

Index Construction

Sven Fiergolla

13. April 2018

Übersicht

Einführung

Hardware Constraints

Index Construction

- 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 und „ad hoc retrieval“



Einführung

Effiziente Suche und „ad hoc retrieval“



zu viel für Main Memory!

Einführung

Typische Systemeigenschaften Desktop-PC (stand 2018)

- ▶ *clock rate* 2-4 GHz, 4-8 Kerne
- ▶ *main memory* 4-32 Gb
- ▶ *disk space* ≤ 1 TB SSD oder ≥ 1 TB HDD

Einführung

Typische Systemeigenschaften Desktop-PC (stand 2018)

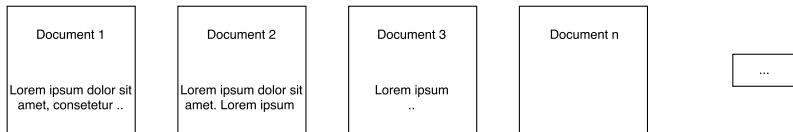
- ▶ *clock rate* 2-4 GHz, 4-8 Kerne
- ▶ *main memory* 4-32 Gb
- ▶ *disk space* ≤ 1 TB SSD oder ≥ 1 TB HDD
 - ▶ HDD (hard disk drive)
 - ▶ *average seek time* zwischen 2 und 10 ms
 - ▶ *transfer* 150 - 300 MB/s

Einführung

Typische Systemeigenschaften Desktop-PC (stand 2018)

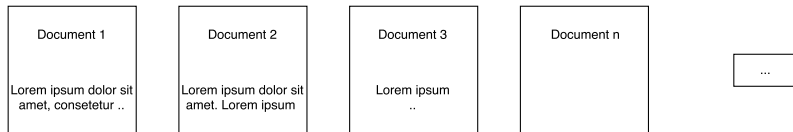
- ▶ *clock rate* 2-4 GHz, 4-8 Kerne
- ▶ *main memory* 4-32 Gb
- ▶ *disk space* ≤ 1 TB SSD oder ≥ 1 TB HDD
 - ▶ HDD (hard disk drive)
 - ▶ *average seek time* zwischen 2 und 10 ms
 - ▶ *transfer* 150 - 300 MB/s
 - ▶ SSD (solid state disk)
 - ▶ *average seek time* zwischen 0.08 und 0.16 ms
 - ▶ *transfer* Lesen: 545 MB/s, Schreiben: 525 MB/s

Hardware Constraints



Zugriffszeit auf Festplatte als Bottleneck

Hardware Constraints



Zugriffszeit auf Festplatte als Bottleneck

→ Erstellung eines Invertieren Index der Daten/Dokumenten auf der Festplatte

Index Creation

geeignete Datenstruktur um Zugriff auf die Festplatte zu minimieren

TermID / DocID

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

Index Creation

geeignete Datenstruktur um Zugriff auf die Festplatte zu minimieren

TermID / DocID

Lorem	1
ipsum	1
consetetur	1
sadipscing	1
amet	1
ipsum	1

Lorem	2
ipsum	2
dolor	2
sit	2
amet	2
...	2



TermID / DocID

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

Index Creation

geeignete Datenstruktur um Zugriff auf die Festplatte zu minimieren

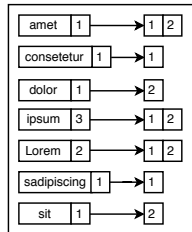
TermID / DocID

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



TermID / DocID

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

TermID, doc Frequenz / Postings

Index Creation - Hardware Constraints

Reuters-RCV1 Modell Sammlung besitzt 100.000.000 Terme...

Sortieren dieser Terme von einer Festplatte:

Index Creation - Hardware Constraints

Reuters-RCV1 Modell Sammlung besitzt 100.000.000 Terme...

Sortieren dieser Terme von einer Festplatte:

- ▶ Annahme
 - ▶ $T \cdot \log_2(T)$ Vergleiche
 - ▶ 2 Zugriffe auf die Hard Drive zum Vergleichen
 - ▶ average seek time 5 ms

Index Creation - Hardware Constraints

Reuters-RCV1 Modell Sammlung besitzt 100.000.000 Terme...

Sortieren dieser Terme von einer Festplatte:

- ▶ Annahme
 - ▶ $T \cdot \log_2(T)$ Vergleiche
 - ▶ 2 Zugriffe auf die Hard Drive zum Vergleichen
 - ▶ average seek time 5 ms
- ▶ $(100.000.000 \cdot \log_2(100.000.000)) \cdot 2 \cdot (5 \cdot 10^{-3})$ Sekunden
- ▶ $2.6575424759... \cdot 10^7$ Sekunden
- ▶ 307.59 Tage

Index Creation - Hardware Constraints

Reuters-RCV1 Modell Sammlung besitzt 100.000.000 Terme...

Sortieren dieser Terme von einer Festplatte:

- ▶ Annahme
 - ▶ $T \cdot \log_2(T)$ Vergleiche
 - ▶ 2 Zugriffe auf die Hard Drive zum Vergleichen
 - ▶ average seek time 5 ms
- ▶ $(100.000.000 \cdot \log_2(100.000.000)) \cdot 2 \cdot (5 \cdot 10^{-3})$ Sekunden
- ▶ $2.6575424759... \cdot 10^7$ Sekunden
- ▶ 307.59 Tage

Mit einer schnellen SSD mit 0,1 ms Zugriffszeit:

- ▶ $(100.000.000 \cdot \log_2(100.000.000)) \cdot 2 \cdot (1 \cdot 10^{-4})$ Sekunden
- ▶ 6.15 Tage

Blocked sort-based indexing (BSBI)

Lösung:

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

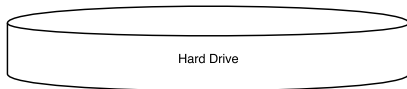
Blocked sort-based indexing (BSBI)

```
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 (BSBI) - merging Blocks

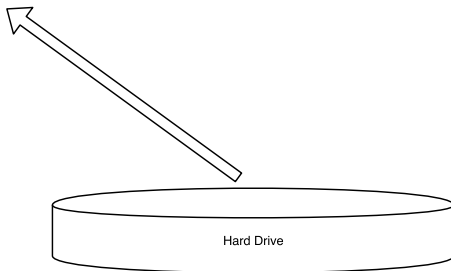
Zusammenführen von
Postings Lists bei
block sort-based
Indexing



Blocked sort-based indexing (BSBI) - 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

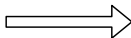


Blocked sort-based indexing (BSBI) - merging Blocks

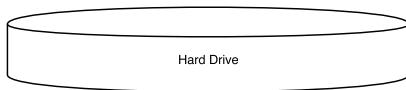
Zusammenführen von
Postings Lists bei
block sort-based
Indexing

amet	d1,d2
dolor	d2
Lorem	d1
ipsum	d2

consetetur	d3, d4
Lorem	d3
ipsum	d3
sit	d4



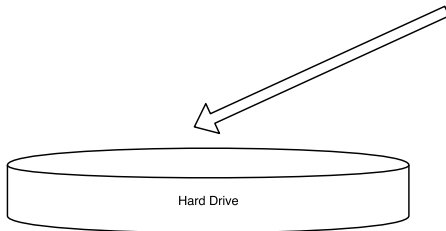
amet	d1,d2
consetetur	d3,d4
dolor	d2
Lorem	d1, d3
ipsum	d2, d3
sit	d4



Blocked sort-based indexing (BSBI) - 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 (BSBI) - Fazit

Reuters-RCV1 Modell Sammlung besitzt 100.000.000 Terme...
Indizieren mit BSBI:

- ▶ Annahme:
 - ▶ Sortieren von (term, doc) - Paaren, je 12 byte
 - ▶ Aufteilung in 64 Blocks, 1600000 Paare pro Block

Blocked sort-based indexing (BSBI) - Fazit

Reuters-RCV1 Modell Sammlung besitzt 100.000.000 Terme...
Indizieren mit BSBI:

- ▶ Annahme:
 - ▶ Sortieren von (term, doc) - Paaren, je 12 byte
 - ▶ Aufteilung in 64 Blocks, 1600000 Paare pro Block
- ▶ $64 \text{ blocks} \cdot 1600000 \text{ Paare} \cdot 12 \text{ bytes} = 1228800000 \text{ bytes}$
- ▶ $N \cdot \log_2(N)$ - Vergleiche = 558457725093
- ▶ 2 Zugriffe pro Vergleich in Main Memory (Zugriffszeit von 10^{-6} sec)
- ▶ 55845,77 Sekunden
- ▶ 15,5 Stunden

Blocked sort-based indexing (BSBI) - 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

Blocked sort-based indexing (BSBI) - 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)

Lösung:

- ▶ einzelne dictionaries für jeden Block
 - ▶ keine Datenstruktur für das Mapping von termen und termID's
- ▶ kein Sortieren der einzelnen Terme eines Blocks
 - ▶ Postings in der Reihenfolge ihres Vorkommens in die PostingsList aufnehmen

Single-pass in-memory indexing (SPIMI)

SPIMI-invert(TokenStream)

outputFile = new File();

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));

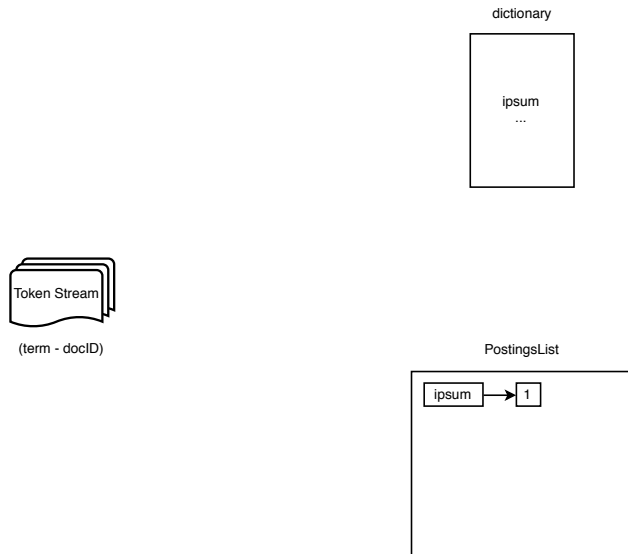
 SortedTerms = SortTerms(dictionary);

 WriteBlockToDisk(SortedTerms, dictionary, OutputFile);

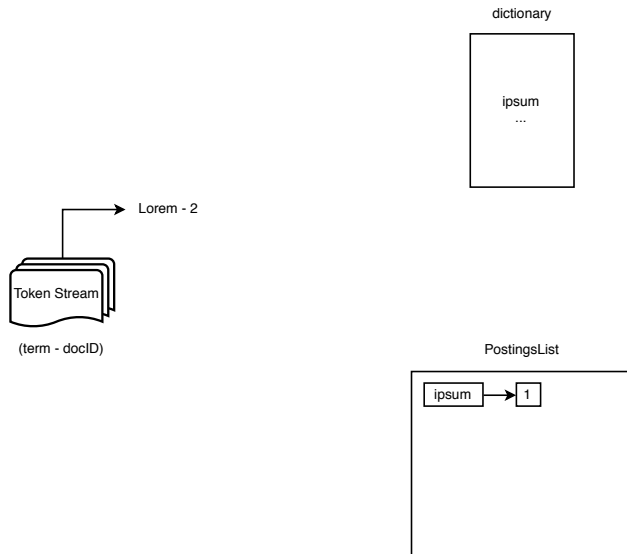
end

Algorithm 2: SPIMI-invert Algorithmus

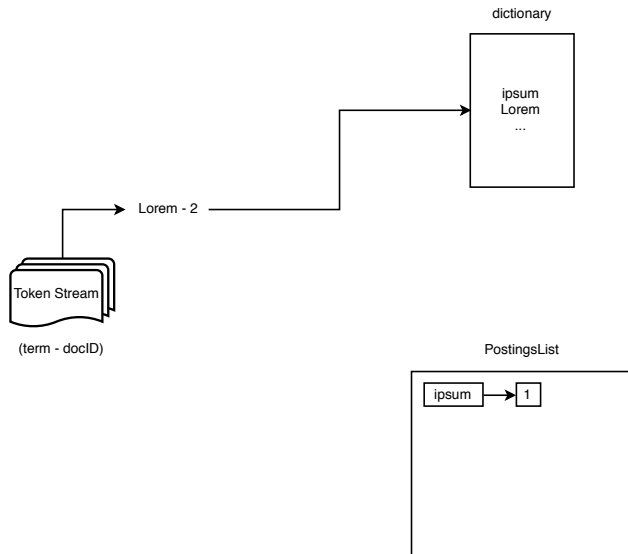
Single-pass in-memory indexing (SPIMI)



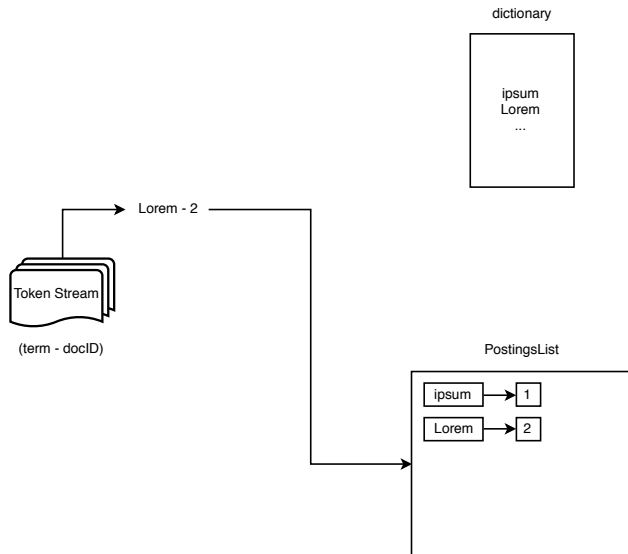
Single-pass in-memory indexing (SPIMI)



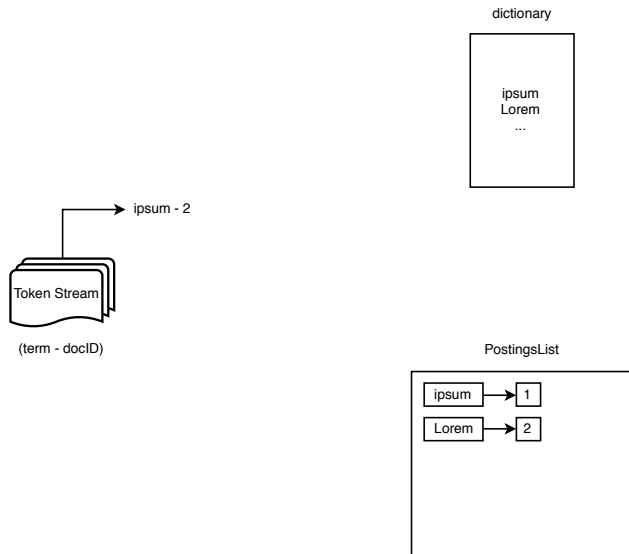
Single-pass in-memory indexing (SPIMI)



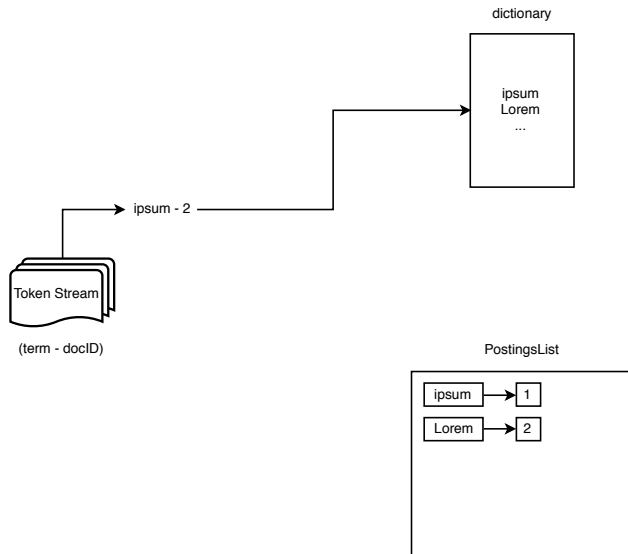
Single-pass in-memory indexing (SPIMI)



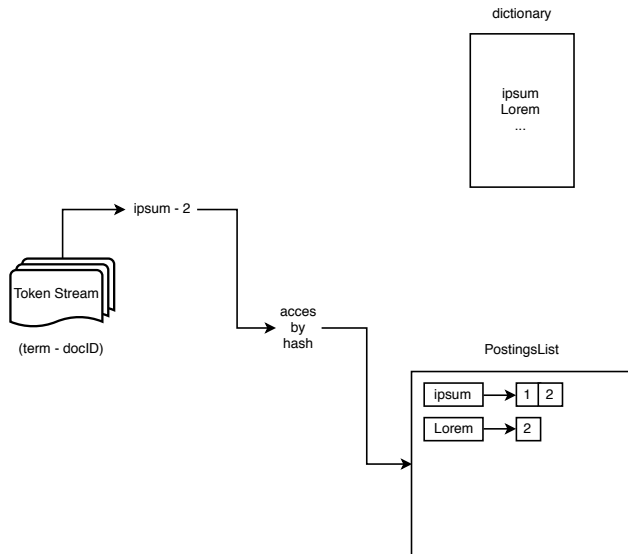
Single-pass in-memory indexing (SPIMI)



Single-pass in-memory indexing (SPIMI)



Single-pass in-memory indexing (SPIMI)



Single-pass in-memory indexing (SPIMI)

Vorteile gegenüber BSI:

- ▶ kann für beliebig große Datenmengen einen Index erstellen

Single-pass in-memory indexing (SPIMI)

Vorteile gegenüber BSI:

- ▶ kann für beliebig große Datenmengen einen Index erstellen
- ▶ einzelne Blöcke können größer sein
 - ▶ Indexerstellung effizienter

Single-pass in-memory indexing (SPIMI)

Vorteile gegenüber BSI:

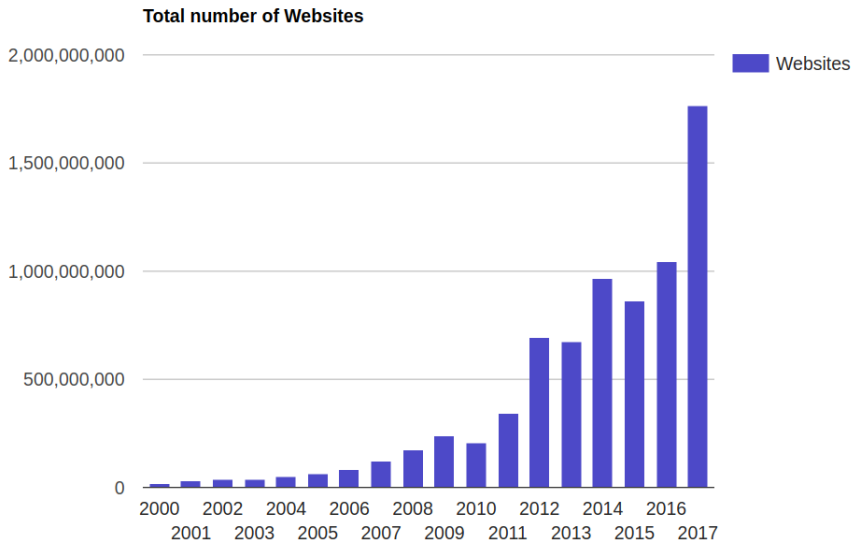
- ▶ kann für beliebig große Datenmengen einen Index erstellen
- ▶ einzelne Blöcke können größer sein
 - ▶ Indexerstellung effizienter
- ▶ dictionaries und die erstellte PostingsList kann sehr kompakt gespeichert werden

Single-pass in-memory indexing (SPIMI)

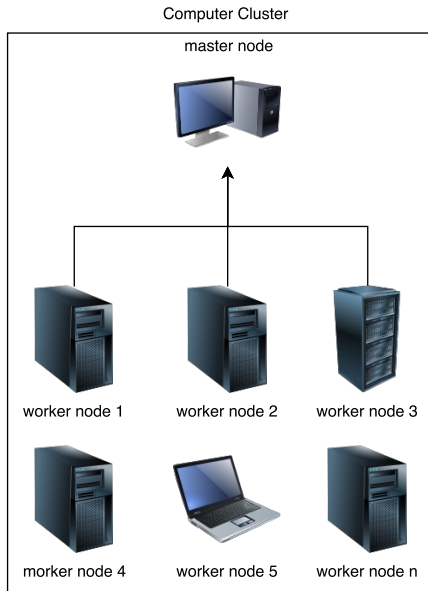
Vorteile gegenüber BSI:

- ▶ kann für beliebig große Datenmengen einen Index erstellen
- ▶ einzelne Blöcke können größer sein
 - ▶ Indexerstellung effizienter
- ▶ dictionaries und die erstellte PostingsList kann sehr kompakt gespeichert werden
- ▶ Zeitkomplexität: $\Theta(T)$, kein Sortieren von TermID-DocID Paaren, alle Operationen linear

Distributed indexing



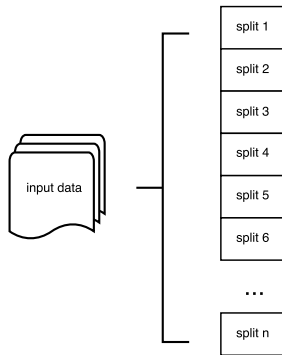
Distributed indexing



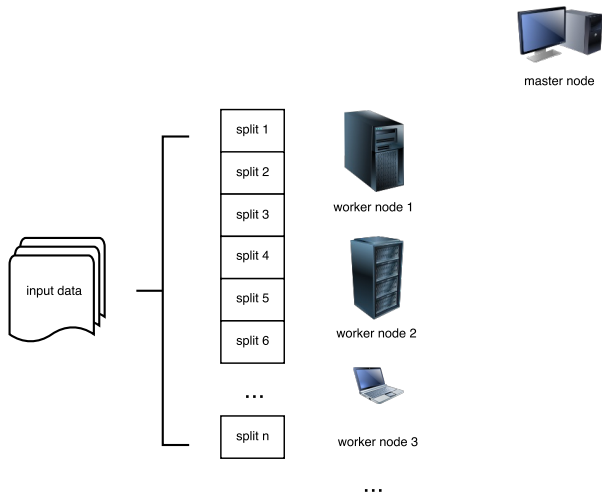
Distributed indexing - MapReduce



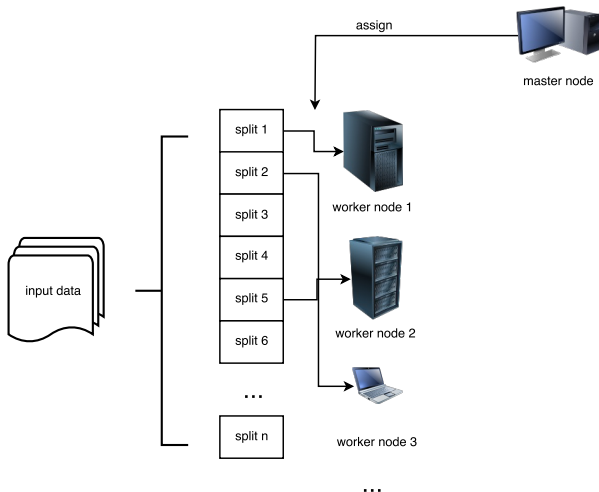
Distributed indexing - MapReduce



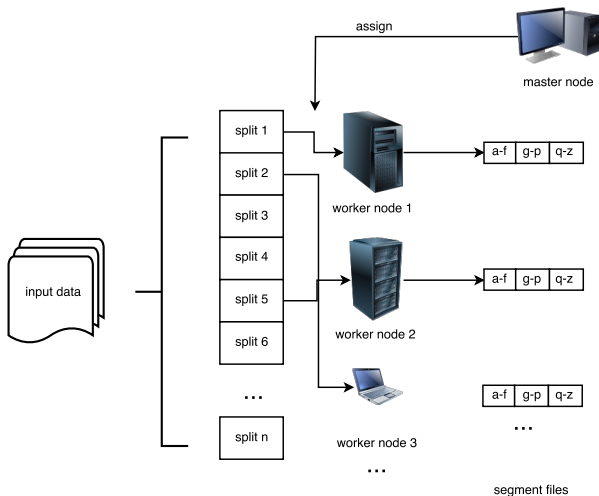
Distributed indexing - MapReduce



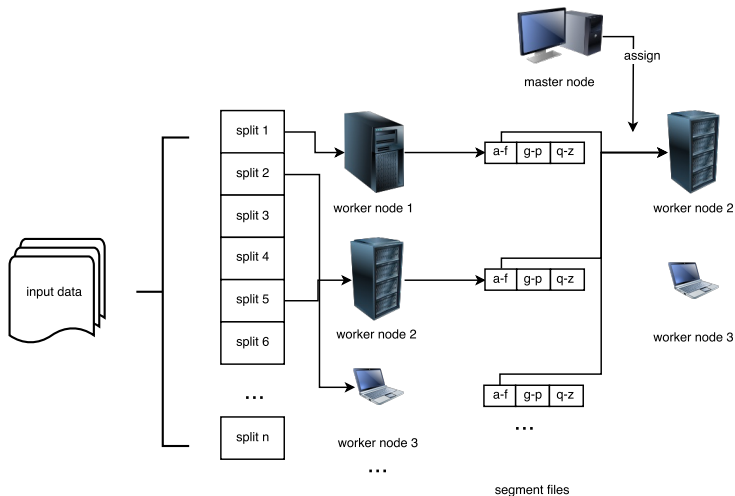
Distributed indexing - MapReduce



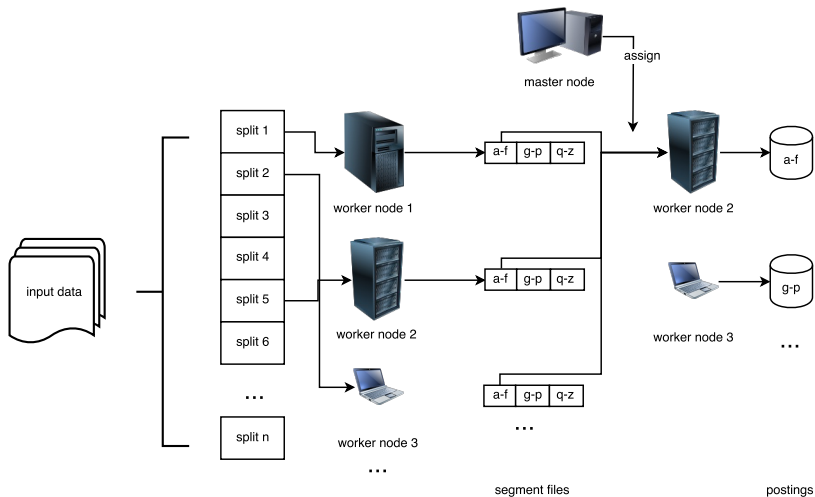
Distributed indexing - MapReduce



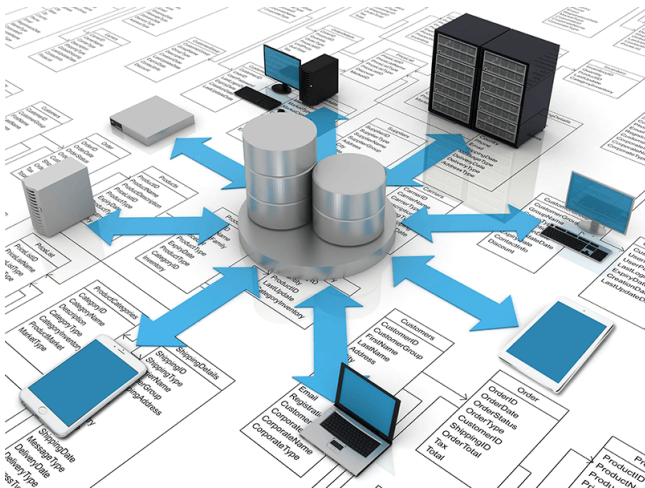
Distributed indexing - MapReduce



Distributed indexing - MapReduce



Dynamic indexing



- ▶ viele Sammlungen von Dokumenten ändern sich häufig
 - ▶ Webseiten werden geändert, gelöscht oder neue hinzugefügt..

Dynamic indexing

Lösung:

- ▶ Index periodisch neu erstellen
 - ▶ akzeptabel wenn Änderungen nicht sehr groß
 - ▶ wenn Änderungen nicht sofort sichtbar sein müssen

Dynamic indexing

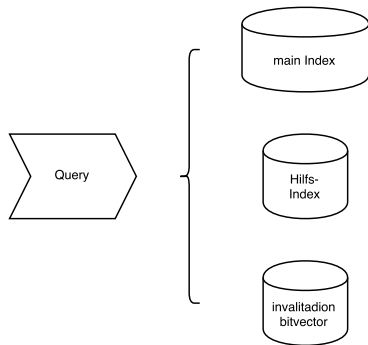
Lösung:

- ▶ Index periodisch neu erstellen
 - ▶ akzeptabel wenn Änderungen nicht sehr groß
 - ▶ wenn Änderungen nicht sofort sichtbar sein müssen
- ▶ Hauptindex behalten und neue Dokumente in einem Hilfsindex speichern
 - ▶ beide Indizes regelmäßig mergen

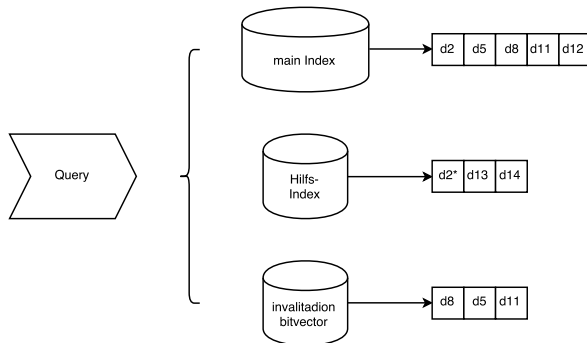
Dynamic indexing



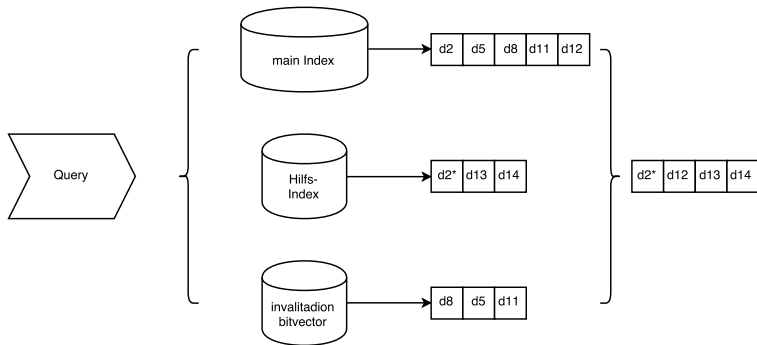
Dynamic indexing



Dynamic indexing



Dynamic indexing



andere Indexierungsverfahren

Alternative Indexverfahren mit unterschiedlichen Vor/Nachteile:

andere Indexierungsverfahren

Alternative Indexverfahren mit unterschiedlichen Vor/Nachteile:

- ▶ ranked retrieval systems

andere Indexierungsverfahren

Alternative Indexverfahren mit unterschiedlichen Vor/Nachteile:

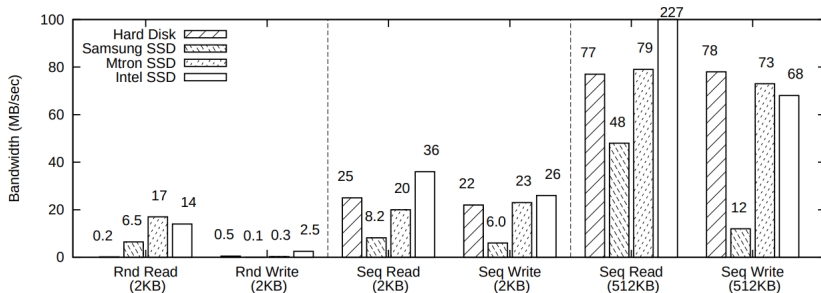
- ▶ ranked retrieval systems
- ▶ zugriffsbeschränkte Indizes

andere Indexierungsverfahren

Alternative Indexverfahren mit unterschiedlichen Vor/Nachteile:

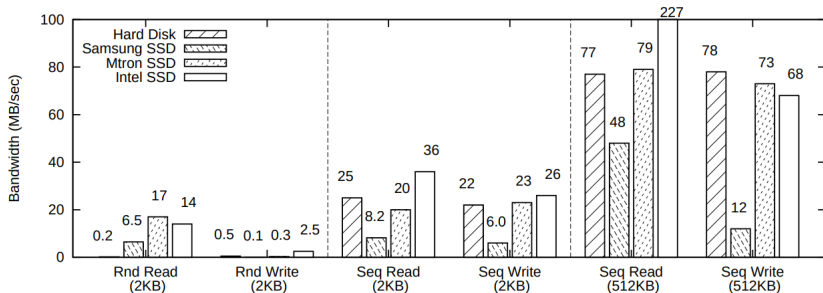
- ▶ ranked retrieval systems
- ▶ zugriffsbeschränkte Indizes
- ▶ „in situ“-Indexerstellung

Indexierung mit Solid State Drives - hardware constraints



- schneller „random read“-Zugriff

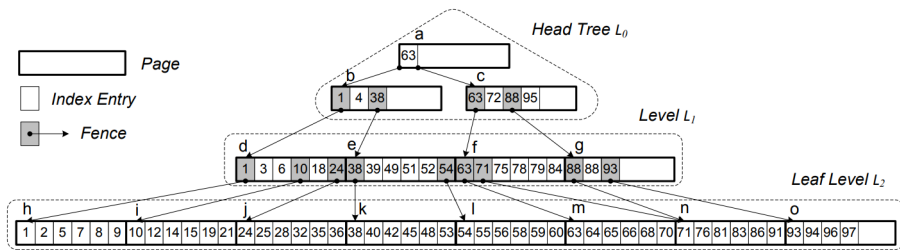
Indexierung mit Solid State Drives - hardware constraints



- ▶ schneller „random read“-Zugriff
- ▶ „random write“-Zugriff deutlich langsamer
 - ▶ wegen dem „erase before write“-Mechanismus

Indexierung mit Solid State Drives

Datenstruktur FD-Baum



Indexierung mit Solid State Drives - Fazit

- ▶ abweichende Hardwareeigenschaften → andere Algorithmen
- ▶ FD-Baumstruktur eliminiert häufige kleine „random read“-Zugriffe

Fazit

- ▶ unterschiedliche Anforderungen → unterschiedliche Indizierungsverfahren
 - ▶ BSBI und SPIMI auf einzelnen Rechnern
 - ▶ MapReduce auf Computer Clustern
 - ▶ alternative Datenstrukturen für neue Hardware

Quellen

- ▶ Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze „Introduction to Information Retrieval“ ¹ 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“ ² Morgan Kaufman Publishers 1999, pp. 223-261.
- ▶ Yinan Li, Bingsheng He, Robin Jun Yang, Qiong Luo, Ke YiTree (Hong Kong University of Science and Technology) „Indexing on Solid State Drives“ ³ The 36th International Conference on Very Large Data Bases, September 13-17, 2010, Singapore.

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

²https://books.google.de/books?id=2F74jyPl48EC&dq=Witten+et+al.+index+1999&lr=&hl=de&source=gbs_navlinks_s

³http://pages.cs.wisc.edu/~yinan/paper/fdtree_pvlodb.pdf