BIG DATA

HÄUFIGKEITSBERECHNUNG

HTW Berlin | Programmierkonzepte und Algorithmen | 30 VI 2020 | Loparev & Nguyen

INHALT

1. Erklärung der Aufgabe
2. Planung und Lösungsbeschreibung
3. Ergebnis
4. Diagrammen für die Leistung und Laufzeit
5. Fazit

Aufgabe 09

Häufigkeitsberechnung

- Berechnen Sie die Vorkommenshäufigkeit für alle Wörter in gegebenen Texten
- Stellen Sie die Ergebnisse als TOP-10 Liste getrennt für jede Sprache vor, mit Ausnahme der Stoppwörter (z.B. “a”, “an”, “the”, “of”, “und”, “mit”, “la”, “и”, “же” usw.)

Planung

INPUT

8 Ordner mit literarischen Werken (8 verschiedene Sprachen)

1

alle Stoppwörter aus den Werken entfernen

2

die Häufigkeit der Wörter berechnen

3

die Wörterhäfigkeit absteigend sortieren

OUTPUT

Top-10 Worte aus jeder Sprache holen und im Hadoop-Kontext ausgeben

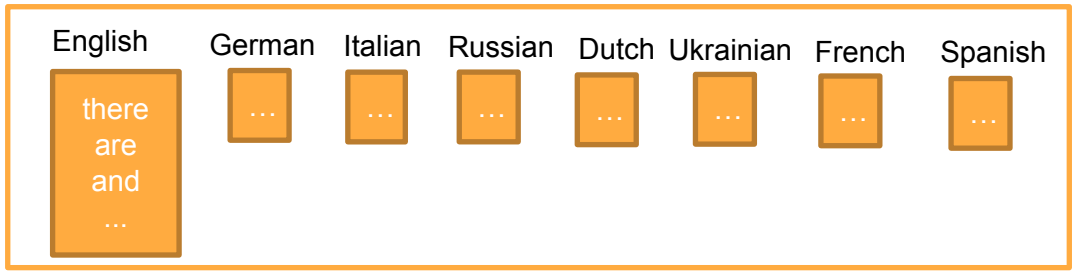
Eingabe

Lösungsbeschreibung

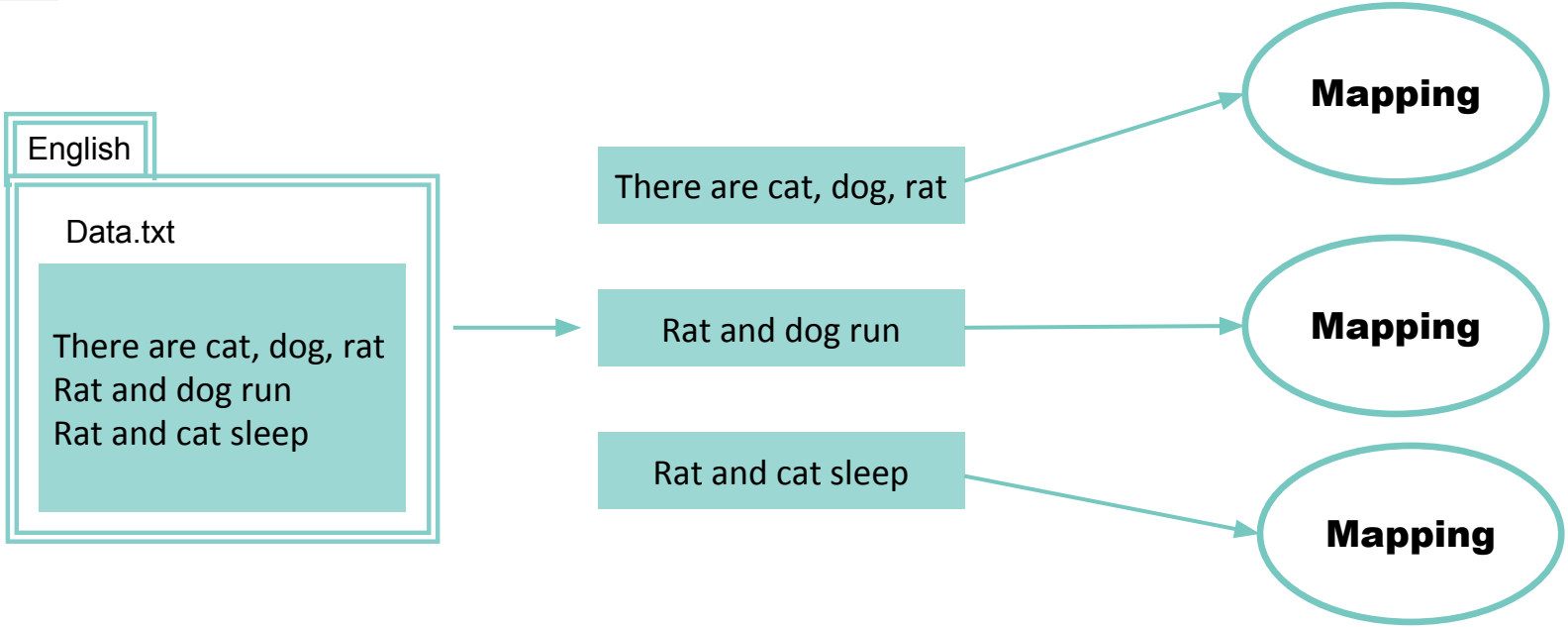
Input

- English
  - Data.txt
- German
- Dutch
- Italian
- French
- Russian
- Ukrainian
- Spanish

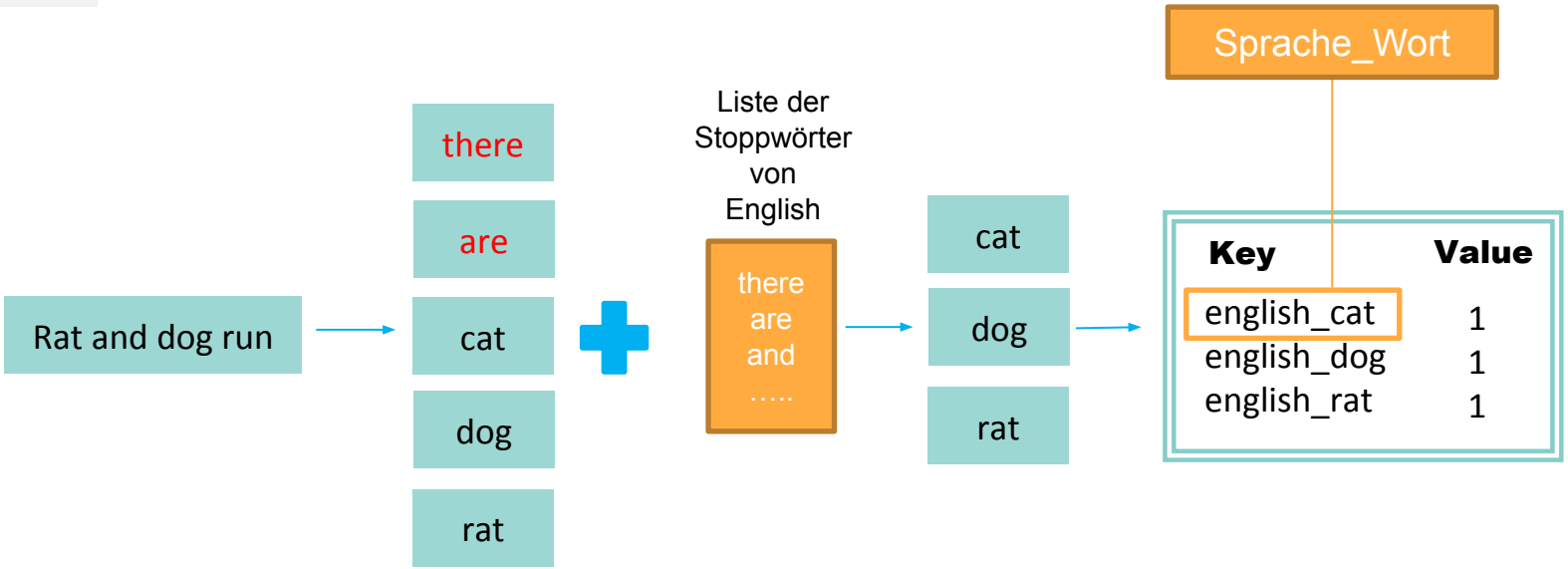
Stopwords



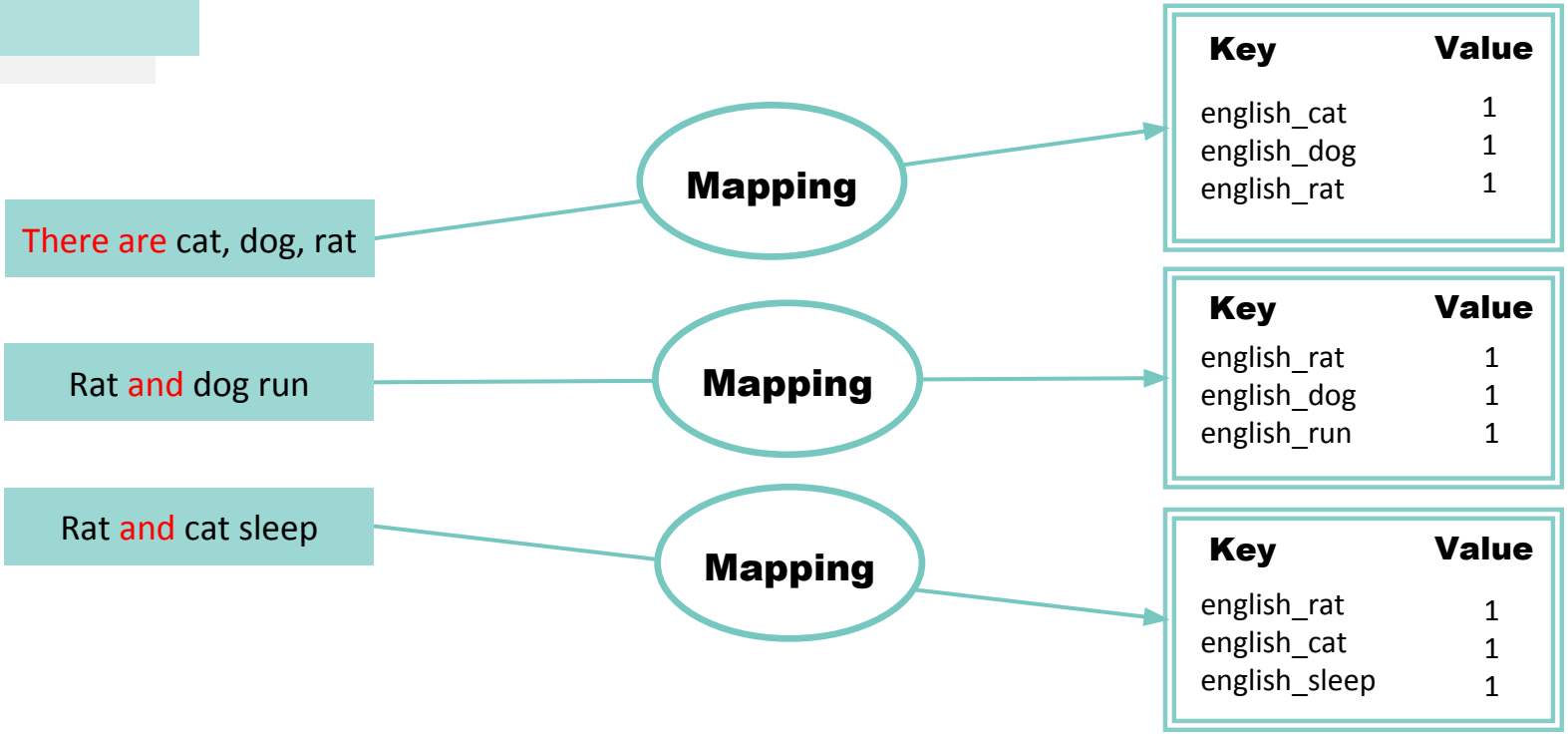
Vorbereitung auf Eingabe für Mapping



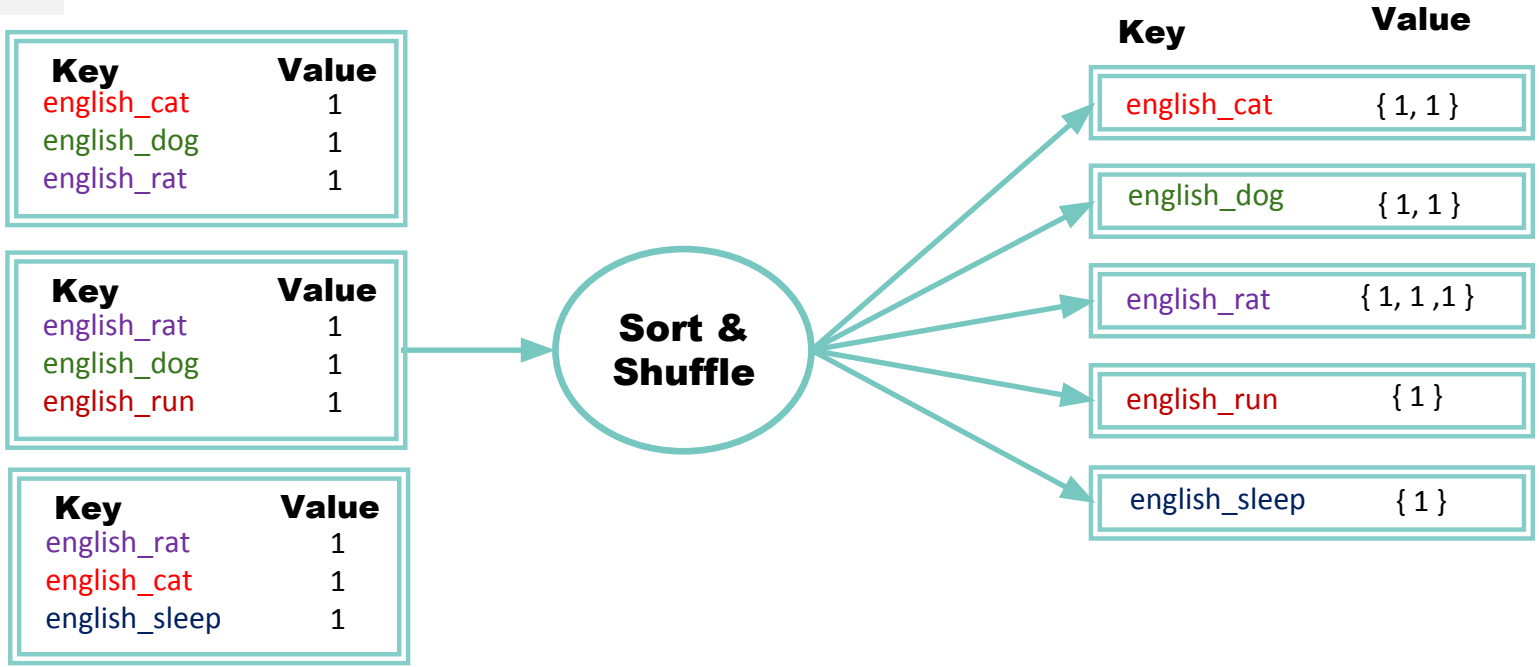
Mapping


Sprache_Wort
Liste der Stoppwörter von English
there
are
and
.....
Rat and dog run
there
are
cat
dog
rat
cat
dog
rat
Key
Value
english_cat
1
english_dog
1
english_rat
1

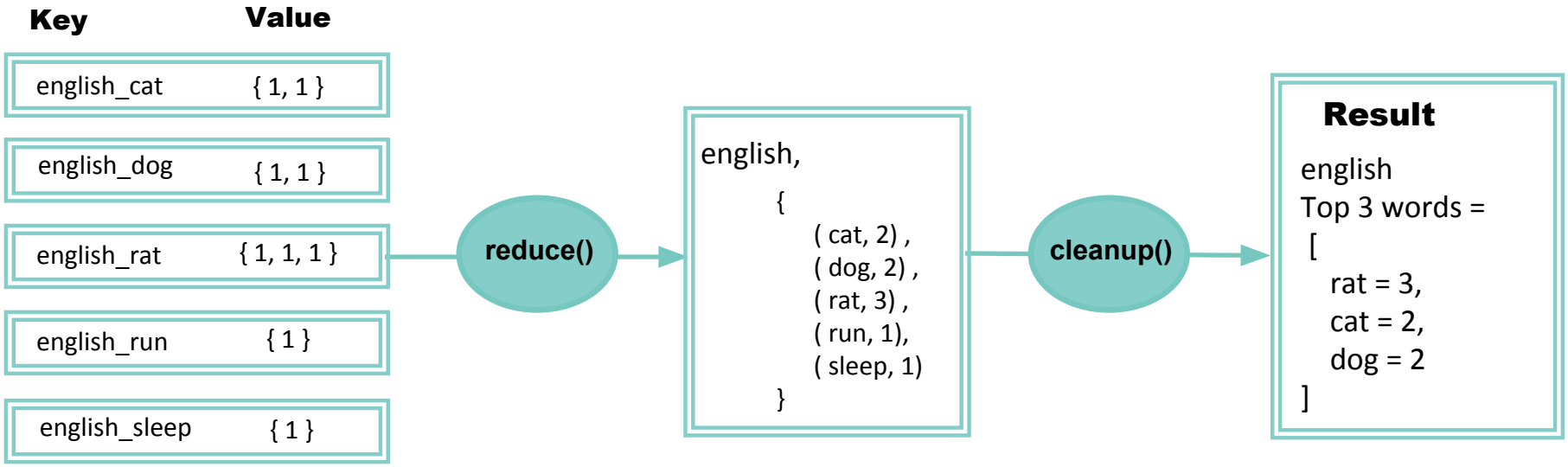
Mapping


There are cat, dog, rat
Rat and dog run
Rat and cat sleep
Mapping
Mapping
Mapping
Key
Value
english_cat 1
english_dog 1
english_rat 1
Key
Value
english_rat 1
english_dog 1
english_run 1
Key
Value
english_rat 1
english_cat 1
english_sleep 1

Sort and Shuffle



Reducer


Key
Value
english_cat
{ 1, 1 }
english_dog
{ 1, 1 }
english_rat
{ 1, 1, 1 }
english_run
{ 1 }
english_sleep
{ 1 }
reduce()
english,
{
(cat, 2) ,
(dog, 2) ,
(rat, 3) ,
(run, 1),
(sleep, 1)
}
cleanup()
Result
english
Top 3 words =
[
rat = 3,
cat = 2,
dog = 2
]

Mapping

```
01. public class MyMapper extends Mapper<LongWritable, Text, Text, Text> {
02.
03.     private final static Text one = new Text("1");
04.     private Text word = new Text();
05.     private HashMap<String, HashSet<String>> stopwordList = new HashMap<>();
06.
07.     public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
08.
09.         String line = value.toString().toLowerCase();
10.         List<String> words = new ArrayList<>();
11.
12.         Collections.addAll(words, line.split("\\P{L}+"));
13.
14.         String filePathString = ((FileSplit) context.getInputSplit()).getPath().toString();
15.         String language = new Path(filePathString).getParent().getName().toLowerCase();
16.
17.         if (!stopwordList.containsKey(language)) {
18.
19.             HashSet<String> stopwords = Utils.readFileStopword(language + ".txt");
20.
21.             if (stopwords == null) {
22.                 return;
23.             }
24.
25.             stopwordList.put(language, stopwords);
26.         }
27.
28.         for (String w : words) {
29.
30.             if (w.length() <= 2) {
31.                 continue;
32.             }
33.
34.             if (stopwordList.get(language).contains(w)) {
35.                 continue;
36.             }
37.
38.             word.set(language + "_" + w);
39.             context.write(word, one);
40.         }
41.     }
42. }
```

Eine Liste "stopwordList" als HashMap erstellen, in der die Stoppwörter jeder Sprache gespeichert werden.

Die Kette in String und in Kleinbuchstaben konvertieren.

die Kette in Wörter teilen

Den Name des Ordners holen, der die Textdatei der Kette enthält und auch der Name der Sprache ist

Checken, ob die Sprache bereits in der Liste der Stoppwörter ist oder nicht?

Laut der Sprache die Daten der Stoppwörter in einer Liste laden

die Liste der Stoppwörter in der gemeinsamen Liste der Stoppwörter hinzufügen.

Wenn das Wort seine Länge <= 2 hat, wird es ignoriert

Checken, ob das Wort Stoppwort ist oder nicht? Wenn Ja, wird es ignoriet

Die Daten in der Hadoop--Kontext als Paaren <"Sprache_Wort", 1> schreiben

Eine Kette als Form "<Sprache>_<Wort>" erstellen, um den Schlüssel zu erstellen

Reducer

reduce()

```java
public class MyReducer extends Reducer<Text, Text, Text, Text> {

    private HashMap<String, HashMap<String, Integer>> allData = new HashMap<>();

    private final int MAX_TOP = 10;

    @Override
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

        String[] language_key = key.toString().split("_");
        String language = language_key[0];
        String str = language_key[1];

        int sum = 0;

        for (Text val : values) {
            sum += Integer.parseInt(val.toString());
        }

        if (allData.containsKey(language)) {

            allData.get(language).put(str, sum);
        } else {

            HashMap<String, Integer> hmap = new HashMap<>();

            hmap.put(str, sum);
            allData.put(language, hmap);
        }
    }
```

eine Liste "allData" als HashMap erstellen, in der die Wörter jeder Sprache gespeichert werden

Die Anzahl von TOP der Wörter, die ausgegeben werden

den "Key" in 2 Teilen schneiden. Der erste ist die Sprache und der zweite ist das Wort

die Häufigkeit des Wortes berechnen

Checken, ob die Sprache des Wortes bereits in der Liste "allData" ist oder nicht?

das Wort und seine Häufigkeit in der Liste hinzufügen.

cleanup()

```java
    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {

        Comparator<Entry<String, Integer>> comparator = new Comparator<Entry<String, Integer>>() {

            @Override
            public int compare(Entry<String, Integer> o1, Entry<String, Integer> o2) {
                return o2.getValue().compareTo(o1.getValue());
            }
        };

        for (Entry<String, HashMap<String, Integer>> entry : allData.entrySet()) {

            List<Entry<String, Integer>> values = new ArrayList<>(entry.getValue().entrySet());

            Collections.sort(values, comparator);

            StringBuilder sb = new StringBuilder("----------------------------------\n");

            sb.append(entry.getKey());
            sb.append("\nTop " + MAX_TOP + " words=");
            sb.append(values.subList(0, values.size() > MAX_TOP ? MAX_TOP : values.size()));

            context.write(new Text(sb.toString()), new Text("\n"));
        }
    }
}
```

Erstellen einen Komparator, um die Häufigkeitszahl zwischen den Wörter zu vergleichen.

Lesen jede Sprache in der Liste allData,

Sortieren die Wörter in absteigender Reihenfolge

Erstellen eine StringBuilder, um zu ausgebenden Informationen zu enthalten

Fügen den Name der Sprache hinzu

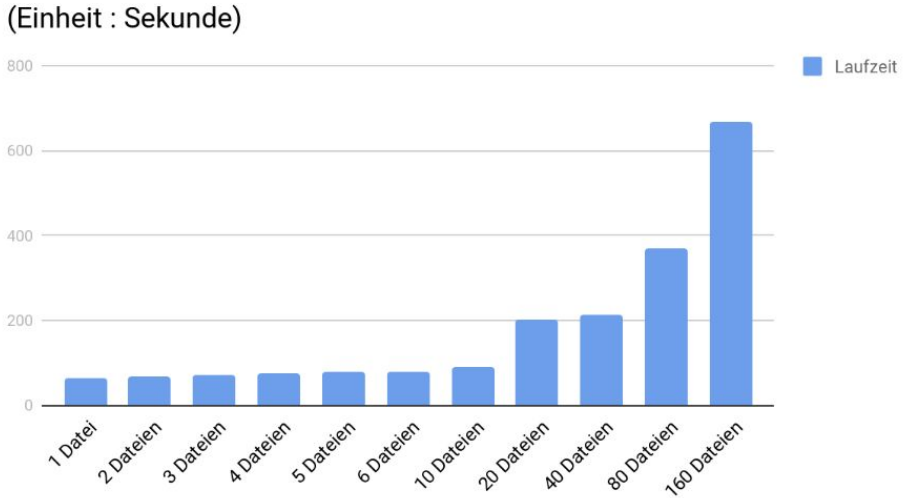
Holen sich Top 10 Wörter

Schreiben die Informationen der Sprache in Hadoop-Kontext

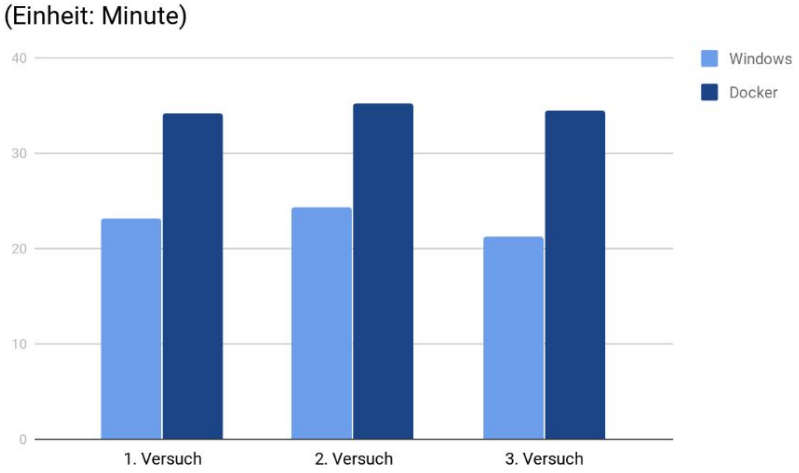
Ergebnis

```
--------------------------------------
dutch
Top 10 words=[den=1520, zoo=512, marten=330, baas=303, zien=230, jan=224, vrouw=201, oogen=194, riep=185, goed=178]

--------------------------------------
ukrainian
Top 10 words=[треба=1029, сказав=810, над=805, мати=783, перед=758, добре=749, серце=738, уже=738, чіпка=738, усе=731]

--------------------------------------
german
Top 10 words=[hand=5610, augen=5172, herr=4815, leben=3924, sprach=3701, stand=3630, vater=3553, ließ=3375, mußte=3243, alte=3102]

--------------------------------------
spanish
Top 10 words=[don=3073, casa=2451, ojos=1816, vida=1599, hombre=1565, dios=1508, señor=1497, padre=1395, mujer=1314, mano=1262]

--------------------------------------
russian
Top 10 words=[глаза=298, сердце=271, жизни=234, знаю=224, горский=224, руки=209, слова=200, вера=192, точно=191, руку=183]

--------------------------------------
english
Top 10 words=[man=11763, time=9585, good=6752, holmes=5899, long=5765, great=5534, day=5295, men=5018, thought=4828, night=4820]

--------------------------------------
italian
Top 10 words=[disse=8759, rispose=4023, occhi=3614, donna=3087, mano=3055, altra=2968, voce=2862, uomini=2851, signor=2710, piú=2626]

--------------------------------------
french
Top 10 words=[homme=10266, faire=9207, monsieur=7084, point=6670, jeune=5891, femme=5866, fit=5731, yeux=5615, jamais=5572, oui=5526]
```

Diagrammen



4.1 Laufzeitvergleich in Abhängigkeit von der Inputgröße



4.2 Laufzeitvergleich zwischen Hadoop und Docker

Fazit

Häufigkeitsberechnung

- Hadoop braucht relativ viel Zeit um einzelne Dateien zu öffnen
- die Verarbeitung an sich dauert verhältnismäßig schnell
- die Laufzeit kann durch Dateienvereinigung verbessert werden

Vielen Dank!

für Ihre Merksamkeit