Big Data Übungsblatt 07

Anton Bulat, Josephine Geiger, Julia Siekiera

December 3, 2017

Aufgabe 1: Monoidhomomorphismen

Handelt es sich bei der Matrix-Vektor-Multiplikation für dünn besetzte Matrizen im Folgenden um Monoidhomomorphismen?

a)

Map (einschlielich Sort/Shuffle):
Die Map-Eingabe besteht aus dem Key (i, j) als Tupel und dem Value a_{ij} als DoubleWritable. Also ist diese vom Typ $Map < Tuple, DoubleWritable >$. Die Map-Eingabe ist ein Monoid bezüglich der Konkatenation mit dem Tripple $(Map < Tuple, DoubleWritable >, \otimes, \{\})$. Der Mapper ist ein Monoid bezüglich der Wertekonkatenation $(Double, \cdot, 1)$, da die Multiplikation im Wertebereich abgeschlossen ist und 1 als neutrales Element besitzt. Die Map-Ausgabe ist ein Monoid bezüglich der Konkatenation mit dem Tripple $(Map < IntegerWritable, DoubleWritable >, \otimes, \{\})$. Sort und Shuffle nehmen keinen Einfluss auf die Monoideigenschaft des Mappers. Nach Sort und Shuffle bleibt es bei einem Monoid und zwar bezüglich der Konkatenation mit dem Tripple $(Map < IntegerWritable, List < DoubleWritable >>, \otimes, \{\})$. Bei einer Aufteilung der Mapper-Eingabe in zwei Stücke $(((i, j), a_{first})$- und $((i, j), a_{second})$-Paare, wobei $a = a_{first} + a_{second})$ verrechnet Map a_{first} und a_{second} mit den x-Werten, die im Distributed Cache gespeichert sind, einzeln und erhält $a_{ij}x_j$, da die Summe der beiden Teilmultiplikation die ganze Multiplikation ergeben. Damit handelt es sich um ein Monoid-Homomorphismus.

b)

Reduce:
Die Reduce-Eingabe besteht aus dem Key i als IntegerWritable und dem Value $l = [a_{i1}x_1, ..., a_{in}x_n]$ als $List < DoubleWritable >$. Also ist diese vom Typ $Map < IntegerWritable, List < DoubleWritable >>$. Die Reduce-Eingabe ist ein Monoid bezüglich der Konkatenation mit dem Tripple $(Map < IntegerWritable, List < DoubleWritable >>, \otimes, \{\})$. Der Reducer ist ein Monoid

bezüglich der Wertekonkatenation ($List < IntegerWritable >, \otimes, []$) sowie der Wertekonkatenation ($Double, +, 0$). Die Reduce-Ausgabe ist ein Monoid bezüglich der Konkatenation mit dem Tripple ($Map < IntegerWritable, DoubleWritable >> , \otimes, \{\}$). Bei einer Aufteilung der Reduce-Eingabe in zwei Stücke ((i, l_{first})- und (i, l_{second})-Paare, $l = l_{first} + l_{second}$) summiert Reduce l_{first} und l_{second} einzeln und erhält l, da die Summe der beiden Teilsummen die ganze Summe ergeben. Damit ist es ein Monoid-Homomorphismus.

c)

MapReduce insgesamt:
Die MapReduce-Eingabe besteht aus dem Key (i, j) als Tupel und dem Value a_{ij} als DoubleWritable. Also ist diese vom Typ $Map < Tuple, DoubleWritable >$. Die MapReduce-Eingabe ist ein Monoid bezüglich der Konkatenation mit dem Tripple ($Map < Tuple, DoubleWritable >, \otimes, \{\}$). Die MapReduce-Ausgabe besteht aus dem Key i als IntegerWritable und dem Value $l = \sum l_j$ als DoubleWritable. Also ist diese vom Typ $Map < IntegerWritable, DoubleWritable >$. Die Ausgabe ist ein Monoid bezüglich der Konkatenation mit dem Tripple ($Map < IntegerWritable, DoubleWritable >, \otimes, \{\}$). Die Ausgabe des MapReduce ist ein Monoid bezüglich der einzelnen Wertekonkatenationen (siehe Aufgabenteil a und b). MapReduce Eingabe und Ausgabe besitzen Monoidstrukturen. Bei einer Aufteilung der MapReduce-Eingabe in zwei Stücke ($((i, j), a_{first})$- und $((i, j), a_{second})$-Paare, wobei $a = a_{first} + a_{second}$) verrechnet Map und Reduce a_{first} und a_{second} einzeln und erhält schlielich Werte, dessen Summe den Verrechnungen von a entspricht. Damit handelt es sich um ein Monoid-Homomorphismus.