Graph-Datenbanken: Nur ein Hype oder das Ende der relationalen Welt?

Hans Viehmann
ORACLE Deutschland B.V. & Co. KG
NL Hamburg

Schlüsselworte
Big Data, Graph-Datenbank, Pattern Matching, Social Network Analysis, Recommendation Engine, Influencer Detection, Data Warehousing, NoSQL, Apache HBase

Zusammenfassung
Graph-Datenbanken erfreuen sich zunehmender Beliebtheit, ihnen wird insbesondere Flexibilität, intuitive Modellierung und die Einfachheit von Abfragen zugeschrieben. Anders als in relationalen Datenbanken werden Entitäten nicht in Tabellen gespeichert, sondern in einem Knoten-Kanten-Modell abgelegt. Hierbei verweisen die gespeicherten Objekte direkt auf mit ihnen in Beziehung stehende Objekte. Seit 2015 bietet Oracle unter dem Produktnamen Oracle Big Data Spatial and Graph auch eine Graph-Datenbank an, die auf dem Konzept von Property Graphs basiert. Für die eigentliche Datenhaltung nutzt sie entweder Oracle NoSQL oder Apache HBase und zur Analyse ist sie mit einer in-memory Engine ausgestattet, die neben der für Graph-Datenbanken üblichen Blueprints Programmierschnittstelle bereits eine Pattern-Matching Abfragesprache, sowie zahlreiche vorgefertigte Algorithmen umfasst.
Dieser Beitrag erläutert die Grundlagen von Property Graph Databases und zeigt diejenigen Einsatzfelder auf, in denen Property Graph Implementierungen der relationalen Technologie überlegen sind. Neben Architektur und Abfragesprache von Oracle Big Data Spatial and Graph werden einige Algorithmen anhand von Beispielen dargestellt. Außerdem wird die Umsetzung von relationalen Tabellen in einen Graph betrachtet und dessen Visualisierung mit typischen Werkzeugen gezeigt. Abschließend sind noch Performancevergleiche angefügt, die mit gängigen Datenbeständen durchgeführt wurden.

Einleitung

Graph-Datenbanken (auch: graphenorientierte Datenbanken) unterscheiden sich von relationalen Technologien vor allem darin, dass sie Entitäten und deren Beziehungen nicht in Tabellen, sondern in Form eines Knoten-Kanten-Modells speichern, wie es schematisch in Abb. 1 dargestellt ist. Sie werden immer häufiger benutzt, um beispielsweise stark vernetzte Datenbestände zu verwalten und auszuwerten. Hierbei spielen insbesondere so genannte Property Graph-Datenbanken eine Rolle, bei denen sowohl Knoten, als auch Kanten mit Eigenschaften (properties) verknüpft sind und für die in der Literatur zahlreiche aus der Graphentheorie bekannte Algorithmen beschrieben sind. Dieser Ansatz unterscheidet sie etwa von Triple Stores (RDF Graph-Datenbanken), wie sie im Umfeld des Semantic Web eingesetzt werden und wie sie beispielsweise in der Oracle Spatial and Graph Option zur Oracle Datenbank enthalten sind.
Property Graph Technologien werden gern zur Modellierung und Analyse sozialer Netzwerke eingesetzt, werden aber auch zur Auswertung von Kunde-Lieferanten-Beziehungen, zur Analyse physikalischer Netze, wie z.B. Versorgungsnetze oder IP Netzwerke, zur Modellierung von Protein-Wechselwirkungen, o.ä. verwendet. Knowledge Graphs, wie sie Apple SIRI oder der Google Suche zugrunde liegen, basieren ebenfalls darauf.
Im Gegensatz zu relationalen Technologien, die eine sorgfältige Modellierung erfordern, sind Property Graph Datenbanken sehr flexibel. Sie erfordern kein vordefiniertes Schema, sondern können mit einer einfachen Skizze vom Flipchart beginnen und sich im Verlauf des Projekts entwickeln. Neue

Entitäten, Beziehungen oder Eigenschaften können laufend hinzugefügt werden, ohne die bestehende Funktionalität zu beeinträchtigen. Da Entitäten keinen Platzhalter für alle möglicherweise auftretenden Eigenschaften oder Beziehungen benötigen, eignet sich dieses Schema der Speicherung insbesondere für „sparse data“, also Einsatzfälle, bei denen die Tabellen in relationalen Datenbanken zahlreiche NULL Werte enthalten würden.

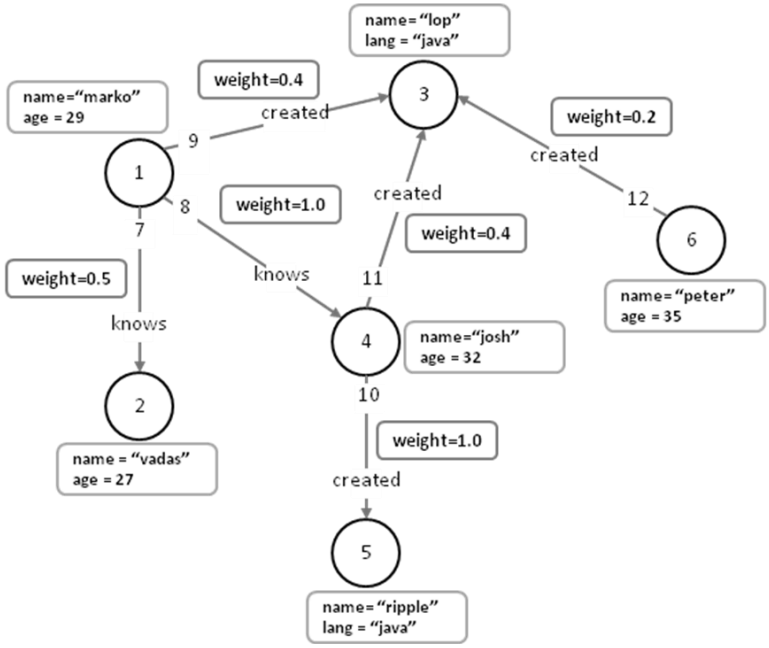

Abb. 1: Allgemeines Modell eines Property Graph (nach: github.com/tinkerpop/blueprints/wiki/Property-Graph-Model)

Der wesentliche Mehrwert von Graph-Datenbanken besteht darin, dass sie Analysen ermöglichen, die mit konventionellen Datenbanken gar nicht oder nur mit hohem rechnerischen Aufwand umsetzbar wären. Im folgenden ist eine Reihe von entsprechenden Einsatzfällen aufgeführt. Diese teilen sich grundsätzlich in zwei Kategorien – entweder handelt es sich um rechnerische Verfahren, die je Knoten bzw. Kante Indikatoren berechnen, die aus (meist wiederholter) Iteration über den gesamten betrachteten Graphen abgeleitet werden, oder es handelt sich um Pattern Matching Verfahren, bei denen ein vorgegebenes Muster genutzt wird, um alle gleichartigen Teilgraphen im Datenbestand zu identifizieren.

Einsatzfall Analyse sozialer Netze

Graph-Datenbanken lassen sich sehr elegant zur Auswertung von sozialen Netzwerken nutzen, wobei hier nicht nur Facebook oder LinkedIn gemeint sind, sondern auch Beziehungen zwischen Kunden in CRM Systemen, Beziehungen zwischen Kunden und Lieferanten, oder auch Netzwerke von Kriminellen oder Verdächtigen in der Verbrechensbekämpfung betrachtet werden. Möchte man etwa die „Wichtigkeit“ eines Kunden bestimmen, kann man ermitteln, wie dieser Kunde mit anderen Kunden in Beziehung steht. Für diese Art der Analyse verwendet die Graphentheorie den Begriff der Zentralität eines Knoten im Graphen. Im einfachsten Fall stellt sie ein Maß dafür dar, wie viele Beziehungen - also Kanten im Graphen - von einem Kunden ausgehen (Degree Centrality). Es gibt aber auch komplizietere Varianten wie das Pagerank Verfahren, das neben der Zahl von Beziehungen auch berücksichtigt, ob ein Knoten mit einem „wichtigen“ Knoten verbunden ist und diesen dann entsprechend aufwertet. Mit diesem Algorithmus wird gern im Umfeld von CRM Systemen gearbeitet, um gezielter Marketing betreiben zu können. Baut man etwa im Bereich der Telekommunikation einen

Graph aus allen Mobilfunkverbindungen (Call Data Records) über einen gegebenen Zeitraum auf, indem man jedem Kunden einen Knoten zuordnet und jede Verbindung durch eine Kante repräsentiert, kann man aus dem Graphen mit diesem Algorithmus die am intensivsten vernetzten Kunden identifizieren und gezielt für Werbemaßnahmen auswählen. Diese Art von Berechnung lässt sich auf relationalen Systemen aufgrund der hohen Zahl rekursiver Joins nicht realistisch durchführen.

Einsatzfall Ähnlichkeitsanalyse für Recommendation Engines
Recommendation Engines sind im Umfeld von eCommerce sehr gängig und etwa bei Amazon seit Jahren im Einsatz. Sie basieren darauf, dass zu einem Nutzer mit gegebenen Kundenverhalten eine Gruppe ähnlicher Kunden aus dem Bestand gefunden wird und auf Basis der Präferenzen in der Gruppe dem Nutzer Produkte vorgeschlagen werden. Technisch lässt sich das über verschiedene Ansätze erreichen; im folgenden wird ein Matrix-basierter (Collaborative Filtering) und ein Ranking-basierter (Personalized Pagerank) Ansatz kurz dargestellt.

1. Recommendation Engines auf Basis Collaborative Filtering
Trägt man für eine eCommerce Plattform alle Produkte und alle Kunden in einer Matrix gegeneinander auf und markiert jeweils diejenigen Kombinationen von Produkt und Kunde, für die eine Interaktion vorhanden war (z.B. Produkt gekauft, Produkt angesehen, Produkt negativ bewertet, ...), entsteht eine sehr große Matrix, die ganz überwiegend aus leeren Zellen besteht. Diese Art von Matrix lässt sich nicht ohne weiteres in einer relationalen Datenbank abbilden, kann aber sehr kompakt als Graph modelliert und mittels Algorithmen aus der Graphentheorie ausgewertet werden. Der zugehörige Lösungsalgorithmus wird als Matrix Factorization bezeichnet und ist Bestandteil des Produktumfangs von Oracle Big Data Spatial and Graph. Er dient dazu, eine Art Fingerabdruck („taste signature“) zu berechnen, der die Präferenzen des Kunden widerspiegelt, bzw. im Gegenzug rekursiv eine „taste signature“ des Produkts zu ermitteln, der die Kunden, die das Produkt bevorzugen repräsentiert.
Um diese Funktionalität zu nutzen, muss nur ein Graph aufgebaut werden, der alle Interaktionen von Kunde und Produkt enthält. Der Matrix Factorization Algorithmus kann dann einfach aufgerufen werden, um anschließend zu einem gegebenem Kunden genau das Produkt vorgeschlagen zu bekommen, das er/sie gemäß dem Algorithmus mit der höchsten Wahrscheinlichkeit kauft.

2. Recommendation Engines auf Basis Personalized Pagerank
Hat man alle Interaktionen von Kunde und Produkt in einem Graph modelliert, lassen sich auch andere Vorhersagemodelle verwenden, um Produkte vorzuschlagen. Insbesondere der Personalized Pagerank Algorithmus wäre hier zu nennen. Dieser Algorithmus startet bei einem gegebenen Knoten (Kunden) und folgt einem zufälligen Pfad von Kunde zu Produkt zu Kunde, usw., bis zu einem willkürlichen Abbruchzeitpunkt. Anschließend springt er zum Ausgangspunkt zurück und legt – wieder und wieder – einen zufälligen Weg entlang des Graphen zurück. Wählt man die Anzahl Iterationen groß genug und zählt die Anzahl der Male, die ein Produkt bzw. Kunde im Laufe der Iterationen auf dem Pfad gelegen hat, erhält man ein Maß dafür, wie „ähnlich“ der jeweilige Kunde bzw. wie „naheliegend“ das jeweilige Produkt ist. Diejenigen Produkte, die von Kunden gekauft wurden, die mit dem Startknoten über viele Produkt-Kunden Interaktionen verbunden sind, werden wesentlich häufiger auf dem zufälligen Pfad liegen, als ein Produkt, das nur über eine große Zahl von Zwischenschritten erreicht werden kann.
Genau wie im Fall von Collaborative Filtering ist der Algorithmus bereits im Produktumfang enthalten und kann zu einem gegebenen Startknoten direkt die nächstliegenden Produkte zurückliefern. In relationaler Technologie wäre dieser Algorithmus aufgrund der zahlreichen Joins jeweils zwischen Produkt und Kunde in Kombination mit den vielfachen Iterationen sehr langsam.

Einsatzfall Anomaly Detection
Der Personalized Pagerank Algorithmus ist nicht nur dafür geeignet, in einem Graph mit Beziehungen zwischen Kunden und Produkten die zu einem Knoten besonders „ähnlichen“ Knoten zu ermitteln. Er kann ebenso dazu genutzt werden, diejenigen Knoten in einem Graphen zu identifizieren, die auffallend „anders“ sind, als die übrigen Knoten. Sucht man zu einer Vergleichsgruppe diejenigen Knoten mit einem geringen Personalized Pagerank Wert, erhält man sozusagen die „Außenseiter“. In einem Graphen in dem beispielsweise Ärzte und deren Behandlungen in Beziehung gesetzt sind, kann man so diejenigen Ärzte identifizieren, die Behandlungen durchführen, die in ihrer Vergleichsgruppe besonders unüblich sind.

Einsatzfall Errichbarkeit und kürzester Pfad
Ein weiterer typischer Einsatzbereich von Graph-Datenbanken ist die Verwendung zur Erreichbarkeitsanalyse bzw. zur Berechnung des kürzesten oder optimalen Pfades zwischen gegebenen Punkten in einem Netzwerk. Diese Fragestellung stammt herkömmlich aus dem Bereich der Versorgungsnetze oder Routenplaner, kann aber vielfältig auch für andere Zwecke eingesetzt werden. Betrachtet man beispielsweise ein Stromverteilnetzwerk und möchte feststellen, welche Haushalte durch den Ausfall eines Netzwerkknotens betroffen sind, muss man von der Quelle (Kraftwerk) kommend nur das gesamte Netzwerk mit Ausnahme des defekten Knotens traversieren und alle erreichbaren Knoten als solche markieren, um festzustellen, welche Haushalte noch versorgt werden. In einem relationalen Umfeld könnte man bei einem rein hierarchischen Netzwerk mittels rekursiven WITH statements ähnlich vorgehen, aber sobald der Graph Zyklen enthält, versagt dieser Ansatz und muss durch prozedurale Logik ergänzt werden.

Oracle Big Data Spatial and Graph – Grundlagen
Mit Oracle Big Data Spatial and Graph (BDSG) liefert Oracle eine Property Graph Datenbank samt in-memory Analysis Engine an, die für alle diese Einsatzfälle ausgelegt ist. Neben der Datenhaltung, die in Oracle NoSQL oder Apache HBase erfolgt und die mit den in diesem Umfeld gängigen Programmierschnittstellen ausgestattet ist, besteht der wesentliche Wert vor allem in den über 40 vorgefertigten Algorithmen aus den Bereichen Ranking, Community Detection, Path Finding, Recommendation, usw., die bereits mit dem Produkt ausgeliefert werden. Die Ergebnisse von Analysen im Hauptspeicher lassen sich in der Datenhaltung persistieren und und stehen über Big Data

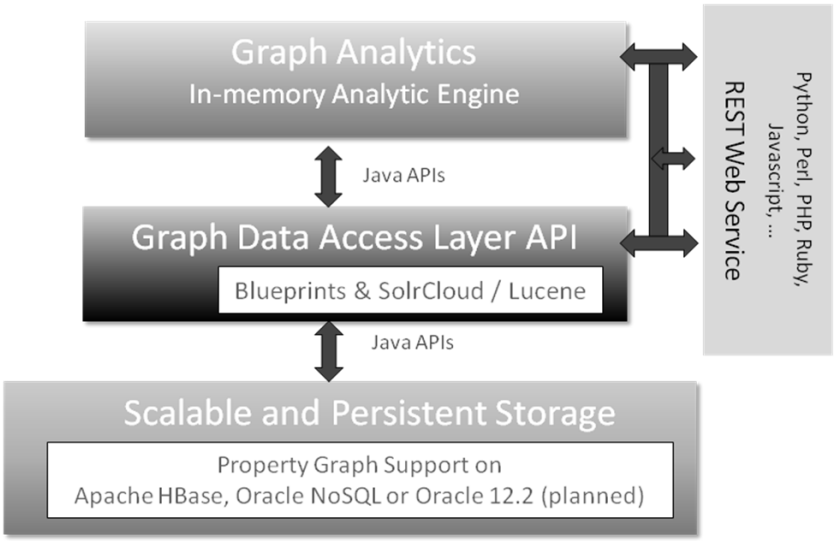

Abb. 2: Schematische Architektur von Oracle Big Data Spatial and Graph

SQL oder External Tables (Big Data Connectors) auch für den Zugriff aus der Oracle Datenbank zur Verfügung. Eine schematische Architektur ist in Abb. 2 dargestellt.

Aufbau eine Graphen

Die Struktur eines Property Graph ist relativ simpel – zu jedem Knoten gibt es einen eindeutigen Schlüssel, eine Menge von Kanten, gerichtet oder ungerichet, sowie eine Menge von Eigenschaften als Key-value Paare. Ebenso gibt es zu jeder Kante einen eindeutigen Schlüssel, einen Anfangs- und einen Endknoten, einen Label, der die Art der Beziehung enthält, sowie wiederum eine Menge von Eigenschaften als Key-value Paare (vgl. Abb. 1).
Liegt bereits ein relationales Datenmodell vor, gibt es verschiedene Möglichkeiten, diese als Property Graph zu modellieren. Ein erster Ansatz ist üblicherweise, alle Zeilen einer Tabelle in Knoten zu übersetzen, die Spaltenwerte als Eigenschaften zu modellieren und Primärschlüssel-Fremdschlüssel Beziehungen als Kanten abzubilden. Im Falle von n:m Beziehungen fallen die Join Tabellen weg, die Beziehungen werden direkt an die Knoten angefügt und alle Spaltenwerte der Join Tabelle können als Eigenschaften der jeweiligen Kante modelliert werden.
Das eigentliche Anlegen eines Graphen kann entweder programmatisch über elementare Operationen wie AddVertex, AddProperty, ... erfolgen, indem schrittweise Elemente und Eigenschaften hinzugefügt werden, oder über graphische Tools stattfinden, die ihrerseits diese Methoden aufrufen. Liegt ein Datenbestand schon als Graph vor, gibt es eine Reihe von Datenaustauschformaten (GraphSON, GraphML, GML), die produktseitig unterstützt werden. Auch zum Laden aus Tabellen oder CSV Dateien ist ein entsprechendes API vorhanden, es können aber ebensogut Werkzeuge wie der Oracle Data Integrator verwendet werden.

Zugriff auf den Graphen

Da sich bislang keine Standards ähnlich SQL für die Arbeit mit Graph-Datenbanken etabliert haben, erfolgt der Zugriff über Programmierschnittstellen (APIs). Im Zuge des Apache Tinkerpop Projekts hat sich mit den Blueprints APIs immerhin ein Industriestandard ähnlich JDBC entwickelt, der für Java und REST Schnittstellen definiert, die von verschiedenen Anbietern unterstützt werden. Im Falle von Oracle Big Data Spatial and Graph ist neben den reinen Blueprints APIs auch die Unterstützung von Groovy und Python samt iNotebook Integration, sowie ein Node.js Language Binding enthalten. Möchte man in Groovy etwa zu jedem Knoten im Netzwerk berechnen, wie viele Knoten innerhalb von zwei Schritten erreichbar sind (entspricht z.B. der Anzahl Freunde von Freunden je Person), sieht das mit Hilfe des Blueprints API etwa folgendermaßen aus:

```
sum([v.query() \
    .direction(blueprints.Direction.OUT).count() \
    for v in OPGIterator(v0.query() \
      .direction(blueprints.Direction.OUT) \
      .vertices().iterator())])
```

Diese low-level Programmierung ist für einfache Einsatzfälle nicht allzu schwierig, kann aber für komplexere Fragestellungen unübersichtlich werden. Daher wurden in Oracle Big Data Spatial and Graph über 40 Algorithmen in jeweils einem einzigen API call implementiert. Möchte man beispielsweise die nach dem „Betweenness Centrality“ Ansatz wichtigsten 15 Knoten im Netzwerk bestimmen, reduziert sich die Programmierung auf:

```
b = analyst.betweenness().topK(15)
```

wobei natürlich zuvor der Zugriff auf die Graph-Datenbank konfiguriert, eine Session instanziiert und eine Instanz des Analyst erzeugt worden sein muss, die mit dem Graph aus der eigentlichen Datenhaltung befüllt wurde. Die Analyst-Instanz stellt die in-memory engine bereit, in der die Auswertung stattfindet. Der gesamte Graph befindet sich dabei im Hauptspeicher, was für die

performante Analyse enorm hilfreich ist. Diese Analyst-Instanz kann sich lokal im Hauptspeicher befinden (embedded), oder in einer Java-Laufzeitumgebung auf einem separaten Server. Alternativ kann sie auch auf einem Hadoop Cluster liegen und durch YARN als Resource Manager verwaltet werden. Die Abbildung des Graphen in memory ist sehr kompakt – so findet beispielsweise ein Graph mit etwa 23 Mrd. Kanten auf einem Server der Big Data Appliance Platz – aber naturgemäß ist Hauptspeicher begrenzt. Sollte der Graph mehr Speicherplatz erfordern, gibt es seit BDSG 2.0 eine Implementierung auf verteilten Servern.

Pattern Matching mittels PGQL

Für die Suche nach vorgegebenen Mustern im Datenbestand enthält Oracle Big Data Spatial and Graph eine Pattern Matching Abfragesprache (PGQL=Property Graph Query Language, siehe http://pgql-lang.org/), die aktuell zur Standardisierung vorgeschlagen ist. Diese Abfragesprache ist von der Syntax her an SQL angeleht und vor allem für die interaktive Analyse gedacht. Typische Einsatzbereiche sind Fraud Detection – etwa zum Aufspüren von Betrugsringen – , die Extraktion von Sub-Graphen, oder die Ähnlichkeitssuche.

Mittels PGQL lassen sich topologische Bedingungen formulieren, die in einer WHERE Klausel in der Form `()-[]->()` abgebildet werden. Hierin repräsentieren runde Klammern Knoten, der Pfeil die (gerichtete) Beziehung und die eckige Klammer dient zur Formulierung von Bedingungen entlang der Kante. Die genaue Spezifikation der Sprache ist unter der obigen URL zu finden, daher sei hier nur ein einfaches Beispiel angeführt. Will man etwa in einem sozialen Netzwerk den Namen und das Alter aller Personen herausfinden, die Freunden von Klaus bekannt sind, könnte eine Abfrage wie folgt aussehen:

```
SELECT v3.name, v3.alter
FROM 'myGraph'
WHERE
  (v1:Person WITH name = 'Klaus') -[:befreundetMit]-> (v2:Person)
  -[:kennt]-> (v3:Person)
```

Liegt nur ein Graph im System vor und ist dieser unbenannt („unnamed graph“), entfällt die FROM Klausel, was für Entwickler relationaler Systeme zunächst etwas ungewohnt aussieht.

Um effizient auf Knoten mit Eigenschaften in Textform zugreifen zu können, ist in Oracle Big Data Spatial and Graph die Text Search Engine Lucene bzw. SolrCloud integriert. Dies ermöglicht den Aufbau von Textindices und darüber eine große Zahl verschiedener Suchoperationen, inklusive unscharfer Suche. So kann man beispielsweise statt nur nach `'Klaus'` als String auch nach `'*Klaus*'`, `'Claus~'`, usw., suchen.

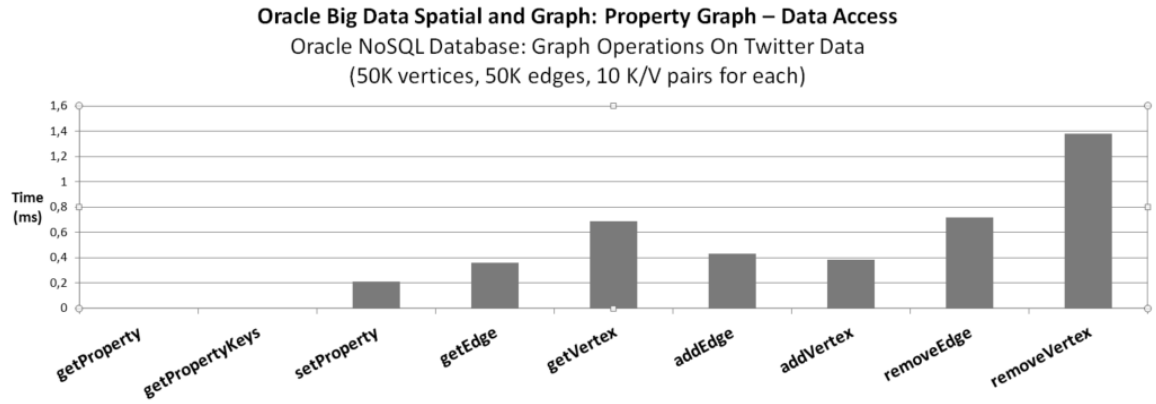

Abb. 3: Gemessene Zugriffszeiten auf einen kleinen Graph

Performance
Bei der Betrachtung der Ausführunsgeschwindigkeit von Operationen muss man zwischen den elementaren Operationen auf der Graph-Datenbank (getEdge, getVertex, getProperty, addVertex, removeEdge, usw.) und der Ausführung der Algorithmen im Hauptspeicher unterscheiden.
Da der Graph in einer Oracle NoSQL Database oder in Apache HBase gespeichert ist und daher in einer hochskalierbaren Datenhaltung mit im Verhältnis zu konventionellen relationelen Datenbanken geringem Overhead liegt, sind die elementaren Operationen zur Abfrage und Änderung einzelner Datensätze im Graph extrem schnell. Bereits auf einem kleinen System erreicht man üblicherweise Ausführungszeiten unterhalb einer Millisekunde, wobei Operationen wir getProperty kaum messbar sind. Für komplexe Analyse-Algorithmen auf größeren Datenmengen (ShortestPath, PageRank, ...) benötigt man natürlich signifikante Ausführungszeiten, wobei die kompakte Verwaltung des Datenbestands im Hauptspeicher hier große Vorteile bringt. Ein detaillierter Benchmark würde den Umfang dieses Beitrags sprengen. Es gibt aber ein umfassendes White Paper in dem die Performance verschiedener Algorithmen auf gängigen Datenbeständen untersucht und im Vergleich zu Dato GraphLab oder Spark GraphX ausgewertet wurden. Hierbei zeigte sich, dass die Implementierung in BDSG teilweise um Größenordnungen performanter war. Auch im Vergleich zu Neo4j waren die Ausführungszeiten wesentlich kürzer. Für Algorithmen wie ShortestPath oder ähnliche Path Queries

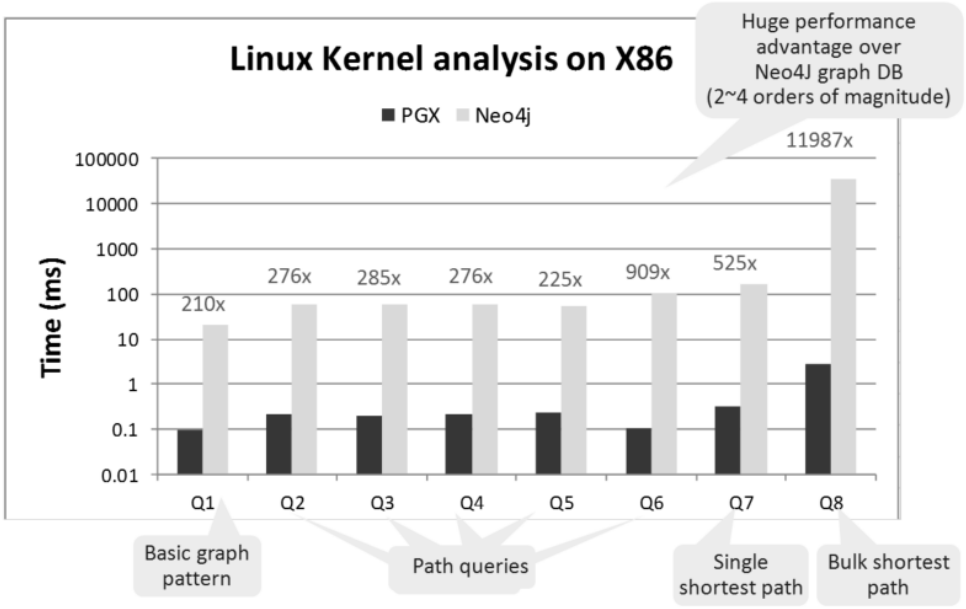

Abb. 4: Performancevergleich von BDSG gegen Neo4j 2.2.1

lagen die Unterschiede üblicherweise zwischen zwei und vier Größenordnungen (!) bei einem Graph, der den Linux Kernel code repräsentiert und auf einem Server mit zwei x86 CPUs à 8 cores ausgewertet wurde (s. Abb. 4). Auch im Falle der verteilten Analyse, also der Ausführung der Graph Algorithmen auf mehreren veteilten Rechnern, skaliert BDSG ausgezeichnet. Allerdings ist eine schnelle Netzwerkverbindung zwischen den Rechnerknoten angebracht, da naturgemäß im Zuge der verteilten Berechnungen Datenaustausch zwischen den Servern erfolgen muss.

Graphische Werkzeuge
Graph-Datenbanken erfordern nicht immer eine Benutzeroberfläche, um den Graph auch visualisieren zu können, aber für bestimmte Einsatzszenarien, in denen interaktive Analyse erforderlich ist, kann dies schon hilfreich sein. Im Lieferumfang von Oracle Big Data Spatial and Graph ist daher Cytoscape

(s. www.cytoscape.org) als Open Source Visualisierungstool enthalten. Für die Software ist ein Plug-in vorhanden, das die Anbindung an BDSG übernimmt und es ermöglicht, interaktiv auf den Datenbestand zuzugreifen und Algorithmen auf dem Graph auszuführen. Daneben existieren kommerzielle Tools wie Tom Sawyer Perspectives, das seit Version 7.5 standardmäßig mit einer Anbindung an BDSG ausgestattet ist (s. Abb. 5).

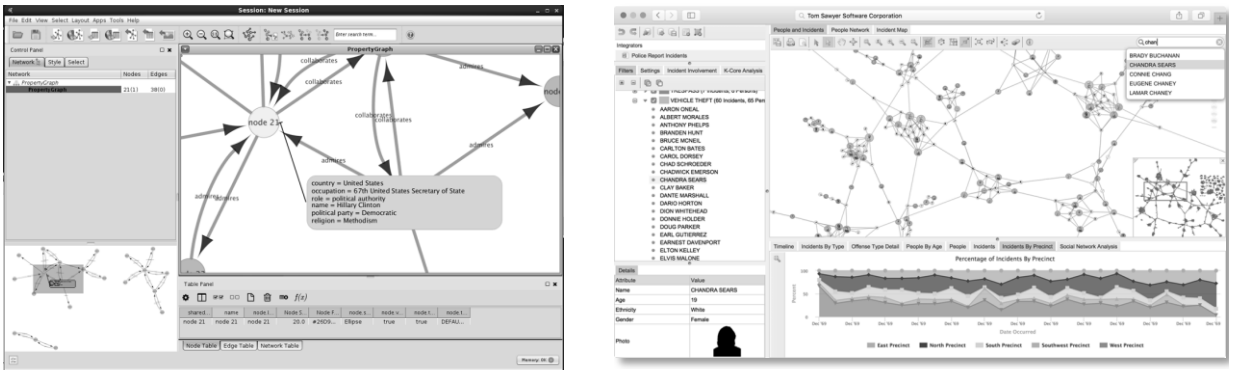

Abb. 5: Screenshots von Cytoscape (links) und Tom Sawyer Perspectives (rechts)

Zusammenfassung

Graph-Datenbanken bieten zahlreiche Analysemöglichkeiten, die mit relationalen Technologien nicht oder nur nur sehr aufwändig umgesetzt werden können. Überall dort, wo Beziehungen oder Abhängigkeiten ausgewertet werden müssen, vor allem, wenn sie die Topologie des gesamten Beziehungsnetzwerks betreffen oder Beziehungen über mehrere Stufen aufgedeckt werden sollen, sind sie ausgesprochen hilfreich. Sie können daher insbesondere als wertvolle Ergänzung zu bestehenden Data Warehouse Implementierungen und Reporting-Lösungen dienen. Aufgrund ihrer generischen Implementierung könnten sie auch als Ersatz für relationale Datenbanken eingesetzt werden. Dem sprechen bislang aber noch Aspekte wie die fehlende Standardisierung, mangelnde Tool- und Lösungsunterstützung, fehlende Skills, usw., entgegen. Außerdem sind relationale Datenbanken nach wie vor in der Verarbeitung von Set-Operationen auf großen Datenmengen im Vorteil.

Weitere Informationen

Nähere Informationen zu Oracle Big Data Spatial and Graph sind auf dem Oracle Technology Network zu finden (www.oracle.com/technetwork/database/database-technologies/bigdata-spatialandgraph); daneben gibt es unter blogs.oracle.com/bigdataspatialgraph/ einen Blog mit hilfreichen Beiträgen, der vom Development Team gepflegt wird. Zum Einstieg eignet sich die BigDataLite Virtual Machine (siehe www.oracle.com/technetwork/community/developer-vm/index.html), in der die komplette Umgebung konfiguriert und eingerichtet ist. Ein Hands-on lab (unter /opt/oracle/oracle-spatial-graph) wird mit dieser virtuellen Umgebung mit ausgeliefert und enthält auch einen kleinen Demo-Datenbestand.

Kontaktadresse:

Hans Viehmann
ORACLE Deutschland B.V. & Co. KG
Kühnehöfe 5
D-22761 Hamburg

Telefon:	+49 (0) 40-89091-173
Fax:	+49 (0) 40-89091-250
E-Mail	hans.viehmann@oracle.com
Internet:	www.oracle.com/goto/bigdata