B. Mitschang et al. (Hrsg.): BTW 2017 – Workshopband,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 111

Effizienz-Optimierung daten-intensiver Data Mashups am Beispiel von Map-Reduce

Pascal Hirmer[1]

Abstract: Data Mashup-Ansätze und -Tools bieten einen einfachen und schnellen Weg, um Daten zu verarbeiten und zu analysieren. Über eine grafische Oberfläche können dabei – in der Regel grafisch – Datenquellen und Datenoperationen sowie der Datenfluss einfach modelliert werden. Hierdurch ergeben sich vor allem Vorteile durch einfache Bedienbarkeit durch Domänennutzer sowie einer explorativen Vorgehensweise. Jedoch legen vorhandene Data Mashup-Ansätze und -Tools wenig Wert auf die Effizienz der Ausführung, was dadurch begründet wird, dass durch Data Mashups in der Regel kleine Datenmengen verarbeitet werden. Zu Zeiten von Big Data gilt dies jedoch nicht mehr; schon scheinbar kleine Szenarien enthalten oftmals eine Vielzahl an Daten. Um mit diesem Problem zukünftig umzugehen, stellen wir in diesem Paper eine Konzeptidee am Beispiel von Map-Reduce vor, mit der die Ausführung von Data Mashups bzgl. Effizienz optimiert werden kann.

Keywords: Data Mashups, Map-Reduce, Big Data, Effizienzoptimierung

1 Einführung

In der heutigen Zeit steigen die Datenmengen immer weiter an, wodurch viele Vorteile durch die Generierung von Wissen aus diesen Daten gewonnen werden können [Mc12]. Zum Beispiel führen große Konzerne wie Facebook und Google komplexe Analysen auf einer riesigen Datenmenge durch, um den Nutzern personalisierte Werbung vorzuschlagen. Die Hauptherausforderungen sind hierbei die sich ständig ändernden Daten, deren Verteiltheit sowie deren Heterogenität. Jedoch kann wichtiges Wissen nicht nur von großen Konzernen sondern auch im Kleinen gewonnen werden. Insbesondere kleine Firmen sowie Privatnutzer sehnen sich nach einer einfachen Möglichkeit Daten zu verarbeiten sowie Analysen auszuführen. Um dies zu ermöglichen, wurden in der Vergangenheit viele Data Mashup Tools geschaffen, oftmals auch als ETL-Tools oder SPSS-Systeme bezeichnet [DM14]. Diese bieten in der Regel eine grafische Oberfläche basierend auf dem Pipes and Filters Pattern [Me95], um einen Datenflussgraphen bestehend aus Datenquellen sowie Datenoperationen (Filter, Aggregation, Analysen) zu modellieren. Dies ermöglicht nicht nur die Nutzung durch Domänenexperten, wie z.B. Wirtschaftsanalysten, sondern auch eine explorative Vorgehensweise bei der Ergebnisfindung. Das Datenflussmodell kann dabei leicht angepasst und erneut ausgeführt werden. Dies wird so lange wiederholt, bis das gewünschte Ergebnis eingetreten ist. Als Konsequenz kann dieses Ergebnis anschließend robust, z.B. als ETL-Prozess mit angeschlossenem Data Warehouse, umgesetzt werden.

Bestehende Data Mashup-Lösungen fokussieren sich nicht auf eine effiziente Ausführung sondern legen Wert auf die grafische Oberfläche und deren Bedienbarkeit. Des Weiteren

[1] Universität Stuttgart, Institut für Parallele und Verteilte Systeme, Universitätsstraße 38, 70569 Stuttgart, Pascal.Hirmer@ipvs.uni-stuttgart.de

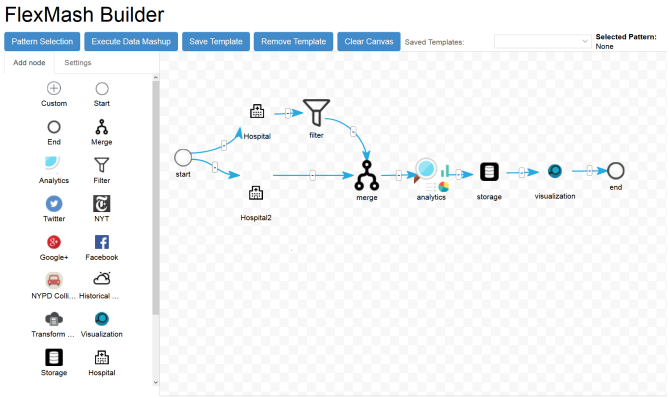

Abb. 1: Grafische Oberfläche des Data Mashup Tools FlexMash

ist die Datenmenge aufgrund von proprietären, nicht-skalierbaren Implementierungen oftmals stark begrenzt. Eine Verknüpfung der einfachen, benutzerfreundlichen Modellierung mit einer skalierbaren Verarbeitung sehr großer Datenmengen kann die Anwendbarkeit dieser Ansätze stark erhöhen. So können beispielsweise große Datenbanken angeschlossen werden. In dieser Arbeit stellen wir eine Konzeptidee am Beispiel von Map-Reduce vor, die den an der Universität Stuttgart entwickelten Data Mashup-Ansatz *FlexMash* sowie dessen prototypische Implementierung um eine optimierte, skalierbare Ausführung von Datenoperationen und -analysen erweitert. Dabei sollen für auszuführende Datenoperationen Map-Reduce Jobs erstellt werden, die anschließend verteilt auf einem großen Rechencluster ausgeführt werden können. Hierdurch kann die Effizienz der Ausführung stark optimiert werden. Eine ähnliche Vorgehensweise wurde bereits in verwandten Tools und Ansätzen verfolgt, z.B. durch eine Transformation von Datenfluss-Sprachen wie Pig oder JaQL auf Map-Reduce. Für die Domäne Data Mashups und deren grafische Modellierung von Datenflüssen wurde dies jedoch noch nicht umgesetzt.

Dieses Paper ist wie folgt aufgebaut: In Abschnitt 2 wird kurz der Data Mashup-Ansatz FlexMash beschrieben, der als Grundlagen für dieses Paper dient. Anschließend wird der Hauptbeitrag in Abschnitt 3 vorgestellt. Abschnitt 4 beschreibt verwandte Arbeiten und Abschnitt 5 fasst den Beitrag dieses Papers zusammen und gibt einen Ausblick auf zukünftige Arbeiten.

2 FlexMash

FlexMash [HM16] ist ein Data Mashup-Ansatz und eine Toolimplementierung entwickelt an der Universität Stuttgart, welches eine ad-hoc Modellierung von Data Mashups sowie eine explorative Vorgehensweise der Ausführung ermöglicht. Die Modellierung der Datenflussgraphen basiert auf dem Pipes and Filters-Pattern. Im Vergleich zu anderen Arbeiten bietet FlexMash einige Vorteile, darunter die Abstraktion von technischen Details bei der Modellierung durch sogenannte *Modellierungs-Patterns* [Hi15; HM16]. Diese ermöglichen

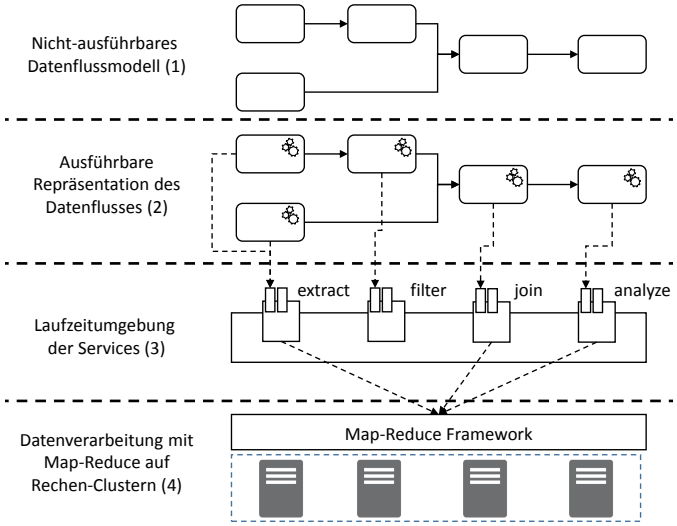

Abb. 2: Gesamtansatz zur Ausführung mit Map-Reduce

eine weitestmögliche Abstraktion, sodass Domänenexperten nur mit Konzepten arbeiten müssen, die sie auch kennen. Beispielsweise können Wirtschaftsanalysten Systeme und Programme modellieren (z.B. SAP-Systeme), mit denen sie täglich arbeiten und mit deren Daten sie vertraut sind. Ein weiteres Alleinstellungsmerkmal von FlexMash ist die flexible Ausführung der Datenverarbeitung basierend auf nicht-funktionalen Nutzeranforderungen. Nach der Modellierung kann der Anwender aus einem Katalog seine spezifischen Anforderungen an die Ausführung auswählen, beispielsweise das diese besonders robust gegen Datenverluste oder besonders sicher (verschlüsselt) durchgeführt werden soll. Die Art der Ausführung hängt dann von diesen Anforderungen ab, d.h., die Technologien und Softwarekomponenten werden dynamisch, baukasten-artig zusammengesetzt, um eine personalisierte Ausführungsumgebung zu schaffen [HM16]. Die grafische Oberfläche des Tools zur Modellierung der Datenflussgraphen ist in Abb. 1 zu sehen. In dem dargestellten Beispiel sollen Daten von zwei Krankenhäusern zusammengeführt und analysiert werden. Da eines der Krankenhäuser für die Analyse unwichtige Daten erfasst, werden diese vor der Zusammenführung gefiltert. Eine mögliche Analyse wäre bspw. zu untersuchen, welche Nebenwirkungen durch die Einnahme eines bestimmten Medikaments entstehen.

3 Effizienz-Optimierung daten-intensiver Data Mashups

In diesem Abschnitt wird der Hauptbeitrag dieses Papers beschrieben: eine Konzeptidee für die optimierte Ausführung von Data Mashups am Beispiel von Map-Reduce. Im bisherigen FlexMash-Konzept wird das nicht ausführbare grafische Datenflussmodell zur Beschreibung der Datenverarbeitung und -analyse in eine ausführbare Repräsentation überführt (bspw. in Workflowsprachen wie BPEL). Diese ausführbare Repräsentation ruft dann für jeden Datenoperator im Modell einen dazugehörigen Service auf, der die Daten verarbeitet. Dies

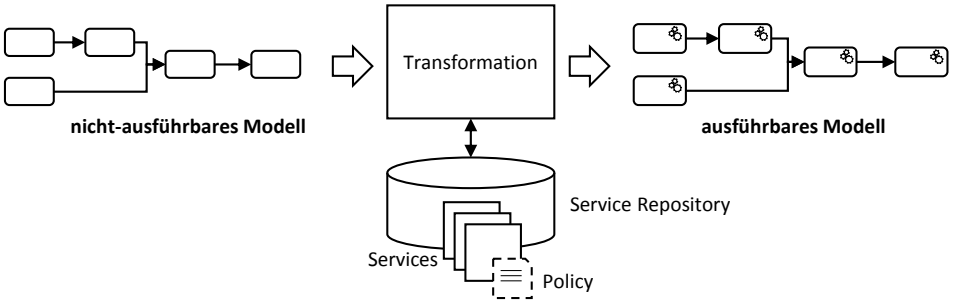

Abb. 3: Policy-basierte Auswahl der Services

skaliert jedoch nicht, da die Services die Daten immer als Ganzes verarbeiten. Um die erforderliche Skalierbarkeit zu erreichen wird daher eine neue Ebene der Datenverarbeitung eingeführt, die von den Services abstrahiert wird. Der Vorteil hiervon ist es, dass das bestehende Konzept nicht angepasst werden muss.

Dies ist in Abb. 2 dargestellt. Dabei ist zu sehen, dass mehrere Abstraktionsebenen existieren: auf oberster Ebene (1) liegt der nicht-ausführbare Datenflussgraph, der durch Transformation auf eine ausführbare Repräsentation abgebildet wird, die eine Ebene darunter liegt (2). Diese Repräsentation kann beispielsweise auf etablierten Workflow-Sprachen wie BPEL basieren. Bei der Ausführung dieser werden für jeden Datenoperator im Modell entsprechende Services aufgerufen (3), die die Daten verarbeiten. In bisherigen Ansätzen wird die Datenverarbeitung von jedem Service direkt selbst ausgeführt (z.B. in Java), wodurch die Skalierbarkeit jedoch stark eingeschränkt wird. In diesem Paper führen wir eine zusätzliche Ausführungsschicht auf unterster Ebene ein (4), die die Datenverarbeitung verteilt mittels Map-Reduce ausführen kann. Wie in Abb. 2 zu sehen ist, sollte jedoch nicht pauschal jede Datenoperation mit Map-Reduce ausgeführt werden. Für Datenoperationen mit geringer Laufzeitkomplexität (Filterung, Sortierung, etc.) macht es Sinn die Datenverarbeitung direkt in dem Service auszuführen, um den durch das Map-Reduce Framework entstehenden Overhead zu vermeiden. Jedoch muss dies dynamisch entscheidbar sein, da sich der Overhead ab einer bestimmten Datenmenge wiederum lohnen kann. Wichtig ist, dass die Daten für die Verarbeitung mittels Map-Reduce oftmals in einem verteilten Dateisystem abgelegt werden müssen (z.B. HDFS in Hadoop). Dies erfordert jedoch, abhängig von der verwendeten Verarbeitungsplattform, eine Verschiebung von Daten in das entsprechende Dateisystem. Der hierdurch resultierende Overhead und wie hiermit umgegangen werden kann wird in zukünftigen Arbeiten betrachtet.

Um zu entscheiden welche Services ausgeführt werden annotieren wir diese mit Policies (z.B. Servicebeschreibungen wie WS-Policys), die definieren für welche Art von Daten und für welche Datenmenge sie geeignet sind. Bei geringen Datenmengen sowie einer geringen Komplexität kommen Services in Frage die Daten ohne die Verwendung von Map-Reduce verarbeiten, im umgekehrten Fall sollten diese Techniken wiederum angewendet werden. Die Entscheidung welche Services letztendlich für die Datenverarbeitung genutzt werden wird bei der Transformation des nicht-ausführbaren Modells in die ausführbare Repräsentation getroffen, dies ist in Abb. 3 dargestellt. Dabei werden die Art der Daten sowie

die Datenmenge mit den Policies geeigneter Services abgeglichen. Erfüllt eine Service-Policy alle Anforderungen, die durch die Daten gegeben sind wird ein entsprechender Service-Aufruf in die ausführbare Repräsentation eingefügt. Erfüllen mehrere Services eine Policy sollte die Auswahl entweder zufällig geschehen oder es müssen weitere Faktoren wie z.B. Kosten, Laufzeitkomplexität, benötigte Rechenressourcen (z.B. Größe der Cluster) etc. der Implementierung der Services hinzu genommen werden. Dies erfordert komplexe Kostenmodelle, die in Zukunft entwickelt werden. Erfüllt keiner der Services die Policy in vollem Umfang wird der bestmögliche Service ausgewählt. Wir fokussieren uns in diesem Paper nicht auf die genauen Charakteristiken der Map-Reduce Jobs, da die Datenverarbeitung mittels Map-Reduce bereits durch eine Vielzahl von Arbeiten abgedeckt wurde. Wir gehen davon aus, dass die Datenoperationen sowohl mittels Map-Reduce, als auch ohne umgesetzt werden können. Eine erste Untersuchung hat ergeben, dass dies auf die gängigen Datenoperationen (z.B. Filter) zutrifft.

4 Verwandte Arbeiten

Hadoop[3] ist eine bekannte Plattform für die Ausführung von Map-Reduce, welches auf dem Hadoop File System (HDFS), einem verteilten Dateisystem, arbeitet. Basierend auf dem HDFS ermöglicht Hadoop eine auf mehreren Rechenclustern verteilte Verarbeitung großer Datenmengen mittels Map-Reduce. In den letzten Jahren wurde Hadoop zum de-facto Standard für die Ausführung von Map-Reduce. Für eine erste Implementierung des in diesem Paper vorgestellten Ansatzes wurde Hadoop verwendet. Seit einiger Zeit sind jedoch moderne, effizientere Ansätze auf dem Vormarsch. Zu diesen gehören Apache Spark und Apache Flink[4]. Apache Spark ermöglicht eine deutliche Effizienzsteigerung im Vergleich zu Hadoop [GA15]. Des Weiteren bietet Apache Spark eine Vielzahl weiterer Features, die vor allem auch Grundfunktionen der Datenverarbeitung (Filterung, Aggregation) beinhalten. Außerdem ermöglicht Spark eine Unterstützung von strombasierter Datenverarbeitung, was eine höhere Abdeckung verschiedenartiger Szenarien ermöglicht. Eine weitere Plattform für effiziente Datenverarbeitung ist Apache Flink. Apache Flink ist auf strombasierte Datenverarbeitung fokussiert und bietet eine ähnliche Funktionalität wie Spark. Flink hat somit wie Spark einen Effizienzvorteil gegenüber Hadoop. Ein umfassender Vergleich von Spark und Flink ist in [Ma16] beschrieben.

Zusammengefasst ergibt diese Diskussion, dass es sinnvoll ist, abhängig vom Anwendungsfall, verschiedene Frameworks und Datenverarbeitungstechniken, z.B. strombasiert, zu verwenden. Durch den in diesem Paper beschriebenen Ansatz ist dies einfach möglich. Die Abstraktion der eigentlichen Datenverarbeitung durch Services erlaubt es, verschiedene Service-Implementierungen für dieselben Datenoperationen anzubieten. Diese können dann beispielsweise mit Hadoop, Spark, Flink, oder anderen Plattformen ausgeführt werden. Wichtig ist, dass die Policys für jede zusätzlich unterstützte Plattform erweitert werden müssen. Dies kommt daher, dass diese Plattformen zusätzliche Funktionalitäten bieten könnten, die in den Policys repräsentiert sein sollten.

[3] http://hadoop.apache.org/
[4] https://flink.apache.org/

5 Zusammenfassung und Ausblick

In diesem Paper wird eine Konzeptidee für eine effizienz-optimierte Ausführung von Data Mashups am Beispiel von Map-Reduce beschrieben. Dabei steht insbesondere die Transformation des nicht-ausführbaren Modells in die ausführbare Repräsentation im Vordergrund. Durch das Anheften von Policies an die Services, die definieren für welche Art Daten und für welche Datenmenge sie am besten geeignet sind, können wir die Transformation der nicht-ausführbaren Datenflussmodelle auf eine ausführbare Transformation dahingehen anpassen, dass immer die zeiteffizienteste Ausführung ausgewählt wird. Die Ausführung selbst kann dabei beispielsweise verteilt mittels Map-Reduce durchgeführt werden, wodurch die Effizienz vor allem bei großen Datenmengen optimiert werden kann.

In Zukunft werden wir die vorgestellte Konzeptidee konkretisieren sowie eine zugehörige Implementierung anfertigen. Dabei muss auch das Format der Policy genauer definiert werden sowie mehrere in Frage kommende Policy-Frameworks untersucht werden.

Literatur

[DM14] Daniel, F.; Matera, M.: Mashups - Concepts, Models and Architectures. Springer, 2014.

[GA15] Gopalani, S.; Arora, R.: Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means. International Journal of Computer Applications 113/1, S. 8–11, März 2015.

[Hi15] Hirmer, P.; Reimann, P.; Wieland, M.; Mitschang, B.: Extended Techniques for Flexible Modeling and Execution of Data Mashups. In: Proceedings of the 4th International Conference on Data Management Technologies and Applications (DATA). 2015.

[HM16] Hirmer, P.; Mitschang, B.: Rapid Mashup Development Tools: First International Rapid Mashup Challenge, Revised Selected Papers. In. Springer International Publishing, Kap. FlexMash – Flexible Data Mashups Based on Pattern-Based Model Transformation, 2016.

[Ma16] Marcu, O. C.; Costan, A.; Antoniu, G.; Pérez-Hernández, M. S.: Spark Versus Flink: Understanding Performance in Big Data Analytics Frameworks. In: 2016 IEEE International Conference on Cluster Computing (CLUSTER). S. 433–442, 2016.

[Mc12] McAfee, A.; Brynjolfsson, E.; Davenport, T. H.; Patil, D.; Barton, D.: Big data. The management revolution. Harvard Bus Rev/, 2012.

[Me95] Meunier, R.: The pipes and filters architecture. In: Pattern languages of program design. 1995.