بازیابی اطلاعات

دکتر امین گلزاری اسکوئی

a.golzari@azaruniv.ac.ir

a.golzari@tabrizu.ac.ir

https://github.com/Amin-Golzari-Oskouei

دانشگاه صنعتی ارومیه

پاییز ۱۴۰۲

فصل ۴

ساخت شاخص

مطالب این فصل

- BSBI(simple) و SPIMI(more realistic) (دو الگوریتم ساخت شاخص)
- MapReduce (ساخت شاخص توزیعی)
- ساخت شاخص پویا (چگونه شاخص را هنگامی که مجموعه تغییر می‌کند بروز نگه داریم؟)

مفاهیم پایه‌ای سخت افزار

- بسیاری از تصمیمات طراحی در بازیابی اطلاعات بر اساس محدودیت سخت افزاری انجام می‌شود.
- ما با بررسی مفاهیم پایه‌ای سخت افزار که در این دوره نیاز داریم، شروع می‌کنیم.

❖دسترسی به داده‌های در حافظه نسبت به دیسک بسیار سریع‌تر است. (تقریبا 10 برابر)

❖هنگام جستجو در دیسک زمان تلف می‌شود. (هیچ داده‌ای از دیسک هنگامی که هد دیسک شروع به تغییر مکان می‌کند، انتقال نمی‌یابد.)

❖یک تکه بزرگ بسیار سریع‌تر از چند تکه کوچک است. (این کار برای بهینه کردن زمان انتقال از دیسک به ممووری است).

❖عملیات I/O، block-base است. (خواندن و نوشتن از کل block ها، و سایز آن ها ازkb 8 تا 256kb است.)

❖سرورهای استفاده شده در سیستم‌های بازیابی اطلاعات به طور معمول دارای چند گیگابایت حافظه اصلی هستند، گاهاً ده‌ها گیگابایت، و دارای ترابایت‌ها یا صدها گیگابایت فضای دیسک هستند.

❖تحمل‌خطا پرهزینه است:
(It’s cheaper to use many regular machines than one fault tolerant machine)

نماد	شرح	مقدار
s	متوسط زمان استوانه جویی	$5\ \text{ms} = 5 \times 10^{-3}\ \text{s}$
b	زمان انتقال در هر بایت	$0.02\ \mu\text{s} = 2 \times 10^{-8}\ \text{s}$
		$10^{9}\ \text{s}^{-1}$
P	عملیات سطح پایین (مانند مقایسه و جابجایی یک کلمه)	$0.01\ \mu\text{s} = 10^{-8}\ \text{s}$
	اندازه حافظه اصلی	several GB
	اندازه فضای دیسک	1 TB or more

RCV1 Collection

- آثار جمع‌آوری شده شکسپیر برای نشان دادن بسیاری از نکات این دوره به اندازه کافی بزرگ نیست.
- به عنوان مثال برای اعمال الگوریتم‌های مقیاس‌پذیر ساخت شاخص، ما از مجموعه Reuters RCV1 استفاده خواهیم کرد.
- مقالات اخبار انگلیسی در سال ۱۹۹۵ و ۱۹۹۶ (یک سال) از طریق سیم انتقال داده شده‌اند.

A Reuters RCV1 document

REUTERS

You are here: Home > News > Science > Article

Go to a Section: U.S. International Business Markets Politics Entertainment Technology Sports Oddly En

Extreme conditions create rare Antarctic clouds

Tue Aug 1, 2006 3:20am ET

Email This Article | Print This Article | Reprir

[-] Text [-

SYDNEY (Reuters) - Rare, mother-of-pearl colored clouds caused by extreme weather conditions above Antarctica are a possible indication of global warming, Australian scientists said on Tuesday.

Known as nacreous clouds, the spectacular formations showing delicate wisps of colors were photographed in the sky over an Australian

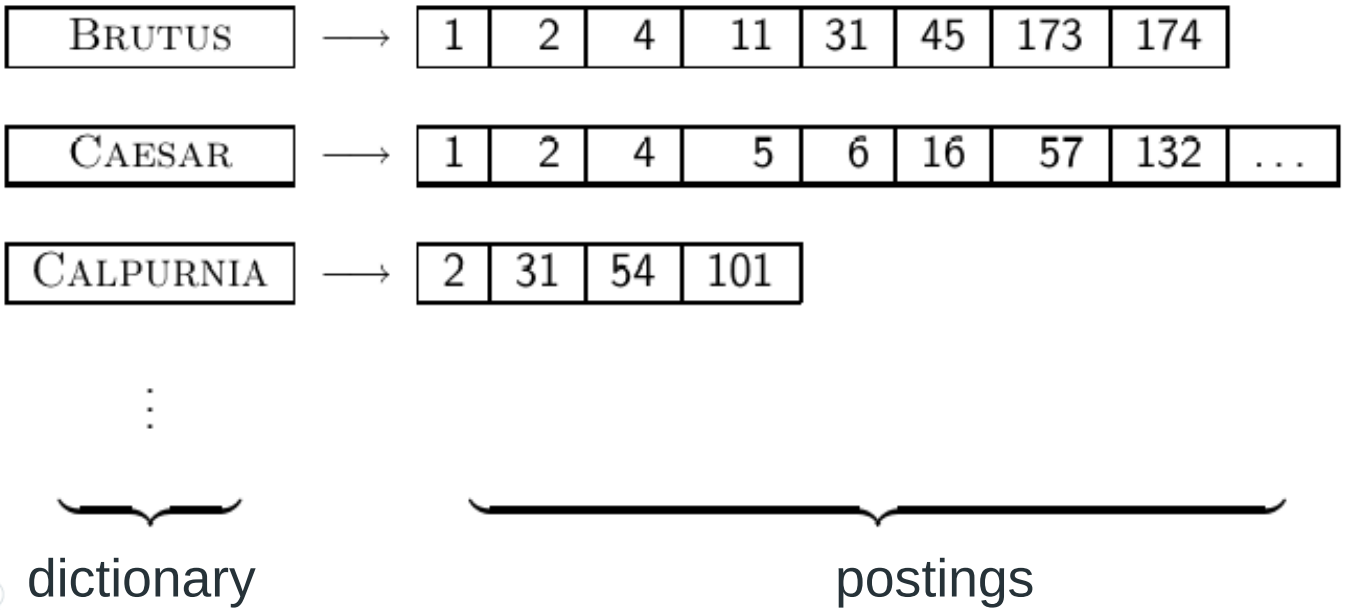
آمار مربوط به Reuters RCV1

N	documents	800,000
L	tokens per document	200
M	terms (= word types)	400,000
	bytes per token (incl. spaces/punct.)	6
	bytes per token (without spaces/punct.)	4.5
	bytes per term (= word type)	7.5
T	non-positional postings	100,000,000

❖ تمرین: میانگین فراوانی یک واژه چقدر است؟(چند توکن است؟) 4.5

bytes per word token vs 7.5 bytes per word type: why the difference? How many positional postings?

هدف: ساخت شاخص معکوس



ساخت شاخص در بازیابی اطلاعات :
مرتب کردن پست‌ها در حافظه اصلی

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

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

ساخت شاخص براساس مرتب‌سازی

❖ همانطور که فهرست را می‌سازیم، اسناد را یکی‌یکی تجزیه می‌کنیم.

❖ برای هر اصطلاح، posting ها تا پایان ناقص است.

❖آیا می‌توانیم تمام posting ها را در حافظه نگه داشته و در آخر(درون حافظه)مرتب کنیم؟
. برای مجموعه های بزرگ نمی‌توانیم.

❖برای ورودی 10-12 bytes per postings ، برای مجموعه های بزرگ به فضای زیادی نیاز داریم.

❖با T=100,000,000 در موردی از RCV1، ما می‌توانیم این کار را درون مموری روی یک ماشین ساده سال 2010 انجام دهیم.

❖ اما ساخت شاخص درون حافظه، برای مجموعه های بزرگ‌تر مناسب(مقیاس‌پذیر) نیست.

❖ما باید نتایج میانی را در دیسک ذخیره کنیم.

آیا همان الگوریتم برای دیسک نیز است؟

❖ آیا می‌توانیم از همان الگوریتم ساخت شاخص برای مجموعه‌های بزرگ‌تر استفاده کنیم، اما به جای حافظه از دیسک استفاده کنیم؟

. جواب خیر است، زیرا مرتب سازی $T=100,000,000$ رکورد روی دیسک خیلی کند است.

❖ ما به یک الگوریتم مرتب‌سازی خارجی نیاز داریم.

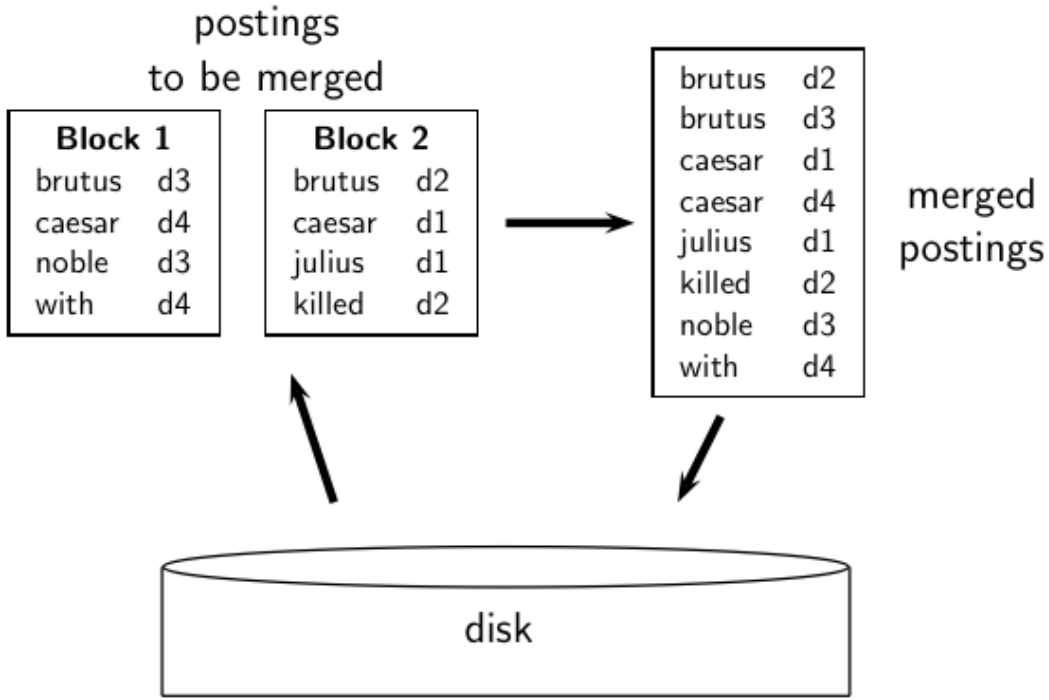
الگوریتم مرتب‌سازی خارجی (با استفاده از تعداد کمی disk seeks)

- باید T=100,000,000، posting را به صورت non-positional مرتب کنیم.
  - هر posting 12 بایت اندازه دارد. (4+4+4: termID, docID, Document Frequency)

- بلوکی را تعریف کنید که شامل 10,000,000 پست باشد:
  - ما به راحتی می‌توانیم این تعداد پست را درون حافظه‌اصلی(ممووری) جا دهیم.
  - همینطور 10 بلوک برای RCV1 خواهیم داشت.

- ایده اصلی الگوریتم:
  - برای هر بلوک: 1-جمع آوری پست‌ها 2-مرتب‌سازی در حافظه 3-نوشتن در دیسک
  - سپس بلوک ها را به یک ترتیب، به صورت مرتب ادغام کنید.

ادغام دو بلوک



Blocked Sort-Based Indexing

```
BSBIndexConstruction()
1  n ← 0
2  while (all documents have not been processed)
3  do n ← n + 1
4      block ← ParseNextBlock()
5      BSBI-Invert(block)
6      WriteBlockToDisk(block, f_n)
7  MergeBlocks(f_1, ..., f_n; f_merged)
```

- تصمیم کلیدی: اندازه یک بلوک چقدر باشد؟

مشکل الگوریتم Sort-based

- فرض اینگونه بود که، ما می‌توانیم لغت‌نامه را درون مموری نگه داریم.
- ما به لغت‌نامه(که به صورت پویا رشد می‌کند) نیاز داریم تا نگاشت اصطلاح به شناسه اصطلاح را انجام دهیم.
- در واقع می‌توانیم با term, docID postings به جای termID, docID postings کار کنیم.
- اما در این صورت فایل های میانی بسیار بزرگ می‌شوند. (در نهایت به یک روش ساخت شاخص مقیاس‌پذیر اما بسیار کند خواهیم رسید.)

Single-pass in-memory indexing

- مخفف عبارت بالا: SPIMI
- ایده کلیدی 1: برای هر بلوک دیکشنری‌های جداگانه‌ای ایجاد کنید، نیازی به حفظ نگاشت term – termID در سراسر بلوک‌ها نیست.
- ایده کلیدی 2: مرتب نکنید. به محض وقوع، پست‌ها را در لیست‌پست‌ها جمع‌آوری کنید.
- برای هر بلوک با این دو ایده می‌توانیم یک شاخص معکوس کامل ایجاد کنیم.
- سپس این شاخص های جدا می‌توانند به یک شاخص بزرگ ادغام شوند.

SPIMI-Invert

```
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
```

Merging of blocks is analogous to BSBI.

فشرده‌سازی در SPIMI

- فشرده سازی باعث افزایش کارآیی SPIMI میشود.

. فشرده سازی اصطلاحات

. فشرده سازی پست ها

Exercise: Time 1 machine needs for google size collection

```
BSBINDEXCONSTRUCTION()
1  n ← 0
2  while (all documents have not been processed)
3  do n ← n + 1
4     block ← PARSENEXTBLOCK()
5     BSBI-INVERT(block)
6     WRITEBLOCKTODISK(block, f_n)
7  MERGEBLOCKS(f_1, ..., f_n; f_merged)
```

symbol	statistic	value
s	average seek time	5 ms $= 5 \times 10^{-3}$ s
b	transfer time per byte	0.02 μs $= 2 \times 10^{-8}$ s
	processor's clock rate	10^9 s^{-1}
p	lowlevel operation	0.01 μs $= 10^{-8}$ s
	number of machines	1
	size of main memory	8 GB
	size of disk space	unlimited
N	documents	10^{11} (on disk)
L	avg. # word tokens per document	10^3
M	terms (= word types)	10^8
	avg. # bytes per word token (incl. spaces/punct.)	6
	avg. # bytes per word token (without spaces/punct.)	4.5
	avg. # bytes per term (= word type)	7.5

Hint: You have to make several simplifying assumptions – that's ok, just state them clearly.

شاخص توزیع شده

- برای ایجاد شاخص در مقیاس وب: باید از یک کلاستر کامپیوتری توزیع شده استفاده کنیم.
- ماشین‌های تک مستعد خطا هستند.

. ممکن است به طور غیرمنتظره‌ای کند شوند یا شکست بخورند.

- چگونه از چنین مجموعه‌ای از ماشین‌ها بهره ببریم؟

مراکز داده گوگل (برآورده‌های سال ۲۰۰۷؛ Gartner)

- مراکز داده گوگل عمدتاً (commodity machines) را شامل می‌شوند.
- مراکز داده در سراسر جهان پخش شده‌اند.
- یک ملیون سرور، سه ملیون processors/cores .
- گوگل هر سه ماه، 100,000 سرور نصب می‌کند.
- بر اساس مخارج 200-250 ملیون دلار در به صورت سالانه.
- این 10% از ظرفیت محاسباتی(computing) جهان خواهد بود.
- اگر در یک سیستم non-fault-tolerant با 1000 گره، هر گره 99.9% uptime باشد، uptime بودن کل سیستم چقدر خواهد بود؟

❖ فرض کنید یک سرور پس از سه سال از کار خواهد افتاد، برای نصب هر یک ملیون سرور،

فاصله میان machine failures چقدر است؟

پاسخ : کمتر از دو دقیقه.

شاخص توزیع شده

- حفظ یک ماشین اصلی (master) که کار شاخص سازی را انجام می‌دهد- considered “safe”
- شاخص سازی را به مجموعه task (وظایف)های موازی تقسیم بندی کنید.
- ماشین اصلی هر task را به ماشینی از مجموعه ماشین های بیکاری که وجود دارند اختصاص می‌دهد.

وظایف موازی (Parallel tasks)

❖دو مجموعه از parallel task ها را تعریف خواهیم کرد، و دو نوع از ماشین‌ها را برای حل آن ها گسترش خواهیم داد.

- Parsers
- Inverters

❖ مجموعه اسناد ورودی را به دو قسمت تقسیم کنید.(corresponding to blocks in BSBI/SPIMI)

❖ هر قسمت یک زیر مجموعه‌ای از اسناد است.

Parses

- ماشین اصلی یک قسمت(از سند) را به parser ماشینی که خالی است اختصاص می‌دهد.
- Parser به ترتیب(در یک زمان) یک سند را می‌خواند و جفت (term,docID) را تولید می‌کند.
- Parser جفت ها را داخل j term-partitions می‌نویسد.
- هر کدام برای یک دامنه از اولین حروف اصطلاحات:

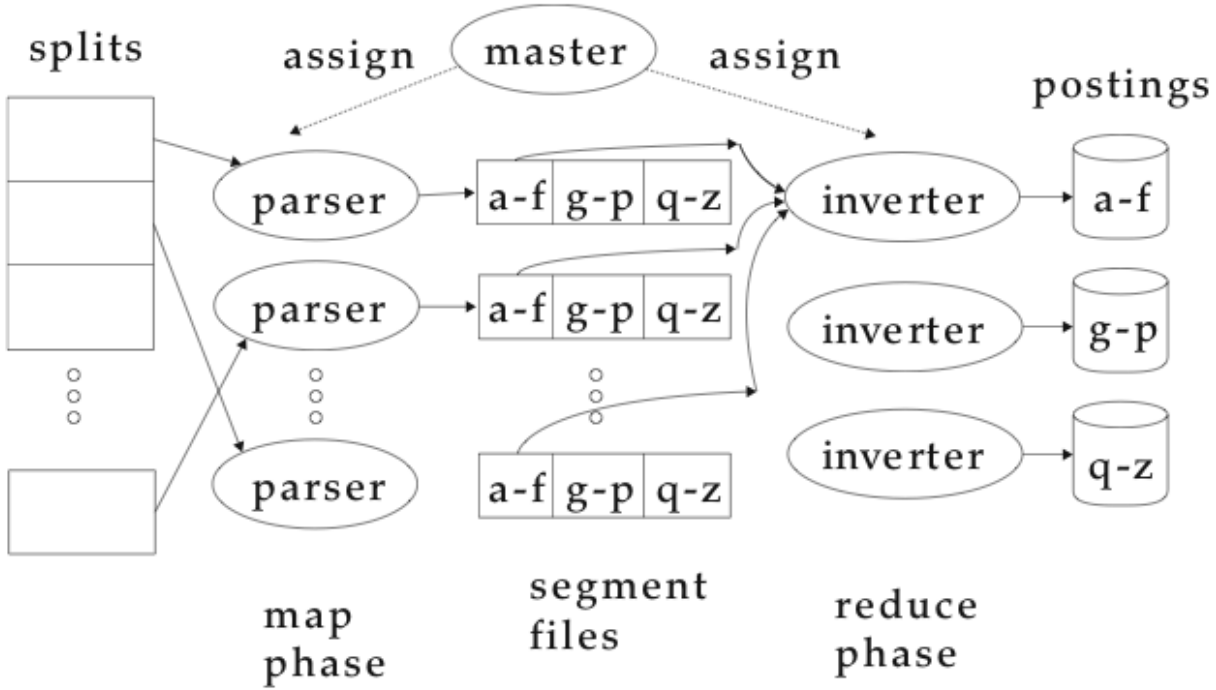
E.g., a-f, g-p, q-z (here: j = 3)

Inverters

- یک inverter تمام جفت (term,docID) ها را برای یک term-partition جمع‌آوری می‌کند.(e.g., for a-f)

- لیست‌پست‌ها را مرتب می‌کند و می‌نویسد.

جریان داده


splits
assign
master
assign
postings
parser
parser
parser
a-f g-p q-z
a-f g-p q-z
a-f g-p q-z
inverter
inverter
inverter
a-f
g-p
q-z
map phase
segment files
reduce phase

MapReduce

- الگوریتم ساخت شاخص که به تازگی توضیح داده شد، یک نمونه از MapReduce است.
- MapReduce یک چارچوب قدرتمند و از نظر مفهومی ساده برای محاسبات توزیع شده است.
- سیستم شاخص سازی گوگل (حدود 2002) شامل چند مرحله بود، هر کدام از طریق MapReduce پیاده‌سازی شد.
- ساخت شاخص تنها یک مرحله بود.
- تبدیل term-partitioned به شاخص document-partitioned

ساخت شاخص در MapReduce

Schema of map and reduce functions

map: input → list(k, v)

reduce: (k,list(v)) → output

Instantiation of the schema for index construction

map: web collection → list(termID, docID)

reduce: (⟨termID$_1$, list(docID)⟩, ⟨termID$_2$, list(docID)⟩, ...) → (postings_list$_1$, postings_list$_2$, ...)

Example for index construction

map: d_2 : C DIED. d_1 : C CAME, C C'ED. → (⟨C, d_2⟩, ⟨DIED,d_2⟩, ⟨C,d_1⟩, ⟨CAME,d_1⟩, ⟨C,d_1⟩, ⟨C'ED,d_1⟩)

reduce: (⟨C,(d_2,d_1,d_1)⟩,⟨DIED,(d_2)⟩,⟨CAME,(d_1)⟩,⟨C'ED,(d_1)⟩) → (⟨C,(d_1:2,d_2:1)⟩,⟨DIED,(d_2:1)⟩,⟨CAME,(d_1:1)⟩,⟨C'ED,(d_1:1)⟩)

تمرین:

- اطلاعاتی که توضیحات task شامل آن‌ها می‌باشد که ماشین اصلی به یک parser ارائه می‌دهد، چیست؟
- اطلاعاتی که parser به ماشین اصلی پس از اتمام task گزارش می‌دهد، چه اطلاعاتی است؟
- شرح task شامل چه اطلاعاتی است که Master به یک inverter می دهد؟
- اطلاعاتی که inverter به ماشین اصلی پس از اتمام task گزارش می‌دهد، چه اطلاعاتی است؟

شاخص پویا

- تا به اینجا فرض کرده‌ایم که مجموعه ها ایستا هستند.
- به ندرت اینگونه‌اند : اسناد درج، حذف و اصلاح می‌شوند.
- این به این معنا است که لغت‌نامه‌ها و لیست‌پست‌ها باید به صورت پویا اصلاح شوند.

ساخت شاخص پویا(ساده ترین رویکرد)

- شاخص اصلی بزرگ را روی دیسک نگه‌داری کنید
- اسناد جدید به فهرست کمکی کوچک در حافظه می روند.
- در هر دو جستجو کنید، نتایج را ادغام کنید
- به صورت دوره‌ای، شاخص کمکی را در شاخص بزرگ ادغام کنید
- حذفیات:

. بردار بیت (bit-vector) بی اعتبار برای اسناد حذف شده

. با استفاده از این بیت-بردار(bit-vector) ، اسناد بازگردانده شده توسط فهرست را فیلتر کنید

مسئله شاخص کمکی و شاخص اصلی

❖ ادغام های مکرر

❖ عملکرد ضعیف جستجو در طول ادغام فهرست

❖ در حقیقت:

.اگر برای هر لیست پست ها یک فایل جداگانه نگه داریم ادغام شاخص کمکی در فهرست اصلی چندان پرهزینه نیست

. Merge همان افزودن است

. اما در این صورت به فایل های زیادی نیاز خواهیم داشت

❖ فرض برای بقیه سخنرانی: فهرست یک فایل بزرگ است.

❖ در واقع: از طرحی در جایی در این بین استفاده کنید (به عنوان مثال، لیست های پست های بسیار بزرگ را به چندین فایل تقسیم کنید، لیست های پست های کوچک را در یک فایل جمع آوری کنید و غیره)

ادغام لگاریتمی

- ادغام لگاریتمی، هزینه ادغام شاخص‌ها را در طول زمان کم می‌کند.
  . کاربران در زمان پاسخ، تاثیر کمتری می‌بینند.
- یک سری شاخص را که هر کدام دو برابر شاخص قبلی است نگه دارید.
- کوچکتری (Z_0) را در حافظه نگه دارید.
- Larger ones (I_0, I_1, . . .) on disk
- اگر Z_0 بزرگتر از حد معمول شد روی دیسک به صورت I_0 بنویسید.(یا با I_0 (در صورت وجود) ادغام کنید و جواب را درون I_1 بنویسید.)

```
LMergeAddToken(indexes, Z_0, token)
 1  Z_0 ← Merge(Z_0, {token})
 2  if |Z_0| = n
 3    then for i ← 0 to ∞
 4         do if I_i ∈ indexes
 5               then Z_{i+1} ← Merge(I_i, Z_i)
 6                    (Z_{i+1} is a temporary index on disk.)
 7                    indexes ← indexes − {I_i}
 8               else I_i ← Z_i     (Z_i becomes the permanent index I_i.)
 9                    indexes ← indexes ∪ {I_i}
10                    Break
11         Z_0 ← ∅
```

```
LogarithmicMerge()
1  Z_0 ← ∅     (Z_0 is the in-memory index.)
2  indexes ← ∅
3  while true
4  do LMergeAddToken(indexes, Z_0, getNextToken())
```

Binary numbers: $I_3I_2I_1I_0 = 2^3 2^2 2^1 2^0$

- 0001
- 0010
- 0011
- 0100
- 0101
- 0110
- 0111
- 1000
- 1001
- 1010
- 1011
- 1100

ادغام لگاریتمی

- Number of indexes bounded by $O(\log T)$ (T is total number of postings read so far)
- So query processing requires the merging of $O(\log T)$ indexes
- Time complexity of index construction is $O(T \log T)$.
- . . . because each of T postings is merged $O(\log T)$ times.
- Auxiliary index: index construction time is $O(T^2)$ as each posting is touched in each merge.

  Suppose auxiliary index has size a

$$a + 2a + 3a + 4a + \ldots + na = a\frac{n(n+1)}{2} = O(n^2)$$

- So logarithmic merging is an order of magnitude more efficient.

شاخص سازی پویا در موتورهای جستجوی بزرگ

- اغلب یک ترکیب

. تغییرات تدریجی مکرر

. چرخش بخش‌های بزرگی از شاخص که می‌توان آن‌ها را جایگزین کرد

. بازسازی کامل گاه به گاه (با افزایش اندازه سخت تر می شود-مشخص نیست که آیا Google می تواند بازسازی کامل انجام دهد؟)

ایجاد شاخص های موقعیتی

- اساساً همین مشکل جزساختارهای داده میانی بزرگ هستند.

تشکر

سوال؟

a.golzari@azaruniv.ac.ir

a.golzari@tabrizu.ac.ir

https://github.com/Amin-Golzari-Oskouei