

Web-Informationssysteme, WS 2009/10

Seminar-Übung 4: Map/Reduce, PageRank

Besprechung am Mi 18.11.2009

— SÜ-4.1 —

Was Ihr wissen müßt über “Map/Reduce”

Wiederholung

- Welches **Problem** versucht Map/Reduce zu lösen?

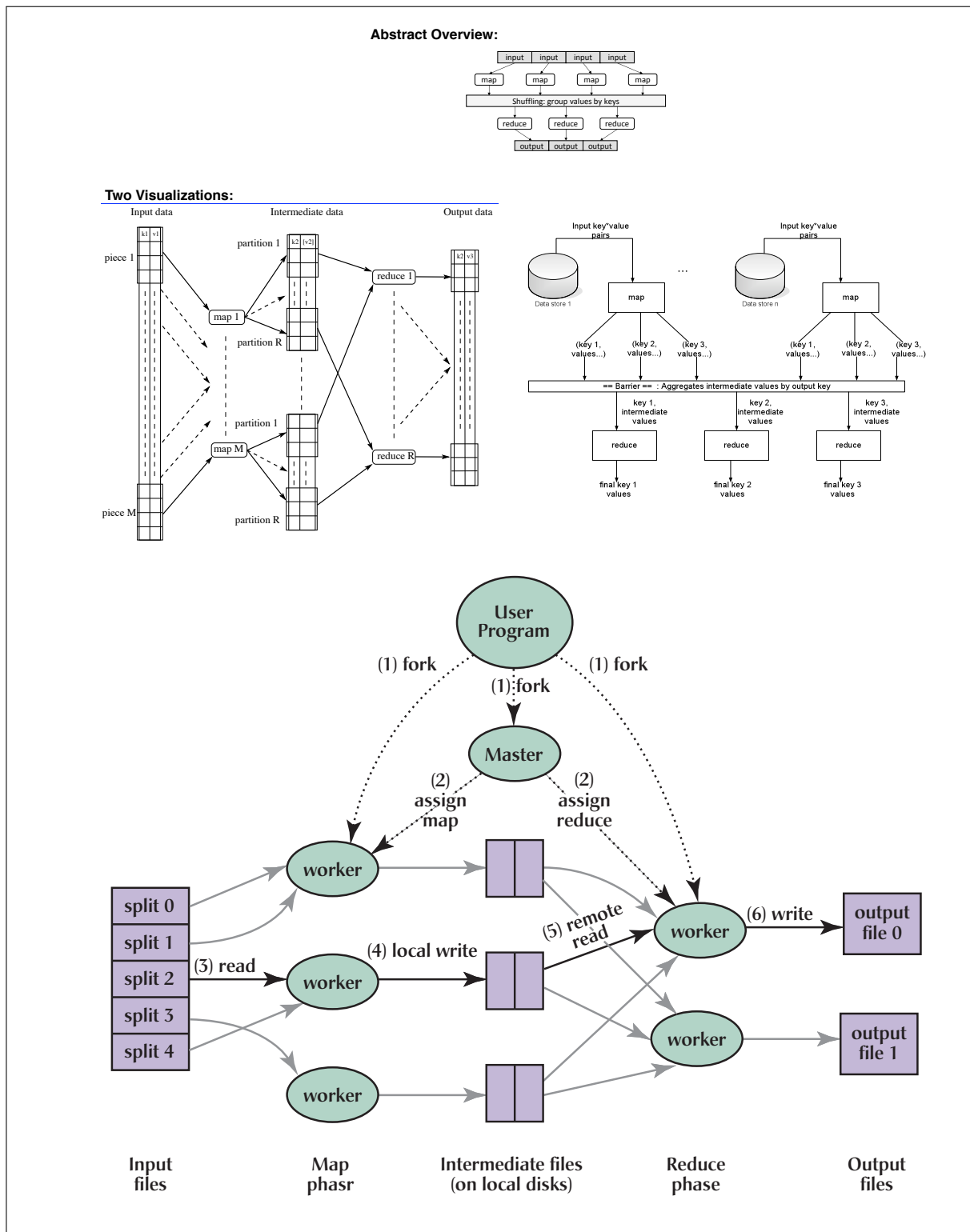
Die *automatische* Verteilung von Daten-Verarbeitungsprozessen auf (viele) verteilte Rechner. Besonders Prozesse, die auf großen Datenmengen operieren, so dass die zugrundeliegenden Daten so partitioniert werden können, dass der gleiche/ähnliche Verarbeitungsschritte auf allen Partitionen ausgeführt werden können.

- Erklären Sie die Herkunft des **Namens** Map/Reduce!

Ein Map/Reduce Prozess besteht aus zwei Teilen:

- In der **Map** Phase wird eine Funktion auf jedes Element der Eingabe angewandt.
- In der **Reduce** Phase werden die Ergebnisse der **Map** Phase nach bestimmten Kriterien gruppiert und/oder aggregiert.

- Beschreiben Sie den **Datenfluss** in einem Map/Reduce System mit mehreren Workern!



- Beschreiben Sie, wie der **PageRank** eines gegebenen Web-Graphen mit Hilfe von Map/Reduce berechnet werden kann, insbesondere wie die Ein- und Ausgabedaten der verschiedenen Map- bzw. Reduce-Tasks aussehen.

VERTEILTE PAGERANK BERECHNUNG:

- Initialisierung:** Sei L_i die Liste der URLs, die von URL_i verlinkt werden (ändert sich nicht während der Berechnung).

$$\{(\text{URL}_i, \text{Inhalt URL}_i)\} \quad \text{-MAP-} \rightarrow \{(\text{URL}_i, [\mathbf{p}_i^0, L_i])\}.$$

– **Iteration:**

$$\{(\text{URL}_i, [\mathbf{p}_i^k, L_i])\} \quad \text{-MAP-} \rightarrow \left\{ (\text{URL}_i, L_i), \left(\text{URL}_j, \frac{\mathbf{p}_i^k}{|L_i|} \right) : \text{URL}_j \in L_i \right\}$$

$$\text{-GROUP-} \rightarrow \left\{ \left(\text{URL}_i, \left[L_i, \frac{\mathbf{p}_j^k}{|L_j|} : \text{URL}_i \in L_j \right] \right) \right\}$$

$$\text{-REDUCE-} \rightarrow \left\{ \left(\text{URL}_i, \left[\mathbf{p}_i^{k+1} = \sum_j \frac{\mathbf{p}_j^k}{|L_j|} : \text{URL}_i \in L_j, L_i \right] \right) \right\}.$$

- Durch die Gruppierung werden alle PageRank-Anteile von auf i verlinkenden Seiten zu i gruppiert (zusammen mit L_i , das für den nächsten Iterations-Schritt bewahrt werden muss). “ $\forall \text{URL}_j$ mit $\text{URL}_i \in L_j$ ” wird nicht neu berechnet, sondern ergibt sich direkt aus der Ausgabe von MAP, die genau dann ein $(\text{URL}_i, \mathbf{p}_j^k/|L_j|)$ Paar enthält, wenn $\text{URL}_i \in L_j$.
- Die Summe ergibt sich durch einfaches Aufsummieren der bereits gruppierten PageRank-Anteile. Man beobachte, dass die Ausgabe das gleiche “Schema” wie die Eingabe hat und damit die Iteration fortgesetzt werden kann.
- Die Abbruchbedingung muss von außen, z.B. durch den Map/Reduce Controller, überprüft werden.
- Die PageRank-Berechnung mit Map/Reduce ist also ein Beispiel für eine Berechnung, bei der die Anzahl der Map/Reduce-Phasen nicht vorher bekannt ist!

- Geben Sie Beispiele für Probleme, die sich schlecht mit Hilfe von Map/Reduce lösen lassen! Können Sie eine Gemeinsamkeit dieser Probleme finden?

Klassische Beispiele sind Probleme, deren Lösung nur effizient gelingt mit Hilfe eines globalen Speichers (auf den häufig Zugriffen wird). Ein berühmtes Beispiel dafür ist das Sieb des Eratosthenes.

```

1  const N = 10000
   var gestrichen: array [2..N] of boolean
3
   { Initialisierung des Primzahlfeldes: }
5  { Alle Zahlen im Feld sind zu Beginn nicht gestrichen. }
   for i = 2 to N do
7    gestrichen[i] = false
   end
9
   i = 2
11  while i*i <= N
   do
13    if not gestrichen[i]
       then
15      // i ist prim, streiche seine Vielfache mit i*i beginnend:
       for j = i*i to N step i
17        do
           gestrichen[j] = true
19        end
       endif
21      i = i+1
   end
23
   for i = 2 to N do
25     if not gestrichen[i] then
       print i; ", ";
27   end

```

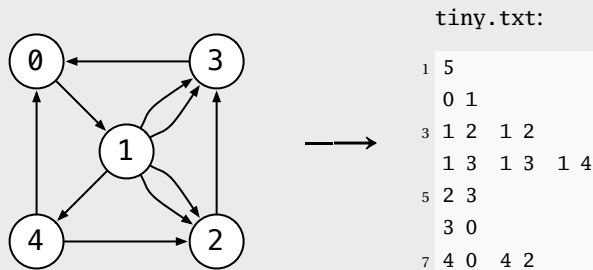
— SÜ-4.2 —

Implementierung PageRank

Programmierung

In dieser Aufgabe sollen Sie den in der Vorlesung vorgestellten Algorithmus zur *iterativen* Berechnung des **PageRanks** implementieren. Dabei gehen wir davon aus, dass nur Web-Graphen ohne *dangling nodes*, also ohne Knoten, die keine ausgehenden Kanten haben, vorkommen.

Als Eingabe des Algorithmus soll eine Repräsentation des Webs in der folgenden Form dienen:



Die erste Zeile gibt die Anzahl der Seiten N , die folgenden N Zeilen Paare von Webseiten zur Repräsentation von Links (z.B. 4 0 4 2 repräsentiert einen Link von Seite 4 auf Seite 0 und einen Link von Seite 4 auf Seite 2). Man beachte, dass mehrere Links zwischen denselben Seiten zugelassen sind und behandelt werden sollen.

Wie in der Vorlesung zerlegen wir den Algorithmus in zwei Teile:

1. *Berechnung der Google-Matrix*: Im ersten Schritt soll die Google-Matrix *mit random leap* wie in der Vorlesung berechnet werden.

EXAMPLE GOOGLE-MATRIX:

Für obigen Web-Graphen ergibt sich die Google-Matrix G (mit *random leap probability* $\alpha = 0.1$) wie folgt:

$$\begin{aligned}
 A^T &= \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}, & D &= \begin{pmatrix} 1 \\ 5 \\ 1 \\ 1 \\ 2 \end{pmatrix}, & H &= \begin{pmatrix} 0 & 0 & 0 & 1 & 0.5 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0.4 & 0 & 0 & 0.5 \\ 0 & 0.4 & 1 & 0 & 0 \\ 0 & 0.2 & 0 & 0 & 0 \end{pmatrix}, \\
 G &= \begin{pmatrix} 0 & 0 & 0 & 0.90 & 0.45 \\ 0.90 & 0 & 0 & 0 & 0 \\ 0 & 0.36 & 0 & 0 & 0.45 \\ 0 & 0.36 & 0.90 & 0 & 0 \\ 0 & 0.18 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0.02 & 0.02 & 0.02 & 0.02 & 0.02 \\ 0.02 & 0.02 & 0.02 & 0.02 & 0.02 \\ 0.02 & 0.02 & 0.02 & 0.02 & 0.02 \\ 0.02 & 0.02 & 0.02 & 0.02 & 0.02 \\ 0.02 & 0.02 & 0.02 & 0.02 & 0.02 \end{pmatrix} = \begin{pmatrix} 0.02 & 0.02 & 0.02 & 0.92 & 0.47 \\ 0.92 & 0.02 & 0.02 & 0.02 & 0.02 \\ 0.02 & 0.38 & 0.02 & 0.02 & 0.47 \\ 0.02 & 0.38 & 0.92 & 0.02 & 0.02 \\ 0.02 & 0.20 & 0.02 & 0.02 & 0.02 \end{pmatrix}
 \end{aligned}$$

2. *Iterative Berechnung des PageRanks*: Mit der soeben berechneten Google-Matrix G und einem beliebigen Startvector (z.B. $\mathbf{p}^0 = (1, 0, \dots, 0)^T$) wird eine Näherung des PageRank-Vektors iterativ berechnet durch die

Formel:

$$\mathbf{p}^{k+1} = G\mathbf{p}^k$$

Der Algorithmus soll nach einer vorgegebenen Anzahl von Iteration abbrechen und die erreichte Approximation für jede der N Seiten ausgeben.

Das Programm soll die Anzahl der Iteration und den Namen der Datei, die eine Repräsentation des Web-Graphens wie oben enthält, von der Kommandozeile lesen.

Bei der Eingabe

```
java t04.t04CLIDriver 30 tiny.txt
```

soll das Programm beispielsweise die folgende Ausgabe liefern:

```
1 Transition Matrix -----
   Dimensions: 5 5
3   0.02000 0.02000 0.02000 0.92000 0.47000
   0.92000 0.02000 0.02000 0.02000 0.02000
5   0.02000 0.38000 0.02000 0.02000 0.47000
   0.02000 0.38000 0.92000 0.02000 0.02000
7   0.02000 0.20000 0.02000 0.02000 0.02000
   PageRank vector (approx.) -----
9   after 1 iterations:    0.02000 0.92000 0.02000 0.02000 0.02000
   after 2 iterations:    0.04700 0.03800 0.36020 0.36920 0.18560
11  after 3 iterations:    0.43580 0.06230 0.11720 0.35786 0.02684
   after 4 iterations:    0.35415 0.41222 0.05451 0.14791 0.03121
13  after 5 iterations:    0.16716 0.33874 0.18245 0.21745 0.09420
   after 6 iterations:    0.25810 0.17045 0.18434 0.30615 0.08097
15  after 7 iterations:    0.33197 0.25229 0.11780 0.24726 0.05068
   after 8 iterations:    0.26534 0.31877 0.13363 0.21684 0.06541
17  after 9 iterations:    0.24459 0.25881 0.16419 0.25503 0.07738
   after 10 iterations:   0.28434 0.24013 0.14799 0.26095 0.06659
19  after 11 iterations:   0.28481 0.27591 0.13641 0.23964 0.06322
   after 12 iterations:   0.26413 0.27633 0.14778 0.24210 0.06966
21  after 13 iterations:   0.26924 0.25771 0.15083 0.25248 0.06974
   after 14 iterations:   0.27862 0.26231 0.14416 0.24852 0.06639
23  after 15 iterations:   0.27355 0.27075 0.14431 0.24418 0.06722
   after 16 iterations:   0.27001 0.26619 0.14772 0.24735 0.06874
25  after 17 iterations:   0.27354 0.26301 0.14676 0.24878 0.06791
   after 18 iterations:   0.27446 0.26619 0.14524 0.24677 0.06734
27  after 19 iterations:   0.27239 0.26701 0.14613 0.24655 0.06791
   after 20 iterations:   0.27245 0.26515 0.14669 0.24764 0.06806
29  after 21 iterations:   0.27351 0.26521 0.14608 0.24747 0.06773
   after 22 iterations:   0.27320 0.26616 0.14595 0.24695 0.06774
31  after 23 iterations:   0.27274 0.26588 0.14630 0.24717 0.06791
   after 24 iterations:   0.27302 0.26546 0.14628 0.24739 0.06786
33  after 25 iterations:   0.27318 0.26571 0.14610 0.24722 0.06778
   after 26 iterations:   0.27300 0.26587 0.14616 0.24715 0.06783
35  after 27 iterations:   0.27296 0.26570 0.14623 0.24726 0.06786
   after 28 iterations:   0.27306 0.26566 0.14619 0.24726 0.06783
37  after 29 iterations:   0.27306 0.26576 0.14616 0.24721 0.06782
   after 30 iterations:   0.27300 0.26575 0.14619 0.24722 0.06784
39 -- final page rank vector (after 30 iterations):
   0.27300 0.26575 0.14619 0.24722 0.06784
```

Hinweis: In den Beilagen zu diesem Übungsblatt finden Sie zwei Eingaben (**tiny.txt** und **medium.txt**) zum Testen Ihrer Implementation. Wie immer finden Sie auch `.java` Dateien, die Sie als Ausgangspunkt für eine eigenen Lösung verwenden können, indem Sie die mit `@TODO` markierten Teile ersetzen.

Command-Line Driver:

```

2 package t04;

4 import java.io.BufferedReader;
  import java.io.FileReader;
6 import java.io.IOException;

8 /**
   * Command-line driver for solutions of topic sheet 02.
10  *
   * @author Tim Furch
12  */
  public class t04CLIDriver {

14      private static final boolean STEP_BY_STEP = true;

16      public static void main(String[] args) throws IOException {
18          // number of iterations
          int numIters = Integer.parseInt(args[0]);

20          PageRankMatrix ptm = new PageRankMatrix(
22              new BufferedReader(new FileReader(args[1])));

24          System.out.println("Transition Matrix -----");
          System.out.print(ptm.toString());

26          System.out.println("PageRank vector (approx.) -----");
          double[] pr = ptm.pageRank(numIters, STEP_BY_STEP);

28          if (STEP_BY_STEP) {
30              System.out.printf("-- final page rank vector (after %d iterations):\n",
                                   numIters);
32              PageRankMatrix.printVector(pr);
          }

34      }

36  }

38 }

```

PageRankMatrix:

```

1 package t04;

3 import java.io.BufferedReader;
  import java.io.IOException;
  import java.util.Formatter;

7 /**
   * This class provides methods for computing the page rank transition matrix as
   * well as the approximated page rank vector.
11  */
  public class PageRankMatrix {

13      public double[][] matrix;

```

```

15     public final static double DEFAULT_RANDOM_JUMP_PROBABILITY = 0.1;
17
18     public final double rndJumpProbability;
19
20     /**
21      * Reads a page rank matrix from the given reader.
22      */
23     PageRankMatrix(BufferedReader reader) throws IOException {
24         this(reader, DEFAULT_RANDOM_JUMP_PROBABILITY);
25     }
26
27     /**
28      * Reads a page rank matrix from the given reader using the given random jump
29      * probability.
30      */
31     PageRankMatrix(BufferedReader reader, double rndJumpProbability) throws IOException {
32         this.rndJumpProbability = rndJumpProbability;
33         this.computeFromInput(reader);
34     }
35
36     /**
37      * Performs the actual computation of the page rank matrix from the given reader.
38      */
39     private void computeFromInput(BufferedReader reader) throws IOException {
40         int numPages = Integer.parseInt(reader.readLine().trim());
41
42         // counts[i][j] = # links from page i to page j
43         int[][] counts = new int[numPages][numPages];
44         // outDegree[j] = # links from page i to anywhere
45         int[] outDegree = new int[numPages];
46
47         // Read link counts.
48         String line;
49         while ((line = reader.readLine()) != null) {
50             String[] nrs = line.trim().split("\\s");
51             int i = 0;
52             while(i < nrs.length) {
53                 while (i < nrs.length && nrs[i].equals("")) i++;
54                 if (i == nrs.length) break;
55                 int start = Integer.parseInt(nrs[i]);
56                 i++;
57                 while (i < nrs.length && nrs[i].equals("")) i++;
58                 if (i == nrs.length) break;
59                 int end = Integer.parseInt(nrs[i]);
60                 i++;
61                 outDegree[start]++;
62                 counts[start][end]++;
63             }
64         }
65
66         matrix = new double[numPages][numPages];
67         // Compute, store probability distribution for row i.
68         for (int i = 0; i < numPages; i++) {
69             // Compute transition probabilities for column j.

```

```

71     for (int j = 0; j < numPages; j++) {
72         /* Probability for transition from j to i is (a) the probability for a
73         * random jump divided by the number of pages + (b) the portion of links
74         * from j to i among all links in j times the inverse of the random jump
75         * probability.
76         */
77         matrix[i][j] = (1-this.rndJumpProbability)*counts[j][i]/outDegree[j] +
78             this.rndJumpProbability/numPages;
79     }
80 }
81
82 /**
83 * Returns the approximated page rank vector after the given number of
84 * iterations. If <code>step_by_step</code> is <code>true</code> the
85 * final and intermediate page rank vectors are printed to {@link System#out}.
86 */
87 public double[] pageRank(int numIterations, boolean step_by_step) {
88     int numPages = this.matrix.length;
89     // Use the power method to compute page ranks.
90     double[] rank = new double[numPages];
91     //for (int i = 0; i < numPages; i++) rank[i] = 1.0;
92     rank[0] = 1.0;
93
94     for (int t = 0; t < numIterations; t++) {
95         // Compute effect of next move on page ranks.
96         double[] newRank = new double[numPages];
97         for (int i = 0; i < numPages; i++) {
98             // New rank of page j is dot product of old ranks and column j of p[][].
99             for (int j = 0; j < numPages; j++)
100                 newRank[i] += this.matrix[i][j] * rank[j];
101         }
102
103         // Update page ranks.
104         rank = newRank;
105         if(step_by_step) {
106             System.out.printf("    after %3d iterations:    ", t+1);
107             // print page ranks
108             printVector(rank);
109         }
110     }
111
112     return rank;
113 }
114
115 @Override
116 public String toString() {
117     StringBuilder sb = new StringBuilder();
118     Formatter transitionValueFormatter = new Formatter(sb);
119
120     sb.append("    Dimensions: ");
121     sb.append(matrix.length).append(" ").append(matrix[0].length).append("\n");
122     for (int i = 0; i < matrix.length; i++) {
123         sb.append("    ");
124         for (int j = 0; j < matrix[i].length; j++) {
125             transitionValueFormatter.format("%7.5f ", matrix[i][j]);

```



```
127         }
128         sb.append("\n");
129     }
130     return sb.toString();
131 }
132
133 public static void printVector(double[] vector) {
134     // print page ranks
135     for (int i = 0; i < vector.length; i++) {
136         System.out.printf("%8.5f", vector[i]);
137     }
138     System.out.println();
139 }
140
141 }
```