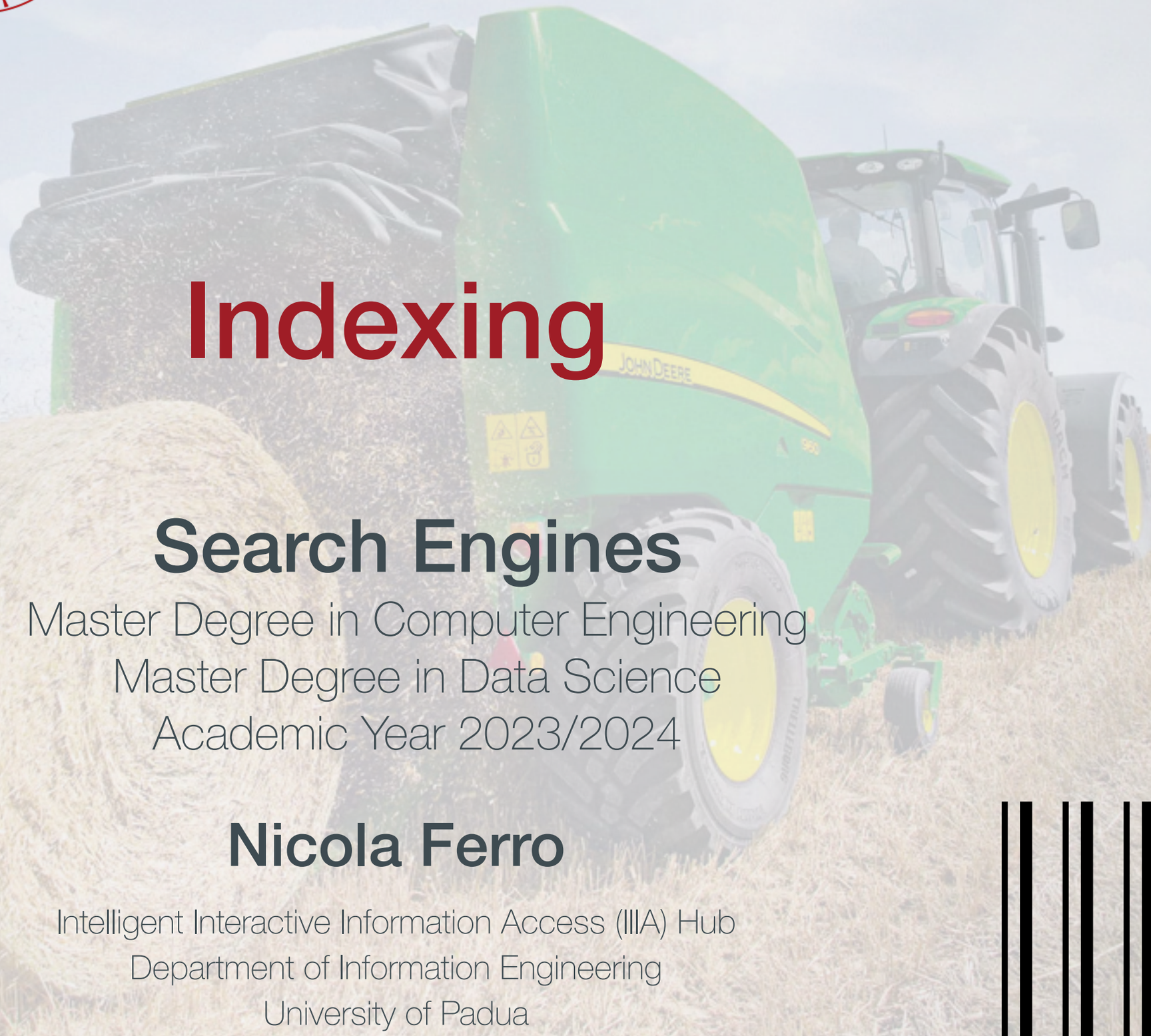


Indexing

Search Engines

Master Degree in Computer Engineering
Master Degree in Data Science
Academic Year 2023/2024

Nicola Ferro

Intelligent Interactive Information Access (IIIA) Hub
Department of Information Engineering
University of Padua

Outline

- Basics of Indexing
- Index Construction
- Basics of Query Processing

INDEX in database: struttura dati ausiliaria per offrire metodi alternativi di accesso senza cambiare layout fisico

⇓

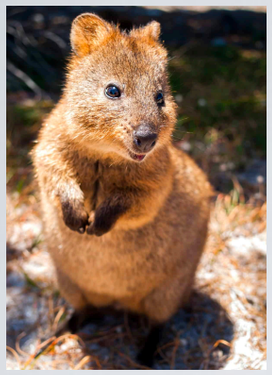
molto + piccolo di DB, tenuto sorted, servono ad accelerare query ⇒ ad ogni modifica dati, cambia index

composto da puntatori a record ⇒ valori che ci interessano e puntatori

possiamo averne di più

Basics of Indexing

General Model of Ranking



Document

d1

The quokka, the only member of the genus Setonix, is a small marsupial about the size of a domestic cat.

Indexing

a	0.7
about	1.4
cat	2.7
domestic	1.8
genus	3.2
is	0.9
marsupial	9.7
member	3.9
of	0.6
only	1.1
quokka	10.5
setonix	11.3
size	4.7
small	3.6
the	0.6

Topical Features

incoming links	16
page rank	37
update counts	18
clicks	150
likes	20
…	…

Quality Features

d_i

q_i

Matching

$$R(q,d) = \sum_i q_i d_i$$

201.17

Document Score

marsupial	4.5
small	3.2
mammal	1.3
little	2.2

Topical Features

incoming links	1
page rank	0
update counts	5
clicks	0
likes	2
…	…

Quality Features

Indexing

Query

small marsupial

Indexes and Ranking

solo index, no index+database ⇒ ne abbiamo solo uno

- **Indexes** are **data structures** designed to support search

  pulizia testo ⇓ |index| = |data| / 2

  - avoid linear scan, faster response time, supports updates
- Text search engines use a particular form of search: **ranking**
  - documents are retrieved in sorted order according to a score computing using the document representation, the query, and a ranking algorithm

- **Efficiency** of an IR system:
  - **indexing time**: time needed to build the index
  - **indexing space**: space used during the generation of the index;
  - **index storage**: space required to store the index, once it has been generated;
  - **query latency**: time interval between the arrival of the query in the IR system and the generation of the answer. Alternatively **query throughput**, i.e. number of queries per second

Inverted Index

- **Inverted index** s a data structure providing a mapping between **terms** (tokens) and their locations in documents
  - "inverted" because documents are associated with terms, rather than terms with documents
- **Dictionary (vocabulary, lexicon):** list of all distinct index terms in the collection
- Components of an inverted index
  - Each document in the collection is given a **unique number**
  - Each term is associated with a **posting list**: contains list of documents (or locations) where the term appears and other information (**feature scores**)
  - The part of the posting that refers to a specific document or location is called a **pointer**
  - Lists are usually **document-ordered** (sorted by document number)

Index

Term Posting List Pointer

(Toy) Corpus

- `d1` – The quokka, the only member of the genus Setonix, is a small marsupial about the size of a domestic cat.
- `d2` – Wombats are small, short-legged, muscular quadrupedal marsupials that are native to Australia.
- `d3` – Quokkas have little fear of humans and commonly approach people closely, particularly on Rottnest Island in Australia, where a prevalent population exists.

[credits to Maria Maistro]

Inverted Index: Just Documents

boolean query

- It allows for retrieving documents containing any or all query terms
  - Q: `small marsupial`
    → **intersection** if searching for all the terms;
    union if searching for any terms
- No ranking possible
- No proximity match/ phrase match possible

no info su posizione

Dictionary	Posting List
a	1 3
about	1
and	3
approach	3
are	2
australia	2 3
cat	1
closely	3
commonly	3
domestic	1
exists	3
fear	3
genus	1
have	3
humans	3
in	3
is	1
island	3
legged	2
little	3
marsupial	1
marsupials	2
member	1
muscular	2
native	2
of	1 3
on	3
only	1
particularly	3
people	3
population	3
prevalent	3
quadrupedal	2
quokka	1
quokkas	3
rottnest	3
setonix	1
short	2
size	1
small	1 2
that	2
the	1
to	2
where	3
wombats	2

Inverted Index: Documents and Frequencies

vector space model

- It allows for ranking documents by their relevance
- No proximity match/phrase match possible

Dictionary	Posting List
a	1:2 3:1
about	1:1
and	3:1
approach	3:1
are	2:2
australia	2:1 3:1
cat	1:1
closely	3:1
commonly	3:1
domestic	1:1
exists	3:1
fear	3:1
genus	1:1
have	3:1
humans	3:1
in	3:1
is	1:1
island	3:1
legged	2:1
little	3:1
marsupial	1:1
marsupials	2:1
member	1:1

Dictionary	Posting List
muscular	2:1
native	2:1
of	1:2 3:1
on	3:1
only	1:1
particularly	3:1
people	3:1
population	3:1
prevalent	3:1
quadrupedal	2:1
quokka	1:1
quokkas	3:1
rottnest	3:1
setonix	1:1
short	2:1
size	1:1
small	1:1 2:1
that	2:1
the	1:4
to	2:1
where	3:1
wombats	2:1

Inverted Index: Documents and Positions

scelte che si possono fare in Lucene

- It allows for ranking documents by their relevance
- It allows for phrase match
  - Q: `"small marsupial"`
- It allows for proximity match
  - Q: `find small within 5 words of marsupials`

Dictionary	Posting List
a	1:10,17 3:18
about	1:13
and	3:6
approach	3:8
are	2:1,9
australia	2:12 3:16
cat	1:19
closely	3:10
commonly	3:7
domestic	1:18
exists	3:21
fear	3:3
genus	1:7
have	3:1
humans	3:5
in	3:15
is	1:9
island	3:14
legged	2:4
little	3:2
marsupial	1:**12**
marsupials	2:**7**
member	1:4
muscular	2:5
native	2:10
of	1:5,16 3:4
on	3:12
only	1:3
particularly	3:11
people	3:9
population	3:20
prevalent	3:19
quadrupedal	2:6
quokka	1:1
quokkas	3:0
rottnest	3:13
setonix	1:8
short	2:3
size	1:15
small	1:**11** 2:**2**
that	2:8
the	1:0,2,6,14
to	2:11
where	3:17
wombats	2:0

Auxiliary Data Structures

- Inverted lists usually stored together in a single file for efficiency
  - **Inverted file**
- **Vocabulary** or **lexicon**
  - Contains a lookup table from index terms to the byte offset of the inverted list in the inverted file
  - Either hash table in memory or B-tree for larger vocabularies
- Term statistics stored at start of inverted lists
- Collection statistics stored in separate file

Lucene salva in index info utili per similarity
↳ risparmiare tempo, ranking semplice per approx.
e scelte implementative

⇒ file separato

Index Construction

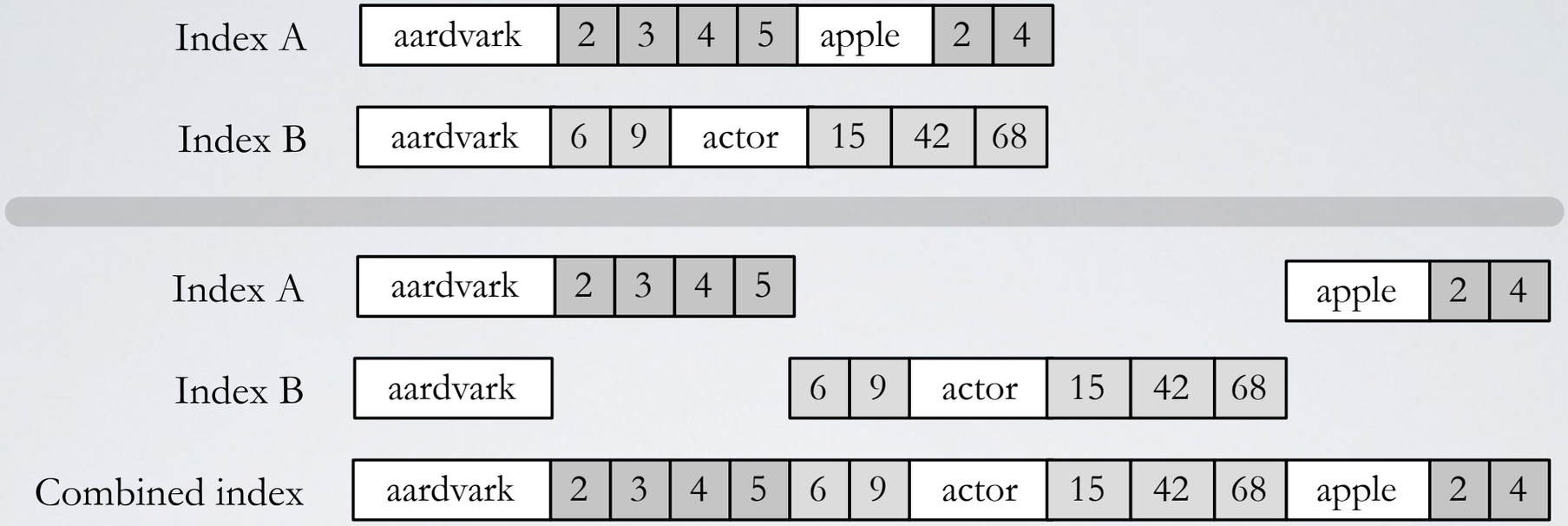
In-memory Index Construction

modo iterativo => sequenziale

procedure BUILDINDEX(D)
  $I \leftarrow$ HashTable()
  $n \leftarrow 0$
  for all documents $d \in D$ **do**
    $n \leftarrow n + 1$
    $T \leftarrow$ Parse(d)
    Remove duplicates from T
    for all tokens $t \in T$ **do**
      if $I_t \notin I$ **then**
        $I_t \leftarrow$ Array()
      end if
      I_t.append(n)
    end for
  end for
  return I
end procedure

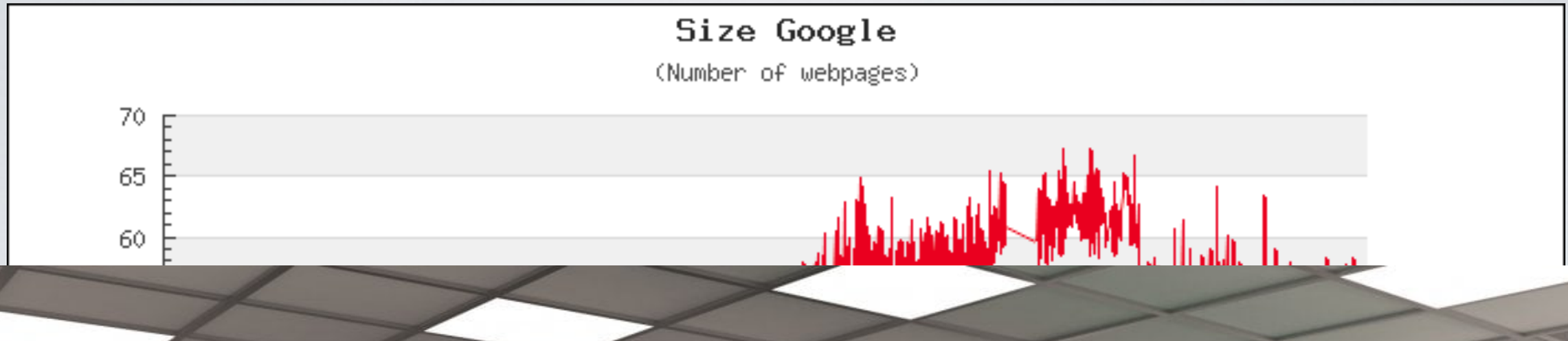
- It works only for "small" amounts of documents
- It does not parallelize well

Index Merging



- Merging addresses limited memory problem
  - Build the inverted list structure until memory runs out
  - Then write the partial index to disk, start making a new one
  - At the end of this process, the disk is filled with many partial indexes, which are merged
- The algorithm is essentially the same as the standard merge sort algorithm
  - Partial lists must be designed so they can be merged in small pieces, e.g., storing in alphabetical order
- It can be parallelized (to some extent): many machines can build their own partial indexes and a single machine can combine all of those indexes together into a single final index

Indexing at Web Scale



Frontier, jointly developed by Oak Ridge National Laboratory and Hewlett Packard, was completed on May, 2022
1.194 Exaflops
https://www.hpe.com/uk/en/compute/hpc/cray/oak-ridge-national-laboratory.html

van den Bosch, A., Bogers, T., and de Kunder, M. (2016). Estimating search engine index size variability: a 9-year longitudinal study. *Scientometrics*, 107(2):839–856.
https://www.worldwidewebsize.com/

Distributed Indexing

- Distributed processing driven by need to index and analyze huge amounts of data (i.e., the Web)
- Large numbers of **inexpensive/commodity servers** used rather than larger, more expensive machines
  - Data centers are distributed around the world.
  - Estimate: Google ~2.5 million servers (Gartner 2016)
- Suppose a non-fault-tolerant system where each node as an uptime v = 99.99% (52.6 minutes of downtime per year)
  - 1,000 nodes → 9.52% of the time one or more servers is down
  - 10,000 nodes → 63.2% of the time one or more servers is down
  - 50,000 nodes → 99.33% of the time one or more servers is down
  - 100,000 nodes → 100.00% of the time one or more servers is down

+ server, + probabilità di avere server down
meglio per manutenzione

$$\mathbb{P}\left[\text{1 Node Up}\right] = \nu$$

$$\mathbb{P}\left[\text{All Nodes Up}\right] = \prod_{i=1}^{n} \nu = \nu^n$$

$$\mathbb{P}\left[\text{At Least One Node Down}\right] = 1 - \nu^n$$

MapReduce

- **MapReduce** is a robust and conceptually simple framework for distributed computing… without having to write code for the distribution part.
  - **Mapper**: transforms a list of items into another list of items
  - **Reducer**: transforms a list of items into a single item
  - Sort of beginning of the Big Data hype
- Many mapper and reducer tasks on a cluster of machines
- Dean and Ghemawat describe the Google indexing system (ca. 2002) as consisting of a number of phases, each implemented in MapReduce

Dean, J. and Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. In Brewer, E. A. and Chen, P., editors, *Proc. 6th Symposium on Operating System Design and Implementation (OSDI 2004)*, pages 137–149. USENIX Association, USA.

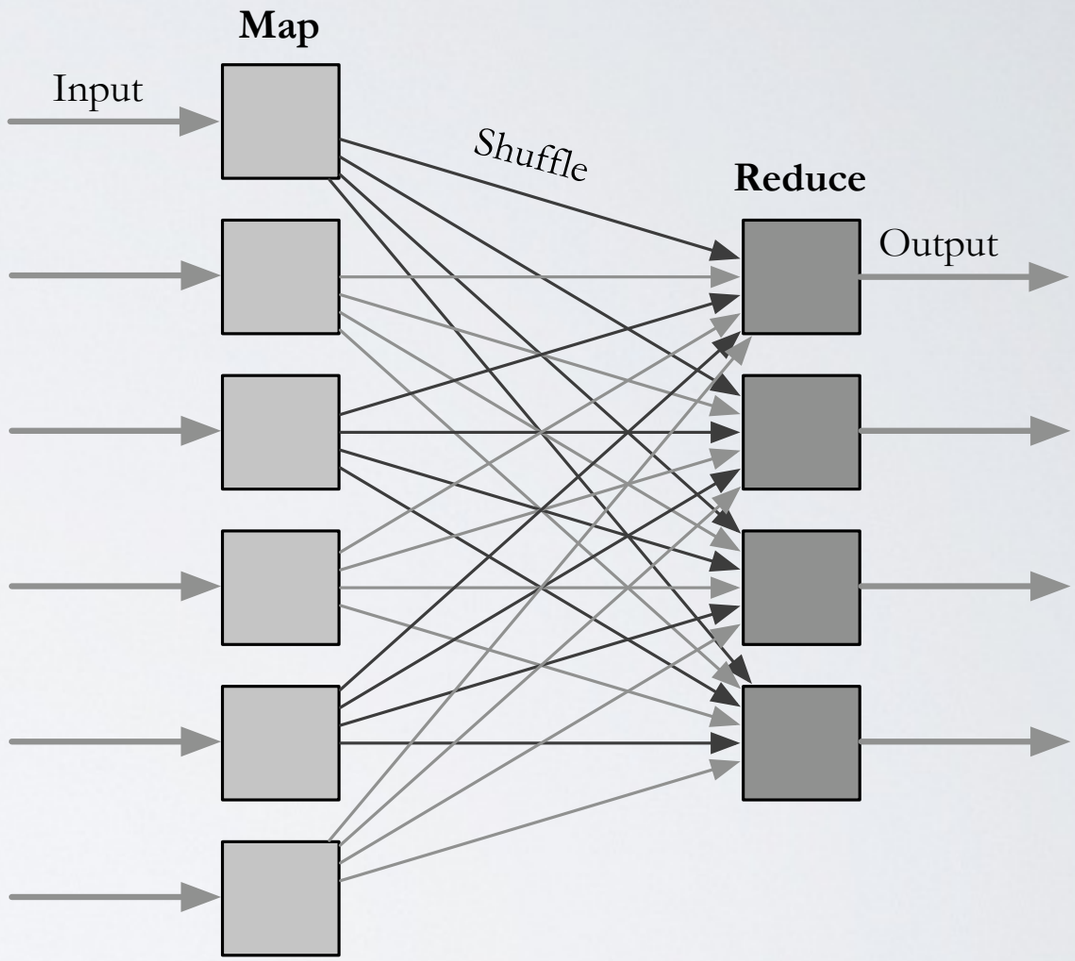
MapReduce: Basics

- Basic process
  - Map stage which transforms data records into pairs, each with a key and a value
  - Shuffle uses a hash function so that all pairs with the same key end up next to each other and on the same machine
  - Reduce stage processes records in batches, where all pairs with the same key are processed at the same time
- **Idempotence** of Mapper and Reducer provides fault tolerance
  - multiple operations on same input give same output, so if a node is down, the framework can simply (re-)run the same operations on another node
  - *speculative execution*: the framework may execute the same task more than once on different machines, retaining the first output that emerges

MapReduce: Indexing

```
procedure MapDocumentsToPostings(input)
    while not input.done() do
        document ← input.next()
        number ← document.number
        position ← 0
        tokens ← Parse(document)
        for each word w in tokens do
            Emit(w, document:position)
            position = position + 1
        end for
    end while
end procedure

procedure ReducePostingsToLists(key, values)
    word ← key
    WriteWord(word)
    while not input.done() do
        EncodePosting(values.next())
    end while
end procedure
```



Index Update

- Document collections grow and change
  - **static collections**: collections with few changes over time can be **re-indexed** every so often.
    Inverted file update not an option, as it requires writes in the middle of the file
  - **dynamic collections**: change a lot and re-indexing is not an option
    Twitter is an extreme example: 500+ million tweet/day; 6,000 tweet/second (as of May 2020)
- The following strategies are applied:
  - **Index merging**: when new documents arrive in large numbers at a time, they are indexed and then the existing index is merged with the new one
  - **Result merging**: when new documents arrive in small numbers at a time, a separate, small index is maintained and updated (it can also be in-memory).
    Queries are processed against both the existing index and the small one containing the new arrivals, fusing the results.
  - **Deletions list**: deletions are recorded in a deletions list, and deleted documents are removed from search results before results are shown.
- Modifications are done by inserting a new document, and deleting the previous version

Basics of Query Processing

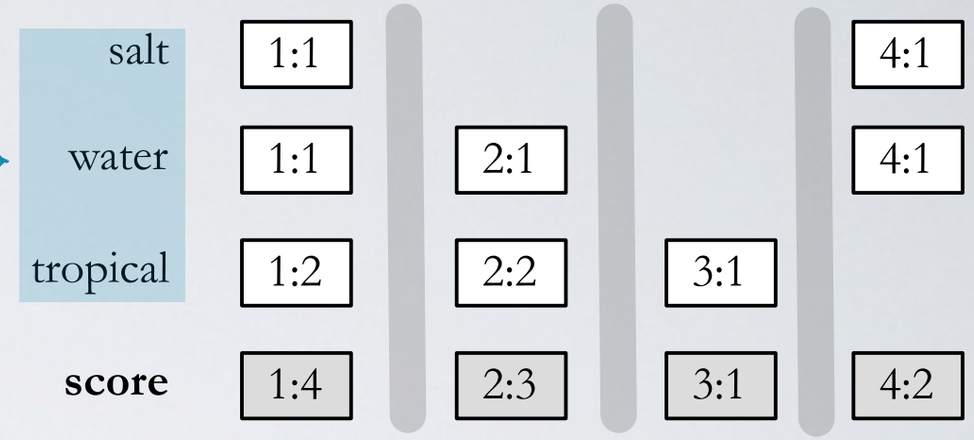
Query Processing

- **Document-at-a-time** (**DAAT**)
  - Calculates complete scores for documents by processing all term lists, one document at a time
- **Term-at-a-time** (**TAAT**)
  - Accumulates scores for documents by processing term lists one at a time
- Both approaches have optimization techniques that significantly reduce time required to generate scores

Turtle, H. and Flood, J. (1995). Query evaluation: Strategies and optimizations. *Information Processing & Management*, 31(6):831–850.

Document-at-a-time (DAAT)

Posting Lists corresponding to query terms

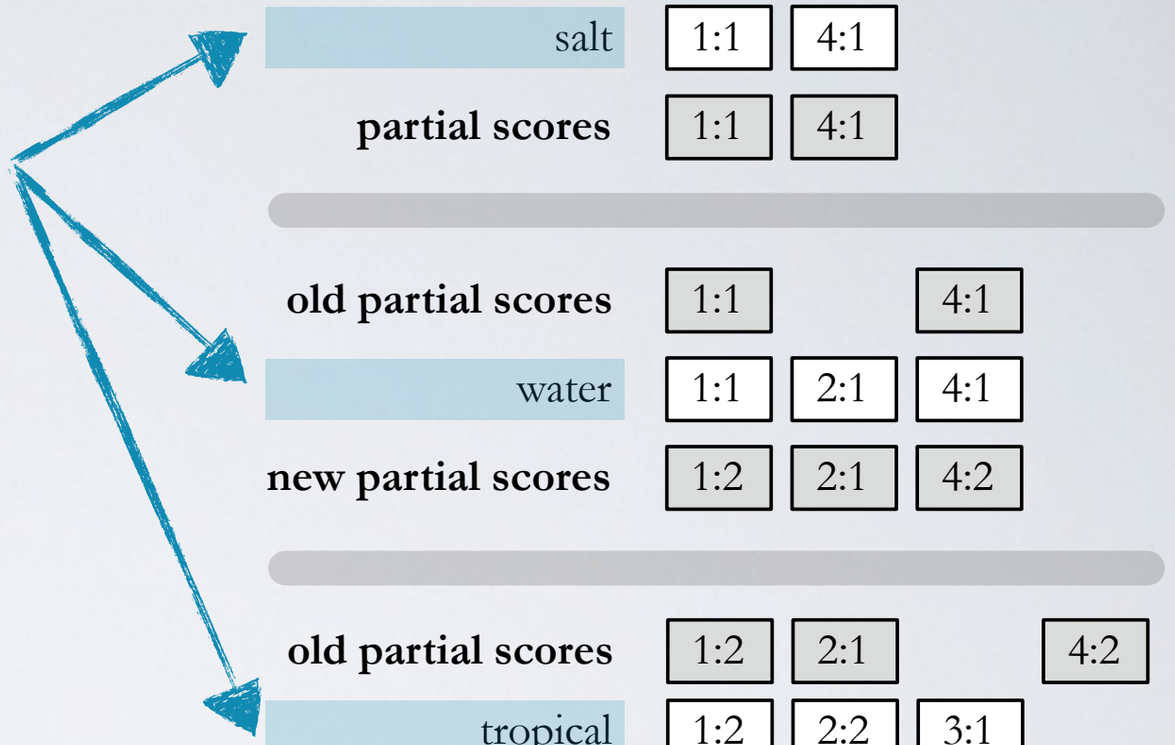


procedure DOCUMENTATATIMERETRIEVAL(Q, I, f, g, k)
  $L \leftarrow$ Array()
  $R \leftarrow$ PriorityQueue(k)
  for all terms w_i in Q **do**
    $l_i \leftarrow$ InvertedList(w_i, I)
    L.add(l_i)
  end for
  for all documents $d \in I$ **do**
    $s_d \leftarrow 0$
    for all inverted lists l_i in L **do**
      if l_i.getCurrentDocument() = d **then**
        $s_d \leftarrow s_d + g_i(Q) f_i(l_i)$
      end if
      l_i.movePastDocument(d)
    end for
    R.add(s_d, d)
  end for
  return the top k results from R
end procedure

- The priority queue contains only the top k documents
  - small memory footprint
- Switching between posting lists increase the cost of seeking at disk level
  - high disk access
  - may be compensated by buffering at expense of increased memory footprint

Term-at-a-time (TAAT)

Posting Lists corresponding to query terms



procedure TERMATATIMERETRIEVAL($Q, I, f, g\ k$)
  $A \leftarrow$ HashTable()
  $L \leftarrow$ Array()
  $R \leftarrow$ PriorityQueue(k)
  for all terms w_i in Q **do**
    $l_i \leftarrow$ InvertedList(w_i, I)
    L.add(l_i)
  end for
  for all lists $l_i \in L$ **do**
    while l_i is not finished **do**
      $d \leftarrow l_i$.getCurrentDocument()
      $A_d \leftarrow A_d + g_i(Q) f(l_i)$
      l_i.moveToNextDocument()
    end while
  end for
  for all accumulators A_d in A **do**
    $s_d \leftarrow A_d$
    R.add(s_d, d)
  end for
  return the top k results from R
end procedure

- The accumulators for all the documents need to remain in memory until the very end
  - high memory footprint
- Sequential scan of each inverted list, reducing the cost of seeking at disk level
  - low disk access

Distributed Query Processing

- Basic process
  - All queries sent to a director machine
  - Director then sends messages to many index servers
  - Each index server does some portion of the query processing
  - Director organizes the results and returns them to the user
- Two main approaches
  - Document distribution
    by far the most popular
  - Term distribution

Document Distribution

- Also known as **distributed local indexing**
- Each index server acts as a search engine for a small fraction of the total collection
- Director sends a copy of the query to each of the index servers, each of which returns the top-k results
- Results are merged into a single ranked list by the director
- Issue
  - Collection statistics should be shared for effective ranking

Term Distribution

- Also known as **distributed global indexing**
- Single index is built for the whole cluster of machines
- Each inverted list in that index is then assigned to one index server
  - in most cases the data to process a query is not stored on a single machine
- One of the index servers is chosen to process the query
  - usually the one holding the longest inverted list
  - Other index servers send information to that server
- Final results sent to director
- Issue
  - load balancing depends on the distribution of query terms and their co-occurrence

Caching

- About 50% of the queries each day are unique, but some are very popular. Moreover, about 15% of the queries per day have never occurred before
- Caching can significantly improve effectiveness
  - Cache popular **query results**
  - Cache common **inverted lists**
- Inverted list caching can help with unique queries
- Cache must be refreshed to prevent stale data

Questions?


GOOGLE HEADQUARTERS
ISN'T IT A LITTLE BIT EVIL TO KILL DILBERT WITH OUR DEATH RAY?
www.dilbert.com scottadams@aol.com
GOOD POINT. . . WHAT IF I JUST BLAST THE SPACE STATION OUT OF ORBIT AND MAKE IT LAND ON HIS HOUSE?
I'LL BET YOU TEN BILLION DOLLARS YOU CAN'T.
AND THE LOSER HAS TO INTRO-DUCE HIM-SELF AS "THE DUMB ONE."
©2006 Scott Adams, Inc./Dist. by UFS, Inc.