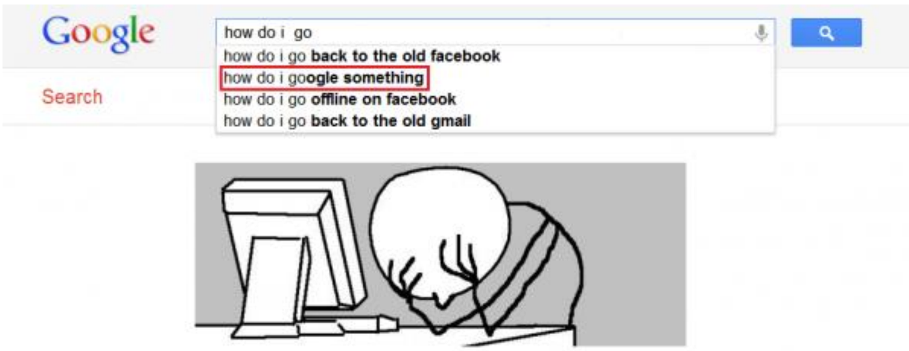
Data-Intensive Distributed Computing

CS 431/631 451/651 (Fall 2021)

Part 4: Analyzing Text (2/2)

Ali Abedi

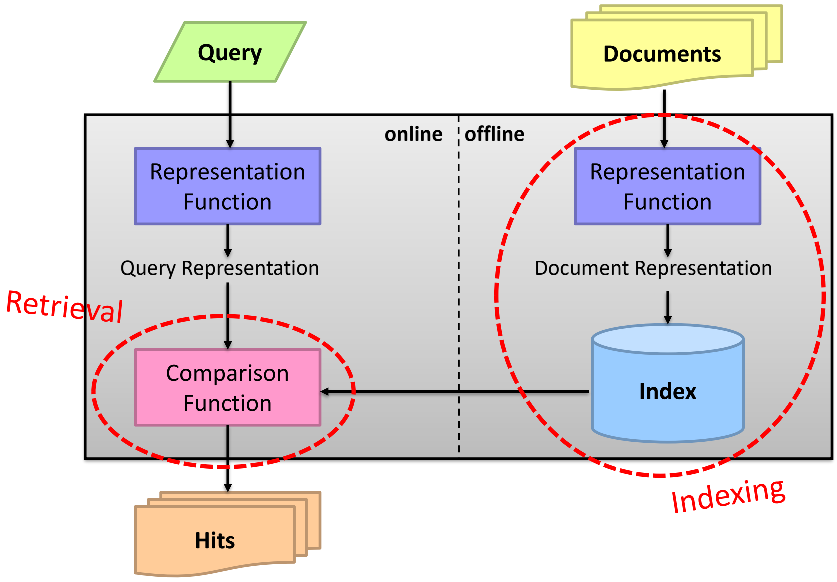
These slides are available at https://www.student.cs.uwaterloo.ca/~cs451


Google
how do i go
how do i go back to the old facebook
how do i google something
how do i go offline on facebook
how do i go back to the old gmail
Search

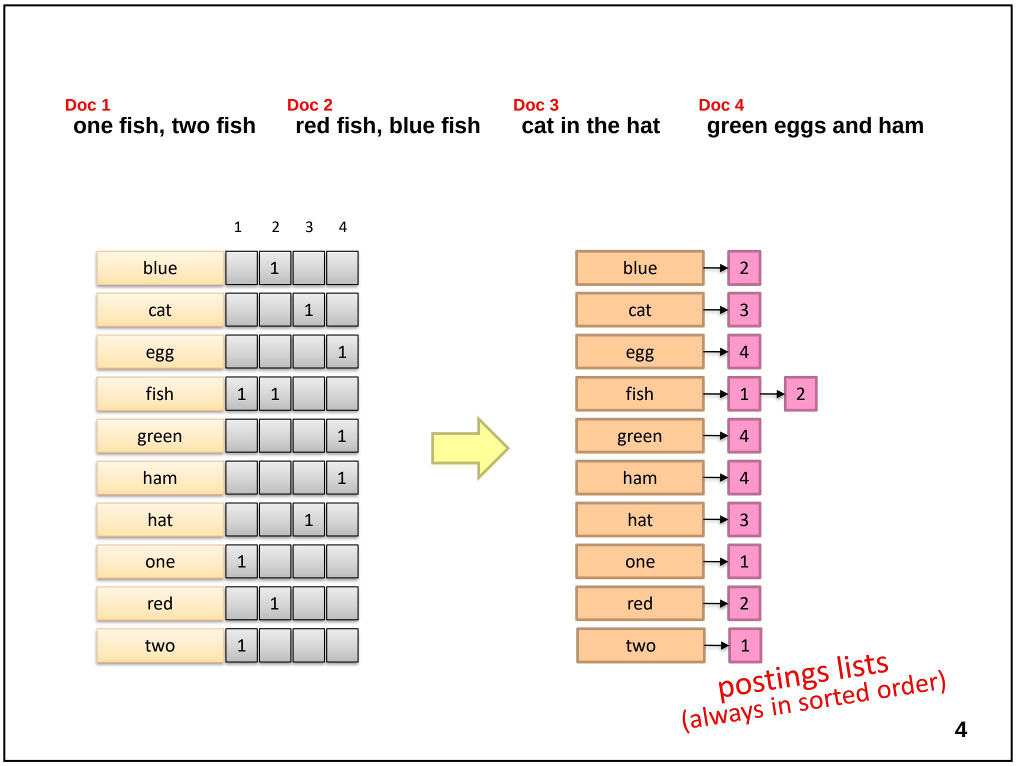
Search!

2

Abstract IR Architecture



3


Doc 1
one fish, two fish
Doc 2
red fish, blue fish
Doc 3
cat in the hat
Doc 4
green eggs and ham
1 2 3 4
blue 1
cat 1
egg 1
fish 1 1
green 1
ham 1
hat 1
one 1
red 1
two 1
blue 2
cat 3
egg 4
fish 1 2
green 4
ham 4
hat 3
one 1
red 2
two 1
postings lists
(always in sorted order)
4

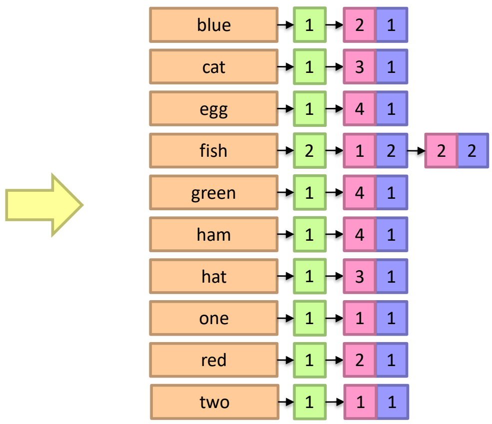
Doc 1 one fish, two fish

Doc 2 red fish, blue fish

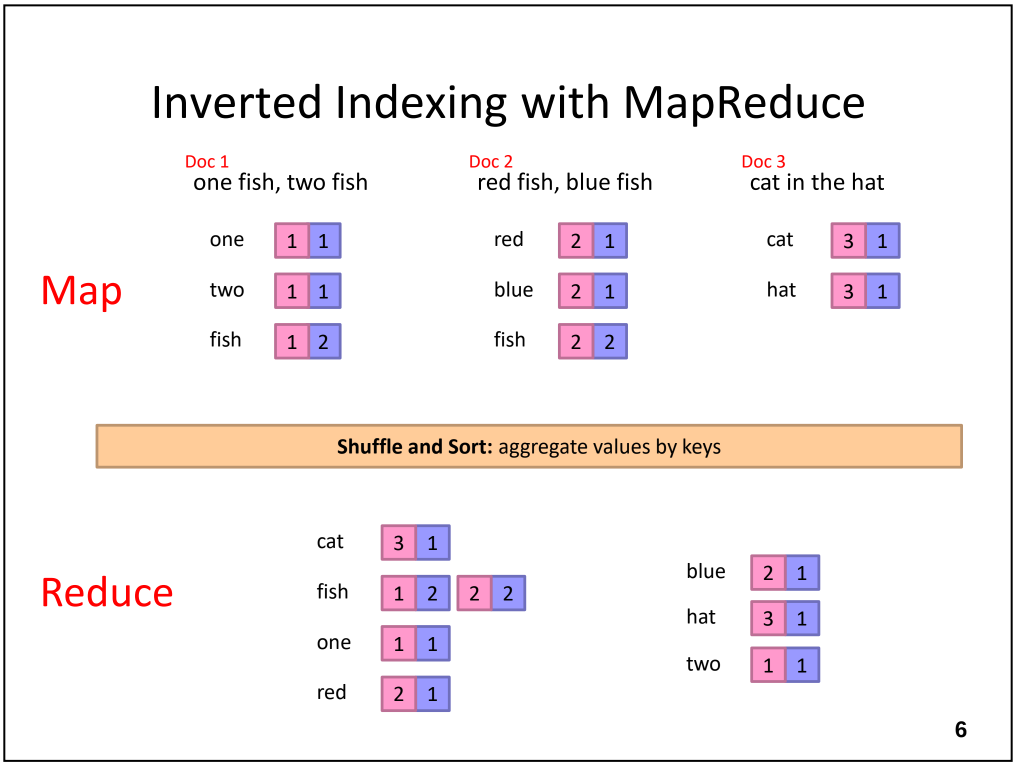
Doc 3 cat in the hat

Doc 4 green eggs and ham

	tf 1	2	3	4	*df*
blue		1			1
cat			1		1
egg				1	1
fish	2	2			2
green				1	1
ham				1	1
hat			1		1
one	1				1
red		1			1
two	1				1



5

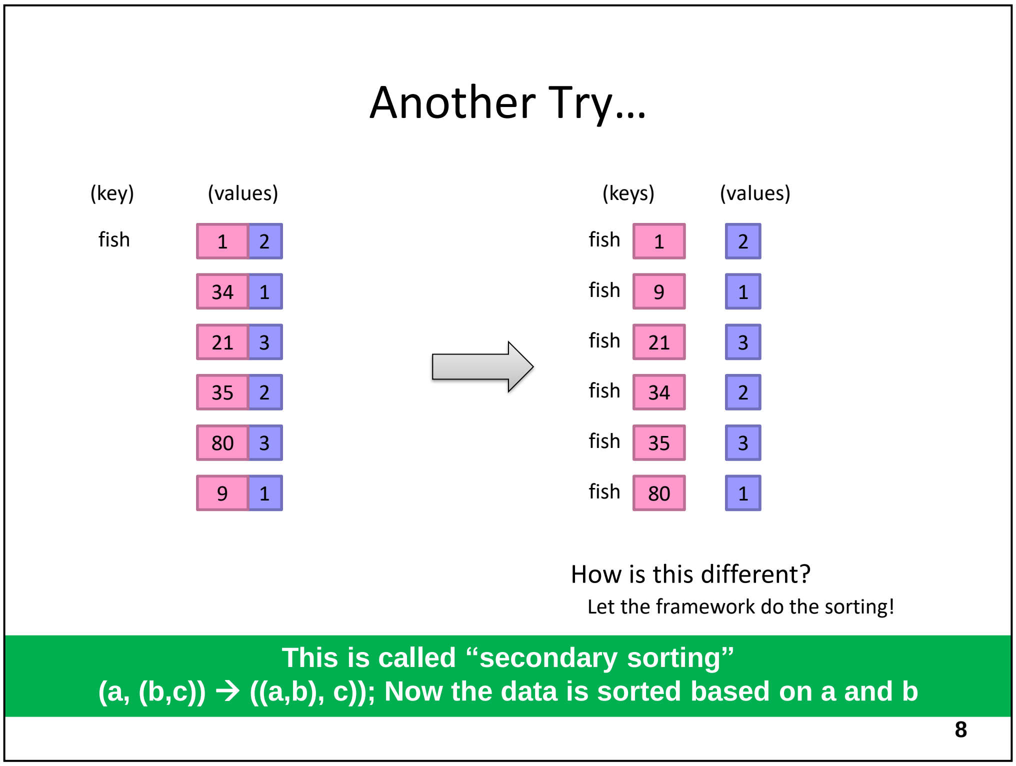

Inverted Indexing with MapReduce
Doc 1
one fish, two fish
Doc 2
red fish, blue fish
Doc 3
cat in the hat
Map
one 1 1
two 1 1
fish 1 2
red 2 1
blue 2 1
fish 2 2
cat 3 1
hat 3 1
Shuffle and Sort: aggregate values by keys
Reduce
cat 3 1
fish 1 2 2 2
one 1 1
red 2 1
blue 2 1
hat 3 1
two 1 1
6

Inverted Indexing: Pseudo-Code

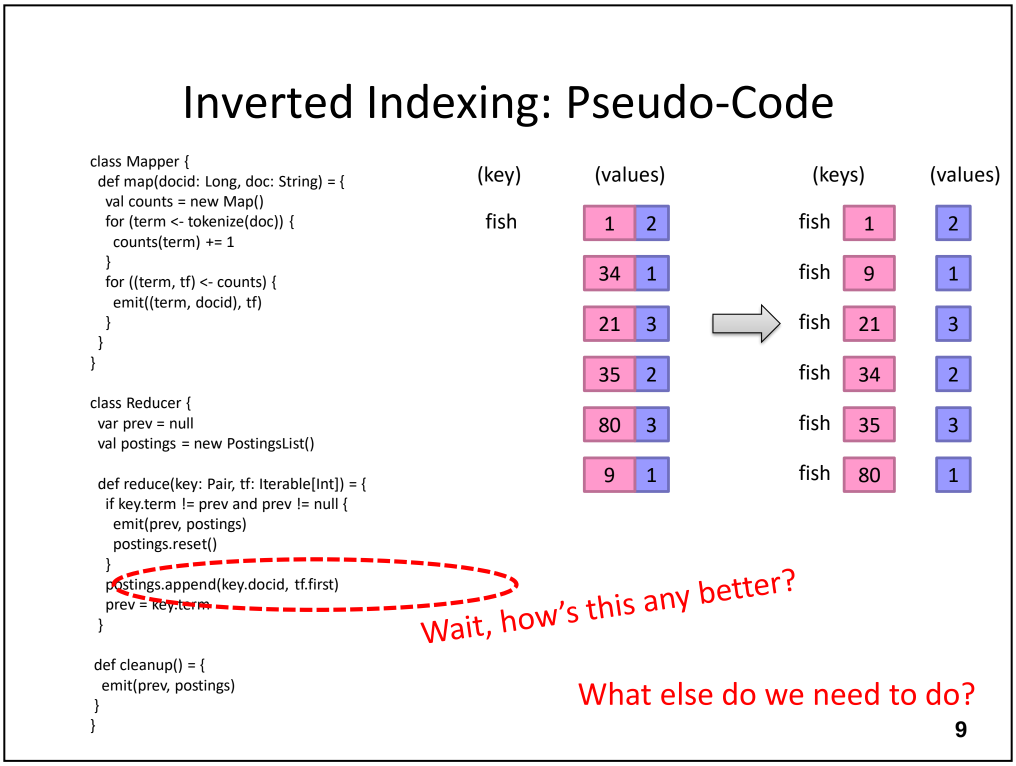
```
class Mapper {
  def map(docid: Long, doc: String) = {
    val counts = new Map()
    for (term <- tokenize(doc)) {
      counts(term) += 1
    }
    for ((term, tf) <- counts) {
      emit(term, (docid, tf))
    }
  }
}

class Reducer {
  def reduce(term: String, postings: Iterable[(docid, tf)]) = {
    val p = new List()
    for ((docid, tf) <- postings) {
      p.append((docid, tf))
    }
    p.sort()
    emit(term, p)
  }
}
```

What's the problem?

7



MapReduce sorts the data only based on the key. So if we need the data to be sorted based on a part of the value, we need to move that part to the key.



We still have the memory overflow issue, but the different is that now key.docid is sorted when we add them to the list. As a result, we can compress these values using integer compression techniques to reduce the size of the list.

Postings Encoding

Conceptually:

fish	1	2	9	1	21	3	34	1	35	2	80	3	…

In Practice:

Don’t encode docids, encode gaps (or *d*-gaps)
But it’s not obvious that this save space…

fish	1	2	8	1	12	3	13	1	1	2	45	3	…

= delta encoding, delta compression, gap compression

Overview of Integer Compression

Byte-aligned technique

VarInt (Vbyte)

Group VarInt

Word-aligned

Simple family

Bit packing family (PForDelta, etc.)

Bit-aligned

Unary codes

γ/δ codes

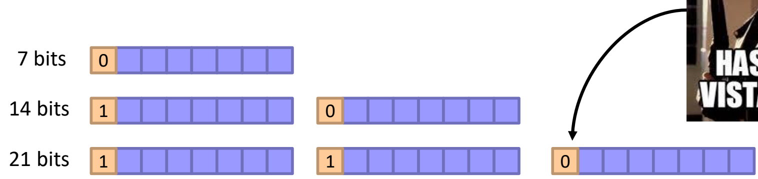
Golomb codes (local Bernoulli model)

VarInt (Vbyte)

Simple idea: use only as many bytes as needed

Need to reserve one bit per byte as the "continuation bit"

Use remaining bits for encoding value



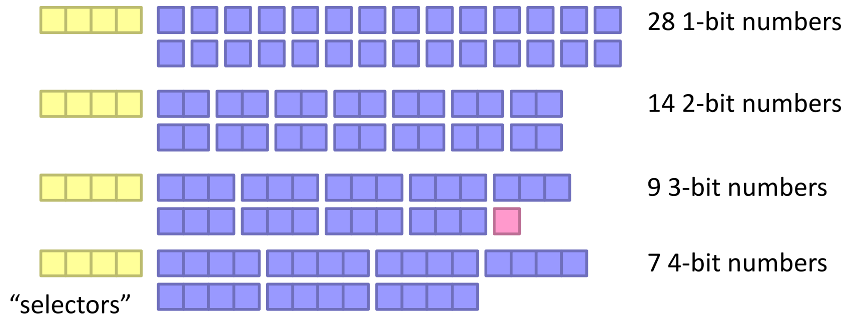
Works okay, easy to implement…

Beware of branch mispredicts!

12

Simple-9

How many different ways can we divide up 28 bits?



(9 total ways)

Efficient decompression with hard-coded decoders

Simple Family – general idea applies to 64-bit words, etc.

13

Golomb Codes

$x \geq 1$, parameter M: $q = \left\lfloor \frac{(x-1)}{M} \right\rfloor$ **Encoded in unary**

$r = x - qM - 1$ **Encoded in truncated binary**

Final result: (q + 1) r

Example:

- $M = 3$, $r = 0, 1, 2$ (0, 10, 11)
- $M = 6$, $r = 0, 1, 2, 3, 4, 5$ (00, 01, 100, 101, 110, 111)
- $x = 9$, $M = 3$: $q = 2$, $r = 2$, code = 110:11
- $x = 9$, $M = 6$: $q = 1$, $r = 2$, code = 10:100

Punch line: optimal $M \sim 0.69\,(N/df)$

Different M for every term!

14

N = Number of documents

Df = document frequency (the number of documents a term appears in)

Inverted Indexing: Pseudo-Code

```
class Mapper {
 def map(docid: Long, doc: String) = {
  val counts = new Map()
  for (term <- tokenize(doc)) {
   counts(term) += 1
  }
  for ((term, tf) <- counts) {
   emit((term, docid), tf)
  }
 }
}

class Reducer {
 var prev = null
 val postings = new PostingsList()

 def reduce(key: Pair, tf: Iterable[Int]) = {
  if key.term != prev and prev != null {
   emit(prev, postings)
   postings.reset()
  }
  postings.append(key.docid, tf.first)
  prev = key.term
 }

 def cleanup() = {
  emit(prev, postings)
 }
}
```

Ah, now we know why this is different!

15

We can perform integer compression now!

Chicken and Egg?

	(key)	(value)
fish	1	2
fish	9	1
fish	21	3
fish	34	2
fish	35	3
fish	80	1

…

Write postings *compressed*

But wait! How do we set the Golomb parameter *M*?

Recall: optimal *M* ~ 0.69 (*N/df*)

We need the *df* to set *M*…

But we don't know the *df* until we've seen all postings!

Sound familiar?

16

The problem is that we cannot calculate df until we see all fish *s

Getting the *df*

In the mapper:

Emit "special" key-value pairs to keep track of *df*

In the reducer:

Make sure "special" key-value pairs come first: process them to determine *df*

Remember: proper partitioning!

Getting the *df*: Modified Mapper

Doc 1
one fish, two fish

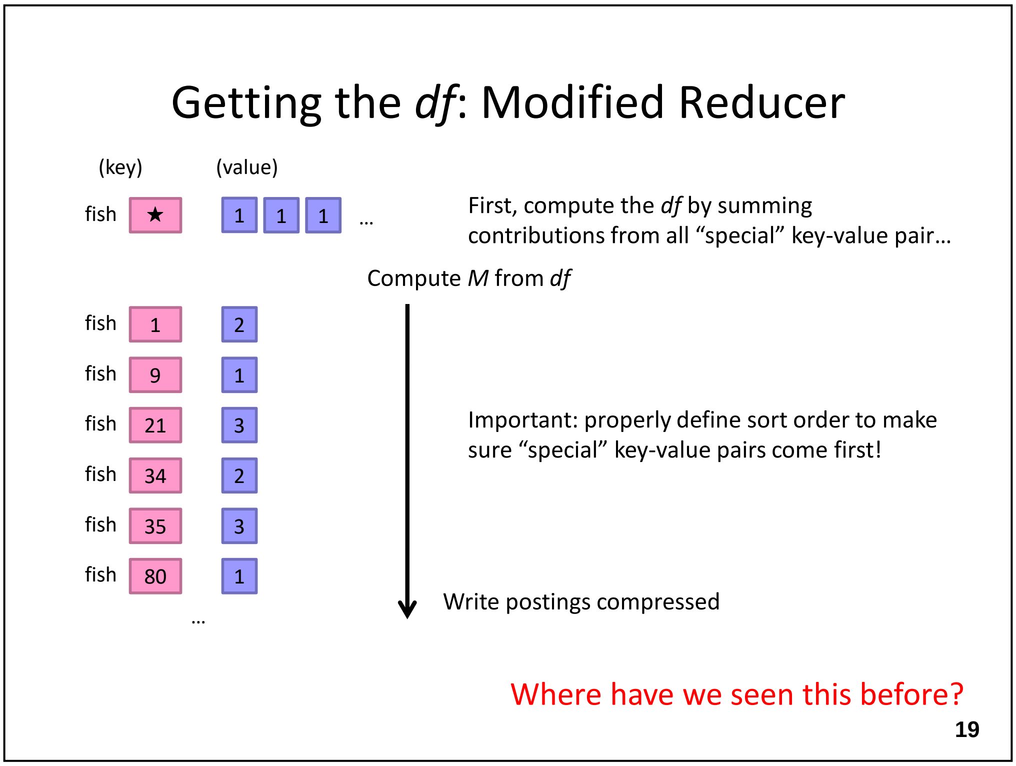
Input document…

	(key)	(value)
fish	1	2
one	1	1
two	1	1

Emit normal key-value pairs…

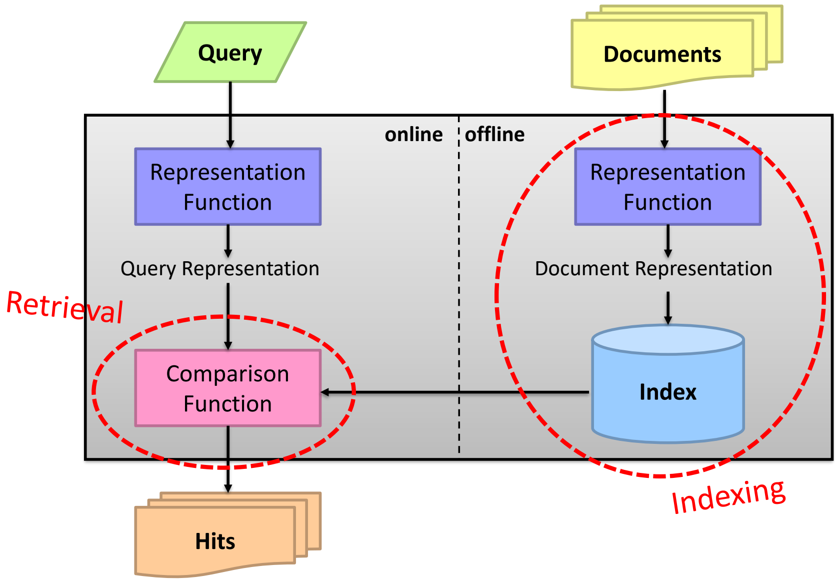
	(key)	(value)
fish	★	1
one	★	1
two	★	1

Emit “special” key-value pairs to keep track of *df*…



We have see this before in the pairs implementation of f(B|A) i.e., part 2b

Abstract IR Architecture



20

MapReduce it?

The indexing problem

Perfect for MapReduce!

- Scalability is critical
- Must be relatively fast, but need not be real time
- Fundamentally a batch operation
- Incremental updates may or may not be important
- For the web, crawling is a challenge in itself

The retrieval problem

- Must have sub-second response time
- For the web, only need relatively few results

Uh… not so good…

21

Assume everything fits in memory on a single machine…

Boolean Retrieval

Users express queries as a Boolean expression

AND, OR, NOT

Can be arbitrarily nested

Retrieval is based on the notion of sets

Any query divides the collection into two sets: retrieved, not-retrieved

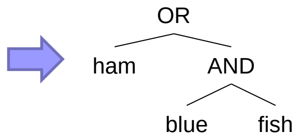
Pure Boolean systems do not define an ordering of the results

Boolean Retrieval

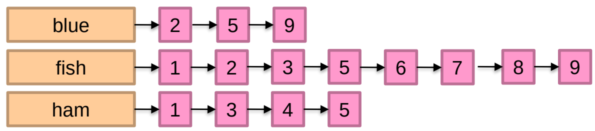
To execute a Boolean query:

Build query syntax tree

(blue AND fish) OR ham



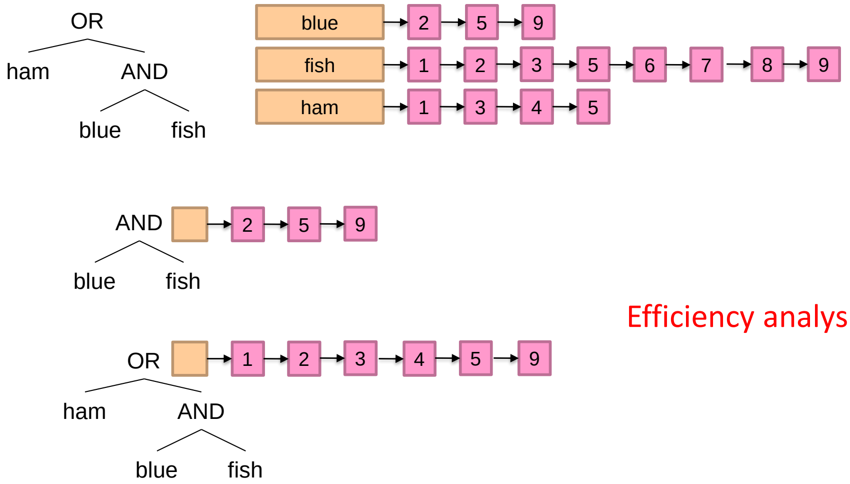
For each clause, look up postings



Traverse postings and apply Boolean operator

24

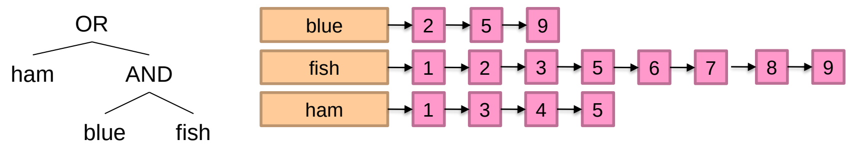
Term-at-a-Time

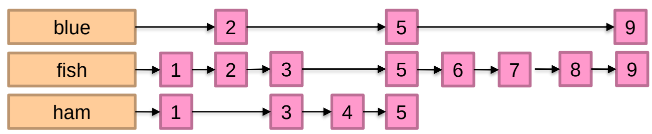


Efficiency analysis?

25

Document-at-a-Time





Tradeoffs?
Efficiency analysis?

Boolean Retrieval

Users express queries as a Boolean expression

AND, OR, NOT

Can be arbitrarily nested

Retrieval is based on the notion of sets

Any query divides the collection into two sets: retrieved, not-retrieved

Pure Boolean systems do not define an ordering of the results

What's the issue?

Ranked Retrieval

Order documents by how likely they are to be relevant

- Estimate relevance(q, d_i)
- Sort documents by relevance

Term Weighting

Term weights consist of two components

Local: how important is the term in this document?

Global: how important is the term in the collection?

Here's the intuition:

Terms that appear often in a document should get high weights

Terms that appear in many documents should get low weights

How do we capture this mathematically?

Term frequency (local)

Inverse document frequency (global)

29

TF-IDF* Term Weighting

$$w_{i,j} = \mathrm{tf}_{i,j} \cdot \log \frac{N}{n_i}$$

$w_{i,j}$ weight assigned to term *i* in document *j*

$\mathrm{tf}_{i,j}$ number of occurrence of term *i* in document *j*

N number of documents in entire collection

n_i number of documents with term *i*

*Term Frequency-Inverse Document Frequency

Retrieval in a Nutshell

Look up postings lists corresponding to query terms

Traverse postings for each query term

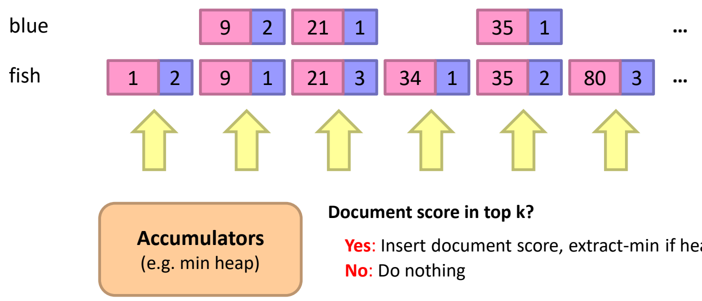
Store partial query-document scores in accumulators

Select top *k* results to return

31

Retrieval: Document-at-a-Time

Evaluate documents one at a time (score all query terms)



Document score in top k?

- **Yes**: Insert document score, extract-min if heap too large
- **No**: Do nothing

Tradeoffs:

Small memory footprint (good)

Skipping possible to avoid reading all postings (good)

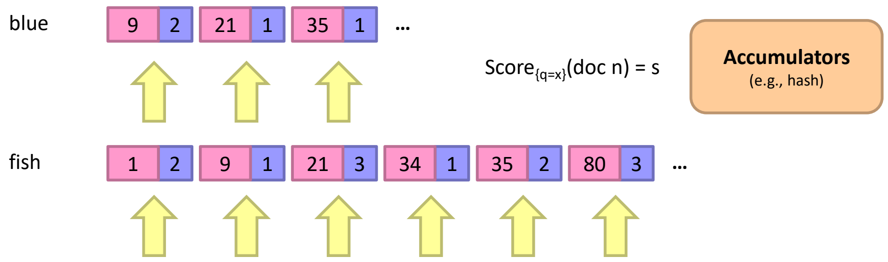
More seeks and irregular data accesses (bad)

32

Retrieval: Term-At-A-Time

Evaluate documents one query term at a time

Usually, starting from most rare term (often with *tf*-sorted postings)



Tradeoffs:

Early termination heuristics (good)

Large memory footprint (bad), but filtering heuristics possible

33