B. Mitschang et al. (Hrsg.): BTW 2017 – Workshopband,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 105

Distributed FoodBroker: Skalierbare Generierung graphbasierter Geschäftsprozessdaten

Stephan Kemper[1], André Petermann[2] und Martin Junghanns[2]

Abstract:

Graphen eignen sich zur Modellierung und Analyse komplexer Zusammenhänge zwischen beliebigen Objekten. Eine mögliche Anwendung ist die graphbasierte Analyse von Geschäftsprozessen. Für die Entwicklung und Evaluierung entsprechener Analysetools werden Datensätze benötigt. FoodBroker ist ein Datengenerator, welcher vordefinierte Geschäftsprozesse simuliert und die Daten in Form von Graphen lokal auf einem Rechner erzeugt. Um Graphen beliebiger Größer erstellen zu können, zeigen wir in diesem Beitrag wie FoodBroker mit Hilfe der Open-Source-Frameworks GRADOOP und Apache Flink auf verteilten Systemen implementiert werden kann.

Keywords: Datengenerierung, Geschäftsprozesse, Verteilte Systeme, Gradoop, Apache Flink

1 Einleitung

Für die Entwicklung und Evaluierung von Tools zur graphbasierten Analyse von Prozessdaten werden Datensätze benötigt. Selbst für rein wissenschaftliche Zwecke ist es jedoch schwer an reale Geschäftsdaten zu gelangen. Dies kann an Datenschutzgründen liegen, aber auch daran, dass Firmen selbst gewinnbringende Erkenntnisse aus den Daten ziehen wollen. Eine Alternative ist die Verwendung eines Generators, der Daten mit realäquivalenten Eigenschaften erzeugt. FoodBroker [Pe14] ist ein solcher Generator, der auf der Simulation von konkreten Geschäftsprozessen basiert. Die aktuelle Implementierung unterstützt jedoch nur die Parallelisierung über mehrere Threads eines einzelnen Rechners und ist daher ungeeignet um Datenvolumen zu erzeugen, wie sie in sehr großen Unternehmen anfallen und für Skalierbarkeitsuntersuchungen neuer analytischer Verfahren benötigt werden. Hierdurch motiviert entstand der in diesem Beitrag beschriebene Ansatz, FoodBroker unter Verwendung von BigData-Technologien neu zu implementieren. Dies geschieht in der Absicht die horizontal skalierbare Ausführung auf einem Rechencluster zu ermöglichen.

Die vorliegende Arbeit beschreibt Distributed FoodBroker. In Kapitel 2 wird die FoodBroker-Simulation zunächst kurz eingeführt. Die für die verteilte Ausführung verwendeten Frameworks, sowie die Implementierung des Generators werden in Kapitel 3 beschrieben. Kapitel 4 fasst die Ergebnisse verschiedener Experimente zur Skalierbarkeit zusammen. In Kapitel 5 werden verwandte Arbeiten diskutiert und in Kapitel 6 wird der Artikel zusammengefasst.

[1] Universität Leipzig, Abteilung Datenbanken, kemper@studserv.uni-leipzig.de
[2] Universität Leipzig & ScaDS Dresden/Leipzig, [petermann,junghanns]@informatik.uni-leipzig.de

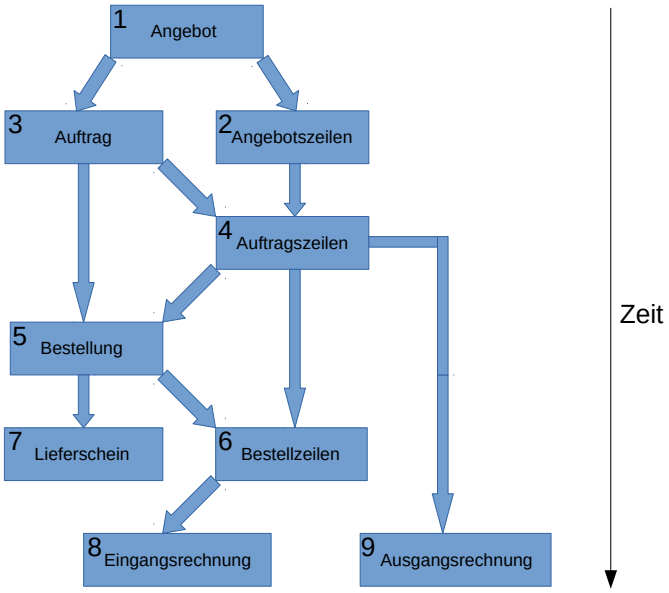


Abb. 1: Zeitlicher Ablauf des Brokerage-Prozesses. Die Reihenfolge der Erstellung wird durch die Zahlen wiedergegeben.

2 FoodBroker

FoodBroker simuliert vordefinierte Geschäftsprozesse eines Unternehmens, welches Nahrungsmittel handelt. Dabei werden Daten über alle wesentlichen Schritte, vom Angebot bis hin zur Lieferung und Tickets, z.B. aufgrund von Qualitätsmängeln oder verspäteter Lieferung, erstellt. Die Simulation erzeugt zwei Arten von Daten. Zunächst gibt es Stammdaten, die während der gesamten Simulation vorhanden sind und für jeden Kaufvorgang gleichwertig zur Verfügung stehen. Beispiele für Stammdaten sind Zulieferer, Kunden oder Produkte. Jedes dieser Objekte erhält einen der Qualitätswerte *gut*, *normal* oder *schlecht*, welche zur Steuerung der Prozesssimulation benötigt werden. Des Weiteren gibt es transaktionale Daten, die erst während der Ausführung eines Geschäftsprozesses erstellt werden. Hierzu gehören unter anderem das Verkaufsangebot, die Rechnung oder ein mögliches Ticket. Beide Arten von Daten können als Knoten und deren Beziehungen als Kanten eines Graphen modelliert werden. Ein vollständiges Schema aller Datenobjekte und deren Beziehungen ist in [Pe14] dargestellt.

Der Geschäftsprozess ist in zwei wesentliche Teilprozesse unterteilt: die Vermittlung (*Brokerage*) als Hauptvorgang und die optional nachgelagerte Problembehandlung (*Complaint Handling*). Der zeitliche Ablauf der Generierung transaktionaler Daten während des Brokerage-Prozesses wird in Abbildung 1 dargestellt. Des Weiteren zeigt die Abbildung die Abhängigkeiten der einzelnen Objekte voneinander. Betrachtet man die Ausgangsrechnung, so wird deutlich, dass diese vom Auftrag abhängt. Zeitlich gesehen wird sie jedoch erst zum Schluss erstellt.

3 Verteilte Ausführung

Nachfolgend wird zunächst eine kurze Einführung zu den BigData-Frameworks Apache Flink [Ca15] und GRADOOP [Ju15, Ju16] gegeben. Diese werden verwendet um die Ausführung FoodBrokers auf mehrere Maschinen zu verteilen.

Apache Flink ist ein Open-Source-Framework, welches für Big Data-Analysen eine Laufzeitumgebung zur Verfügung stellt. Diese ermöglicht es, analytische Programme durch Transformationen verteilter Objektmengen (sog. *DataSets*) als Datenfluss zu beschreiben. Die Ausführung eines solchen Programms wird von Flink koordiniert und automatisch auf die Maschinen eines Rechnerclusters verteilt. Mögliche Datenquellen und -senken für Flink-Programme sind bspw. das verteilte Dateisystem HDFS [Sh10] oder relationale bzw. NoSQL Datenbanken wie Apache HBase.

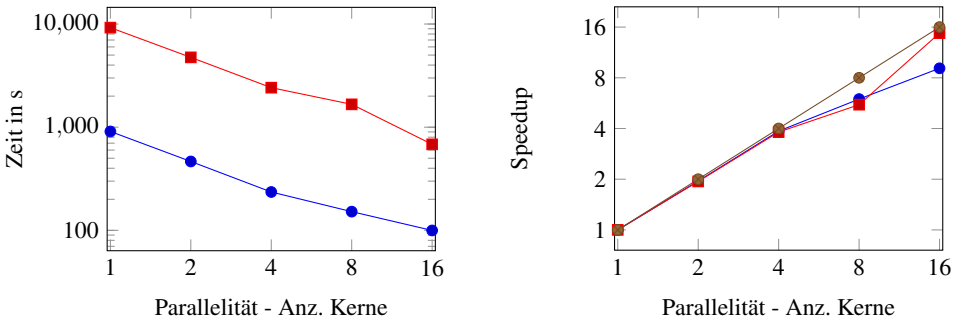
GRADOOP ist ein Open-Source-Framework, welches an der Universität Leipzig entwickelt und für die verteilte Analyse von Graphen eingesetzt wird. GRADOOP implementiert das schema-freie ***E**xtended **P**roperty **G**raph **M**odel* (EPGM) [Ju16] und ermöglicht hierdurch die Darstellung mehrerer, möglicherweise überlappender Graphen innerhalb einer Datenbank. So können die während einer Geschäftsprozess-Simulation erzeugten Daten zusammengefasst und als Graph, nachfolgend als *Transaktion* bezeichnet, gespeichert werden. Hierbei können Überlappungen enstehen, wenn ein Stammdatum in mehreren Transaktionen referenziert wird. Neben der Repräsentation bietet das EPGM analytische Graph-Operatoren, welche auf einzelnen Graphen und Graphmengen ausgeführt werden können [Ju16].

Um mittels FoodBroker eine verteilte Datengenerierung zu ermöglichen, müssen die zeitlichen Abhängigkeiten aller Objekte berücksichtigt werden. Hierdurch ergibt sich eine Aufteilung in drei Phasen. Zunächst müssen die Stammdaten erstellt und in DataSets gespeichert werden. Diese stehen in der nachfolgenden Brokerage-Phase zur Verfügung in welcher der Großteil der transaktionalen Daten erzeugt wird. Abhängig von den Ergebnissen der Transaktionen wird die dritte Phase, das Complaint Handling, aufgerufen. Anschließend müssen alle relevanten Daten aus den einzelnen Transaktionen mit den Stammdaten zusammengeführt werden.

Der **Vermittlungsprozess** (s. Abb. 1) bietet wenig Potenzial zur Parallelisierung. So könnte bspw. die Erstellung der Eingangs- und Ausgangsrechnung gleichzeitig stattfinden, da beide auf der gleichen zeitlichen Ebene liegen. In der bisherigen Implementierung[3] wird der Prozess linearisiert und mehrere Simulationen parallel (auf Prozessorebene) ausgeführt. Dieser Ansatz hat sich auch als beste Lösung für die Verteilung auf mehrere Rechenknoten herausgestellt. Hierzu wird die Flink-Transformation *MapPartition* verwendet.

Zu Beginn wird ein N-elementiges DataSet erzeugt, wobei jedes Element Startpunkt einer Distributed FoodBroker-Simulation ist. Durch den Aufruf der *MapPartition*-Transformation lässt sich die Ausführung des Vermittlungsprozesses gleichmäßig auf R Recheneinheiten verteilen. Jede Recheneinheit bearbeitet eine Partition des initialen DataSet und führt nun $N \div R$ Simulationen unabhängig aus. In jeder Simulation muss auf die

[3] `https://github.com/dbs-leipzig/foodbroker`, commit: 548e51d



(a) Ausführungszeiten bei einem Skalierungsfaktor von *100* (blau) und *1000* (rot).

(b) Speedup beider Skalierungsfaktoren im Vergleich zu linearem Speedup (braun).

Abb. 2: Ergebnisse

Stammdaten zugegriffen werden, was erfordert, dass diese an alle Rechner gesendet werden müssten. Um die zu verteilende Datenmenge zu minimieren, werden diese vorher mittels *Map*-Transformation auf ihre Id und den zugehörigen Qualitätswert reduziert und als Parameter an die *MapPartition*-Transformation übergeben. Innerhalb dieser Operation ist der Ablauf analog zur bisherigen Implementierung.

4 Auswertung

Im Folgenden wird die Performanz der verteilten Datengenerierung betrachtet. Hierbei wird sowohl die Skalierbarkeit hinsichtlich wachsender Datenmengen als auch unter Hinzunahme von Ressourcen evaluiert. Um die produzierten Datenmengen zu erhöhen wird in FoodBroker ein Skalierungsfaktor verwendet. Die Anzahl der simulierten Geschäftsprozesse hängt linear und die Anzahl der Stammdaten linear mit Dämpfungsfaktor von diesem Skalierungsfaktor ab. Die Parallelität wird abhängig von der Anzahl verwendeter Prozessorkerne angegeben. Für die Messung wurden jeweils 10 unabhängige Distributed FoodBroker Abläufe mit der Standardkonfiguration gestartet und die durchschnittliche Ausführungszeit ermittelt.

Das **Testsystem** ist ein Cluster von fünf PC-Systemen, jeder Rechner besteht aus einem Intel XEON W3520 (4 x 2.6 Ghz) und besitzt 6 GB RAM. Das Betriebssystem ist Ubuntu Server 14.04 (64-Bit) und es ist eine 500 GB Samsung Seagate HDD für die Datenspeicherung verbaut. Für die Ausführung verwenden wir Apache Flink 1.1.2 (1 Jobmanager, 4 Taskmanager), Hadoop 2.5.2 und GRADOOP Version 0.3.0-SNAPSHOT. Die Implementierung ist online verfügbar[4].

Die **Ergebnisse** sind in Abbildung 2 dargestellt, wobei Abbildung 2(a) die Ausführungszeit unter verschiedenen Parallelitätsgraden und Skalierungsfaktoren zeigt. Die Tests starten bei sequenzieller Ausführung, was einem Grad von *1* entspricht. Für die Generierung

[4] `https://github.com/dbs-leipzig/gradoop`
package : org.gradoop.flink.datagen.transactions.foodbroker

bei Skalierungsfaktor *100* werden ca. 909 Sekunden, bei Skalierungsfaktor *1000* ca. 9226 Sekunden benötigt. Wird die Parallelität auf *2* erhöht, so dauert die Ausführung im ersten Fall im Durchschnitt ca. 465 Sekunden und ist damit nur bedingt langsamer, als für eine lineare Skalierbarkeit zu erwarten ist. Bei einem Skalierungsfaktor von *1000* ist es ähnlich, hier werden im Schnitt ca. 4753 Sekunden im Vergleich zu 4613 Sekunden benötigt. Wird die Parallelität auf *4* erhöht, so erreichen beide Skalierungsvarianten eine um fast 50% kürzere Ausführungszeit im Vergleich zur vorherigen Messung. Ab einer Parallelität von *8* verringern sich die Ausführungszeiten im ersten Fall jedoch nicht mehr so stark. Bei einem Skalierungsfaktor von *1000* werden jedoch bis zu einem Parallelitätsfaktor von einschließlich *16* signifikante Erhöhungen in der Ausführungsgeschwindigkeit erreicht. Dies ist darin begründet, dass der Einfluss des konstanten Anteils in der Berechnung der Stammdatenanzahl abnimmt.

Betrachtet man den in Abbildung 2(b) dargestellten Speedup beider Ausführungsvarianten (blau, rot) im Vergleich zur linearen Steigerung (braun), so ist zu erkennen, dass sich die resultierenden Kurven bis zu einer Parallelität von *4* überlappen. Bei den anschließenden Parallelitätsgraden nimmt der Speedup bei einem Skalierungsfaktor von *100* immmer weiter ab. Neben dem genannten Verhältnis zu den Stammdaten kann dies an der geringen Anzahl erzeugter Daten je Kern liegen. Denn bei einem Skalierungsfaktor von *1000*, bei dem deutlich mehr Daten erzeugt werden, wird bei einem Parallelitätsgrad von *16* ein Speedup von ca. 13,5 erreicht. Es gibt also für jeden Skalierungsfaktor eine bestimmte Anzahl an Kernen, die verwendet werden sollte, um eine möglichst effiziente Generierung auszuführen.

5 Verwandte Arbeiten

Bisher existieren nur wenige Datengeneratoren für graphbasierte oder prozessorientierte Daten. Netzwerkgeneratoren im Allgemeinen sind bspw. im Bereich des Semantic Web vorhanden. Hier gibt es unter anderem den *Berlin SPARQL Benchmark (BSBM)* [Bi09] für die Simulation eines Internethandels, bei dem Händler Produkte anbieten und Kunden diese bewerten können. Des Weiteren gibt es beispielsweise *SP²Bench* [Sc08], welcher Zitiernetzwerke generiert, sowie *LUBM* [Yu05] für die Generierung einer Universitätsorganisation. Hierbei simuliert lediglich *BSBM* einen vollständigen Geschäftsprozess. Eine weitere wichtige Domäne für graphbasierte Generatoren sind soziale Netzwerke. Hier gibt es u.a. den *Social Network Benchmark (SNB)* der LDBC-Organisation [Er15] sowie *Graph500*[5]. Diese befassen sich jedoch nicht mit einer graphbasierten Geschäftsprozess-Modellierung, zur Datengenerierung für Analysetools, wie es bei FoodBroker der Fall ist. Durch die Einbindung von Distributed FoodBroker in GRADOOP ist es zudem möglich, Graphen verteilt zu generieren und diese anschließend direkt für die Evaluierung von Graph-Operatoren zu verwenden.

[5] `http://www.graph500.org`

6 Zusammenfassung

In diesem Beitrag wurde zunächst ein Überblick zur aktuellen Implementierung des Datengenerators FoodBroker gegeben. Der Fokus lag hierbei auf der Simulation des Brokerage-Prozesses. Da der Datengenerator die Ausführung nur auf einer Maschine unterstützt, ist die maximal mögliche Graphgröße limitert. Der Schwerpunkt dieses Papiers ist die Verteilung der Datengenerierung mittels FoodBroker unter Verwendung von Apache Flink und GRADOOP.

Durch unsere Experimente konnte gezeigt werden, dass sich Distributed FoodBroker annähernd linear skaliert. Bei erhöhter Parallelität mit geringem Skalierungsfaktor bricht der Speedup jedoch ein. Grund hierfür ist das Verhältnis zwischen einer hohen Anzahl an Stammdaten und wenig ausgeführten Simulationen. Das Verhältnis nimmt jedoch bei steigendem Skalierungsfaktor ab. Daher muss für jeden Skalierungsfaktor der Parallelisierungsgrad manuell festgelegt werden. Distributed FoodBroker ist somit auch für große Datenmengen und erhöhter Parallelisierung einsetzbar.

Literaturverzeichnis

[Bi09] Bizer, Christian; Schultz, Andreas: The Berlin SPARQL Benchmark. International Journal on Semantic Web & Information Systems, Vol. 5, Issue 2, S. 1–24, 2009.

[Ca15] Carbone, Paris et al.: Apache Flink™: Stream and Batch Processing in a Single Engine. IEEE Data Eng. Bull., 38(4), 2015.

[Er15] Erling, Orri et al.: The LDBC Social Network Benchmark: Interactive Workload. In: Proc. ACM SIGMOD. SIGMOD '15, ACM, New York, NY, USA, S. 619–630, 2015.

[Ju15] Junghanns, Martin et al.: Gradoop: Scalable Graph Data Management and Analytics with Hadoop. Bericht, University of Leipzig, 2015.

[Ju16] Junghanns, Martin et al.: Analyzing Extended Property Graphs with Apache Flink. Proc. SIGMOD, 2016.

[Pe14] Petermann, André et al.: FoodBroker - Generating Synthetic Datasets for Graph-Based Business Analytics. 5th Works. on Big Data Benchmarking (WBDB), LNCS 8991, 2014.

[Sc08] Schmidt, Michael et al.: SP2Bench: A SPARQL Performance Benchmark. CoRR, abs/0806.4627, 2008.

[Sh10] Shvachko, Konstantin et al.: The Hadoop Distributed File System. In: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). S. 1–10, May 2010.

[Yu05] Yuanbo, Guo et al.: LUBM: A Benchmark for OWL Knowledge Base Systems. Web Semantics: Science, Services and Agents on the World Wide Web, 3(2-3), 2005.