B. Mitschang et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2017),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 609

Emma in Action: Deklarative Datenflüsse für Skalierbare Datenanalyse

Eingeladene Demonstration

Alexander Alexandrov[1], Georgi Krastev[1], Bernd Louis[1],
Andreas Salzmann[1], Volker Markl[1]

Abstract: Schnittstellen zur Programmierung paralleler Datenflüsse, die auf Funktionen höherer Ordnung (wie *map* und *reduce*) basieren, sind in den letzten zehn Jahren durch Systeme wie Apache Hadoop, Apache Flink und Apache Spark populär geworden. Im Gegensatz zu SQL werden solche Programmierschnittstellen in Form eingebetteter Domänenspezifischer Sprachen (eDSLs) realisiert. Im Kern jeder eDSL steht ein dedizierter Typ, der verteilte Datenmengen repräsentiert und Berechnungen auf ihnen ermöglicht, wie z.B. DataSet in Flink oder RDD in Spark. Aufgrund der Integration von eDSLs in einer generischen Programmierumgebung (Java, Scala, oder Python) stellen sie eine flexiblere Alternative zu klassischen Ansätzen (z.B. SQL) dar, um gängige Aufgaben (z.B. ETL-Prozesse) in einer skalierbaren, Cloud-basierten Infrastruktur zu implementieren.

Im Laufe der Zeit hat sich jedoch die Anzahl der Operatoren auf den Schnittstellen vergrößert, um Effizienz und Skalierbarkeit zu gewährleisten. Die Schnittstellen werden somit selbst immer komplexer – Ausführungsaspekte wie Join-Reihenfolge, Partitionierung, Caching von Zwischenergebnissen und Verwendung partieller Aggregate, werden dabei nicht vom System, sondern vom Programmierer entschieden. Diese Tendenz hat zwei Nachteile – (1) Programmierer müssen das zugrunde liegende Ausführungsmodell gut beherrschen, um die jeweils passenden Operatoren auswählen zu können, und (2) der erzeugte Quelltext beinhaltet neben den logischen auch physische Aspekte der beschriebenen Berechnung. Dies erhöht die Komplexität von Datenanalyseprogrammen und verringert deren Wartbarkeit und Portabilität.

In dieser Demonstration zeigen wir, wie sich die obengennanten Probleme durch einen besseren Einbettungsansatz vermeiden lassen. Die Grundidee hierbei ist, die Einbettung nicht durch Typen, sondern durch sog. Quasi-Quotes zu realisieren. Diese ermöglichen es, Quelltext-Fragmente in der Host-Sprache auszuzeichnen. Sprachkonstrukte, die durch Typen alleine nicht überladen werden können (wie z.B. Kontrollfluss, anonyme Funktionen, `for`-comprehensions), werden in der eDSL somit reflektiert und in eine strukturell reichere Zwischendarstellung übertragen. Auf dieser Darstellung lassen sich dementsprechend Programmtransformationen definieren, welche die erforderlichen physischen Operatoren und Optimierungen automatisch einführen.

Wir zeigen die oben beschriebenen Ideen in Emma, einer in Scala eingebetteten und auf Quasi-Quotes basierenden DSL. Zur Modellierung verteilter Datenmengen verwenden wir eine algebraische Formalisierung, welche Daten als Multimengen in der sog. UNION Sicht realisiert – d.h. als Instanzen des polymorphen algebraischen Datentypen Bag *A*. Parallele Berechnungen lassen sich dabei mit der Klasse von Programmen, die sich über strukturelle Rekursion definieren lassen, identifizieren.

Anhand von Beispielalgorithmen wie KMeans und NaiveBayes zeigen wir, wie durch Emma komplexe Datenanalyseprogramme ohne besondere Kenntnisse von APIs, lediglich in klassischem Scala, programmiert werden und dann automatisch optimiert und auf massiv-parallelen Big Data Frameworks wie Apache Flink und Apache Spark zur Ausführung gebracht werden. Auf diese Weise leistet Emma einen Beitrag zur deklarativen Spezifikation von Datenanalsyeprogrammen, die im NoSQL-Kontext verloren gegangen war.

[1] TU Berlin, FG DIMA, Einsteinufer 17. 10587 Berlin, name.nachname@tu-berlin.de