Quest for Efficiency

Remember the scale of Web-scale search engines

- Microsoft Bing uses "hundreds of thousands of servers" in their search engine
- Any new feature/retrieval technique should not increase the response time of the system – as this might lead to a loss in revenues
- Hence, deploying a retrieval technique that causes a 1% increase in response time implies 1000 additional servers must be activated in their data centre(s)
  - This has significant cost and "Green" impact

IR infrastructures are concerned with making effective yet efficient retrieval

3

3

Outline

Efficient infrastructure techniques

- Caching
- Static Pruning
- Query Evaluation (TAAT & DAAT), and Dynamic Pruning
- Index Compression

Distributed IR infrastructure

- Distributed Retrieval
- Query Efficiency Prediction

4

4

Increasing Efficiency

SMALLER, BETTER, FASTER

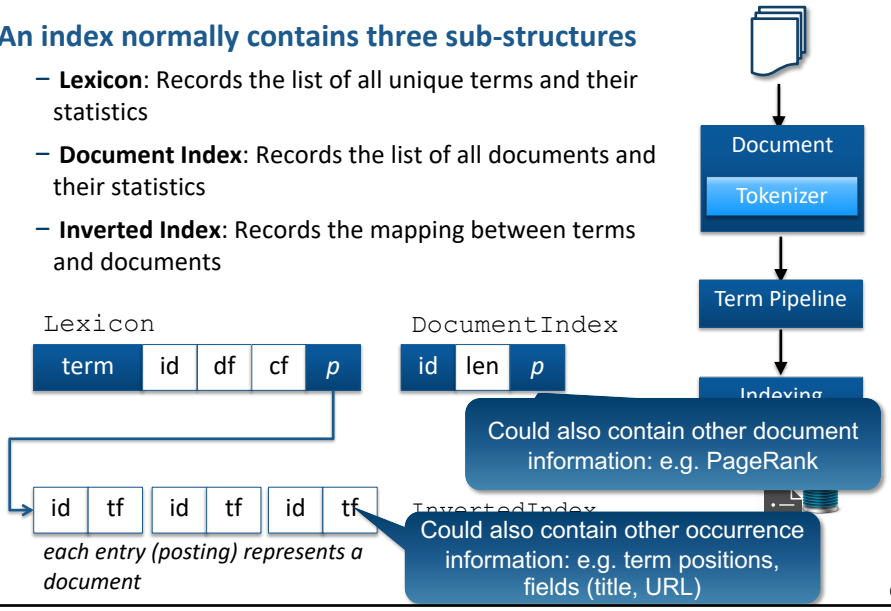
5

5

Recall: The Format of an Index

An index normally contains three sub-structures

- **Lexicon**: Records the list of all unique terms and their statistics
- **Document Index**: Records the list of all documents and their statistics
- **Inverted Index**: Records the mapping between terms and documents



6

6

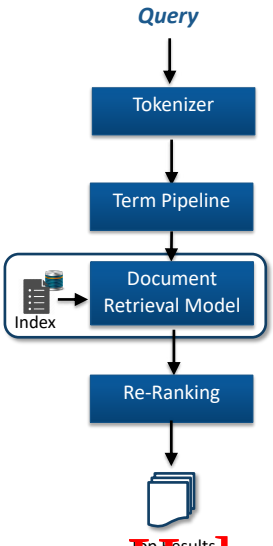
Recall: Search Efficiency

It is important to make retrieval as fast as possible

- Research by bing indicates that even slightly slower retrieval (0.2s-0.4s) can lead to a dramatic drop in the perceived quality of the results [1]

So what is the most costly part of a (classical) search system?

- Scoring each document for the user query



[1] Jeevan et al. Slow Search: Information Retrieval without Time Constraints | HCIR'13

7

7

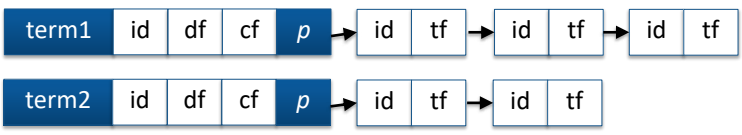
Why is Document Scoring Expensive?

The largest reason for the expense of document scoring is that there are lots of documents:

- A Web search index can contain billions of documents
  - Google currently indexes trillions of pages [1]

More specifically, the cost of a search is dependent on:

- **Query length** (the number of search terms)
- **Posting list length** for each query term
  - i.e. The number of documents containing each term



[1] http://www.statisticbrain.com/total-number-of-pages-indexed-by-google/

8

8

Strategies to Speed-up Search

There are several enhancements to the search architecture that can make search more efficient

Search result/Term caching

- Where possible avoid the scoring process altogether

Pruning

- **Static Pruning**: Skip the ***indexing*** of documents that are not likely to make the first few search result pages
- **Dynamic Pruning**: Skip the ***scoring*** of documents that will not make the first few search result pages

Index compression

- Reduce the time it takes to read a posting list

Discuss a number of strategies to speed-up the retrieval process along with their pros and cons.

9

CACHING

10

Search Result/Term Caching

Caching strategies are built on the idea that we should store answers to past queries

- Past answers can be used to bypass the scoring process for subsequent queries
- For popular queries, caching is very beneficial

There are two types of caching strategy

- **Search Result Caching** stores the final ranked list of documents for a query
- **Term Caching** stores the posting lists for each of the query terms in memory

Caching is beneficial to efficiency [1]:

- Search Result caching can avoid scoring for 50% of queries – so called head queries
- Term caching can avoid loading one or more posting lists for 88% of queries

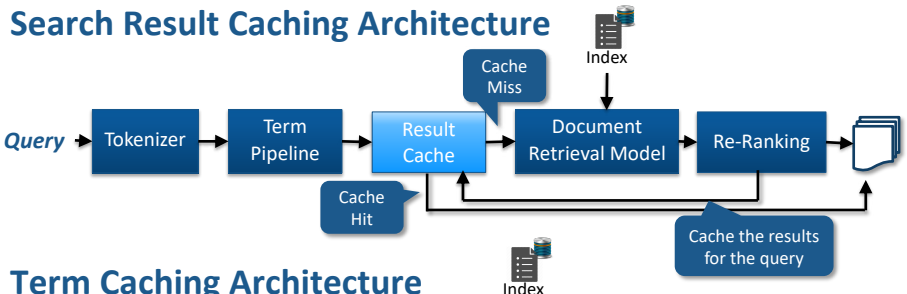
[1] Baeza-Yates et al. The impact of caching on search engines. SIGIR 07

11

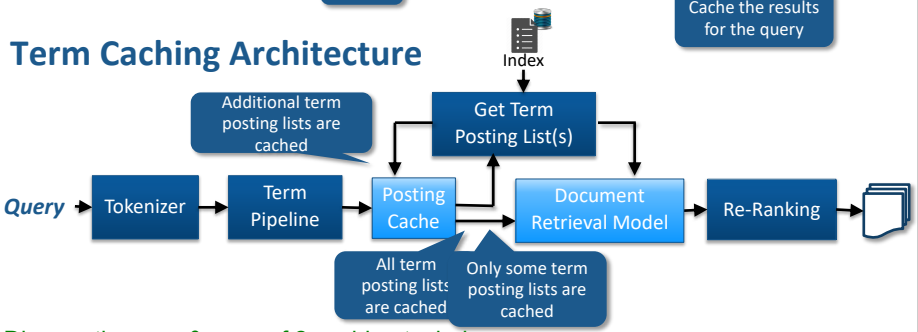
11

Search Result/Term Caching

Search Result Caching Architecture



Term Caching Architecture



Discuss the pros & cons of 2 caching techniques.

12

12

More on Caching

Memory is cheap…

- The logical consequence is that many search engines keep the entire index in memory

SSDs and hard drives offer slower storage tiers

For many queries, phrases can be used to help the ranking

- If we had bigram posting lists, we could score these queries much quicker
- But we cannot store postings for all bigrams

Instead, frequent bigrams from the query log can be selected, and then SSD and disk space can be used to cache and store these "term pair" posting lists [1]

- Then decide on a per-query basis to use them or not [2]

[1] Yan et al. Efficient Term Proximity Search with Term-Pair Indexes. CIKM 2010
[2] Risvik et al. Maguro, …. WSDM 2010

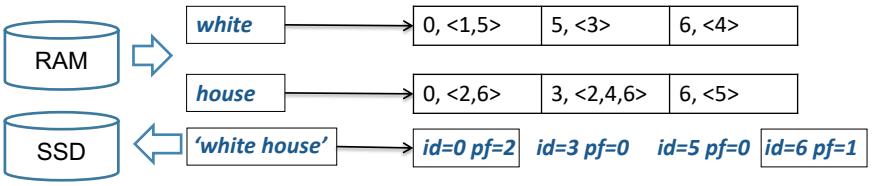
13

13

Paired Posting Lists

Consider the example query `white house` from Lect. 8

- We can retrieve for the phrase "white house" by intersecting the inverted index posting lists for 'white' and 'house',
  - calculating a 'pair frequency' (pf) for each document



- In essence, we can simulate a posting list for "white house", without indexing it as a bigram

Then if 'white house' occurs frequently in the query stream, it can be cached, e.g. to SSD

14

14

STATIC PRUNING

15

15

Top-K Retrieval

K is usually very small viz. the size of the collection (N)

- K << 1000, i.e. 20 for the first page of results
- Even if we are re-ranking results, K << N, e.g. K=5000

Question: do we really need to keep ALL of the information in the index for a good-quality top-K search for common queries?

- There should be a way to remove some of the less important data, while (hopefully) retaining the quality of top-K results!

What is (not) important?

- Will some documents never be retrieved, for any query?
- Will some terms never help retrieval?

16

16

Static Pruning Concepts

Document-based pruning:

- Discards terms from a document that are less representative of a document's content
- Can be applied on-the-fly during indexing, IF we have reasonable collection statistics

Term-based pruning:

- Discards term occurrences that are less likely to affect the retrieval performance of a specific weighting model (e.g. BM25)
- Done after the index is completed

What is static pruning? Discuss 2 possible static pruning techniques.

17

17

Pruning Algorithm

***Foreach* common query:**

- Run it against the full index
- Record the top-K matching documents
- *Foreach* document:
  - Record the terms and term positions that contributed to the score

Finally: remove all non-recorded postings and terms

First proposed by D. Carmel (2001) for single term queries

18

18

Static Pruning
Advantages & Disadvantages

Some results claim a modest negative impact on effectiveness when pruning up to 60% of postings

- Static pruning is a lossy compression of the inverted index
- Index is smaller; retrieval is faster; pruning is done offline

An application of static pruning is a multi-tier architecture:

- Most common queries are handled by a heavily pruned 1st tier index that fits wholly in RAM
- Less common queries handled by a 2nd tier index (on SSD?)
- Remaining queries handled by a full index on disk

19

19

QUERY EVALUATION & DYNAMIC PRUNING

20

20

Query Evaluation

Even when using caching and/or static pruning, retrieval still needs to score many documents in the index

- i.e. when the query/query terms are not in the cache

Normal strategies make a pass on the postings lists for each query term

- This can be done Term-at-a-Time (TAAT) – one query term at a time
- Or Document-at-a-time (DAAT) – all query terms in parallel

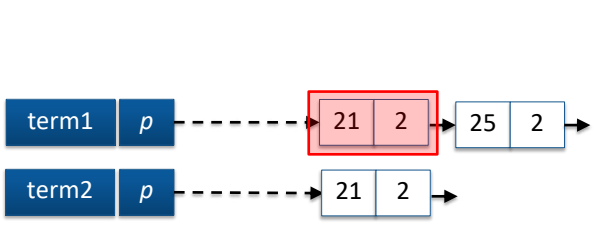
I will explain these, before showing how we can improve them

Compare and contrast the TAAT and DAAT query evaluation techniques

21

21

Term-at-a-Time (TAAT)

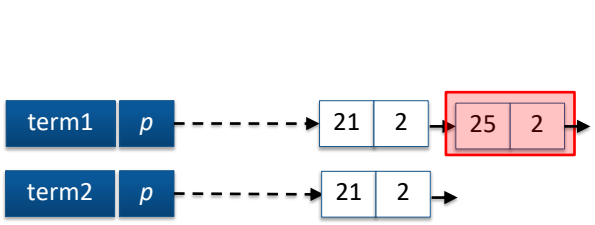


rank	docid	score
1	*21*	*2*
2		
…		

22

22

Term-at-a-Time (TAAT)

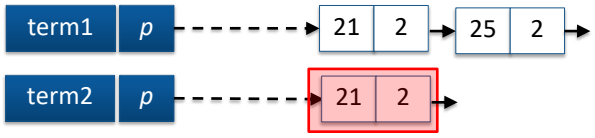


rank	docid	score
1	21	2
2	25	2
…		

23

23

Term-at-a-Time (TAAT)



rank	docid	score
1	21	4
2	25	2
…		

Advantages:

- Simple

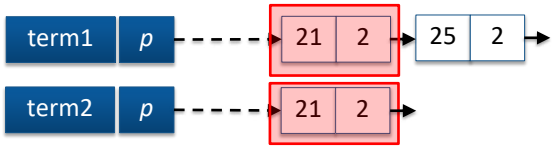
Disadvantages:

- Requires lots of memory to contain partial scores for all documents
- Difficult to do Boolean or phrase queries, as we don't have all postings for a given document at the same time

24

24

Document-at-a-Time (DAAT)

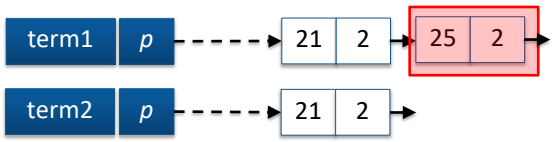


rank	docid	score
1	*21*	*4*
2		
…		

25

25

Document-at-a-Time (DAAT)



rank	docid	score
1	*21*	*4*
2	*25*	*2*
…		

Advantages:

- Reduced memory compared to TAAT (and hence faster)
- Supports Boolean query operators, phrases, etc.

Disadvantages:

- Slightly more complex to implement

Most commercial search engines are reported to use DAAT

26

26

Dynamic Pruning during Query Evaluation

Dynamic pruning strategies aim to make scoring faster by only scoring a subset of the documents

- The core assumption of these approaches is that the user is only interested in the top K results, say K=20
- During query scoring, it is possible to determine if a document cannot make the top K ranked results
- Hence, the scoring of such documents can be terminated early, or skipped entirely, without damaging retrieval effectiveness to rank K

We call this *"safe-to-rank K"*

27

27

Dynamic Pruning Techniques

The two most well-known methods for DAAT dynamic pruning are MaxScore and WAND

MaxScore [1]

- **Early termination**: does not compute scores for documents that won't be retrieved by comparing **upper bounds** with a score **threshold**

WAND [2]

- **Approximate evaluation**: does not consider documents with approximate scores (sum of **upper bounds**) lower than **threshold**
- Therefore, it focuses on the combinations of terms needed (w**AND**) for a document to be retrieved

[1] H Turtle & J Flood. Query Evaluation : Strategies and Optimisations. IPM: 31(6). 1995.
[2] A Broder et al. Efficient Query Evaluation using a Two-Level Retrieval Process. CIKM 2003.

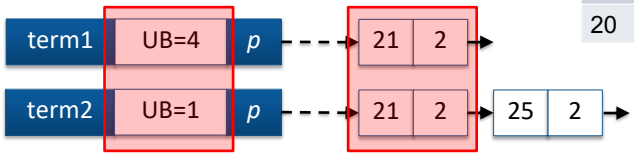
28

28

MaxScore Example

Require K=20 documents
Weighting model BM25

rank	docid	score
1	20	5
	…	
19	8	4.75
20	5	4.5



docid	21
term1	<=4
term2	<=1
score	<=5
threshold	4.5

term scores determined using upper bounds

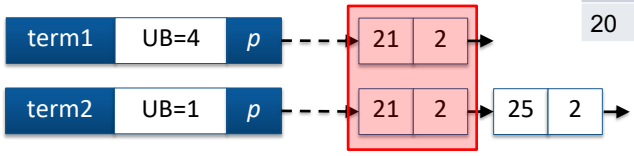
Could make top K

29

MaxScore Example

Require K=20 documents
Weighting model BM25

rank	docid	score
1	20	5
	…	
19	8	4.75
20	5	4.5



docid	21
term1	3.1
term2	<= 1
score	<= 4.1
threshold	4.5

term score calculated using BM25

PRUNE: Can't make top K!

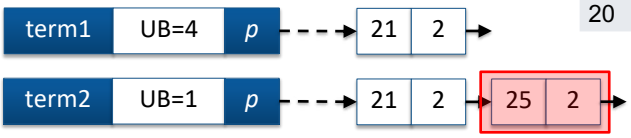
30

30

MaxScore Example

Require K=20 documents
Weighting model BM25

rank	docid	score
1	20	5
	…	
19	8	4.75
20	5	4.5



docid	25
term1	0
term2	<=1
score	<=1
threshold	4.5

term scores determined using upper bounds

PRUNE: Can't make top K!

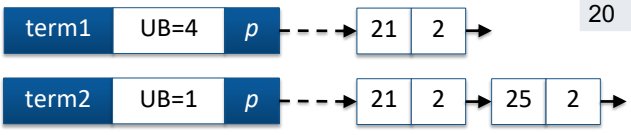
31

31

WAND Example

Require K=20 documents
Weighting model BM25

rank	docid	score
1	20	5
	…	
19	8	4.75
20	5	4.5



WAND focuses on the combinations of terms needed (c.f. Weighted AND) to reach the threshold

- With threshold 4.5, any document without term1 cannot make the retrieved set. Hence, we can **skip** docid 25 in the term2 posting list
- Hence, it will focus retrieval using term1, and only score term2 for documents that could exceed the threshold

For both MaxScore & WAND, smaller K => faster retrieval

32

32

Some numbers…

To demonstrate the benefit of dynamic pruning, we report experiments from [1]

- Retrieve K=1000 document for BM25 on the ClueWeb09 collection
- 1000 real search engine queries

Query Evaluation	Mean Response Time	Postings Scored
Exhaustive DAAT	1.36	100%
MaxScore	1.24	43.0%
WAND	0.96	10.8%

This is for "safe-to-rank 1000". Both WAND & MaxScore can be configured to be faster, but unsafe, i.e. permit losses in effectiveness above rank K

- This is achieved by over-inflating the threshold

There are also unsafe dynamic pruning techniques for TAAT

Overall, dynamic pruning is an important component of modern search engine deployments

[1] N. Tonellotto, C. Macdonald, and I. Ounis. Effect of different docid orderings on dynamic pruning retrieval strategies. SIGIR 2011.

33

INDEX COMPRESSION

34

Index Compression

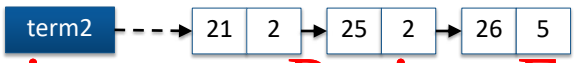
The previous approaches make retrieval faster by scoring fewer documents

- However it is also possible to make the scoring of each document faster!

This can be achieved by applying index compression [1]

- **Motivation**: it physically takes time to read the term posting lists, particularly if they are stored on a (slow) hard disk
- Using **lossless** compressed layouts for the term posting lists can save space (on disk or in memory) and reduce the amount of time spent doing IO
- But decompression can also be expensive, so efficient decompression is key!

1 integer = 32 bits = 4 bytes
total = 24 bytes



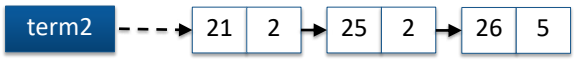
Do we need 32 bits?

[1] I.H.Witten et al. Managing Gigabytes: Compressing and Indexing Documents and Images. Morgan Kaufmann 1999.

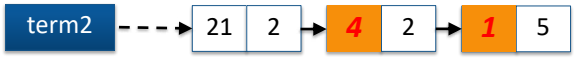
Delta Gaps

One component of the information stored in a posting list is the docids…

- …in ascending order!



We can make smaller numbers by taking the differences



So each docid in the posting lists could be represented using less bits

- How to represent these numbers?
- 32 bits has a range -2147483648 .. 2147483648
- Using a fixed number of bits is wasteful (192 bits = 24 bytes)

Elias Unary & Gamma Encoding

Unary:

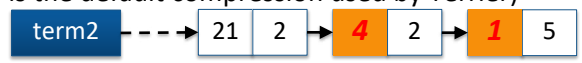
- Use as many 0s as the input value *x*, followed by a 1
- E.g.: 5 is 000001

Gamma:

- Let N=$\lfloor \log_2 x \rfloor$ be the highest power of 2 that *x* contains;
- Write N out in unary representation, followed by the remainder ($x - 2^N$) in binary
- E.g.: 5 is represented as 00101

Let's represent docids as gamma, and tf as unary

- (This is the default compression used by Terrier)



Repr.	0000 1 0101	001	00100	001	1	000001
Bits	9	3	5	3	1	6

= 27 bits (< 4 bytes), down from 24 bytes!

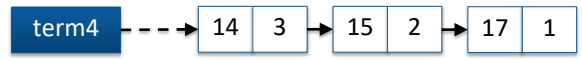
Achieved Compression rate = 4.5 bits/int

37

37

Exercise

Consider the following posting list:



id=14 tf=3	id= 15 tf=2	id=17 tf=1

Encode the posting list using Unary encoding, and calculate the achieved compression rate

Firstly take the delta-gaps

14,3,1,2,2,1

Encode each in unary:

000000000000001 0001 01 001 001 01

Encoding 6 integers as 32bit integers = 192 bits (i.e. 32 bits per integer)

Encoding 6 integers as using Unary = 29 bits =~ 4.8 bit per integer

38

38

Other Compressions Schemes

Elias Gamma & Elias Unary are moderately expensive to decode: lots of bit twiddling!

- Other schemes are byte-aligned, e.g.
  - Variable byte [1]
  - Simple family [2]

Documents are often clustered in the inverted index (e.g. by URL ordering)

- Compression can be more effective in blocks of numbers
- *List-adaptive* techniques work on blocks of numbers
  - Frame of reference (FOR) [3]
  - Patched frame of reference (PFOR) [4]

[1] H.E. Williams and J. Zobel. Compressing Integers for Fast File Access. Comput. J. 1999
[2] V. Anh & A. Moffat. Inverted Index Compression using Word-aligned Binary Codes. INRT. 2005
[3] A. Goldstein et al. Compressing relations and indexes. ICDE 1998
[4] M. Zukowski et al. Super-scalar RAM-CPU cache compression. ICDE 2006.

39

39

Frame of Reference

Idea: pick the minimum *m* and the maximum *M* values of the block of numbers that you are compressing.

- Then, any value *x* can be represented using *b* bits, where b = $\lceil \log_2(M-m+1) \rceil$.

Example: To compress numbers in range {2000,...,2063}

- $\lceil \log_2(64) \rceil = 6$
- So we use 6 bits per value:

2000 6 xxxxxxxxxxxxxxxxxx...
2 2

40

40

Compression: Some numbers [1]

ClueWeb09 corpus – 50 million Web documents

- 12,725,738,385 postings => 94.8GB inverted file uncompressed – NO retrieval numbers: WAY TOO SLOW!
- Terrier's standard Elias Gamma/Unary compression = 15GB

	Time (s)	Size	Time (s)	Size
	docids		tfs	
Gamma/Unary	1.55	-	1.55	-
Variable Byte	+0.6%	+5%	+9%	+18.4%
Simple16	-7.1%	-0.2%	-2.6%	+0.7%
FOR	**-9.7%**	+1.3%	**-3.2%**	+4.1%
PForDelta	-7.7%	+1.2%	-1.3%	+3.3%

Compression is essential for an efficient IR system

- List adaptive compression: slightly larger indices, markedly faster

[1] M. Catena, C. Macdonald, and I. Ounis. On Inverted Index Compression for Search Engine Efficiency. ECIR 2014: Best Paper Award.

41

41

Efficient Query Evaluation

Caching, Pruning & Compression all form essential aspects of an efficient IR system

- Each provides important improvements to **efficiency**
- Some techniques like static pruning or *unsafe* dynamic pruning can **degrade effectiveness**
  - A search engine implementer must be aware of the tradeoffs to achieve their desired effectiveness within cost constraints

Other state-of-the-art approaches I haven't included:

- Impact ordered posting lists: an alternative index layout
  - Requires efficiency/effectiveness tradeoff
- Block-Max WAND: integrates WAND more tightly with the index compression format

42

42

Scaling Up

TALL vs. WIDE SYSTEMS

43

43

Scaling Up Information Retrieval

Even with the aforementioned efficiency improvements, indexing and search is computationally and (disk) IO intensive

To satisfy high query loads, the retrieval process needs to be spread over many CPUs and hard disks

There are 2 main paradigms to scale up:

- Vertical: Buy a large mainframe machine with lots of CPU cores and storage
- Horizontal: Buy many machines and distribute the search process over them

44

44

Vertical vs. Horizontal Scaling

Vertical Scaling

- Advantages
  - All resources are local to the processing
  - Some applications do not lend themselves to a distributed computing model
- Disadvantages
  - Expensive infrastructure required
  - Fault tolerance is hard to achieve

Horizontal Scaling

- Advantages
  - Nodes can be added in an ad-hoc manner as processing power is needed
  - Multi-core processing nodes are inexpensive
- Disadvantages
  - Additional communication and coordination overheads are incurred

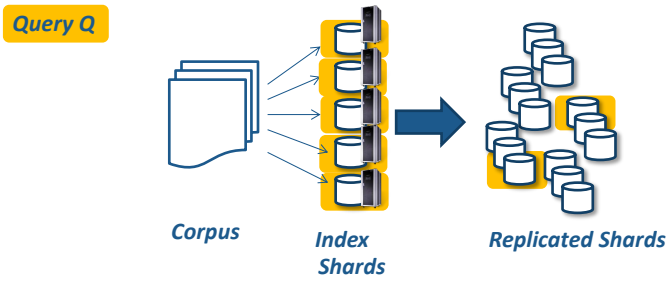
45

45

Parallelised Indexing and Retrieval

Horizontal scaling is used by large search engines to parallelise the indexing process and the index itself

- Spread the index out into shards running on many machines
- Replicate each shard multiple times to allow for multiple queries to be processed in parallel and for fault tolerance



In the following, we cover distributed retrieval

46

46

Distributed Retrieval Architectures (1)

So how do we partition data between nodes?

	t_1	t_2	t_3	t_4
d_1		3	1	
d_2	5	2	1	3
d_3		4		
d_4	4	5		4

Each row represents a document d_j and each column represents an indexing term t_i

Option 1: Term Partitioning

- Different nodes (or *query servers*) are associated to different terms: e.g. A-J K-Q, R-Z
- During query processing, different queries *touch* different query servers
- So querying load is spread across different query servers

Baeza-Yates et al. Challenges on distributed web retrieval. ICDE 2007. 47

47

Distributed Retrieval Architectures (2)

So how do we partition data between nodes?

	t_1	t_2	t_3	t_4
d_1		3	1	
d_2	5	2	1	3
d_3		4		
d_4	4	5		4

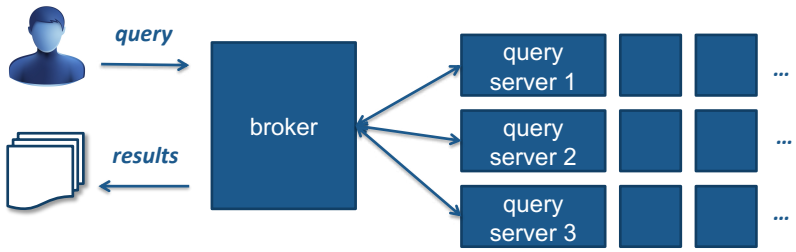
Option 2: Document Partitioning

- Different documents are allocated to P different query servers
- During query processing, each server executes the query on N/P documents, so (for even partitioning), load is even on each query server
- The results from each of the servers are combined into a final result list

Baeza-Yates et al. Challenges on distributed web retrieval. ICDE 2007. 48

48

Distributed Architectures



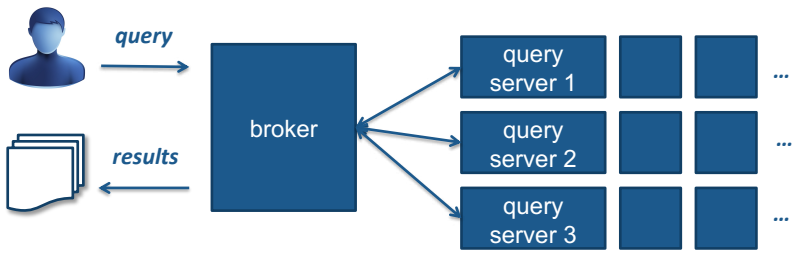
A distributed retrieval setting is coordinated by the *broker*

- The broker passes queries to query servers, and collates the top K results for the user.
- Query servers can be replicated: increases throughput and fault tolerance

49

49

Distributed Querying (1)



A distributed retrieval setting is coordinated by the *broker*

- Document partitioning: for K results, collect top K from each query server
- Term partitioning: broker collects ALL results from each query server to ensure effective retrieval:
  - Why all? A document with low score on some terms may have high scores on others
    - Just like TAAT query evaluation

A Moffat, W Webber, J Zobel, R Baeza-Yates (2005). A pipelined architecture for distributed text query evaluation. IR Journal 10(3)

50

50

Distributed Querying (2)

Key message:

The need to collect more than K results from each query server is a key disadvantage of term partitioning, which is rarely used in practice

However, hybrid architectures such as pipelining are possible

51

51

Document Partitioning Strategies

A few document partitioning strategies exist:

- **Random** – good for efficiency on large collections: all queries touch all partitions
- **Semantic/topic** – e.g. if collection is already organised into semantically meaningful sub-collections; a query targets particular sub-collections
  - News, tweets, webpages, images

We want each collection to be "well separated", such that query maps to a distinct collection containing the largest number of relevant documents

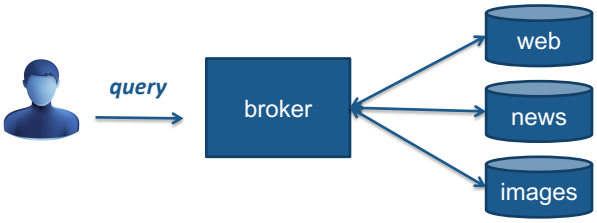
- E.g. by language – permits geographically distributed data centres: keep the Chinese index in Hong Kong
- E.g. by examining a query log, and clustering documents by the queries that touched them

52

52

Resource Selection & Aggregated Search

How do we tell which partitions can answer a query?



1. **Resource selection** techniques (CORI, ReDDE): statistical predictors if a sub-collection can answer a query
2. **Learn** if a sub-collection is good for a query: e.g. present users with news results, see if they click

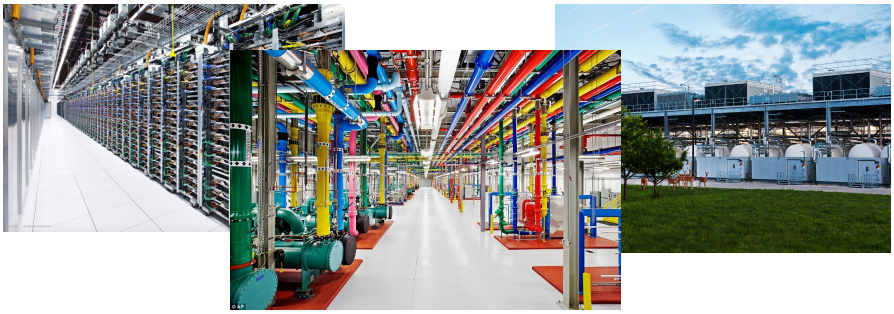
M Shokouhi and L Si (2011). Federated Search. Foundations and Trends in IR 5(1).
A Arguello, J Diaz, J Callan, F Crespo (2009). Sources of evidence for vertical selection. SIGIR.

53

Keep Up Efficiency!

Large search engines may need hundreds of thousands of query servers…

- …representing a significant consumption of power – data centres must be near cheap, green energy sources
- Green IR is therefore important: Keep your search engine as efficient as possible => more throughput, less servers, less €!



54

Advanced Technique: Query Efficiency Prediction

How long does a query take to execute on a query server?

- Depends on: how many query terms?
- Depends on: how long the postings lists are (sum, maximum)?
- Depends on: how many documents can be pruned during scoring?
- Depends on: how many K documents to retrieve?

You met query performance predictors in lecture 6

- They aim to predict the effectiveness of queries

It's also possible to predict the efficiency of a query

Craig Macdonald, Nicola Tonellotto and Iadh Ounis (2012). Learning to Predict Response Times for Online Query Scheduling. SIGIR 2012.

55

55

Query Efficiency Prediction Applications

If a query is predicted to be too slow, there are actions that can be taken:

1. Scheduling queries to least busy query servers [1]
2. Changing the efficiency/effectiveness tradeoff according to query expense or current volume [2, 3]
   - E.g. adjusting K, or dynamic pruning safeness
3. Selective parallelisation: using more CPU cores for expensive queries [4]
   - We might route queries between fast, energy-hungry CPUs, and slower greener CPUs

[1] C Macdonald et al (2012). Learning to predict response times for online query scheduling. SIGIR.
[2] D Brocoolo et al (2013). Load-Sensitive Selective Pruning for Distributed Search. CIKM.
[3] N Tonellotto et al (2013). Efficient and effective retrieval using selective pruning. WSDM.
[4] M Jeon et al. (2014). Predictive parallelization: taming tail latencies in web search. SIGIR.

56

56

Query Efficiency Prediction at Bing

Microsoft Bing has deployed the Query Efficiency Predictors proposed by the University of Glasgow across *"a few hundred thousand query servers"*.

"This potentially saves one-third of production servers" – significant $$ saving in energy consumption

57

57

Summary: Distributed Retrieval

Distributed Retrieval environments…

- …permit efficient retrieval across large-scale collections
- …are particularly applicable for Web-scale search engines

We examined partitioning schemes and resource selection

- How to decide which part of the index we should examine…
- How to address efficiency in a distributed retrieval environment…

What about distributed indexing?

- Techniques like MapReduce can help to distribute indexing across many machines – see the Big Data course for more information

58

58

Acknowledgements

These slides are based on a presentation by Craig Macdonald at the ESSIR 2015 European Summer School in Athens

The following people gave input to the creation of these slides:

- Iadh Ounis
- Richard McCreadie
- Matteo Catena

Thanks to Andrezj Bialecki for a few examples

66