeschnittene Werbung zu ermöglichen.[1]
Auch beim Spielzeughersteller Schleich kommt Neo4j zum Einsatz. Dort wird die Graphdatenbank für das Produktdatenmanagement genutzt. Abbildung 1.1 zeigt die Darstellung der Produktdaten innerhalb eines Graphen.

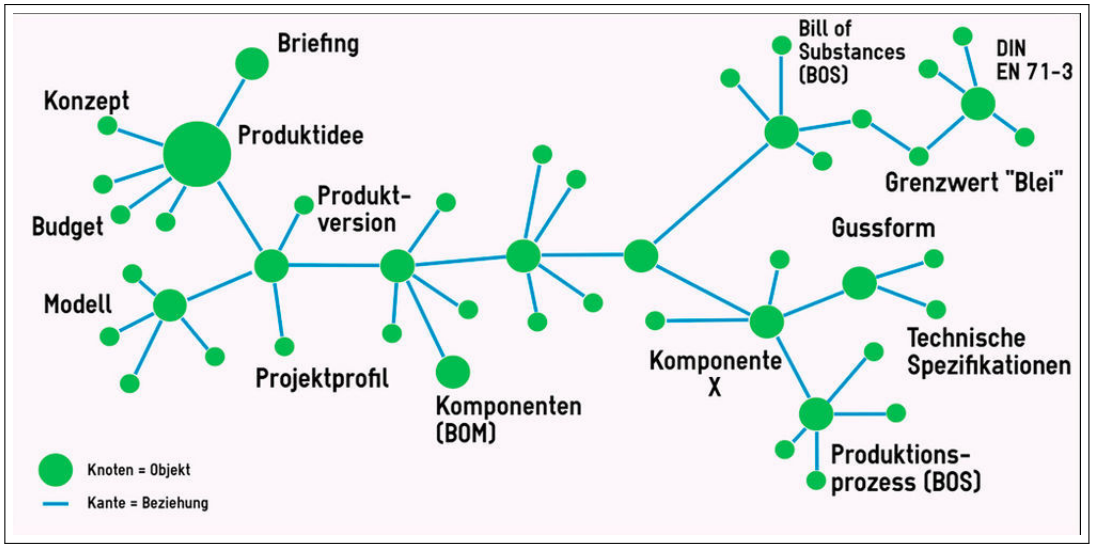

Abb. 1.1 Produktdaten innerhalb eines Graphen [2]

Dadurch lassen sich die Produktdaten verglichen mit der Geschwindigkeit bei Nutzung einer relationalen Datenbank extrem schnell abrufen und die Mitarbeiter können die Daten zu einem Produkt mit einem Klick einsehen. [1]
Da der aktuelle Trend immer mehr in Richtung Big Data geht und es deshalb auch einen großen Bedarf an Systemen gibt, welche große Mengen an Daten inklusive ihrer Zusammenhänge verarbeiten können, erleben Graphdatenbanken seit einigen Jahren einen Aufschwung. Wie in Abbildung 1.2 zu sehen ist, steigt die Popularität von Graphdatenbanken seit 2014 deutlich stärker an, als die der anderen Datenbankmodelle.

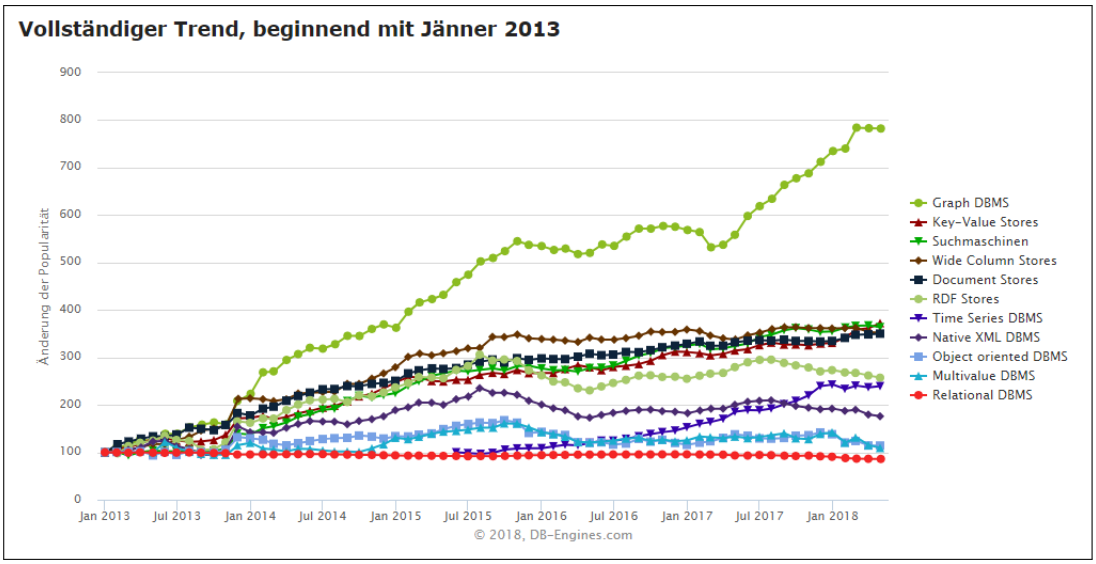

Abb. 1.2 Änderung der Popularität von Datenbankmodellen [3]

2 | Anforderungen an Graphdatenbanksysteme

Damit Graphdatenbanksysteme sinnvoll eingesetzt werden können, müssen sie einige Anforderungen erfüllen. Dazu zählt die Nutzung eines geeigneten Graphdatenmodells. Dieses muss Knoten und Kanten mit verschiedenen Attributen ohne ein festes Schema abbilden können. Außerdem müssen auf dem Modell graphorientierte Operationen durchführbar sein. Auf diesen Operationen aufbauend muss das System effiziente Abfrage- und Analysemöglichkeiten bieten. Komplexe Analysen über den kompletten Graphen müssen über eine einfach zu erlernende Abfragesprache möglich sein.
Weitere Anforderungen sind eine gute Performance und hohe Skalierbarkeit. Auch Graphen mit Milliarden Knoten und Beziehungen müssen mit hoher Geschwindigkeit verarbeitet werden können. Um dies zu erreichen ist es sinnvoll, eine Variante zur verteilte Ausführung der Graphenoperationen bereitzustellen. Um dabei den Kommunikationsaufwand möglichst gering zu halten und eine gleichmäßige Rechnerauslastung zu erreichen, ist eine effiziente Partitionierung der Graphen erforderlich.
Zusätzlich zu den bereits genannten Anforderungen muss ein Graphdatenbanksystem eine persistente Speicherung von Daten und Analyseergebnissen garantieren. Außerdem soll Transaktionssicherheit geboten werden.
Als letzte Anforderung ist eine verständliche Graphvisualisierung zu nennen. Vor allem große Graphen oder auch eine sehr große Anzahl kleiner Graphen ist für den Nutzer meist zu komplex und schwer zu überblicken. Hier muss ein Graphdatenbanksystem den Nutzer

unterstützen und eine verständliche Übersicht schaffen. [4, S. 1-3]

3 | Graphdatenmodelle

Wie bereits in Kapitel 2 beschrieben, ist es für ein Graphdatenbanksystem wichtig, ein geeignetes Graphdatenmodell einzusetzen. Ein Graph besteht immer aus einer Menge von Knoten und Kanten. Dabei werden gerichtete und ungerichtete Graphen unterschieden. Letztere drücken eine beidseitige Beziehung aus, während bei ersteren die Beziehung nur in eine Richtung gilt. Bei gewichteten Graphen werden den Kanten numerische Werte zugewiesen, welche das jeweilige Gewicht der Kante angeben. Um den Kanten und auch Knoten andere Attribute - wie zum Beispiel Zeichenketten - zuweisen zu können, werden attributierte Graphen verwendet. Ein Beispiel eines attributierten Graphen ist in Abbildung 3.1 (a) zu sehen. Um Kanten und Knoten eine explizite Bedeutung, also eine Art Datentyp zuzuordnen, werden beschriftete Graphen genutzt. Bei dieser Art von Graphen werden den Knoten und Kanten Labels zugeordnet, wie in Abbildung 3.1 (b) zu sehen.

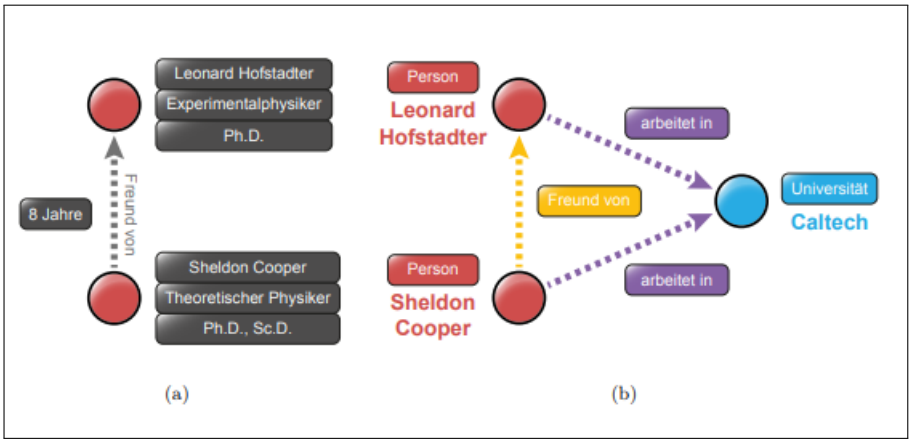

Abb. 3.1 Attributierter (a) und beschrifteter (b) Graph [6]

Für die Abbildung komplexer Daten in einem Graphdatenbanksystem ist eine Kombination aus attributiertem und beschriftetem Graphen sinnvoll. Die Attribute schaffen die nötige Flexibilität alle möglichen Eigenschaften von Knoten und Kanten als Schlüssel-Wert-Paare darzustellen, während die Labels eine Art Typisierung zulassen. Ein solcher Graph, welcher attributierten und beschrifteten Graphen vereint, wird als Property Graph bezeichnet. Ein Beispiel dafür ist in Abbildung 3.2 zu sehen.

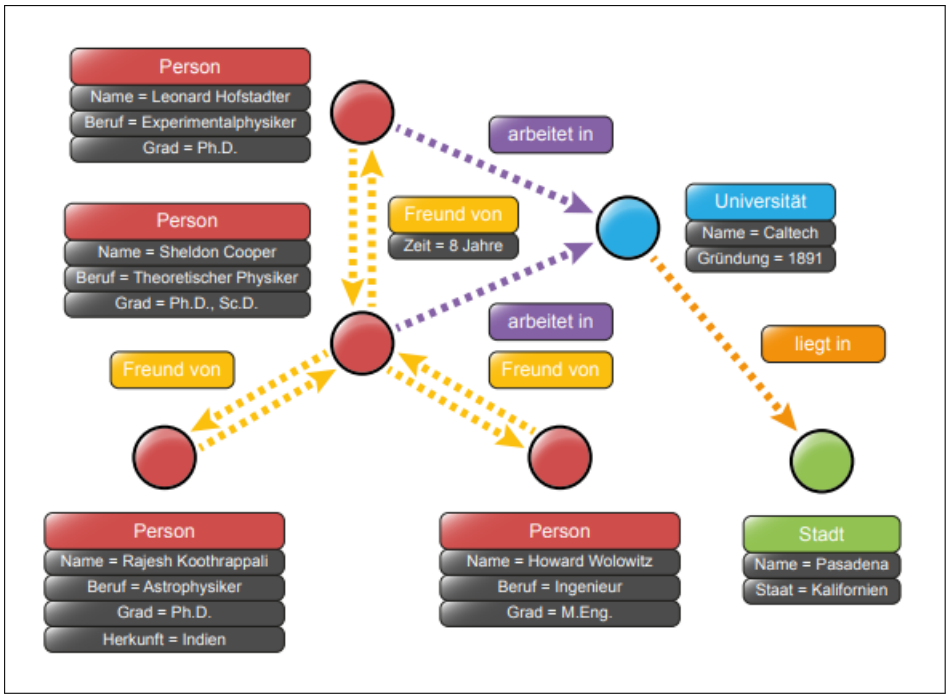

Abb. 3.2 Property Graph [6]

Neben den Property Graphen gibt es noch die RDF Graphen. Diese sind ein grundlegender Baustein des Semantic Web. Die Aussagen in den Graphen bestehen jeweils aus Tripeln. Jedes Tripel setzt sich aus Subjekt, Prädikat und Objekt zusammen. Subjekt und Objekt sind dabei Knoten, welche durch das Prädikat als Kante verbunden sind. Da sich RDF Graphen auch durch Property Graphen darstellen lassen, nutzen die meisten Graphdatenbanksysteme eine Variante des Property Graph Modells. [5, S. 102, 103]

4 | Abfrageschnittstellen

In Kapitel 2 wurde als eine Anforderung an Graphdatenbanksysteme die Möglichkeit der verteilten Ausführung von Graphoperationen genannt. Der Nutzer soll analytische Algorithmen auf dem Graphen parallel ausführen können, jedoch ohne die technischen Hintergründe der parallelen Ausführung zu kennen. Um dies zu ermöglichen, gibt es verschieden Modelle für die Gestaltung von Abfrageschnittstellen. Die erste Variante ist das knoten-zentrische Modell. Dieses setzt sich aus drei Phasen zusammen. In der Gather Phase sammelt jeder Knoten Informationen von all seinen Nachbarknoten. In der Apply Phase wird dann eine benutzerdefinierte Funktion auf jeden einzelnen Knoten angewendet. Die Weitergabe der berechneten Informationen an alle Nachbarknoten erfolgt schließlich in der Scatter Phase.
Das kanten-zentrische Modell ist ähnlich zum knoten-zentrischen Modell. Es gibt ebenfalls Gather, Scatter und Apply Phase. Die benutzerdefinierte Funktion wird jedoch nicht auf den

einzelnen Knoten, sondern auf den Kanten ausgeführt. In den Gather und Scatter Phasen werden Informationen vom Quellknoten der Kante gesammelt und die Ergebnisse der Berechnung an den Zielknoten gesendet. In den Abbildungen 4.1 bis 4.4 werden die Abläufe von knoten- und kanten-zentrischem Modell dargestellt.

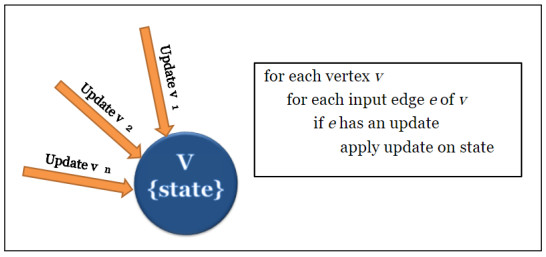

Abb. 4.1 Gather und Apply im knoten-zentr. Modell [7]

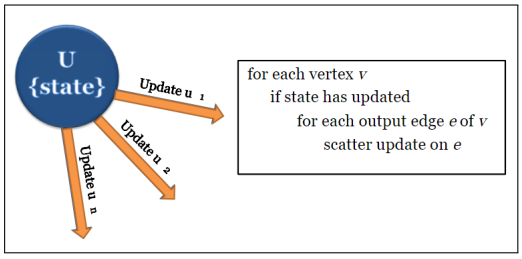

Abb. 4.2 Scatter im knoten-zentr. Modell [7]

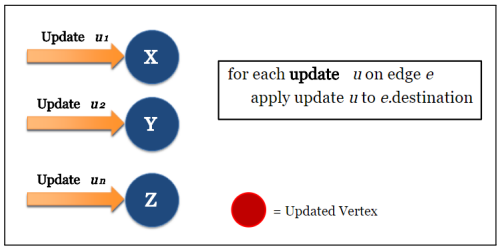

Abb. 4.3 Gather und Apply im kanten-zentr. Modell [7]

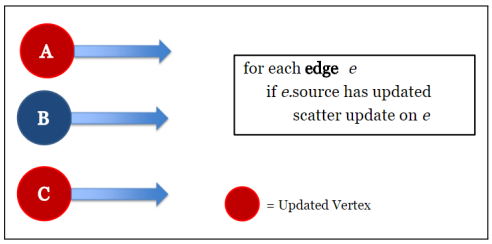

Abb. 4.4 Scatter im kanten-zentr. Modell [7]

Als weitere Variante gibt es das graph-zentrische Modell. Auch hier ist der Ablauf in Gather, Apply und Scatter Phase gegliedert. Die benutzerdefinierte Funktion wird in der Apply Phase nicht nur auf einzelne Knoten oder Kanten, sondern auf einen Teilgraphen angewendet. Voraussetzung hierfür ist also ein partitionierter Graph. Der Informationsaustausch erfolgt nur entlang der Kanten zwischen den Partitionen.
Nachteil des graph-zentrischen Modells ist die komplizierte Implementierung der benutzerdefinierten Berechnungsfunktion. Ein Vorteil ist hingegen die Vermeidung des immensen Kommunikationsoverheads von knoten- und kanten-zentrischem Modell. Außerdem können einige algorithmen-spezifische Optimierungen nur mit dem graph-zentrischen Modell umgesetzt werden. Da jedes der drei Modelle in unterschiedlichen Einsatzzwecken am besten geeignet ist, unterstützen einige Systeme verschiedene Modelle. [5, S. 103]

5 | Speicherung von Graphen

Auch die effektive Speicherung von Graphen stellt eine Herausforderung für Graphdatenbanksysteme dar. Bei der Speicherung eines Graphen als Adjazenzmatrix entstehen bei Graphen mit wenigen Kanten viele Nullen, welche mit abgespeichert werden müssen. Das ist bezüglich des Speichers sehr ineffizient. Um dieses Problem zu lösen kann die Adjazenzliste verwendet werden. Eine noch kompaktere Speicherung bietet der Compressed Row Storage (CRS), welcher für das Speichern dünn besetzter Matrizen besonders geeignet ist. Dabei werden alle Informationen der zu speichernden Matrix in nur drei Arrays festgehalten. Ein Beispiel dafür zeigen Abbildung 5.1 und 5.2.

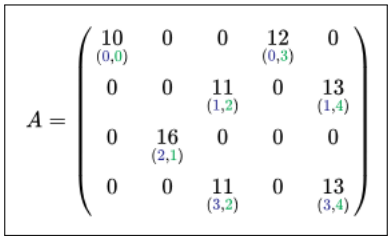

Abb. 5.1 Beispiel Matrix [8]

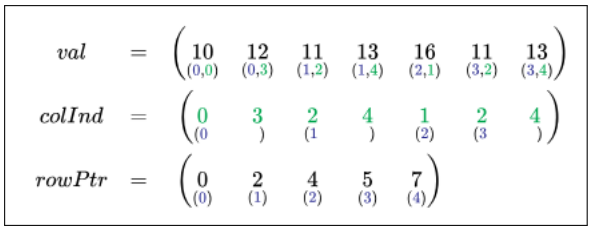

Abb. 5.2 Matrix im CRS [8]

In einer Graphdatenbank müssen neben der Topologie des Graphen auch Attribute der Knoten und Kanten gespeichert werden können. Die Möglichkeit dazu bieten Triple Tables. Diese sind speziell für die Speicherung von RDF Graphen geeignet. Es werden jeweils Subjekt, Prädikat und Objekt als Tripel gespeichert. Meist wird dabei eine Wörterbuchkompression angewandt. Nachteil der Triple Tables ist, dass das Lesen aller Attribute einer Entität viele Joins benötigt. Um eine join-freie Rekonstruktion von Knoten und Kanten mit all ihren Attributen zu schaffen, gibt es den Ansatz der Universal Tables. Dabei werden Knoten und Kanten in zwei Tabellen aufgeteilt. In der Knotentabelle stehen alle Knoten mit einer ID und allen in den Knoten vorkommenden Attributen als Spalten. In der Kantentabelle stehen alle Kanten mit der Quell- und Zielknoten-ID und allen in den Kanten vorkommenden Attributen als Spalten. Dabei entstehen Spalten mit sehr vielen NULL Werten, was es zu vermeiden gilt. Eine Möglichkeit die NULL Werte zu reduzieren, ist die Anwendung von Emerging Schemas. Das Ziel dabei ist es im Graphen enthaltene Schemas zu nutzen. Um diese zu finden können häufig zusammen vorkommende Attribute in einer gemeinsamen Tabelle gruppiert werden. Eine Entität kann in mehreren dieser Tabellen vorkommen. Die Gruppierung kann automatisiert durch Gruppierungsalgorithmen wie zum Beispiel k-means erfolgen.

Neben Emerging Schemas gibt es noch das Schema Hashing, um NULL Werte zu vermeiden. Dabei wird eine Tabelle aus n Spaltengruppen erstellt. Die Spaltengruppen bestehen jeweils aus einer Schlüssel- und einer Wertspalte. Die Zurordnung der Attribute zu den Spaltengruppen erfolgt durch eine Hashfunktion. [5, S. 103, 104]
Zur Verbesserung der Lesegeschwindigkeit kann in den Graphdatenbanken eine Indexierung sinnvoll sein. Vor allem wenn die Graphdaten in relationalen Strukturen wie einer Triple Table oder Knoten- und Kantentabellen vorliegen kann eine erhebliche Geschwindigkeitsverbesserung durch Indexierung erzielt werden. [5, S. 105]

6 | Engines zur Graphdatenverarbeitung

Engines zur Graphdatenverarbeitung basieren neben verschiedenen Speicher- und Datenmodellen auf verschiedenen Modellen zur parallelen Verarbeitung von Anfragen. Beim Bulk Synchronous Parallel Modell (BSP) erfolgt die Graphberechnung in Supersteps. In jedem Superstep erfolgen die drei in Kapitel 4 bereits beschriebenen Phasen Gather, Apply und Scatter. Am Ende jedes Supersteps erfolgt die Synchronisation durch eine globale Barriere. Erst wenn alle Knoten oder Kanten den Halt Status gesetzt haben, kann der nächste Superstep beginnen. Die von einem Knoten in der Scatter Phase gesendeten Informationen können vom Empfänger also immer erst im nächsten Superstep aufgenommen werden. Durch dieses Vorgehen muss immer auf langsamere Prozesse gewartet werden, was die Performance negativ beeinflusst.
Das Asynchronous Parallel Modell (AP) versucht dieses Problem zu entschärfen, indem Nachrichten jeweils direkt für den Empfänger sichtbar sind. Die globalen Barrieren werden dabei jedoch beibehalten. [5, S. 105]

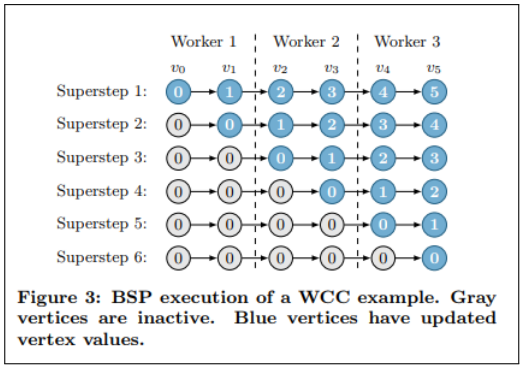

Figure 3: BSP execution of a WCC example. Gray vertices are inactive. Blue vertices have updated vertex values.

Abb. 6.1 BSP Ausführung des Weakly Connected Components Algorithmus [9]

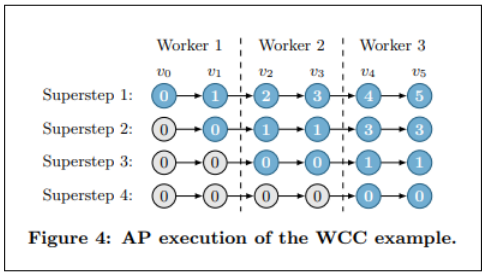

Figure 4: AP execution of the WCC example.

Abb. 6.2 AP Ausführung des Weakly Connected Components Algorithmus [9]

Die Abbildungen 6.1 und 6.2 stellen die Abarbeitung eines Algorithmus mit BSP und AP gegenüber. Dadurch, dass bei AP die Nachrichten direkt empfangen und weiterverarbeitet werden können, sind weniger Supersteps für die Berechnung des Ergebnisses nötig als beim BSP.
Ein weiteres Modell, welches bei Engines zur Graphdatenverarbeitung zum Einsatz kommt, ist das Gather, Apply and Scatter Modell (GAS). Dieses gibt es in einer synchronen und einer asynchronen Variante. Das synchrone GAS ist eine pull-basierte Version des BSP. Die Nachrichten werden also nicht selbstständig gesendet, sondern sie werden vom Empfänger selbst abgerufen. Beim asynchronen GAS muss deshalb zur Vermeidung der Abholung inkonsistenter Daten Distributed Locking eingesetzt werden. Außerdem gibt es beim GAS keine Supersteps, was die Prüfung für die Terminierung aufwändig macht.
Das laut eigener Einschätzung der Entwickler beste Modell, ist das Barrierless Asynchronous Parallel Modell (BAP). Dieses vermeidet die Nutzung lokaler Barrieren weitestgehend durch den Einsatz lokaler Barrieren, welche die Ausführung in logische Supersteps unterteilen. Die lokalen Barrieren sind Pausenpunkte, an denen Aufgaben wie Graph Mutationen oder die Prüfung, ob eine globale Barriere notwendig ist, durchgeführt werden können. Die BSP Programmierschnittstelle wird dabei erhalten. So ist die Programmierung nach dem Motto "program synchronously, execute asynchronously" einfach. Weitere Vorteile des BAP sind der verringerte Kommunikations- und Synchronisationsoverhead, sowie die Vermeidung des Wartens auf langsame Prozesse. Außerdem können Nachrichten sofort empfangen und verarbeitet werden. [9, S. 952-954]
Pregel und Giraph sind Engines zur Graphdatenverarbeitung, welche auf BSP in Kombination mit dem knoten-zentrischen Modell basieren. Abbildung 6.3 zeigt am Beispiel der Berechnung des kürzesten Pfades in einem Graphen die Arbeitsweise dieses Modells. Die Knoten werden bis auf den ersten zunächst alle mit "unendlich" initialisiert. In jedem Superstep werden dann die als Nachricht empfangenen Distanzen der Vorgängerknoten verglichen und der minimale Wert im Knoten gesetzt. Als Nachricht ausgesendet wird dann jeweils der gefundene Wert plus die Distanz zum jeweiligen Empfänger der Nachricht.

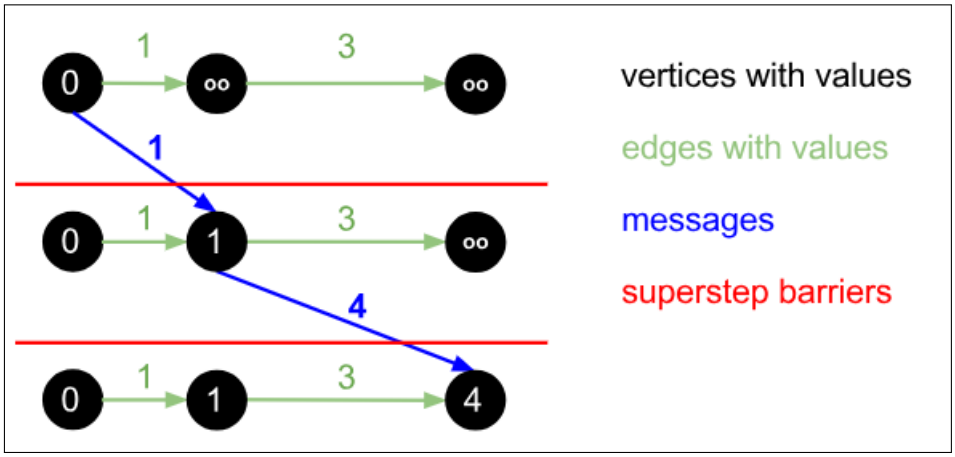

Abb. 6.3 Berechnung des kürzesten Pfades mit BSP [10]

Basierend auf Giraph wurde Giraph++ entwickelt. Diese Engine nutzt AP in Verbindung mit dem graph-zentrischen Modell. Eine weitere Engine, welche auf Giraph basiert, ist GiraphUC. Diese nutzt BAP in Kombination mit dem knoten-zentrischen Modell. Ein Beispiel für die Umsetzung des GAS Modells mit Distributed Locking ist GraphLab.
Abbildung 6.4 zeigt die verschiedenen Modelle im Laufzeitvergleich. Dabei wurden unterschiedliche Algorithmen auf unterschiedlichen Graphen durchgeführt. NS steht dafür, dass der Algorithmus von der Engine nicht unterstütz wird. F bedeutet, dass kein Ergebnis berechnet werden konnte. Die folgende Tabelle enthält Informationen zu den Graphen:

Graph	|V|	|E|	∅ Grad
Straßennetz (US)	23.9 Mio	57.7 Mio	2.4
Social Network (TW)	41.6 Mio	1.46 Mrd	35
Web Graph (UK)	105 Mio	3.73 Mrd	35

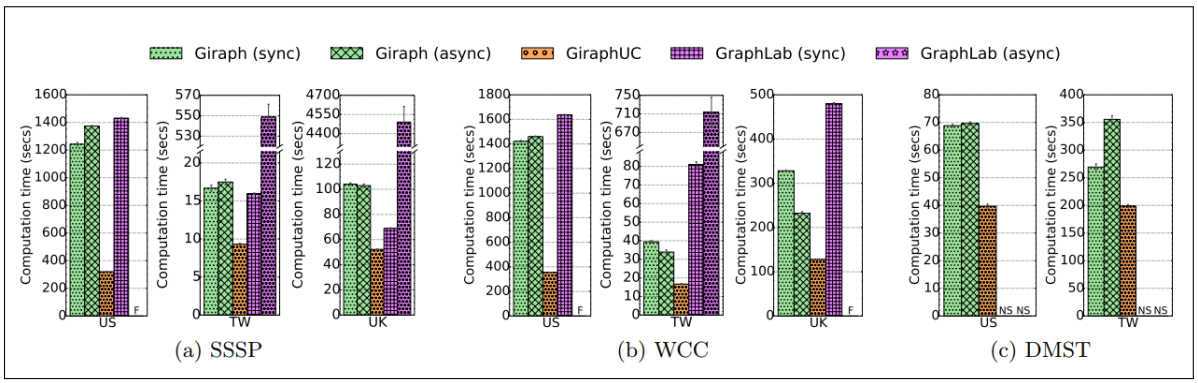

Abb. 6.4 Berechnungszeit für Single-source shortest path (SSSP), Weakly connected components (WCC) und Distributed minimum spanning tree (DMST) Algorithmen [9]

Besonders beim SSSP Algorithmus auf dem Straßennetzgraph kann GiraphUC mit dem BAP Modell seine Vorteile ausnutzen. Da dieser Graph so weit verteilt ist, benötigt Giraph tausende globale Supersteps um zum Ergebnis zu kommen. GiraphUC erreicht hier durch die hauptsächliche Verwendung der lokalen Supersteps eine etwa 4.5 mal bessere Performance. Auch bei allen anderen Messungen schneidet GiraphUC deutlich besser ab, als die anderen Engines. Nur beim DMST ist der Abstand zu den anderen nicht mehr ganz so groß, da hier mehr reine Berechnungszeit notwendig ist und nur wenig Kommunikation und Barrieren.

7 | Ausblick

Graphdatenverarbeitende Systeme werden in Zukunft wohl weiter an Popularität gewinnen. Gerade für die aktuellen Trends wie Big Data Verarbeitung und maschinelles Lernen sind diese Systeme sehr gut geeignet. Allerdings gibt es bereits jetzt viele verschiedene Systeme mit unterschiedlichen Ansätzen bezüglich des Datenmodells, der Speicherung von Graphen oder auch den Berechnungsmodellen. Deshalb ist definitiv eine sehr gründliche Recherche und Einarbeitung in das Thema notwendig, wenn man ein solches System einsetzen will. Schließlich soll es genau für die eigenen Ansprüche und Aufgabenstellungen geeignet sein. In großen Unternehmen mit sehr vielen Daten, wie zum Beispiel Google oder Facebook, lohnt sich dies sicher. Als kleines oder auch mittelständisches Unternehmen sollte man sich jedoch überlegen, ob ein graphdatenverarbeitendes System wirklich nötig ist. Dies nur einzuführen, weil es gerade ein Trend ist, ist nicht sinnvoll.

8 | Verzeichnisse

8.1 Abbildungsverzeichnis

8.2 Quellenverzeichnis

[1] Stefan Kolmar, Neo Technology (19.07.2017) Diese Vorteile bieten Graphdatenbanken https://www.bigdata-insider.de/diese-vorteile-bieten-graphdatenbanken-a-615118/ Abruf am 09.06.2018

[2] Neo Technology / Dr. Andreas Weber Bild Produktdaten innerhalb eines Graphen https://www.bigdata-insider.de/index.cfm?pid=12555&pk=1240706&type=article&fk=615118 Abruf am 09.06.2018

[3] DB-Engines (2018) DBMS Popularität pro Datenbankmodell
https://db-engines.com/de/ranking_categories
Abruf am 14.06.2018

[4] Martin Junghanns, Andre Petermann, Martin Neumann, Erhard Rahm: Management and Analysis of Big Graph Data: Current Systems and Open Challenges, Handbook of Big Data Technologies, 2017

[5] Marcus Paradies, Hannes Voigt: Big Graph Data Analytics on Single Machines – An Overview, in: Datenbank-Spektrum Bd. 17, Heft 2, Juli 2017

[6] Marcus Stuber: Eine Einführung in das Graphdatenbankmodell, 27. März 2012

[7] Myron Carroll, GRAPH PROCESSING
http://slideplayer.com/slide/7406518/#
Abruf am 17.06.2018

[8] Wikipedia, Compressed Row Storage
https://de.wikipedia.org/wiki/Compressed_Row_Storage
Abruf am 17.06.2018

[9] Minyang Han, Khuzaima Daudjee: Giraph Unchained: Barrierless Asynchronous Parallel Execution in Pregel-like Graph Processing Systems

[10] The Apache Software Foundation, Apache Giraph
http://giraph.apache.org/intro.html
Abruf am 18.06.2018