

Tìm kiếm và trình diễn thông tin

Các phương pháp xây dựng chỉ mục ngược
Index Construction

Nội dung chính

Các giải thuật xây dựng chỉ mục ngược:

- a. BSBI
- b. SPIMI
- c. MapReduce
- d. Quản lý bộ dữ liệu động

Xây dựng chỉ mục ngược

- Giải thuật xây dựng chỉ mục trong bộ nhớ chính chỉ phù hợp với những bộ dữ liệu nhỏ.
- Đối với những bộ dữ liệu lớn, kích thước chỉ mục có thể vượt quá dung lượng của bộ nhớ chính:
- Cần sử dụng ổ đĩa cứng, hoặc hơn nữa là phân tán chỉ mục trên nhiều máy.

term	docID		term	docID
I	1		ambitious	2
did	1		be	2
enact	1		brutus	1
julius	1		brutus	2
caesar	1		capitol	1
I	1		caesar	1
was	1		caesar	2
killed	1		caesar	2
i'	1		did	1
the	1		enact	1
capitol	1		hath	1
brutus	1		I	1
killed	1		I	1
me	1	⇒	i'	1
so	2		it	2
let	2		julius	1
it	2		killed	1
be	2		killed	1
with	2		let	2
caesar	2		me	1
the	2		noble	2
noble	2		so	2
brutus	2		the	1
hath	2		the	2
told	2		told	2
you	2		you	2
caesar	2		was	1
was	2		was	2
ambitious	2		with	2

Giải thuật BSBI

- Xây dựng chỉ mục dựa trên sắp xếp theo khối: BSBI: Blocked Sort-Based Indexing
- Các thao tác cơ bản:
 - Đọc dữ liệu, tách từ và sinh thẻ định vị;
 - Tích lũy thẻ định vị thành khối kích thước xác định;
 - Sinh chỉ mục cho khối và lưu tạm thời trên ổ đĩa;
 - Hợp nhất các chỉ mục khối thành chỉ mục bộ dữ liệu.

BSBINDEXCONSTRUCTION()

```
1   $n \leftarrow 0$ 
2  while (all documents have not been processed)
3  do  $n \leftarrow n + 1$ 
4       $block \leftarrow \text{PARSENEXTBLOCK}()$ 
5       $\text{BSBI-INVERT}(block)$ 
6       $\text{WRITEBLOCKTODISK}(block, f_n)$ 
7   $\text{MERGEBLOCKS}(f_1, \dots, f_n; f_{\text{merged}})$ 
```

Hợp nhất chỉ mục (2)

Phương pháp hợp nhất đồng thời:

1. Sử dụng bộ nhớ đệm cho từng khối;
2. Đọc dữ liệu vào bộ nhớ đệm từ các tệp tạm thời;
3. Tại mỗi bước lựa chọn từ nhỏ nhất và hợp nhất tất cả các danh sách thể định vị, nạp thêm dữ liệu vào bộ nhớ đệm nếu cần;
4. Lưu kết quả hợp nhất lên đĩa.

Hợp nhất chỉ mục

postings
to be merged

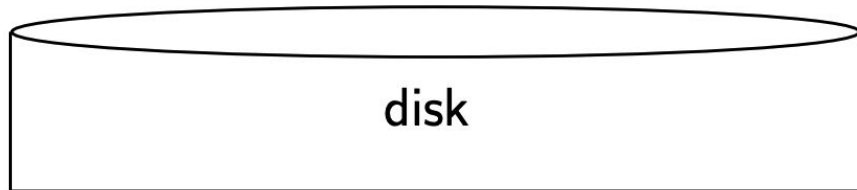
Block 1	
brutus	d3
caesar	d4
noble	d3
with	d4

Block 2	
brutus	d2
caesar	d1
julius	d1
killed	d2



brutus	d2
brutus	d3
caesar	d1
caesar	d4
julius	d1
killed	d2
noble	d3
with	d4

merged
postings



Các hạn chế của BSBI

- Nếu sử dụng <termId,docId>:
 - Cần lưu toàn bộ từ điển trong bộ nhớ chính;
 - Để chuyển đổi từ thành mã từ.
- Nếu sử dụng <term, docId>:
 - Không cần lưu toàn bộ từ điển, nhưng...
 - ...Dữ liệu tạm thời sẽ lớn, làm giảm tốc độ xây dựng chỉ mục;

Giải thuật SPIMI

- Xây dựng chỉ mục một lượt trong bộ nhớ : SPIMI: Single-pass in-memory indexing;
- Sử dụng từ điển riêng biệt cho từng khối;
 - Không cần quản lý mã từ trong quá trình xây dựng chỉ mục khối;
 - Kích thước khối trong bộ nhớ lớn hơn so với BSBI.
- Thêm trực tiếp thẻ định vị vào danh sách
 - Không cần thực hiện sắp xếp danh sách thẻ định vị

Giải thuật SPIMI (2)

SPIMI-INVERT(*token_stream*)

```
1  output_file ← NEWFILE()
2  dictionary ← NEWHASH()
3  while (free memory available)
4  do token ← next(token_stream)
5      if term(token) ∉ dictionary
6          then postings_list ← ADDTODICTIONARY(dictionary, term(token))
7          else postings_list ← GETPOSTINGSLIST(dictionary, term(token))
8          if full(postings_list)
9              then postings_list ← DOUBLEPOSTINGSLIST(dictionary, term(token))
10         ADDTOPOSTINGSLIST(postings_list, docID(token))
11 sorted_terms ← SORTTERMS(dictionary)
12 WRITEBLOCKTODISK(sorted_terms, dictionary, output_file)
13 return output_file
```

- Xây dựng chỉ mục ngược đầy đủ cho mỗi khối
- Sắp xếp từ điển cục bộ
- Ghi ra đĩa
- Hợp nhất khối

MapReduce

MapReduce (Dean and Ghemawat 2004) là một kiến trúc tính toán phân tán

Pha Map: Đọc dữ liệu

- Nút điều khiển thực hiện phân chia công việc đọc dữ liệu:
 - Chia bộ dữ liệu thành nhiều khối và phân chia cho các nút đọc dữ liệu;
 - Nút đọc xử lý tuần tự từng văn bản và sinh thẻ định vị, vd, theo dạng cặp <từ, mã văn bản>
 - Sau đó phân chia thẻ định vị vào j phân đoạn: Mỗi phân đoạn ứng với một khoảng từ (ví dụ, $j = 3$, ứng với ba khoảng: a-f, g-p, q-z).

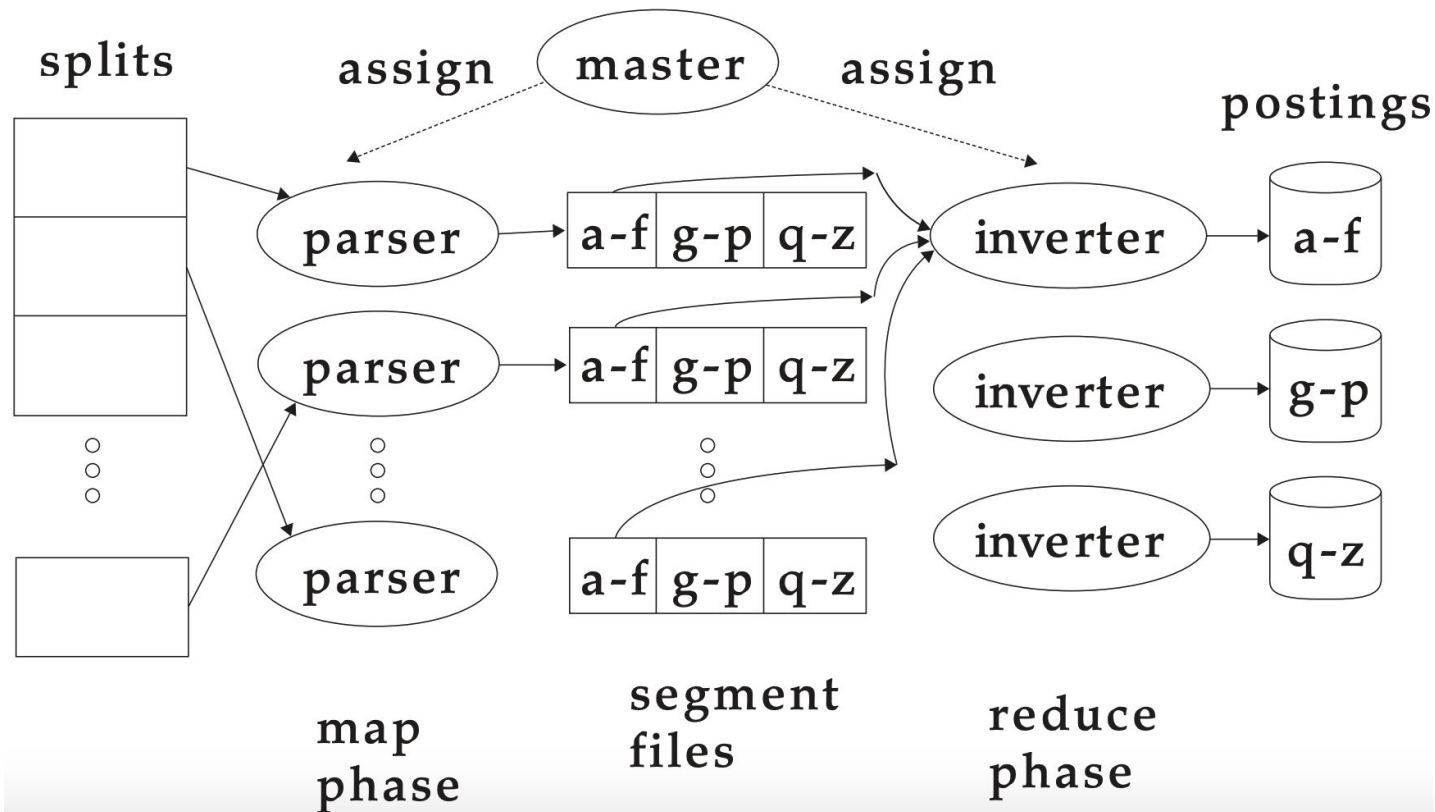
Pha Reduce: Nghịch đảo

- Số lượng nút nghịch đảo bằng số lượng phân đoạn j ;
- Nhiệm vụ của nút nghịch đảo:
 - Tiếp nhận tất cả các phân đoạn tương ứng thu được sau khi đọc dữ liệu;
 - Sắp xếp và thiết lập danh sách thẻ định vị.

Các phương pháp phân tán chỉ mục

- Phân tán theo từ: mỗi máy xử lý một khoảng từ
- Phân tán theo văn bản: mỗi máy xử lý một tập con của bộ văn bản văn bản
- Hầu hết công cụ tìm kiếm sử dụng chỉ mục phân đoạn theo văn bản.
 - Có thể chuyển đổi chỉ mục phân tán theo từ thành chỉ mục phân tán theo văn bản.

Pha Reduce: Nghịch đảo



Ví dụ xây dựng chỉ mục theo MapReduce

Map:

d1 : C ca, C ce.

d2 : C d.

→ <C,d1>, <ca,d1>, <C,d1>, <ce, d1>, <C, d2>, <d,d2>

Reduce:

(<C,(d1,d2,d1)>, <d,(d2)>, <ca,(d1)>, <ce,(d1)>) →

(<C,(d1:2,d2:1)>, <d,(d2:1)>, <ca,(d1:1)>, <ce, (d1:1)>)

Quản lý bộ dữ liệu động

- Bộ dữ liệu tĩnh: Là bộ dữ liệu không thay đổi hoặc rất ít khi thay đổi.
 - Đối với bộ dữ liệu tĩnh khi có sự thay đổi chỉ cần thực hiện xây dựng lại chỉ mục:
 - Cập nhật lại bộ từ vựng;
 - Cập nhật lại danh sách thẻ định vị.
- Đối với những bộ dữ liệu thay đổi thường xuyên cần phải có giải pháp quản lý hiệu quả hơn.

Các thao tác quản lý chỉ mục động

- Xóa (delete). Thường chỉ thực hiện xóa ảo vì xóa thực đòi hỏi xây dựng lại chỉ mục.
 - Đánh dấu văn bản muốn xóa;
 - Lọc những văn bản đã đánh dấu khỏi danh sách kết quả.
- Cập nhật (update). Thường được thực hiện thông qua hai thao tác: xóa và thêm mới.
- Thêm mới (insert). Có nhiều phương pháp với độ phức tạp khác nhau để cho phép thêm mới văn bản vào chỉ mục.

Chỉ mục chính phụ

- Sử dụng chỉ mục chính và chỉ mục phụ:
 - Thêm văn bản mới vào chỉ mục phụ;
- Thực hiện truy vấn trên cả hai chỉ mục và tổng hợp kết quả.
- Định kỳ xây dựng lại toàn bộ chỉ mục.

Nhược điểm của giải pháp chỉ mục chính phụ

- Nếu bộ dữ liệu thay đổi rất thường xuyên, thì kích thước chỉ mục phụ có thể tăng nhanh.
 - Cần nhiều thời gian để hợp nhất chỉ mục chính và chỉ mục phụ;
 - Giải pháp: Sử dụng nhiều chỉ mục có thể giảm thời gian hợp nhất chỉ mục (*tuy nhiên thực hiện truy vấn sẽ phức tạp hơn).

Hợp nhất chỉ mục với độ phức tạp Logarithm

- Sử dụng nhiều cấp chỉ mục:

- Lưu chỉ mục nhỏ nhất (Z_0) trong bộ nhớ;
- Khi Z_0 trở nên quá lớn, sẽ ghi Z_0 lên đĩa và thực hiện hợp nhất với những chỉ mục đã tồn tại.
 - Các chỉ mục trên đĩa ký hiệu là I_0, I_1, \dots, I_n
 - Các chỉ mục tạm trên đĩa ký hiệu là Z_1, Z_2, \dots, Z_n

LMERGEADDTOKEN(*indexes*, Z_0 , *token*)

```
1   $Z_0 \leftarrow \text{MERGE}(Z_0, \{\text{token}\})$ 
2  if  $|Z_0| = n$ 
3    then for  $i \leftarrow 0$  to  $\infty$ 
4      do if  $I_i \in \text{indexes}$ 
5        then  $Z_{i+1} \leftarrow \text{MERGE}(I_i, Z_i)$ 
6          ( $Z_{i+1}$  is a temporary index on disk.)
7           $\text{indexes} \leftarrow \text{indexes} - \{I_i\}$ 
8        else  $I_i \leftarrow Z_i$  ( $Z_i$  becomes the permanent index  $I_i$ .)
9           $\text{indexes} \leftarrow \text{indexes} \cup \{I_i\}$ 
10         BREAK
11      $Z_0 \leftarrow \emptyset$ 
```

LOGARITHMICMERGE()

```
1   $Z_0 \leftarrow \emptyset$  ( $Z_0$  is the in-memory index.)
2   $\text{indexes} \leftarrow \emptyset$ 
3  while true
4  do LMERGEADDTOKEN(indexes,  $Z_0$ , GETNEXTTOKEN())
```

Bài tập 12.1

Cho $n = 2$, và $1 \leq T \leq 30$, hãy thực hiện giải thuật **LogarithmMerge** và vẽ bảng thể hiện ở mỗi thời điểm khi $T = 2 * k$ từ đã được xử lý ($1 \leq k \leq 15$), các chỉ mục nào trong số bốn chỉ mục $I_0 \dots I_4$ được sử dụng.

```
LMERGEADDTOKEN(indexes,  $Z_0$ , token)
1   $Z_0 \leftarrow \text{MERGE}(Z_0, \{token\})$ 
2  if  $|Z_0| = n$ 
3      then for  $i \leftarrow 0$  to  $\infty$ 
4          do if  $I_i \in \text{indexes}$ 
5              then  $Z_{i+1} \leftarrow \text{MERGE}(I_i, Z_i)$ 
6                  ( $Z_{i+1}$  is a temporary index on disk.)
7                   $\text{indexes} \leftarrow \text{indexes} - \{I_i\}$ 
8              else  $I_i \leftarrow Z_i$  ( $Z_i$  becomes the permanent index  $I_i$ .)
9                   $\text{indexes} \leftarrow \text{indexes} \cup \{I_i\}$ 
10                 BREAK
11      $Z_0 \leftarrow \emptyset$ 
```

```
LOGARITHMICMERGE()
1   $Z_0 \leftarrow \emptyset$  ( $Z_0$  is the in-memory index.)
2   $\text{indexes} \leftarrow \emptyset$ 
3  while true
4  do LMERGEADDTOKEN(indexes,  $Z_0$ , GETNEXTTOKEN())
```