Spezielle Kapitel Sozialer Web – Technologien – MapReduce im Einsatz

Seminaristischer Unterricht

Prof. Dr.-Ing. Hendrik Gärtner

Gliederung

- Wiederholung MapReduce/WordCount
- Infrastruktur Hadoop versus Spark
- Einsatz von MapReduce für die Extraktion von Informationen aus großen Datenbeständen

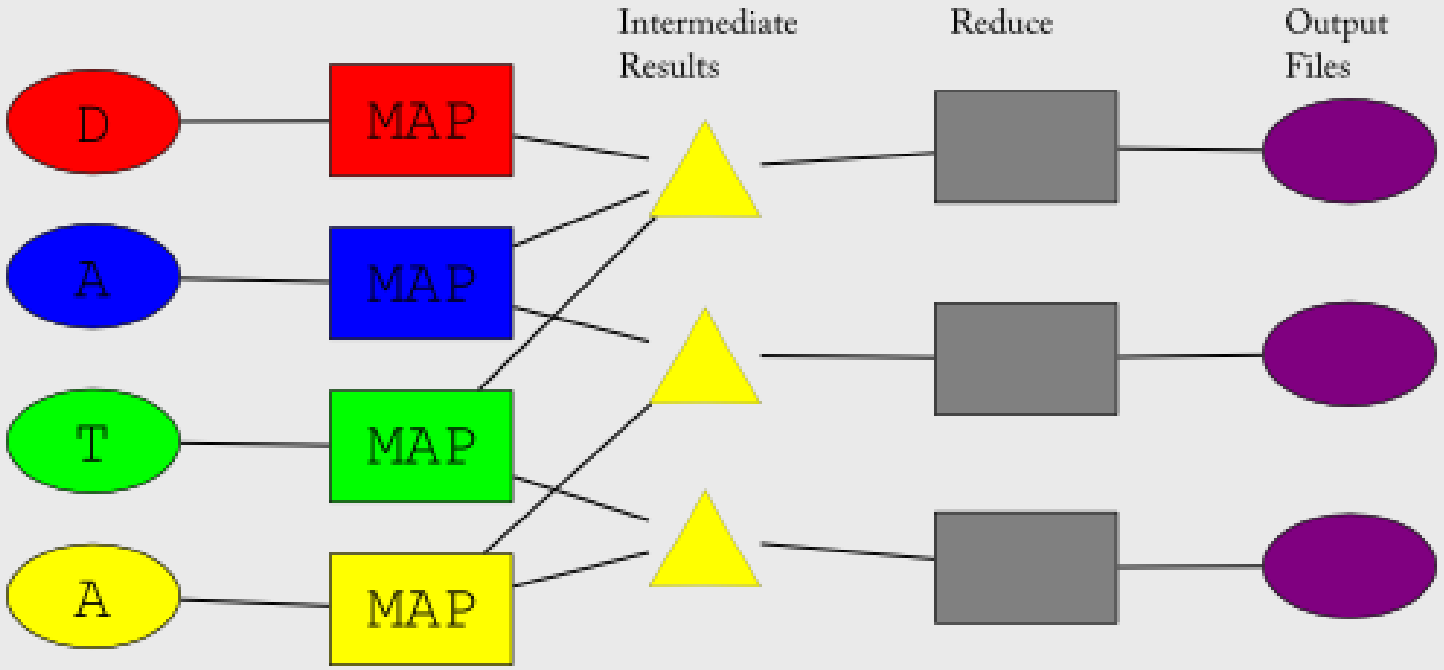
Beispiel

Wandeln Sie das WordCount-Beispiel aus den vorigen Folien so um, dass es mit dem Google-MapReduce läuft.

```
def wordCount(text:List[(Int,String)]):List[(String,Int)] = {

    mapReduce[Int, String, String, Int, String, Int](
        (X)=>{
            val t= X._2.toLowerCase.replaceAll("[^a-z]", " ")
            t.split(" ").toList.map(X=>(X,1)).filter(X=>X._1!="")
        },
        (X) => List((X._1, X._2.sum)),
        text)
}
```

Verteiltes MapReduce



- MapReduce kommt aus der Funktionalen Programmierung
- Paradigma für das Behandeln von Massendaten
- Google führte 2004 ein Framework dafür ein
- Freie Nachimplementierung Hadoop
- Viele NoSQL-DB basieren auf MapReduce in Kombination mit B-Bäumen

Hadoop

Hadoop ist ein Framework zur Verarbeitung von großen Datenmengen. Es besteht aus zwei Komponenten:

- HDFS: Ein verteiltes, replizierendes Filesystem zur Speicherung der Daten
- Hadoop: Framework zur verteilten Umsetzung von MapReduce-Algorithmen

WordCount Hadoop (1/4)

```
public class WordCount extends Configured implements Tool {

    public static void main(String[] args) throws Exception {

        int res = ToolRunner.run(new Configuration(), new WordCount(), args);
        System.exit(res);
}
```

WordCount Hadoop (2/4)

```
 @Override
public int run(String[] args) throws Exception {
    Job job= Job.getInstance(getConf(),"WordCount");
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.waitForCompletion(true);
    return 0;
}
```

WordCount Hadoop (3/4)

```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable ONE = new IntWritable(1);
    private Text word = new Text();

    @Override
    public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {

        for (String token: value.toString().split("\\s+")) {
            word.set(token);
            context.write(word, ONE);
        }
    }
}
```

WordCount Hadoop (4/4)

```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>
    {

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
           sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}}
```

WordCount Spark

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Beispiel MapReduce

Berechnen Sie auf Basis von MapReduce aus einer Liste von Zahlen den Durchschnittswert der Listenelemente.

```
def average(l:List[Int]):Double={

  val start= l map ((-1,_))
  val res::_=
     BasicOperations.mapReduce[Int,Int,Int,Int,Int,Int](
      (X)=> List((1,X._2)),
      (Y)=> List((Y._2.length, Y._2.sum)),
      start
  )
  res._2/res._1.toDouble
}
```

MapReduce im Einsatz

- MapReduce ist nicht nur Basis des Hadoop Frameworks sondern auch vieler NoSQL-Datenbanken
- NoSQL-Datenbanken verbinden Indexierungstechniken mit der verteilten Verarbeitung der Daten
- Indexierungstechniken sind Teil der nächsten Vorlesung sowie MongoDB als NoSQL-Vertreter
- Heute: Auswertung großer Datenbestände mit dem MapReduce-Paradigma

Universitätsbeispiel

```
class UniversityTest extends FunSuite{

  val data_profs= List(("P",List(2125,"Sokrates","C4",226)), ("P",List(2126,
    "Russel", "C4", 232)), ("P", List(2127, "Kopernikus", "C3", 310)),
    ("P",List(2133, "Popper", "C3", 052)),  ("P", List(2134, "Augustinus", "C3",
    309)), ("P", List(2136, "Curie", "C4", 036)), ("P",List(2137, "Kant", "C4", 007)))

  val data_studenten=List(("S",List(24002, "Xenokrates", 18)), ("S", List(25403,
    "Jonas", 12)),  ("S", List(26120, "Fichte", 10)), ("S",List(26830, "Aristoxenos",
    8)), ("S", List(27550, "Schopenhauer", 6)), ("S",List(28106, "Carnap", 3)),
    ("S", List(29120, "Theophrastos", 2)), ("S",List(29555, "Feuerbach", 2)))

  val data_vorlesungen= List(("V", List(5001, "Grundzuege", 4, 2137)),
    ("V",List(5041, "Ethik", 4, 2125)), ("V",List(5043, "Erkenntnistheorie", 3,
    2126)), ("V",List(5049, "Maeeutik", 2, 2125)),  …

val data= data_profs ++ data_studenten ++ data_vorlesungen++...
```

Idee: Adaptieren des SQL-Ansatzes

- Implementieren der Operationen der Relationalen Algebra mittels MapReduce
- Basisoperatoren:
  - Projektion
  - Selektion
  - Kartesisches Produkt
- Erweiterung von MapReduce durch ein Konfigurationsobjekt zur Übergabe von Parametern
- Weitere Operatoren
  - Union, Intersection, Difference
  - Group By, Having

SQL: SELECT Statement

SELECT Attribut1, Attribut2,… oder * (für alle) **FROM** Tabelle1, Tabelle2,…

Π - Projektion (Herausgreifen einzelner Attribute)

X- Kartesisches Produkt (Elemente verknüpfen)

WHERE Bedingung1 **AND/OR** Bedingung2 **AND/OR** …

σ - Selektion (Auswählen von einzelnen Tupeln, die den Bedingungen entsprechen)

Selektion

Projektion: $\sigma_C(R)$

Map-Function: For each tuple t in R, test if it satisfy C. If so, produce the key value pair (t,t). That is, both the key and value are t.

Reduce-Function: The Reduce Function is the identity. It simply passes each key-value pair to the output.[1]

[1]Mining Massive Datasets, page 35

Projektion

Projektion: $\pi_S(R)$

Map-Function: For each tuple t in R, construct a tuple t‘ by eliminating from t those components whose attributes are not in S. Output the key-value pair (t‘,t‘).

Reduce-Function: For each key t‘ produced by any of the Map tasks, there will be one or more key-value-pairs (t‘,t‘). The reduce function turns (t‘,[t‘,t‘,…]) into (t‘,t‘), so it produces exactly one pair (t‘,t‘) for this key t‘.[1]

Achtung: Funktion eliminiert natürlich alle Duplikate.

[1]Mining Massive Datasets, page 36

Vereinigungsmenge

Vereinigung: R ∪ S (Achtung Schemagleichheit)

Map-Function: Turn each tuple t into a key-value pair (t,t)

Reduce-Function: Associated with each key t there will be either one or two values. Produce output (t,t) in either case.[1]

[1]Mining Massive Datasets, page 36

Beispiel

Wie sind die Namen aller Professoren und Studenten?

```
(SELECT Name FROM Studenten) UNION
(SELECT Name FROM Professoren)
```

```
test("union without duplicates"){

  val d1= Basics.projection(("S",List(111,"Sokrates",3))::data_studenten, List(1))
  val d2= Basics.projection(data_profs, List(1))
  Basics.printTable(Basics.union(d1,d2))
 }
```

Differenzmenge

Differenzmenge: R - S

Map-Function: For a tuple t in R, produce key-value pair (t,R) and for a tuple t in S, produce key-value pair (t,S). Note that the intent is, that the value is the name of R or S (or better a single bit indicating whether the relation is R or S), not the entire relation.

Reduce-Function: For each key t, if the associated value list is [R], then produce (t,t). Otherwise produce nothing.[1]

[1]Mining Massive Datasets, page 37

Schnittmenge

Schnittmenge: $R \cap S$

Map-Function: Turn each tuple t into a key-value pair [t,t]

Reduce-Function: If key t has value list (t,t), then produce (t,t). Otherwise produce nothing.[1]

[1]Mining Massive Datasets, page 36

Natural Join

Natürlicher Join: R A S, wobei R aus den Attributmengen A & B besteht und S aus den Attributmengen B und S, d.h. R und S haben die gleichen Attribute B.

Map-Function: For each tuple t (a,b) in R, produce the key-value pair (b,(R,a)). For each tuple t (b,c) in S, produce the key-value pair (b,(S,c))

Reduce-Function: Each key-value b will be associated with a list of pairs that are either of the form (R,a) or (S,c). Construct all pairs consisting of one with first component R and the other with first component S, say (R, a) and (S, c). The output from this key and value is a sequence of key-value pairs. The key is irrelevant. Each value is one of the tiples (a,b,c) such that (R,a) and (S,c) are on the input list of of values.[1]

[1]Mining Massive Datasets, page 37

Anwendungsbeispiel

```
test("intersection"){
    println("-----------------------")
    println("Welche Studenten haben sich in einer VL pruefen lassen, die sie
  auch gehört haben?")
    val p= Basics.projection(data_pruefen, List(0,1))
    val i= Basics.intersection(p, ("H",List(28106, 5001))::data_hoeren)
    val j= Basics.naturalJoin(i, "intersection", List(0), data_studenten, "S",
  List(0),X=>List(X(2),X(3)))
    Basics.printTable(j)
 }
```

Gruppierung

Projektion: $\gamma_{A,\theta(B)}$, wobei A die Menge der Gruppierungsattríbute ist und die Aggregationsfunktion, die auf die Attributmenge B [b_1,b_2,…] angewendet wird.

Map-Function: For each tuple (a,b,c) produce the key-value pair (a,b).

Reduce-Function: Each key represents a group. Apply the aggregation operator θ to the list [b_1,b_2,…, b_n] of B-values associated with key a. The output is the pair (a,x) where x os the result od applying θ to the List. For example, if is the SUM , then x= b_1,b_2,…, b_n, and if θ is MAX then x is the largest of b_1,b_2,…, b_n.[1]

[1]Mining Massive Datasets, page 37/38

Vielen Dank für

Ihre Aufmerksamkeit

Prof. Dr.-Ing. Hendrik Gärtner