Information Retrieval Infrastructures

University of Glasgow

Craig Macdonald

Thanks are due to

Iadh Ounis, Richard McCreadie, Matteo Catena

Contents

Information Retrieval Pipeline

Efficiency

- Caching
- Query evaluation & Dynamic Pruning
- Compression

Infrastructures for Efficient & Effective Learning-To-Rank

Scaling-up

- Vertical Scaling vs Horizontal Scaling
- Distributed Retrieval Architectures
- Distributed Indexing using MapReduce

Efficient Real-time Search

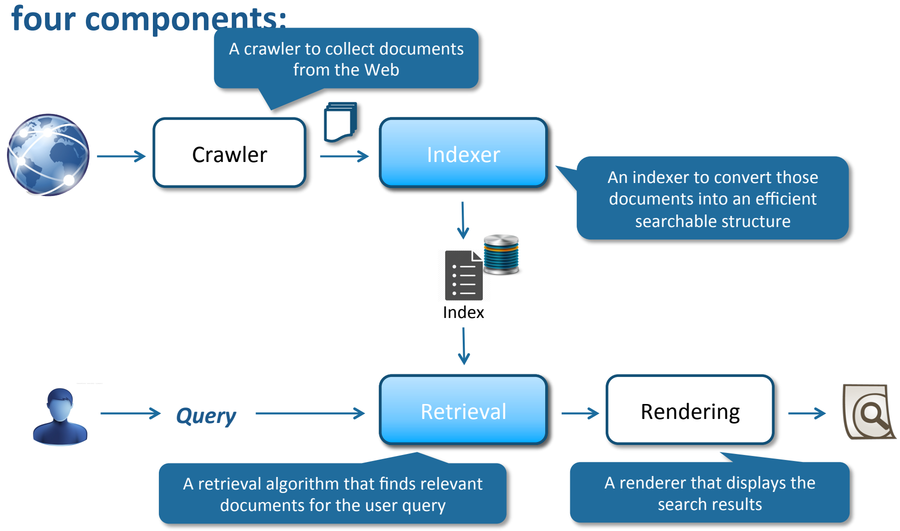
- Twitter Earlybird
- Stream processing architectures

2

The Core Infrastructure of an Information Retrieval System

Terrier Team | University of Glasgow

An information retrieval is classically defined in terms of four components:



3

IR Infrastructure

Terrier Team | University of Glasgow

I would advise that you do USE & extend an existing platform during your research, rather than "re-implement the wheel"!

Much research has been performed to identify the best architectures..., and many platforms with different assumptions exist:

Solr/Lucene	Galago
Terrier	MG4J
Lemur/Indri	

This talk is agnostic to platform choice, but...

4

Terrier

Terrier Team | University of Glasgow

Under development as Glasgow since 2001, with support for classical and modern infrastructures, e.g.:

- Compressed index structures (including MapReduce-based indexing)
- Classical, field-based & proximity models (e.g. Markov Random Fields, BM25F), along with learning-to-rank

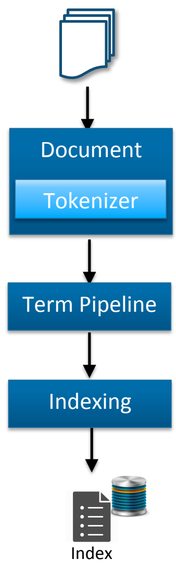
We have researched and tested many infrastructure techniques by building upon and extended Terrier…

… and I'll tell you about a few here today!

Inside a Classical Indexer

Terrier Team | University of Glasgow

The indexer is responsible for building an efficient structure that enables fast search for a user query – called an *index*



This involves:

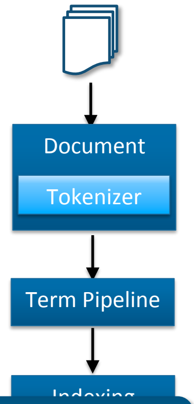
- Extracting the terms from each document (tokenization)
- Applying transformations to the terms to make subsequent search easier
  - e.g. remove stopwords or apply stemming
- Index the terms within each document (record which terms the document contains)

The Format of an Index

Terrier Team | University of Glasgow

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



Lexicon

term	id	df	cf	*p*

DocumentIndex

id	len	*p*

Could also contain other document information: e.g. PageRank

id	tf	id	tf	id	tf

each entry represents a document

InvertedIndex

Could also contain other occurrence information: e.g. term positions, fields (title, URL)

7

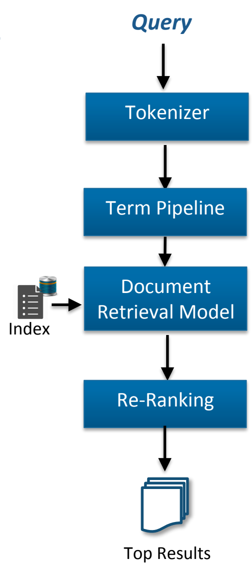
Inside a Search Request

Terrier Team | University of Glasgow

A retrieval algorithm uses the index structures to rank documents that match the user query

This involves:

- Applying the same tokenization and term transformations (e.g. stemming) that were applied to the documents &(possibly other query reformulations)
- For each term in the query, applying a document retrieval model to score each document that contains one or more of those terms
- Optionally applying a re-ranking algorithm to incorporate additional evidence
  - e.g. PageRank scores, or proximity search

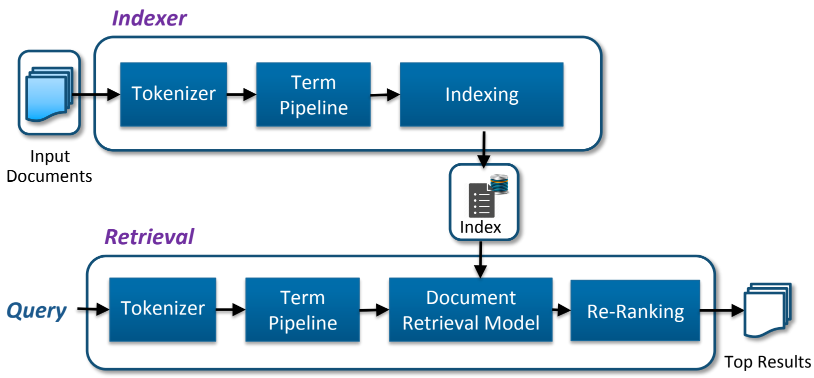


8

Search Architecture

Terrier Team | University of Glasgow

The architecture of a classical information retrieval system can be summarised as:



9

Increasing Efficiency

SMALLER, BETTER, FASTER

10

Search Efficiency

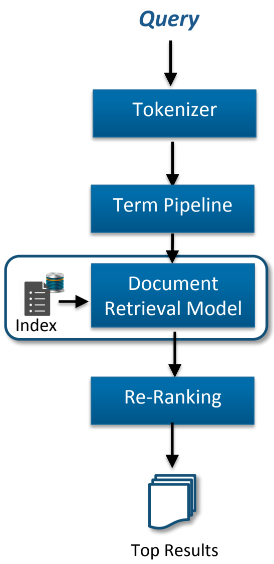
Terrier Team | University of Glasgow

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] Teevan et al. Slow Search: Information Retrieval without Time Constraints. HCIR'13

11

Why is Document Scoring Expensive?

Terrier Team | University of Glasgow

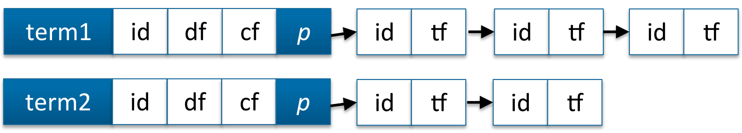
The largest reason for the expense of document scoring is that there are lots of documents:

– A Web search index can contain billions of documents
  - Google currently indexes trillions of pages [2]

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

– **Query length** (the number of search terms)

– **Posting list length** for each query term
  - i.e. The number of documents containing each term



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

12

Strategies to Speed-up Search

Terrier Team | University of Glasgow

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

Dynamic Pruning

- Skip the scoring of documents that are not likely to make the first few search result pages

Index compression

- Reduce the time it takes to read a posting list

13

Search Result/Term Caching

Terrier Team | University of Glasgow

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 effective

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

Caching is effective [3]:

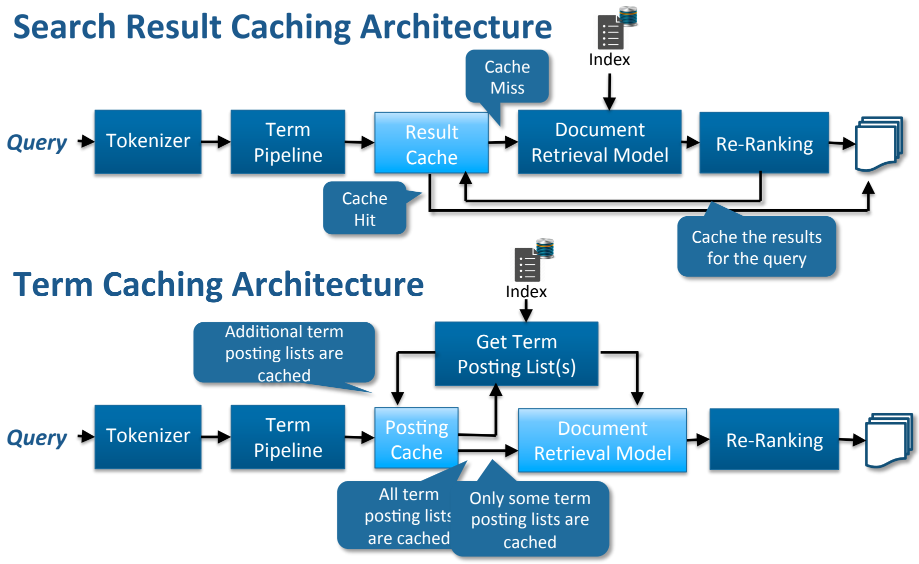
- Search Result caching can avoid scoring for 50% of queries – so called head queries
- Term caching can skip one or more posting lists for 88% of queries

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

14

Search Result/Term Caching

Terrier Team | University of Glasgow



15

More on Caching

Terrier Team | University of Glasgow

Memory is cheap…

- The logical consequence is that many search engines keep 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 much quicker these queries
- But we cannot store postings for all bigrams

Instead, frequently 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 [4]

- And decide on a per-query basis to use them or not [5]

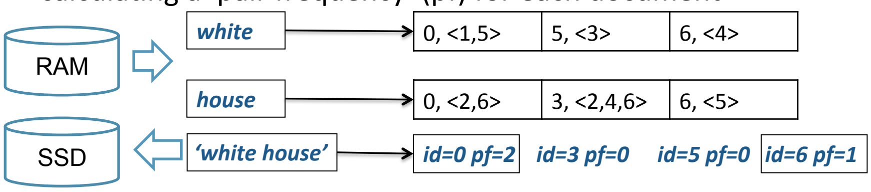
[4] Yan et al. Efficient Term Proximity Search with Term-Pair Indexes. CIKM 2010
[5] Risvik et al. Maguro, a system for indexing and searching over very large text collections. WSDM 2013

16

Paired Posting Lists

Consider a query `white house`

- 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

Query Evaluation & Dynamic Pruning

Even when using caching, retrieval still needs to score many documents, i.e. when the query/query terms are not in the cache

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

- This can be done Term-at-a-time or Document-at-a-time (all query terms in parallel)

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"*

Document Skipping

Terrier Team | University of Glasgow

The two most well known methods for document skipping (dynamic pruning) are MaxScore and WAND

MaxScore [6]

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

WAND [7]

– **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**)

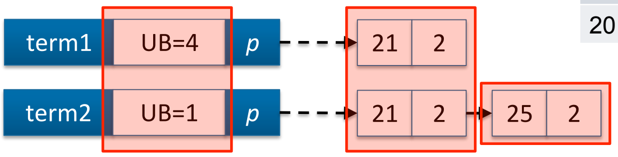
[6] H Turtle & J Flood. Query Evaluation : Strategies and Optimisations. IPM: 31(6). 1995.
[7] A Broder et al. Efficient Query Evaluation using a Two-Level Retrieval Process. CIKM 2003.

MaxScore Example

Terrier Team | University of Glasgow

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	

term scores calculated using BM25
term scores determined using upper bounds

Could make the top K!

WAND Example

Terrier Team | University of Glasgow

Require K=20 documents
Weighting model BM25

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

term1 | UB=4 | *p* → 21 | 2

term2 | UB=1 | *p* → 21 | 2 → 25 | 2

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

– So, 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

21

Some numbers…

Terrier Team | University of Glasgow

To demonstrate the benefit of dynamic pruning, we report experiments from [8]

- 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

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

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

22

Index Compression

Terrier Team | University of Glasgow

The previous two 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 [9]

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

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

term2 - - - → | 21 | 2 | → | 25 | 2 | → | 26 | 5 |

Do we need 32 bits?

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

23

Delta Gaps

Terrier Team | University of Glasgow

Most of the information in a posting list is docids…

- …in ascending order!

We can make smaller numbers by taking the differences

term2 - - - → | 21 | 2 | → | *4* | 2 | → | *1* | 5 |

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

24

Elias Unary & Gamma Encoding

Terrier Team | University of Glasgow

Unary:

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

Gamma:

- Let N= $\lfloor \log_2 x \rfloor$ be the highest power of 2 *x* contains;
- Write N out in unary representation, followed by x – N in binary
- Eg: 5 is represented as 00101

Lets represent docids as gamma, tf as unary

term2 - - -> | 21 | 2 | -> | 4 | 2 | -> | 1 | 5 |

Repr.	00001101
Bits	8

= 20 bits
< 3 bytes
down from 24!

(This is the default compression used by Terrier)

25

Other Compressions Schemes

Terrier Team | University of Glasgow

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

- Other oblivious schemes are byte-aligned, e.g.
  - Variable byte - *Williams, Zobel (1999)*
  - Simple family - *Ahn, Moffat (2005)*

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) [10]
  - Patched frame of reference (PFOR) [11]

[10] J. Goldstein et al. Compressing relations and indexes. ICDE 1998.
[11] M. Zukowski et al. Super-scalar RAM-CPU cache compression. ICDE 2006.

26

Frame of Reference

Terrier Team | University of Glasgow

Idea: pick the minimum *m* and the maximum *M* values of 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

Compression: Some numbers [12]

Terrier Team | University of Glasgow

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	size	time	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

[12] M Catena, C Macdonald, and I Ounis. On Inverted Index Compression for Search Engine Efficiency. ECIR 2014.

Efficient Query Evaluation

Terrier Team | University of Glasgow

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

- Each form important improvements to efficiency, often without affecting effectiveness

Other works I haven't covered include:

- Impact ordered posting lists: an alternative index layout
- Block-Max WAND: integrates WAND more tightly with the index block format

SkyNet

Ranking cascades & Learning to Rank

EFFICIENTLY INCREASING EFFECTIVENESS

Motivations for Learning

There are a plethora of weighting models

- Term weighting models have different assumptions about how relevant document should be retrieved, and varying effectiveness

Also:

- **Field-based models**: term occurrences in different fields matter differently
- **Proximity-models**: close co-occurrences matter more
- **Priors**: documents with particular lengths or URL/inlink distributions matter more
- ***Query Features:*** Long queries, difficult queries, query type

QUESTIONS:
How to combine all these easily and appropriately:
i.e. efficiently & effectively…

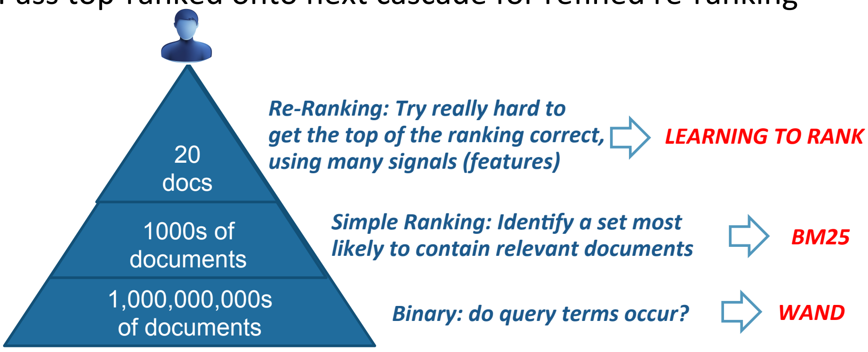
T.-Y. Liu. Learning to rank for information retrieval. Foundation and Trends in Information Retrieval, 3(3), 225–331. 2009
C Macdonald et al. The Whens & Hows of Learning to Rank. IR Journal. 16(5), 2012

31

Ranking Cascades

The ranking can be done as a cascading process [13]

- Rank some documents
- Pass top-ranked onto next cascade for refined re-ranking



[13] J Pederson. Query understanding at Bing. SIGIR 2010 Industry Day.

32

Learning to Rank

Terrier Team | University of Glasgow

Application of specialised machine learning techniques to automatically (select and) weight retrieval ***features***

- Based on training data with relevance assessments

Learning to rank has been popularised by several commercial search engines

- They require large training datasets, possibly instantiated from click-through data
- Click-through data has facilitated the deployment of learning approaches

T.-Y. Liu. (2009). Learning to rank for information retrieval. Foundation and Trends in Information Retrieval, 3(3), 225–331.

33

Infrastructure for Learning to Rank

Terrier Team | University of Glasgow

1. Sample Identification

- Apply BM25 or similar (e.g. DFR DPH) to rank documents with respect to the query
- Hope that the sample contains *enough* relevant documents

2. Compute more *features*

- Query Dependant: more weighting models
- Query Independent: e.g. PageRank, URL length

3A. Learn ranking model

- Based on training data

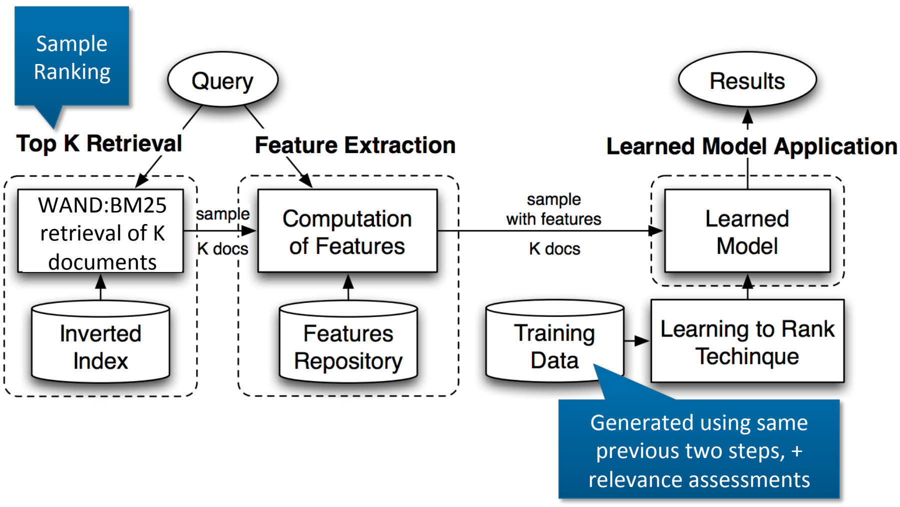
3B. Apply learned model

- Re-rank sample documents

34

Schematically…

Terrier Team | University of Glasgow



N. Tonellotto, C. Macdonald, I. Ounis. Efficient and effective retrieval using selective pruning. WSDM 2013.

35

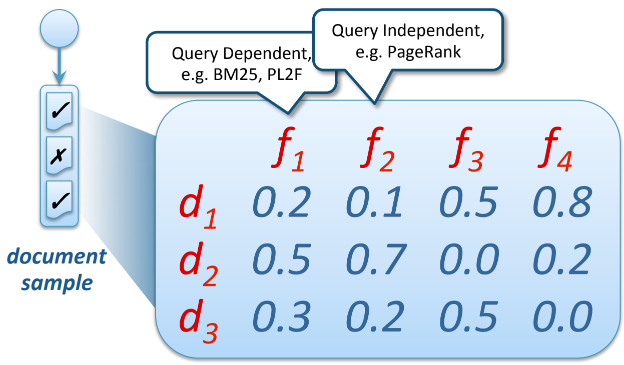
Features

Terrier Team | University of Glasgow

Query dependent
- Weighting models
- … including fields
- Proximity/ngram

Query independent
- Link Analysis, e.g. PageRank Score
- URL length
- Spam Score
- Document Type (PDF, Wikipedia)

Query features
- Query length
- Presence of entities
- Predicted query performance



36

Types of Learned Models (1) Linear Model

Terrier Team | University of Glasgow

Linear Models

- Many learning to rank techniques generate a linear combination of feature values:

$$score(d,Q) = \sum_f w_f \cdot value_f(d)$$

- Very efficient to apply
- These models have limitations:
  - ***Feature Usage***: Linear models assume that the same features are needed by all queries
  - ***Model Form***: Genetic algorithms can learn functional forms, by randomly introducing operators (e.g. try divide feature *a* by feature *b*)

37

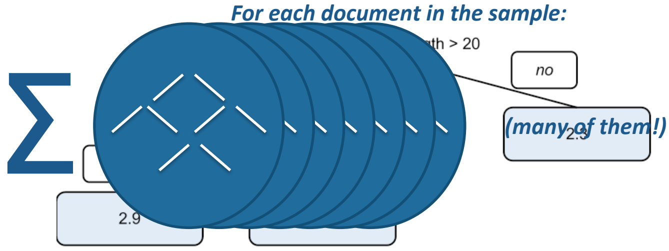
Type of Learned Models (2) Regression Trees

Terrier Team | University of Glasgow

Tree Models

- A *regression tree* is series of decisions, leading to a partial score output
- The outcome of the learner is a "forest" of many such trees, used to calculate the final score of a document for a query
- Their ability to customise branches makes them more effective than linear models
- Several major search engines use regression trees at the heart of their ranking model, e.g. Microsoft's LambdaMART

For each document in the sample:

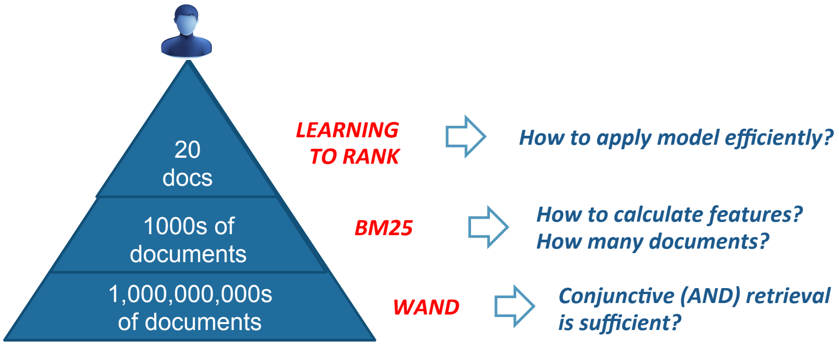


S. Tyree, K. Weinberger, K. Agrawal, J. Paykin. Parallel Boosted Regression Trees for Web Search Ranking. WWW 2011.

38

Making Efficient Ranking Cascades

Terrier Team | University of Glasgow



39

Candidate Generation

Terrier Team | University of Glasgow

Aim: to quickly identify candidates documents in the first cascade to score further

– Enough for high Recall; small enough to be efficient

Typically IR has focussed on disjunctive query processing

– This means term1 OR term2 OR term3

An alternative is using conjunctive processing to identify the first set before re-ranking

(NDCG@20)	BM25	Linear	LambdaMART
Disjunctive	0.21	0.25	0.26
Conjunctive	0.21	0.25	0.26

– Experiments in [14] showed no significant difference in NDCG@20 on ClueWeb09

[14] N Asadi. Effectiveness/Efficiency Tradeoffs for Candidate Generation in Multi-Stage Retrieval Architectures. SIGIR 2013.

40

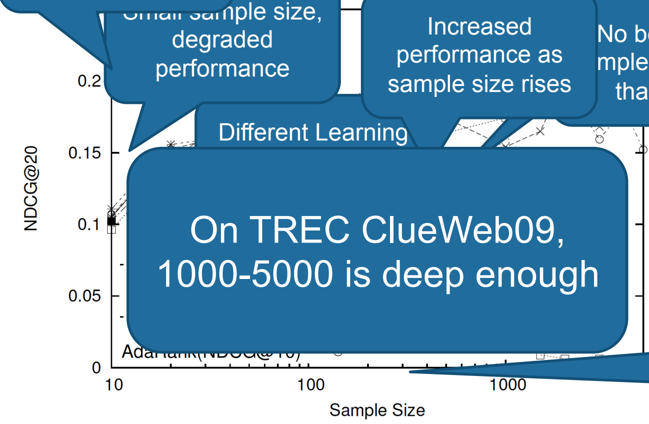
How many documents to rank?

Terrier Team | University of Glasgow

NDCG@20 for different BM25 cascade sizes

laxScore & WAND are faster for smaller K

can we get away with? [15]



ches have been proposed to vary K on a per-query basis: [16,17]

Number of documents being re-ranked

[15] C. Macdonald, R. Santos, B He and I. Ounis. (2012). The Whens & Hows of Learning to Rank, INRT
[16] L Wang, J Lin. A cascade ranking model for efficient ranked retrieval. SIGIR 2011.
[17] N Tonellotto, C Macdonald, I Ounis. Efficient and effective retrieval using selective pruning. WSDM 2013.

Query-Dependent Feature Extraction

Terrier Team | University of Glasgow

We might typically deploy a number of query dependent features

- e.g. more weighting models, proximity

We want the result set passed to the final cascade to have all query dependent features computed

- Once first cascade retrieval has ended, its too late to compute features without re-traversing the inverted file postings lists
- It takes too long to compute all features for all ranked documents

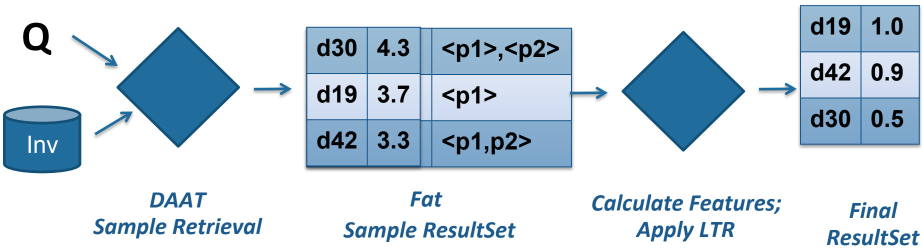
Proposition: cache the postings for documents that *might* make the sample

- Postings contain frequencies, positions, fields, etc.

Query-Dependent Feature Extraction

Terrier Team | University of Glasgow

Proposition: cache the postings for documents that *might* make the sample

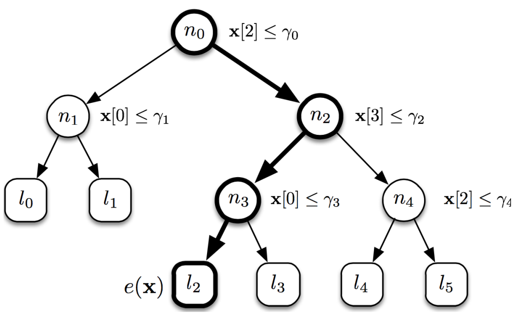


C Macdonald et al. (2012). About Learning Models with Multiple Query Dependent Features. TOIS 31(3).

43

Faster Trees

Terrier Team | University of Glasgow




```
if (x[2] <= 0.5) //γ0
 if (x[0] <= 0.9) //γ1
  return 0.1 //l0
 else
  return 0.5 //l1
else
 if (x[3]) <= 19) //γ2
 …
```

Slow: the processor cannot make good branch predictions

We can do better by vectorization [19]…

```
i0 = nd[i0].index[(v[nd[i0].fid] >= nd[i0].theta)];
i1 = nd[i1].index[(v[nd[i1].fid] >= nd[i1].theta)];
i2 = nd[i2].index[(v[nd[i2].fid] >= nd[i2].theta)];
i3 = nd[i3].index[(v[nd[i3].fid] >= nd[i3].theta)];
```

…or by representing the tree nodes as bitvectors [20]

[19] N Asadi et al. Runtime Optimizations for Tree-based Machine Learning Models. ToKDE 2014.
[20] Lucchese et al. QuickScorer: A Fast Algorithm to Rank Documents with Additive Ensembles of Regression Trees. SIGIR 2015

44

Summary

Terrier Team | University of Glasgow

We can re-rank the retrieved documents in cascades, to increase effectiveness

- Thereby making use of additional ranking features, both query dependent and query independent in nature, as well as query features
  - Weighting models, proximity, PageRank, URL length, Query Length

There are various infrastructure choices about how to design your IR system to give effective results while still remaining sufficiently fast enough:

- For the early cascade, we've seen techniques such as conjunctive retrieval…
- …while for the last cascade, recent research has focussed on fast regression tree application

45

Scaling Up

TALL vs WIDE SYSTEMS

46

Scaling Up Information Retrieval

Terrier Team | University of Glasgow

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 two main paradigms to scale up:

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

47

Vertical vs. Horizontal Scaling

Terrier Team | University of Glasgow

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

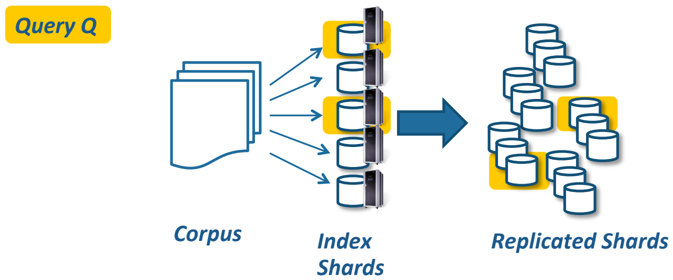
48

Parallelised Indexing and Retrieval

Terrier Team | University of Glasgow

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, then distributed indexing

49

Distributed Retrieval Architectures (1)

Terrier Team | University of Glasgow

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.

50

Distributed Retrieval Architectures (2)

Terrier Team | University of Glasgow

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

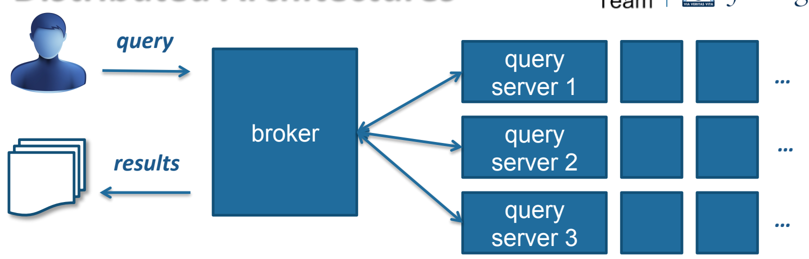
- Different documents are allocated to P different query servers
- During query processing, each server executes the query on N/P documents, so (for even partitioning), load is even on each query server
- The results from each of the servers are combined into a final result list

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

51

Distributed Architectures

Terrier Team | University of Glasgow



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

52

Distributed Querying

Terrier Team | University of Glasgow

This is a key disadvantage of term partitioning, which is rarely used in practice

However, hybrid architectures such as pipelining are possible

A di

- D ... ch q
- T ... ery s
  - W ... high scores on others

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

53

Document Partitioning Strategies

Terrier Team | University of Glasgow

A few variety of document partitioning strategies exist:

- Random – good for efficiency on large collections: all queries touch all partitions
- Semantic/topic – e.g. if collection is already organized 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

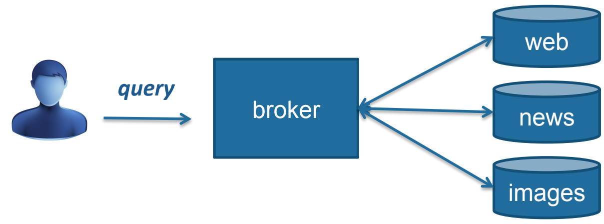
- E.g. by language
  - Also permits geographically distributed data centres: keep the Chinese index in Hong Kong
- E.g. by examining a query log, and clustering documents by the queries that touched them

54

Resource Selection & Aggregated Search

Terrier Team | University of Glasgow

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).
J Arguello, F Diaz, J Callan, J-F Crespo (2009). Sources of evidence for vertical selection. SIGIR.

55

Keep Up Efficiency!

Terrier Team | University of Glasgow

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 €!

Modern trends:

- Scheduling queries to least busy query servers [21]
- Changing the efficiency/effectiveness tradeoff according to query expense or current volume [22, 23]
- Selective parallelisation: using more cores for expensive queries [24]

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

56

Summary: Distributed Retrieval

Terrier Team | University of Glasgow

Distributed Retrieval environments…

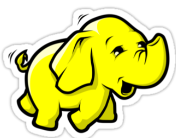
- …permit efficient retrieval across large-scale collections
- …are particularly applicable for Web-scale search engines

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…

Next up: how do we generate distributed indices?

57



Distributed Indexing with MapReduce

THE ELEPHANT IN THE ROOM

58

MapReduce

Terrier Team | University of Glasgow

Horizontal Scaling was popularised by Google and its MapReduce paradigm [25]

– Framework for distributing batch processing of large datasets over multiple machines

– **Idea**: Many tasks involve doing a simple map operation over each record in a large dataset

- E.g. Indexing, hyperlink analysis, spam detection

– MapReduce is not for search: it is for batch operations like inc

– hadoop is a popular open source implementation of M

Map | Map a function over each record in data

Reduce | Sort, Partition and Merge results

[25] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. Communications of the ACM. 51(1), 2008

59

Example: Indexing with MapReduce

Terrier Team | University of Glasgow

Document indexing can be described using MapReduce [26]

– The input collection is divided by document into input splits

Map

- Input <Docid, Document>
- Output <Term, Posting List>

The **Map task** builds partial posting lists for the documents that it sees

Sorting

- Partial posting lists are sorted and grouped by term

Reduce

- Input <Term, Posting List[]>
- Output <Term, Posting List>

The **Reduce task** combines all of the partial posting lists for a single term

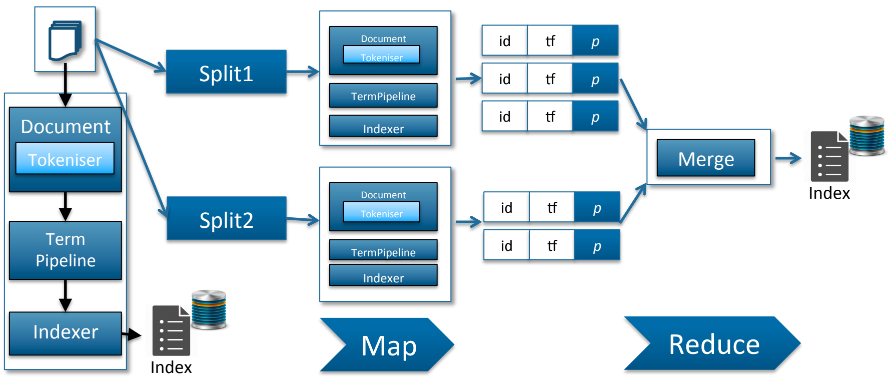
[26] R. McCreadie et al. MapReduce indexing strategies: Studying scalability and efficiency. IPM 48(5), 2011.

60

Example: Indexing with MapReduce

Terrier Team | University of Glasgow

MapReduce Indexing Architecture



Issues with MapReduce

Terrier Team | University of Glasgow

However, MapReduce is a batch-orientated paradigm

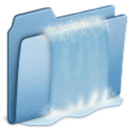
- A MapReduce job processes a fixed set of input data
- Follows a “Store-then-process” model of computation

Disadvantages:

- Complex tasks are difficult to represent as maps and reduces
  - Multiple Map and Reduce tasks may need to be chained together
- Lack of responsiveness
  - Batches need to be built before processing can start
  - Not well suited to document retrieval
  - Wasted time in job setup and for the coordination of processing

Moving toward real-time processing

STREAM ARCHITECTURES



63

Introducing Streams

Terrier Team | University of Glasgow

Modern search systems are not batch-orientated, but rather process data in a streaming manner

- Large volumes of documents are indexed continually over time, as they are crawled
- User queries also arrive in at very high rates and need to be processed quickly and in parallel

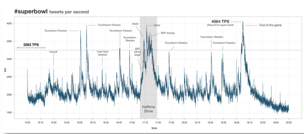
To tackle streaming data, we need to adapt our **search architecture**!

64

Introducing Streams

There are many **streams** that we might want to search:

The Challenges of Stream Processing



Thousands of documents need to be processed every second

- Consistently, forever

Stream input rate is not constant

- Twitter receives ~4600 tweets/second on average, but can burst at 12,000 tweets/second

Responses are needed fast

- E.g. for event detection, its no good detecting an event an hour after it happens
- Users expect new documents to be searchable immediately c.f. Twitter Search

Processing time must remain constant

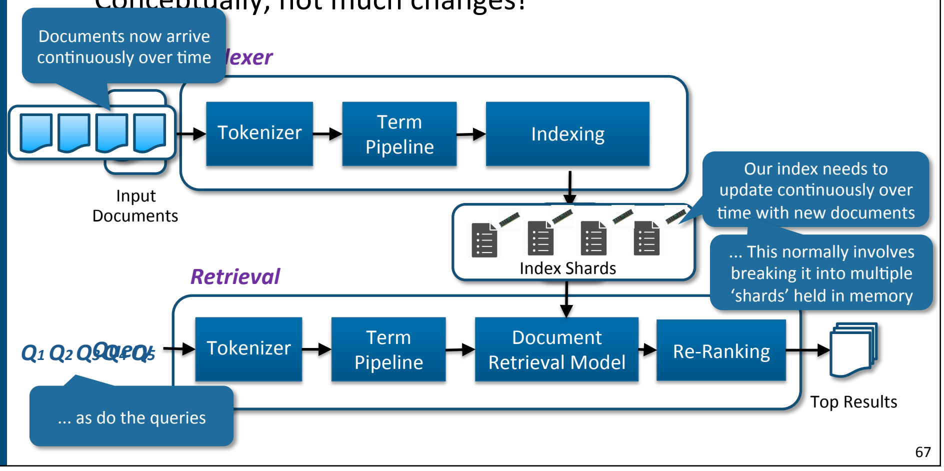
Stonebraker et al. The 8 requirements of real-time stream processing. ACM SIGMOD Record 2005.

Streaming Search Infrastructures

Terrier Team | University of Glasgow

So, how might we adapt the classical search architecture for streaming data?

– Conceptually, not much changes!



67

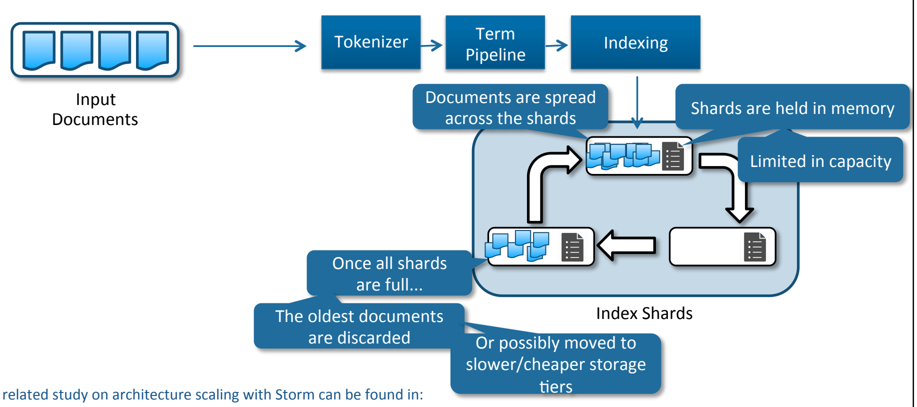
Example: Stream Indexing

Terrier Team | University of Glasgow

When indexing streams

– Documents are added to the index immediately and are searchable immediately

– Documents are partitioned across multiple index shards



A related study on architecture scaling with Storm can be found in:
McCreadie et al. Scalable Distributed Event Detection for Twitter. IEEE Big Data 2013

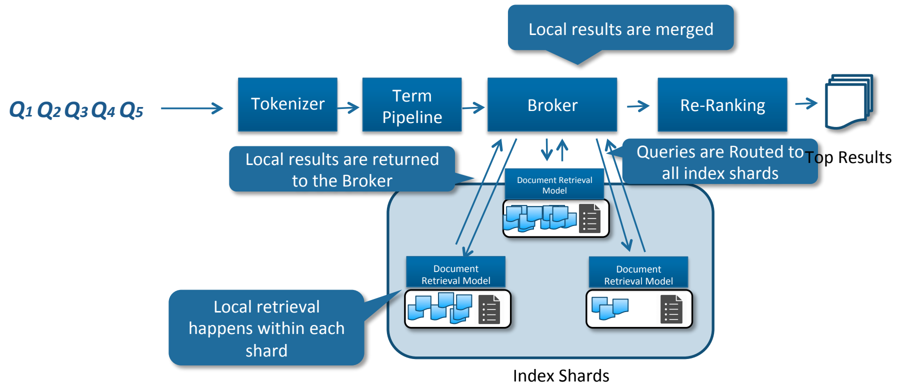
68

Example: Stream Search

Terrier Team | University of Glasgow

When searching streams

- Retrieval is distributed across all of the available index shards
- Partial results from each index are merged to generate the final results



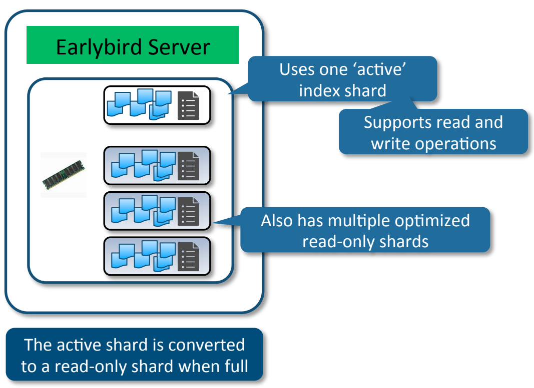
A related study on architecture scaling with Storm can be found in:
McCreadie et al. Scalable Distributed Event Detection for Twitter. IEEE Big Data 2013

69

Twitter Earlybird

Terrier Team | University of Glasgow

Earlybird was the real-time indexing and retrieval system that Twitter used to drive its search engine (circa 2011)

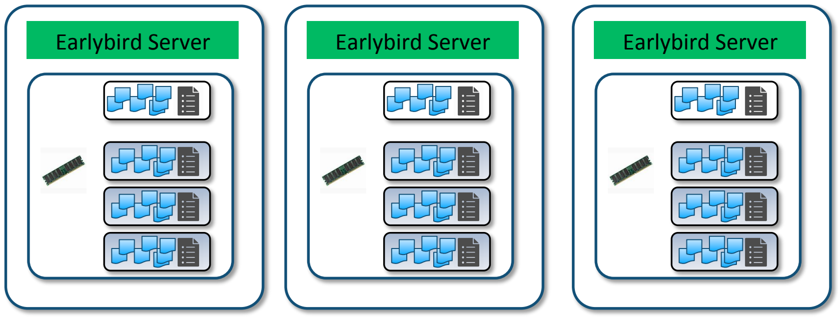


Busch et al. Earlybird: Real-time search at Twitter. In Proceedings of the IEEE 28th International Conference on Data Engineering (ICDE), 2011.

70

Twitter Earlybird

Earlybird was the real-time indexing and retrieval system that Twitter used to drive its search engine (circa 2011)



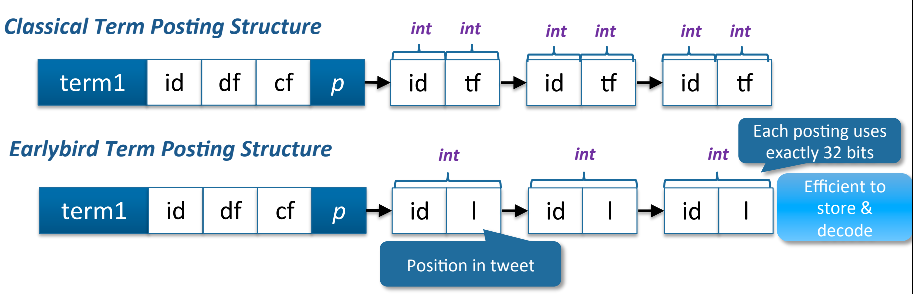
Multiple Earlybird servers (12 at the time) run in parallel, tweets are spread across these servers

71

The Twitter Earlybird Active Index

To enable the processing of very high rate document and query streams, the index structures need to be efficient

- Earlybird uses a unique internal index structure for its active index to make document addition as fast as possible



72

The Twitter Earlybird Active Index

To enable the processing of very high rate document and query streams, the index structures need to be efficient

– Earlybird uses a unique internal index structure for its active index to make document addition as fast as possible
– Postings are arranged into (4) fixed length arrays
  • Fast to traverse – only requires a linear memory scan
  • Predictable access pattern – helps hardware do pre-fetching

73

Complexities when processing streams

The overall simplicity of the above architecture hides significant complexities that do not exist for batch processing

– We need to distribute the architecture components across multiple machines while providing
  • Distributed Continuous Computation
  • Fault Tolerance
  • Avoid overheads when keeping the system synchronized
– When deployed, the indexing and search topology needs to be modular and flexible
  • Avoid bottlenecks on particular components
  • Ideally we want to be able to allocate more resources during hot periods with high document indexing or query loads

74

Stream Processing Platforms to the Rescue

Terrier Team | University of Glasgow

Recently, stream processing platforms have been created that help solve these issues

– Similar to hadoop for streams

Current Platforms

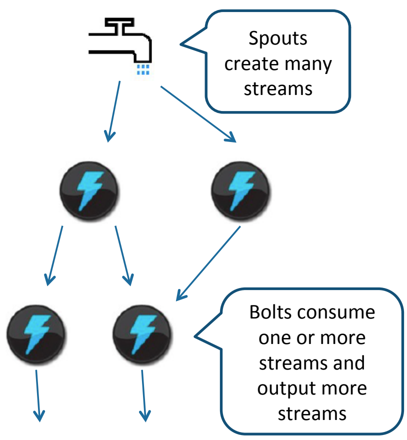
- Apache Storm — Easy to get started with and has an active community
- S4 (YAHOO!)
- Apache Spark — Appears to be quickly gaining traction
- IBM InfoSphere Streams
- TIBCO StreamBase
- Apache samza

See Storm vs S4, available at: McCredie et al. 2012 http://demeter.inf.ed.ac.uk/cross

Apache Storm

Terrier Team | University of Glasgow

Apache's Storm is one such popular processing platform



Storm topologies support:

- Storm defines computation as a graph of interconnected processing units
- Grouping of data passed between bolts and spouts
  - Similar to key grouping in MapReduce
- Fully in-memory computation
- Parallelism of each bolt
  - Can be distributed to different machines in a machine cluster
- Continuous low-latency processing

https://storm.apache.org/

Storm vs MapReduce

Terrier Team | University of Glasgow

When tackling Big Data Streams, Storm has the following advantages:

- Computation is done fully in memory
  - Faster processing by avoiding costly disk-seeks
- Computation is continuous
  - Documents are processed immediately as they arrive
  - Avoid start up costs inherent to MapReduce jobs
- Single complex topologies are possible in contrast to chaining multiple Map->Reduce operations

77

Performance of a Storm-based Search System

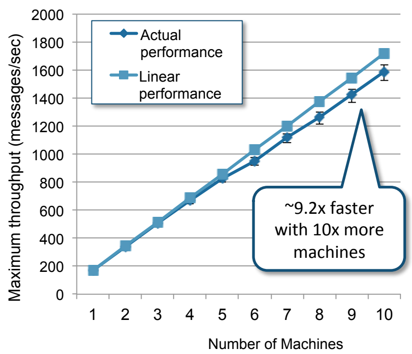
Terrier Team | University of Glasgow

So, how fast is a search system built using Storm?

- Test using a corpus of tweets

Using a cluster of commodity machines

- Scaling Performance
  - Achieves very close performance to perfect linear scaling
- Retrieval Latency
  - Sub-second response times on average for tweet search



Total Number of queries	Average response time	Maximum response time
3,500	171.46 ms	3.42 sec

SMART FP7 consortium. Deliverable D5.1.1, "SmartReduce Engine", 2012.
http://www.smartfp7.eu/public-deliverables

78

Stream Processing Summary

Terrier Team | University of Glasgow

Stream processing introduces new challenges to information retrieval systems

- –Indexing (and retrieving) documents as soon as they occur
- –High volumes and inconsistent (bursty) input rates
- –Retrieval models to identify new, relevant documents

Stream processing frameworks offer methods of easily distributing streamed processing across multiple nodes

- Storm, Spark etc.

79

WRAPUP

80

IR Infrastructure Summary

Terrier Team | University of Glasgow

IR has seen 30+ years of systems development to ensure retrieval that is both effective and efficient

- These two dimensions form a dichotomy: many techniques that can enhance effectiveness may be too expensive to deploy
- Recent years have seen particularly challenging environments, including Web search (scale) and real-time search (velocity)

The lecture covered a range of industry standard and more recent research:

- From caching & MaxScore to learning-to-rank, via distributed retrieval, MapReduce indexing and stream processing

Many techniques described here are widely implemented, including within open source platforms such as Terrier

81

REFERENCES

82

References (1)

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

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

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

[4] Yan et al. Efficient Term Proximity Search with Term-Pair Indexes. CIKM 2010

[5] Risvik et al. Maguro, a system for indexing and searching over very large text collections. WSDM 2013

[6] H Turtle & J Flood. Query Evaluation : Strategies and Optimisations. IPM: 31(6). 1995.

[7] A Broder et al. Efficient Query Evaluation using a Two-Level Retrieval Process. CIKM 2003.

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

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

[10] J. Goldstein et al. Compressing relations and indexes. ICDE 1998.

[11] M. Zukowski et al. Super-scalar RAM-CPU cache compression. ICDE 2006.

[12] M Catena, C Macdonald, and I Ounis. On Inverted Index Compression for Search Engine Efficiency. ECIR 2014.

[13] T.-Y. Liu. Learning to rank for information retrieval. Foundation and Trends in Information Retrieval, 3(3), 225–331. 2009

83

References (2)

[14] C Macdonald et al. The Whens & Hows of Learning to Rank. INRT. 16(5), 2012

[15] J Pederson. Query understanding at Bing. SIGIR 2010 Industry Day.

[16] N. Tonellotto, C. Macdonald, I. Ounis. Efficient and effective retrieval using selective pruning. WSDM 2013.

[17] S. Tyree, K. Weinberger, K. Agrawal, J. Paykin. Parallel Boosted Regression Trees for Web Search Ranking. WWW 2011.

[18] N Asadi. Effectiveness/Efficiency Tradeoffs for Candidate Generation in Multi-Stage Retrieval Architectures. SIGIR 2013.

[19] L Wang, J Lin. A cascade ranking model for efficient ranked retrieval. SIGIR 2011.

[20] N Tonellotto, C Macdonald, I Ounis. Efficient and effective retrieval using selective pruning. WSDM 2013.

[21] C Macdonald et al. (2012). About Learning Models with Multiple Query Dependent Features. TOIS 31(3).

[22] N Asadi et al. Runtime Optimizations for Tree-based Machine Learning Models. ToKDE 2014.

[23] Lucchese et al. QuickScorer: A Fast Algorithm to Rank Documents with Additive Ensembles of Regression Trees. SIGIR 2015.

[24] Baeza-Yates et al. Challenges on distributed web retrieval. ICDE 2007.

[25] A Moffat, W Webber, J Zobel, R Baeza-Yates (2005). A pipelined architecture for distributed text query evaluation. INRT 10

84

References (3)

[26] M Shokouhi and L Si (2011), Federated Search, Foundations and Trends in IR 5(1).

[27] J Arguello, F Diaz, J Callan, J-F Crespo (2009). Sources of evidence for vertical selection. SIGIR.

[28] C Macdonald et al (2012). Learning to predict response times for online query scheduling. SIGIR.

[29] D Brocoolo et al (2013). Load-Sensitive Selective Pruning for Distributed Search. CIKM.

[30] N Tonellotto et al (2013). Efficient and effective retrieval using selective pruning. WSDM.

[31] M Jeon et al. (2014). Predictive parallelization: taming tail latencies in web search. SIGIR.

[32] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. Communications of the ACM 51(1), 2008.

[33] R. McCreadie et al. MapReduce indexing strategies: Studying scalability and efficiency. IPM 48(5), 2011.

[34] Stonebraker et al. The 8 requirements of real-time stream processing. ACM SIGMOD Record 2005.

[35] McCreadie et al. Scalable Distributed Event Detection for Twitter. IEEE Big Data 2013

[36] SMART FP7 consortium. Deliverable D5.1.1, "SmartReduce Engine", 2012. http://www.smartfp7.eu/public-deliverables