Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Subsumption Computation

Gruppe 04: Philipp Berger, Thomas Zimmermann

Seminar „Map/Reduce Algorithms on Hadoop“
13.07.2009

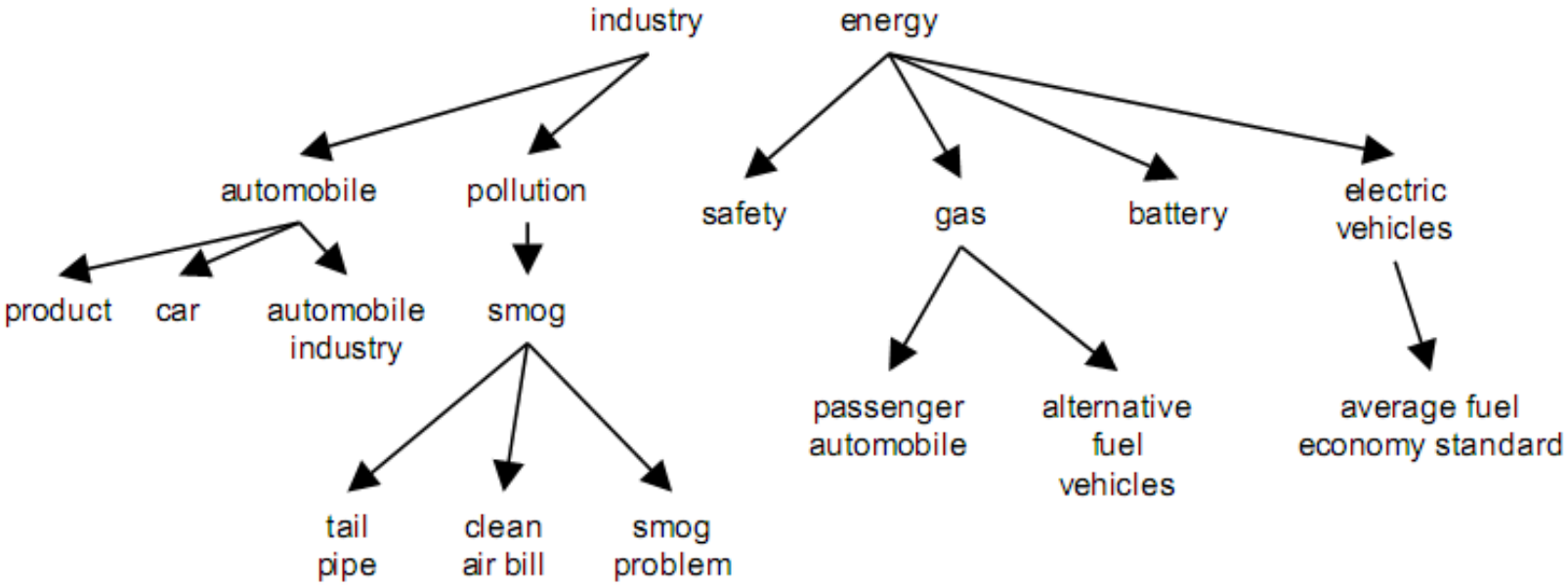
Inhalt

1. Aufgabe
2. Map/Reduce Phasen
3. Laufzeitverhalten

Aufgabe

Subsumption ≙ Klassifizierung, Einordnung

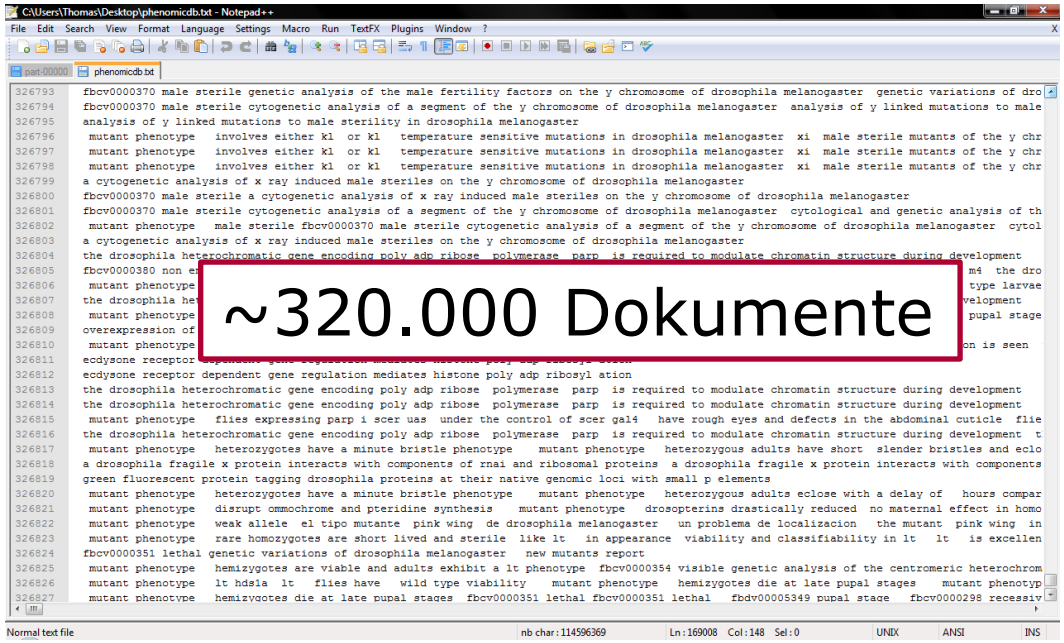
→ Falls P(x|y) >0,8 ist x wahrscheinlich y übergeordnet

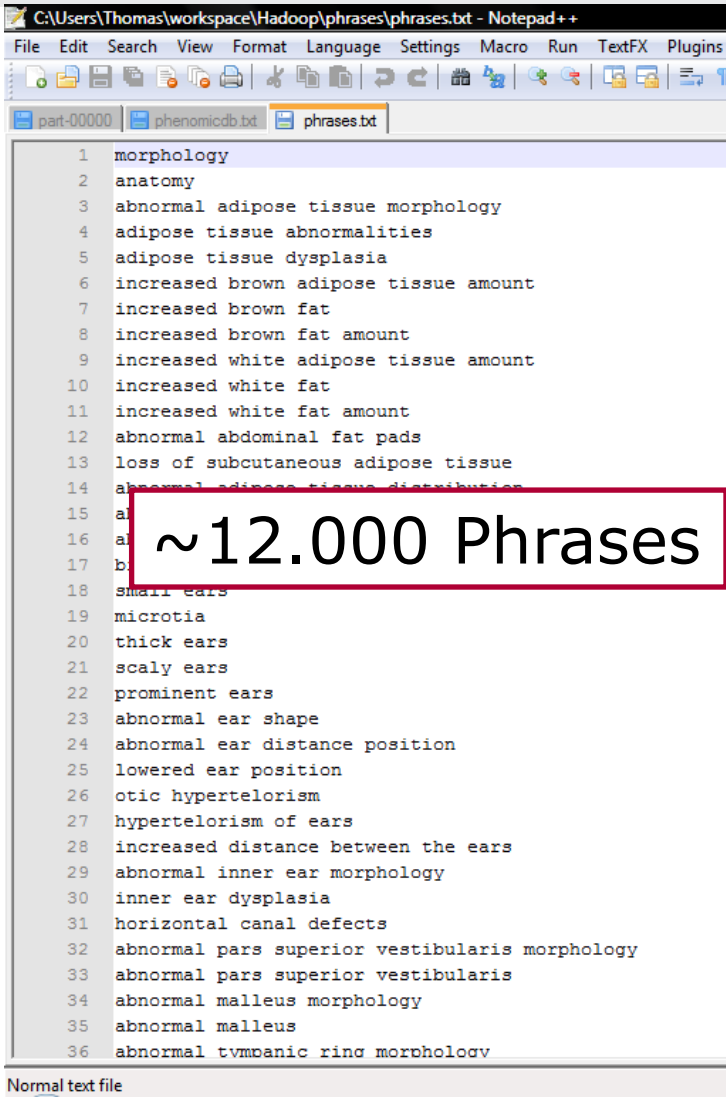


Quelle: Sanderson, Croft "Deriving concept hierarchies from text"

Problem: Bilde P(x|y) über große Dokumentmengen

Aufgabe Beispiel Corpus





Gesucht: P(x|y) für Phrases x und y, die gemeinsam vorkommen

Überblick der 3 Map/Reduce-Phasen

1. Phrases in Dokumenten lokalisieren

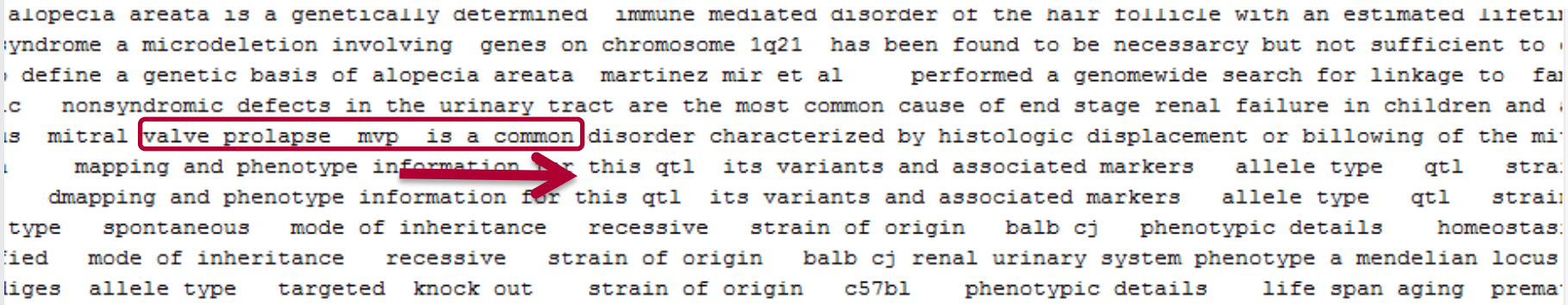
2. Zählen, wie oft Phrases einzeln und mit anderen Phrases zusammen vorkommen

3. Berechnen der bedingten Wahrscheinlichkeiten

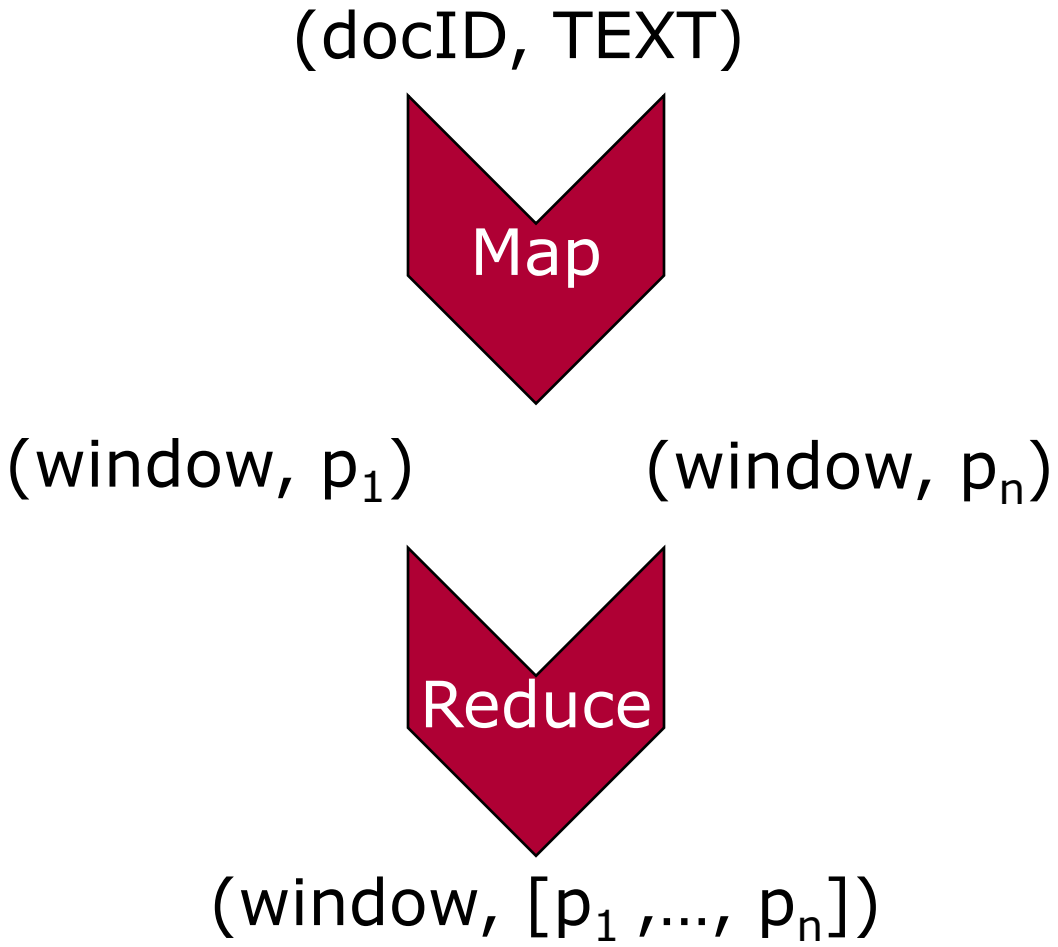
Ansatz – 1. Durchgang

Wann tauchen Phrases mit anderen Phrases **zusammen** auf?

Idee: Wenn sie im selben **Sliding Window** vorkommen



Lösung – 1. Durchgang



Vergleichsalgorithmen


```
public interface Similarity {

        public double computeSimilarity(String window,
                                        String phrase);
}
```

String.contains-Methode

Aufwand: (Länge Window-Länge Phrase) * Länge Phrase

Levenshtein-Distance (Edit-Distanz)

Aufwand: (Länge Window –Länge Phrase) * (Länge Phrase)²

Levenshtein-Distance auf Wort-Ebene

Aufwand: Anzahl Wörter in Window * (Anzahl Wörter in Phrase)²

Ansatz – 2. Durchgang

Wie oft kommen Phrases vor?

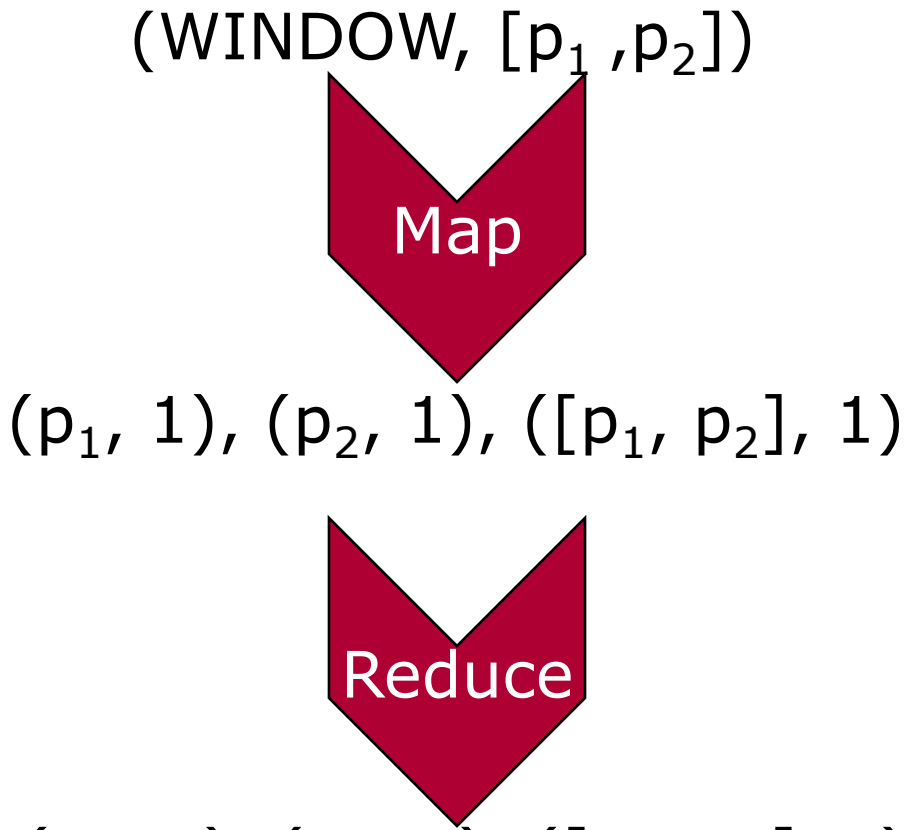
Idee: Mapper bildet Phrase-Teilmengen pro Window

1-elementige: Vorkommen einzelner Phrases

2-elementige: Vorkommen von paarweisen Phrases

Reducer summiert Vorkommen auf

Lösung – 2. Durchgang



Ansatz – 3. Durchgang

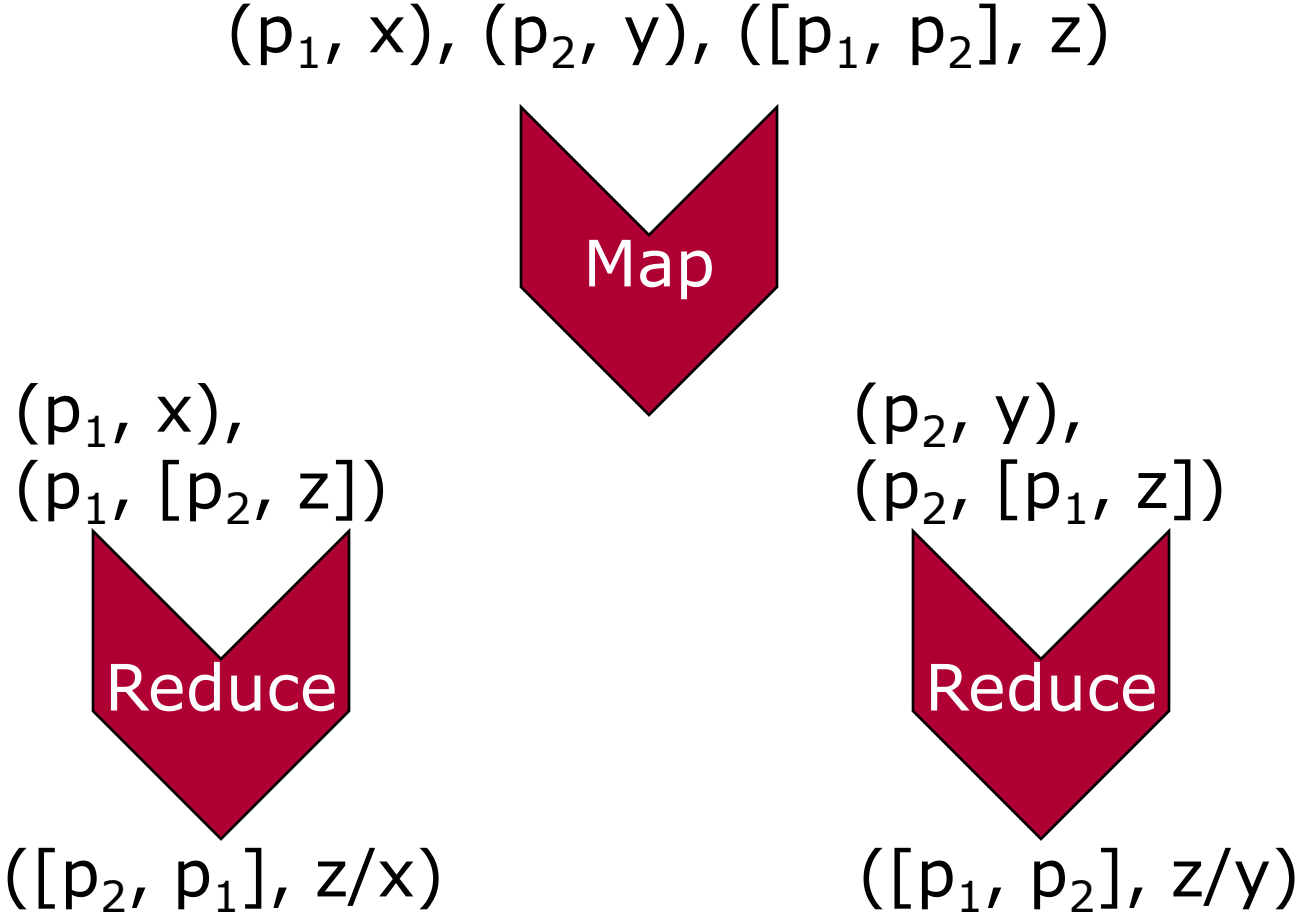
Wie lauten die **bedingten Wahrscheinlichkeiten**?

Idee: Mapper sammelt für alle Phrases:
-Absolutes Vorkommen
-Anzahl Vorkommen mit anderen Phrases

Reducer kann pro Phrase x alle P(x|y) für die y, mit denen x auftaucht, berechnen

$$P(x|y) = \frac{P(x \cap y)}{P(y)}$$

Lösung – 3. Durchgang



Problem beim 3. Reducer

$$P(x|y) = \frac{P(x \cap y)}{P(y)}$$

worst case:

(p_2, [p_1, z])
(p_2, [p_3, z])
(p_2, [p_2, z])
(p_2, [p_8, z])
(p_2, [p_4, z])
(p_2, [p_9, z])
(p_2, [p_{11}, z])
(p_2, y)

12000 phrases x

(4 Byte phraseID + 4 Byte relative Häufigkeit)

= 96 KByte

Lösung

Secondary Sort

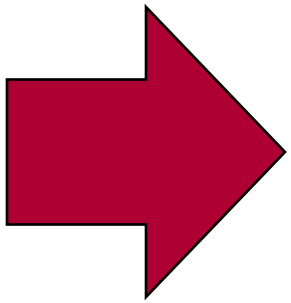
Eigene Pair-Klasse implementiert

→Sortierung nach beiden Werten

→Verteilung auf Reducer nur nach erstem Wert

Output des Mappers

$([p_2, p_1],[p_1, z_1])$
$([p_2, p_{11}],[p_{11}, z_{11}])$
$([p_2, p_3],[p_3, z_3])$
$([p_2, p_9],[p_9, z_9])$
$([p_2, „ “],[„ “, y])$



Input des Reducers

$([p_2, „ “],[„ “, y])$
$([p_2, p_1],[p_1, z_1])$
$([p_2, p_3],[p_3, z_3])$
$([p_2, p_9],[p_9, z_9])$
$([p_2, p_{11}],[p_{11}, z_{11}])$

Theoretischer Aufwand

#windows = (doccount * (avg-doc-length - win-size))

#computeSimilarity-Aufrufe= #phrases * #windows

Aufwand = Similarity-Aufwand * #computeSimilarity-Aufrufe

Theoretischer Aufwand - Beispiel

Levenshtein-Distance auf Wort-Ebene
Aufwand: Anzahl Wörter in Window * (Anzahl Wörter in Phrase)²

*#windows = (320.000 * (66 - 20)) = 14.720.000*

computeSimilarity-Aufrufe = *12.000 * 14.720.000 = 176.640.000.000*

*Aufwand = (20*3²) * 176.640.000.000 =* **3,17952 × 10^{13}**

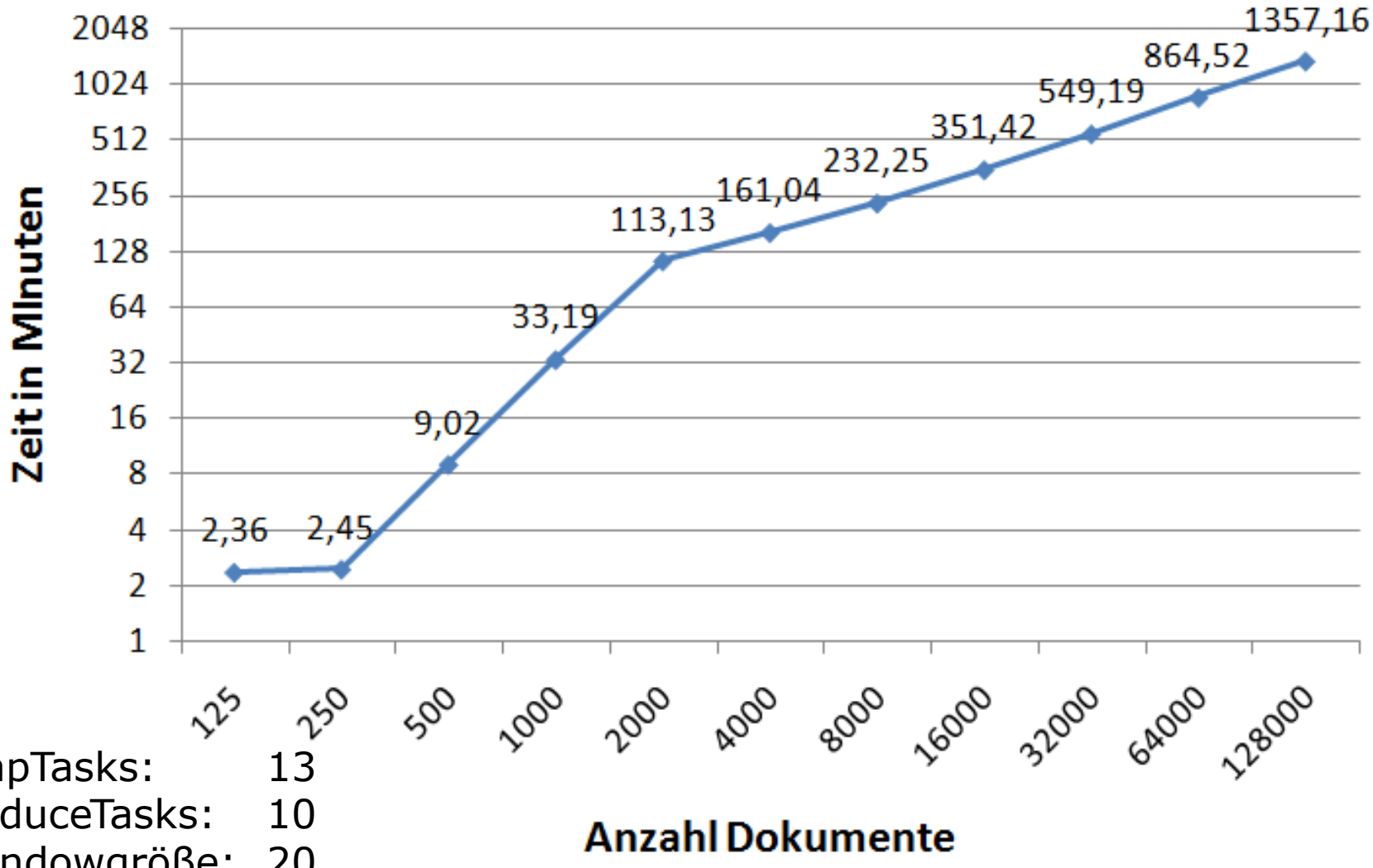
Laufzeitverhalten

Messungen für folgende Größen:

- Anzahl der Dokumente
- Anzahl MapTasks
- Anzahl Reducer
- Größe des Windows (in Worten)

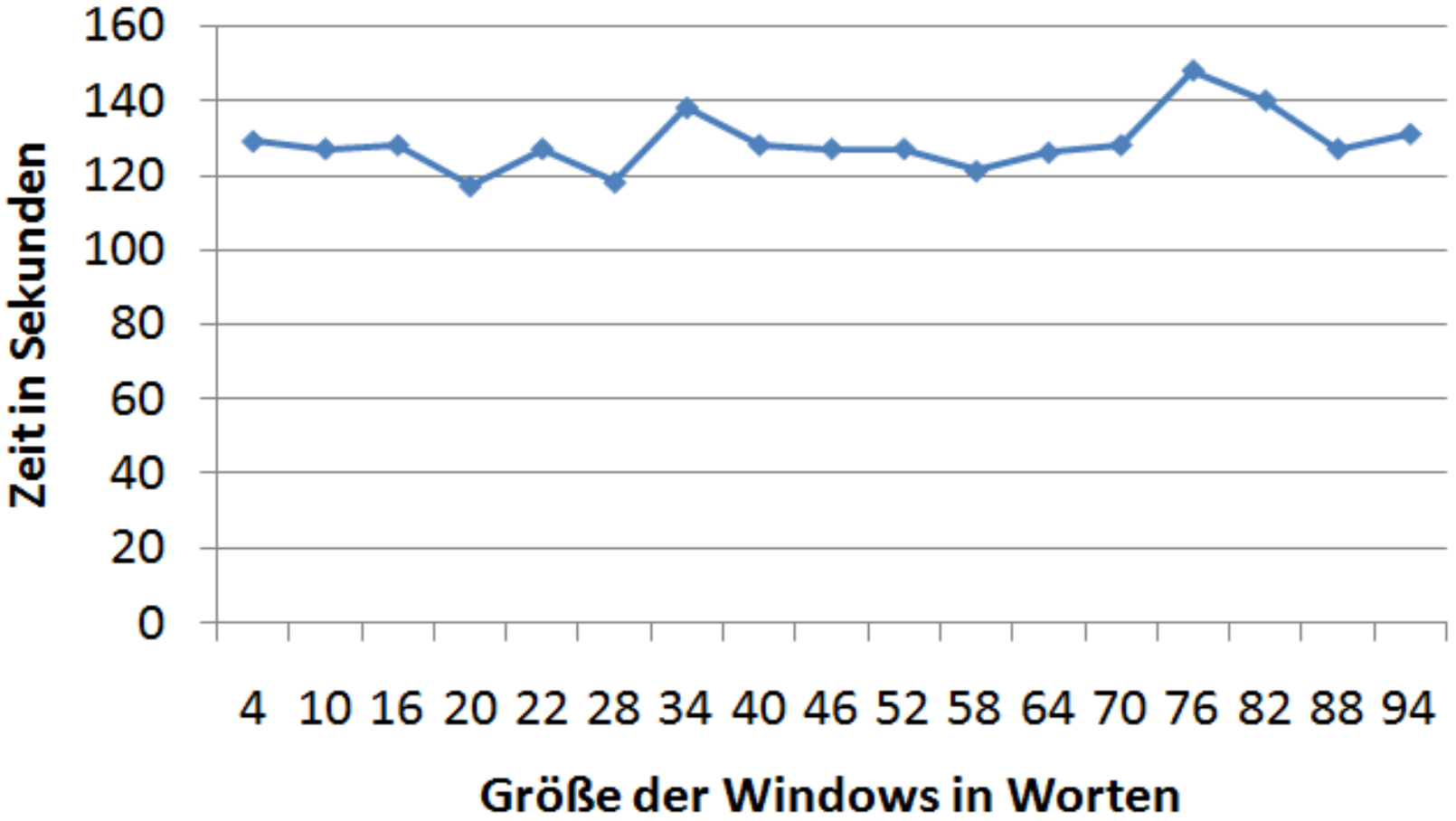
Umgebung:
Cluster mit 9 Knoten

Variable Größe: Dokumentenanzahl



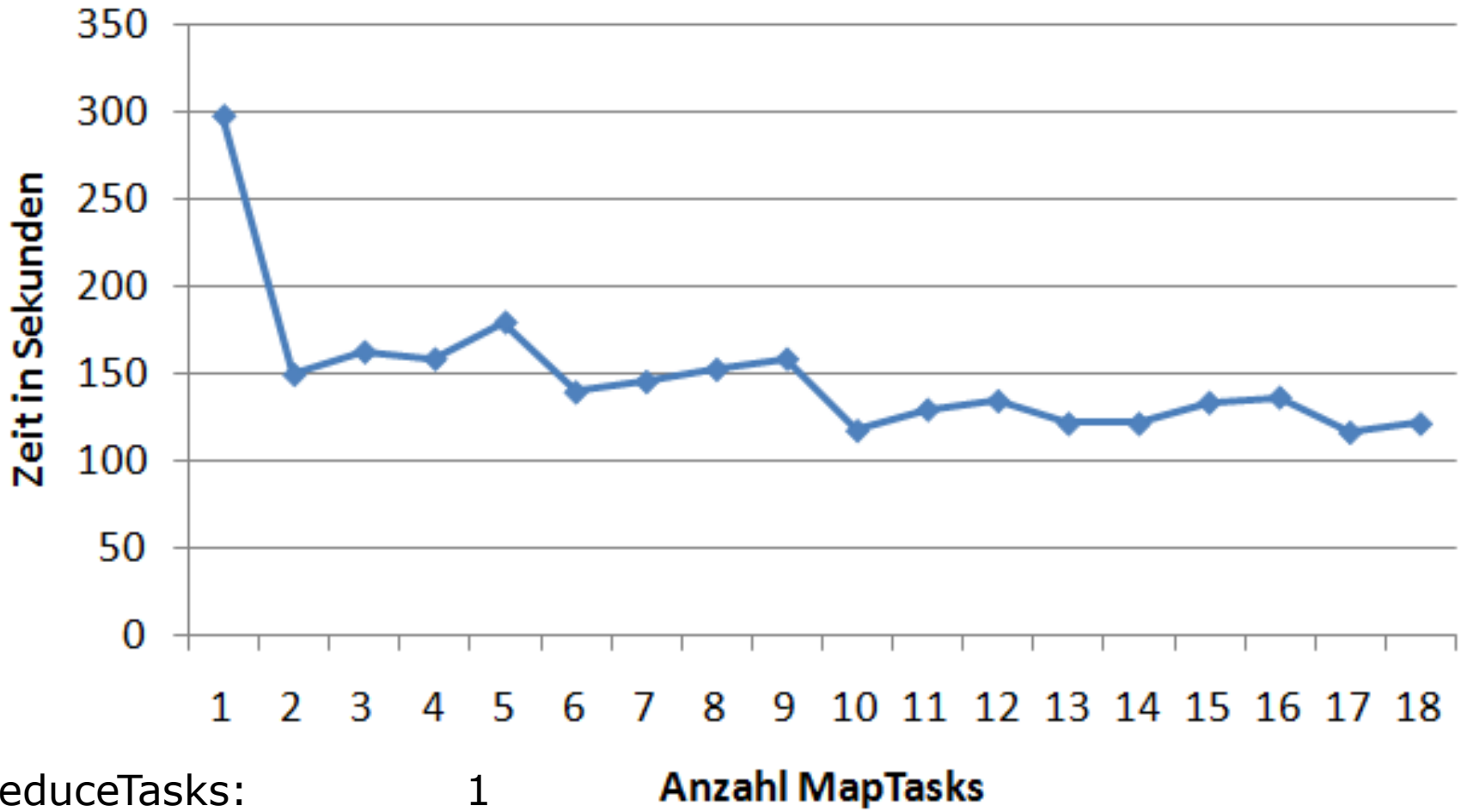
MapTasks: 13
ReduceTasks: 10
Windowgröße: 20

Variable Größe: Window-Größe



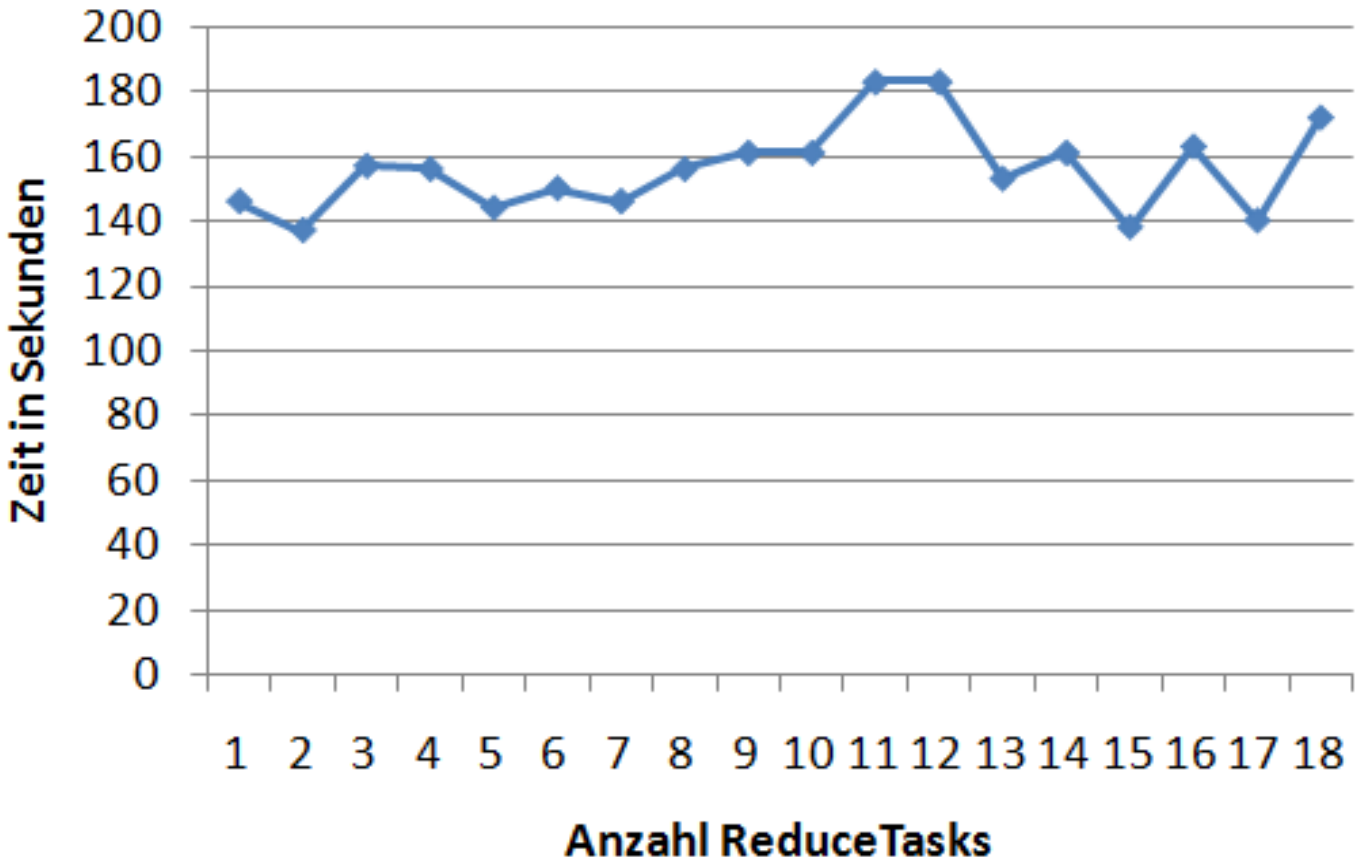
MapTasks: 10
ReduceTasks: 4
Dokumentenanzahl: 125

Variable Größe: MapTasks



ReduceTasks: 1
Dokumentenanzahl: 125
Windowgröße: 20

Variable Größe: ReduceTasks



MapTasks:	13
Windowgröße:	20
Dokumentenanzahl:	125

Dank  keit!