

# Index Creation

Sven Fiergolla

25. März 2018

# Übersicht

Einführung

Hardware constraints

Index Creation

- Blocked sort-based indexing

- Single-pass in-memory indexing

- Distributed indexing

- Dynamic indexing

- andere Indexierungsverfahren

Indexierung mit Solid State Drives

Fazit

Quellen

# Einführung

Effiziente Suche über:

Sammlung von Büchern

das Web

andere große Datenmengen

zu viel für Main Memory!

# Einführung

Effiziente Suche über:

Sammlung von Büchern  
das Web  
andere große Datenmengen

zu viel für Main Memory!

# Einführung

## Typische Systemeigenschaften (stand 2018)

- ▶ *clock rate* 2-4 GHz, 4-8 Kerne
- ▶ *main memory* 4-32 GB
- ▶ *disk space*  $\leq 1$  TB SSD oder  $\geq 1$  TB HDD
  - ▶ HDD (hard disk drive)
    - ▶ *average seek time* zwischen 2 und 20 ms
    - ▶ *transfer time* 150 - 300 MB/s
  - ▶ SSD (solid state disk)
    - ▶ *average seek time* zwischen 0.08 und 0.16 ms
    - ▶ *transfer time* Lesen: 545 MB/s, Schreiben: 525 MB/s

# Einführung

## Typische Systemeigenschaften (stand 2018)

- ▶ *clock rate* 2-4 GHz, 4-8 Kerne
- ▶ *main memory* 4-32 GB
- ▶ *disk space*  $\leq 1$  TB SSD oder  $\geq 1$  TB HDD
  - ▶ HDD (hard disk drive)
    - ▶ *average seek time* zwischen 2 und 20 ms
    - ▶ *transfer time* 150 - 300 MB/s
  - ▶ SSD (solid state disk)
    - ▶ *average seek time* zwischen 0.08 und 0.16 ms
    - ▶ *transfer time* Lesen: 545 MB/s, Schreiben: 525 MB/s

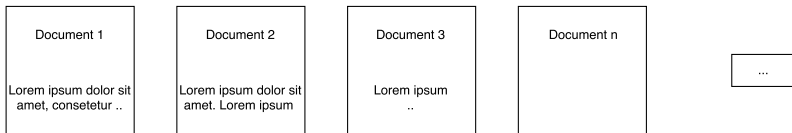
# Einführung

## Typische Systemeigenschaften (stand 2018)

- ▶ *clock rate* 2-4 GHz, 4-8 Kerne
- ▶ *main memory* 4-32 GB
- ▶ *disk space*  $\leq 1$  TB SSD oder  $\geq 1$  TB HDD
  - ▶ HDD (hard disk drive)
    - ▶ *average seek time* zwischen 2 und 20 ms
    - ▶ *transfer time* 150 - 300 MB/s
  - ▶ SSD (solid state disk)
    - ▶ *average seek time* zwischen 0.08 und 0.16 ms
    - ▶ *transfer time* Lesen: 545 MB/s, Schreiben: 525 MB/s

# hardware constraints

## Indizierung einer Sammlung von Daten auf der Festplatte



## Zugriffszeit auf Festplatte als Bottleneck



# Index Creation

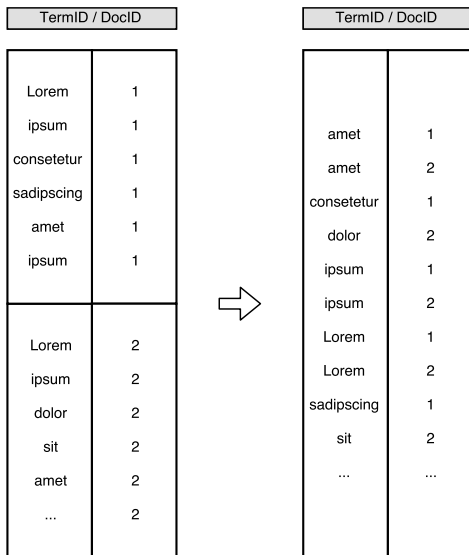
geeignete Datenstruktur um Zugriff auf die Festplatte zu minimieren

TermID / DocID
----------------

Lorem	1
ipsum	1
consetetur	1
sadipscing	1
amet	1
...	1
Lorem	2
ipsum	2
dolor	2
sit	2
amet	2
...	2

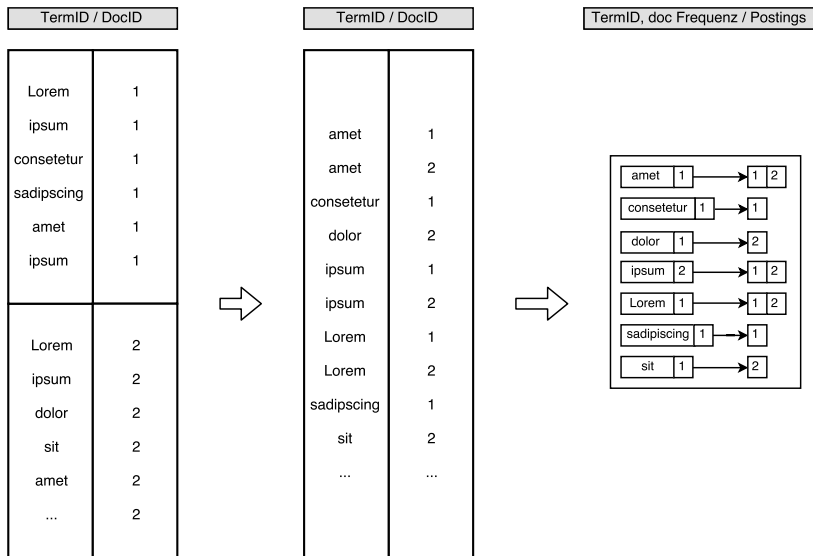
# Index Creation

geeignete Datenstruktur um Zugriff auf die Festplatte zu minimieren



# Index Creation

geeignete Datenstruktur um Zugriff auf die Festplatte zu minimieren



# Index Creation - hardware constraints

in der Regel übersteigt die Datenmenge der Dokumente den Main Memory.

Reuters-RCV1 benötigt ca 0.8 GB für die termID/DocID Paare, bereits die DBLP übersteigt die Dokumentenanzahl, besitzt jedoch weniger Terme.

# Blocked sort-based indexing (BSI)

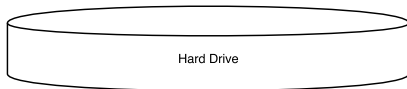
Lösung:

- ▶ Sammlung von Dokumenten in einzelne Blocks unterteilen
- ▶ Index über einzelne Blöcke erstellen
- ▶ Teilindizes mergen

```
n = 0;  
while all documents have not been  
processed do  
    n = n + 1;  
    block = ParseNextBlock();  
    BSBI-INVERT(block);  
    WriteBlockToDisk(block,  $f_n$ );  
end  
MergeBlocks( $f_1, \dots, f_n; f_{merged}$ );  
Algorithm 1: BSI Algorithmus
```

# Blocked sort-based indexing (BSI) - merging Blocks

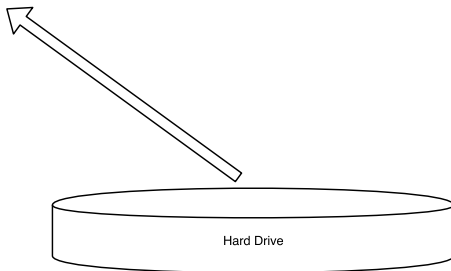
Zusammenführen von  
Postings Lists bei  
block sort-based  
Indexing



# Blocked sort-based indexing (BSI) - merging Blocks

Zusammenführen von  
Postings Lists bei  
block sort-based  
Indexing

amet	d1,d2	consetetur	d3, d4
dolor	d2	Lorem	d3
Lorem	d1	ipsum	d3
ipsum	d2	sit	d4

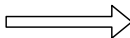


# Blocked sort-based indexing (BSI) - merging Blocks

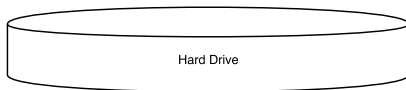
Zusammenführen von  
Postings Lists bei  
block sort-based  
Indexing

amet	d1,d2
dolor	d2
Lorem	d1
ipsum	d2

consetetur	d3, d4
Lorem	d3
ipsum	d3
sit	d4



amet	d1,d2
consetetur	d3,d4
dolor	d2
Lorem	d1, d3
ipsum	d2, d3
sit	d4

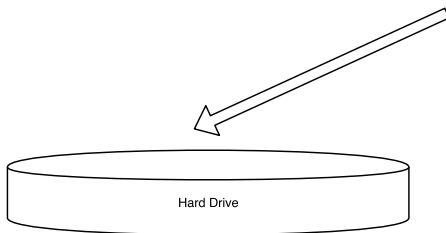




# Blocked sort-based indexing (BSI) - merging Blocks

Zusammenführen von  
Postings Lists bei  
block sort-based  
Indexing

amet	d1,d2
consetetur	d3,d4
dolor	d2
Lorem	d1, d3
ipsum	d2, d3
sit	d4



# Blocked sort-based indexing (BSI) - Fazit

Fazit zu BSBI:

- ▶ Zeitkomplexität:  $\Theta(T \cdot \log(T))$ 
  - ▶ das Sortieren hat die höchste Komplexität
  - ▶ das Parsen und Mergen der Blocks ist jedoch in der Regel am Zeitaufwendigsten
- ▶ Datenstruktur für Mapping zwischen termen und termID's muss in Main Memory liegen
  - ▶ kann für sehr große Datenmengen auch Server überlasten

# Single-pass in-memory indexing (SPIMI)

- ▶ einzelne dictionaries für jeden Block
  - ▶ keine Datenstruktur für das Mapping von termen und termID's
- ▶ kein Sortieren der einzelnen Blöcke
  - ▶ Postings in der Reihenfolge ihres Vorkommens in die Postingslist aufnehmen
  - ▶ PostingsList sollte jedoch sortiert werden, da dann auf Sortieren/Suchen beim mergen der Blöcke verzichtet werden kann

# Single-pass in-memory indexing (SPIMI)

SPIMI-invert(TokenStream)

outputFile = new HashFile();

dictionary = new HashFile();

**while** *free memory available* **do**

    token = next(TokenStream);

**if** *term(token)  $\notin$  dictionary* **then**

        | PostingsList = AddToDictionary(dictionary, term(token));

**else**

        | PostingsList = GetPostingsList(dictionary, term(token));

**end**

**if** *full(PostingsList)* **then**

        | PostingsList = DoublePostingsList(dictionary, term(token));

**end**

    AddToPostingsList(PostingsList, docID(token));

    SortedTerms = SortTerms(dictionary);

    WriteBlockToDisk(SortedTerms, dictionary, OutputFile);

**end**

**Algorithm 2:** SPIMI-invert Algorithmus

# Single-pass in-memory indexing (SPIMI)

Vorteile gegenüber BSI:

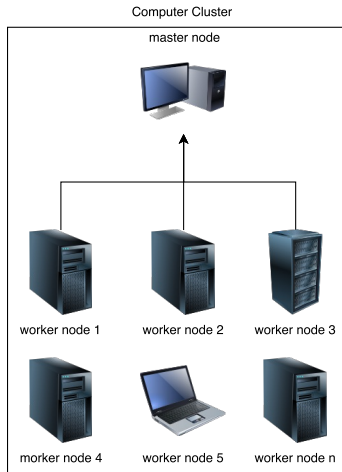
- ▶ kann für beliebig große Datenmengen einen Index erstellen, solange das Festplattenvolumen nicht überstiegen wird
- ▶ einzelne Blöcke können größer sein
  - ▶ Indexerstellung effizienter
- ▶ dictionaries und die erstellte PostingsList kann durch Kompression kompakt auf der Festplatte gespeichert werden

# Distributed indexing

- ▶ manche Sammlungen übersteigen die Leistung eines einzelnen Rechners
  - ▶ beispielsweise das Web
- ▶ um Indizes über solche Sammlungen zu erstellen, muss die Arbeit auf mehrere Rechner verteilt werden

# Distributed indexing

- ▶ manche Sammlungen übersteigen die Leistung eines einzelnen Rechners
  - ▶ beispielsweise das Web
- ▶ um Indizes über solche Sammlungen zu erstellen, muss die Arbeit auf mehrere Rechner verteilt werden

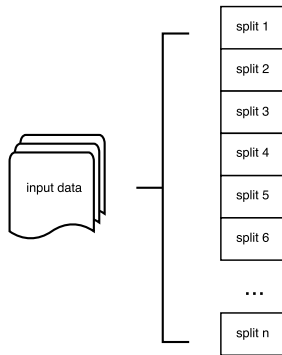


# Distributed indexing - MapReduce

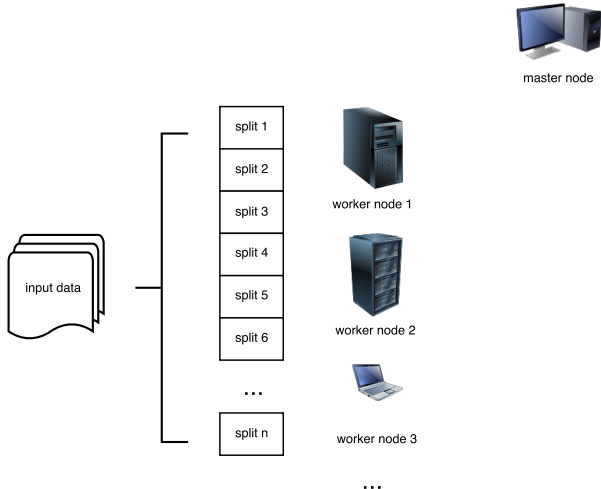




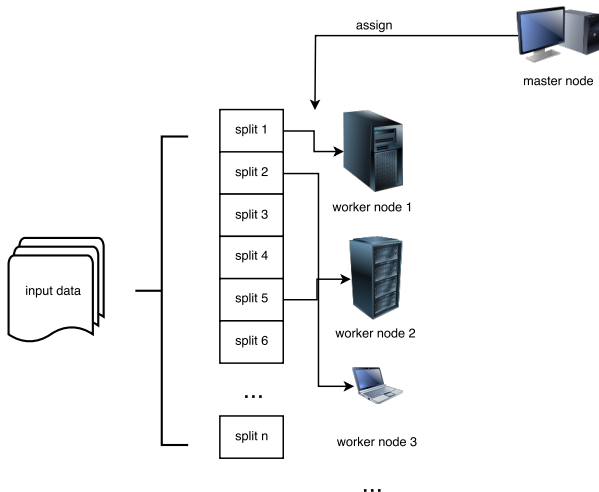
# Distributed indexing - MapReduce



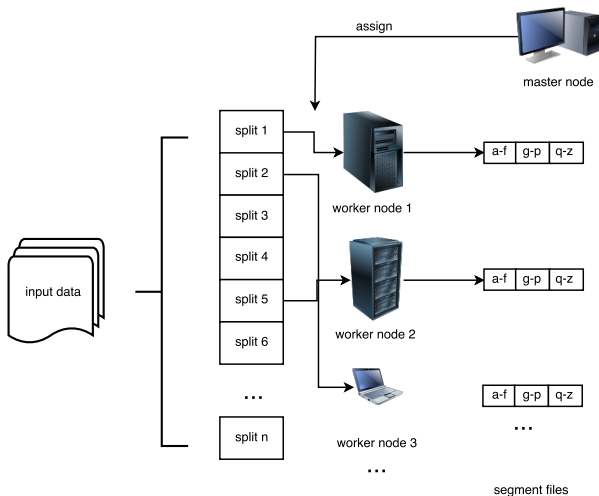
# Distributed indexing - MapReduce



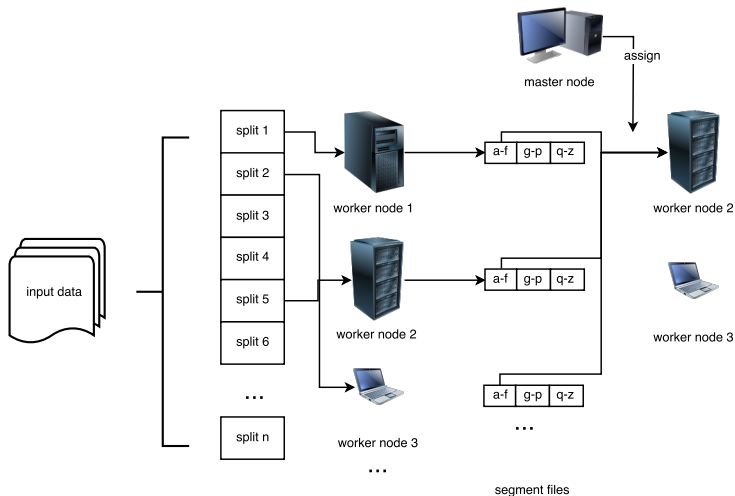
# Distributed indexing - MapReduce



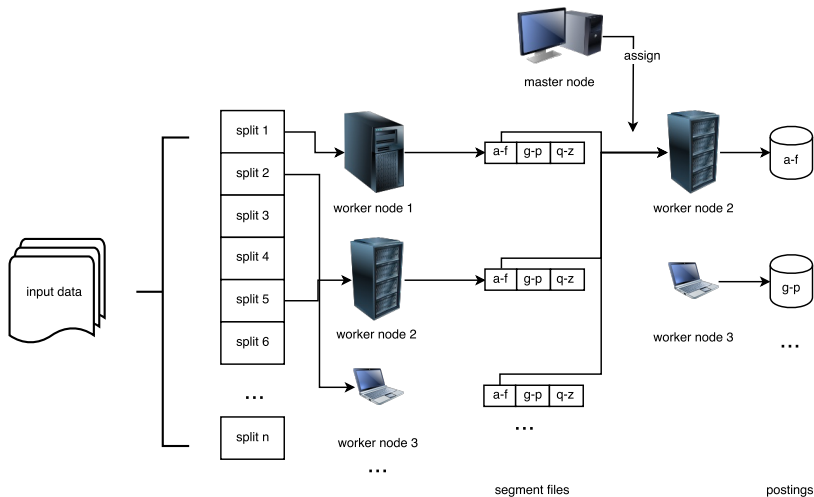
# Distributed indexing - MapReduce



# Distributed indexing - MapReduce



# Distributed indexing - MapReduce



# Distributed indexing - Fazit

todo

# Dynamic indexing

- ▶ viele Sammlungen von Dokumenten ändern sich häufig
  - ▶ beispielsweise werden Webseiten geändert, gelöscht oder neue hinzugefügt
- ▶ daher muss die Indexerstellung über eine solche Sammlung ebenfalls dynamisch sein



# Quellen

- ▶ Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze „Introduction to Information Retrieval“ <sup>1</sup> Cambridge University Press 2008, pp. 1-18 and 67-84.
- ▶ Ian H. Witten, Alistair Moffat, Timothy C. Bell „Managing Gigabytes: Compressing and Indexing Documents and Images“ <sup>2</sup> Morgan Kaufman Publishers 1999, pp. 223-261.
- ▶ Stephen M Ash, King-Ip Lin (University of Memphis) „Optimizing Database Index Performance for Solid State Drives“ <sup>3</sup> IDEAS '14, July 07 - 09 2014, Porro, Portugal.
- ▶ Yinan Li, Bingsheng He, Robin Jun Yang, Qiong Luo, Ke YiTree (Hong Kong University of Science and Technology) „Indexing on Solid State Drives“ <sup>4</sup> The 36th International Conference on Very Large Data Bases, September 13-17, 2010, Singapore.

---

<sup>1</sup><https://nlp.stanford.edu/IR-book/pdf/04const.pdf>

<sup>2</sup>[https://books.google.de/books?id=2F74jyPl48EC&dq=Witten+et+al.+index+1999&lr=&hl=de&source=gbs\\_navlinks\\_s](https://books.google.de/books?id=2F74jyPl48EC&dq=Witten+et+al.+index+1999&lr=&hl=de&source=gbs_navlinks_s)

<sup>3</sup>[http://cs.baylor.edu/~lind/\\_mypaper/ashlin.pdf](http://cs.baylor.edu/~lind/_mypaper/ashlin.pdf)

<sup>4</sup>[http://pages.cs.wisc.edu/~yinan/paper/fdtree\\_pvldb.pdf](http://pages.cs.wisc.edu/~yinan/paper/fdtree_pvldb.pdf)