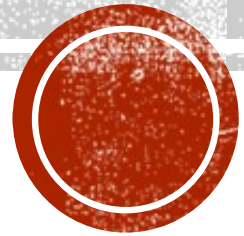
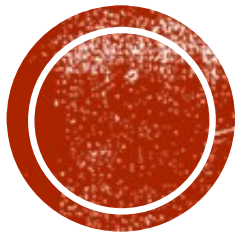
MAPREDUCE PATTERNS



MAPREDUCE PATTERNS

- Filtering Patterns
  - Sampling
  - Generating Top-N
- Summarization Patterns
  - Inverted Indexing
  - Numerical Summarizations
    - Min/Max Values
    - Counting
    - Descriptive Statistics
- Structural Patterns
  - Combining Data Sets



CASE STUDY

Inverted Indexing

INFORMATION RETRIEVAL AND WEB SEARCH

- Given a query, expressed as a small set of terms, retrieve and present relevant objects to users
  - Information retrieval is an online problem that requires sub-second (quick) response time
- Search engines rely on data structure, called "**inverted index**"

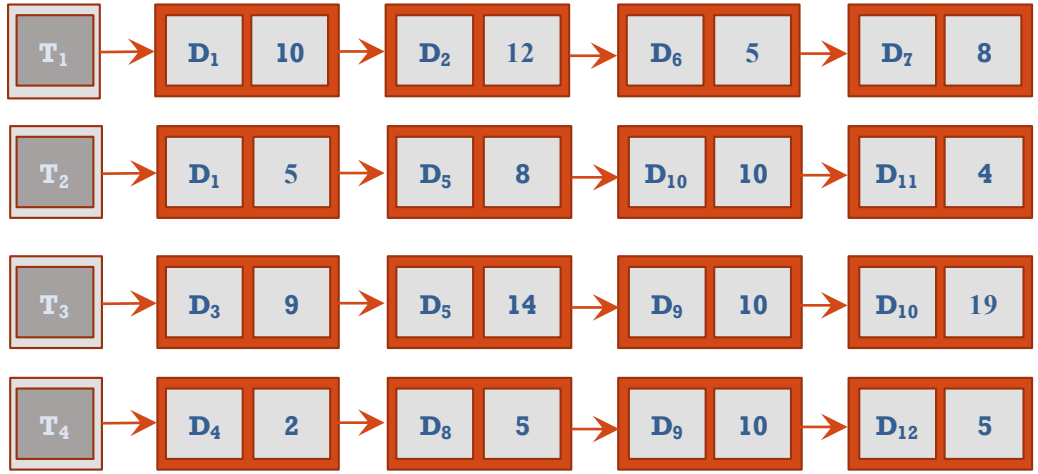
INVERTED INDEX STRUCTURE

- In its basic form, an inverted index consists of postings lists, one associated with each term that appears in the collection of information objects

- An individual posting consists of a document id and a payload
  - Common payload information include
    - term frequency,
    - positions of every occurrence of the term,
    - document properties for ranking purposes or linguistic processing

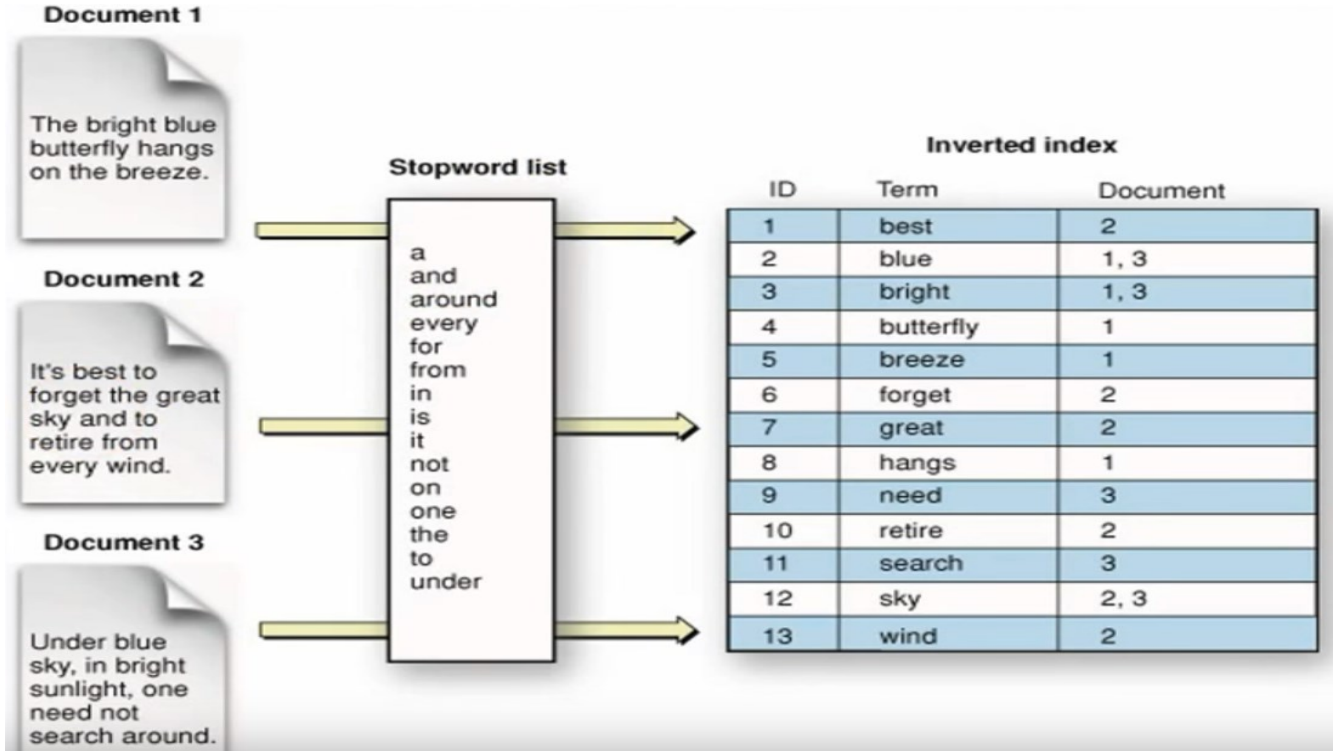
INVERTED INDEX STRUCTURE

- Term T, associated with a **list of posting** (document D, frequency that term T appears in D).
- Your posting list may include other data as needed.



T = Term, D = Document

INVERTED INDEX EXAMPLE



HOW TO USE MAPREDUCE FOR BUILDING INVERTED INDEX?

INVERTED INDEX - MAPPER

- Individual Documents are processed in parallel by the mappers
- Input to the mapper <key, value> pairs

- Document is analyzed
  - Term frequencies, **f**, are computed by iterating over all analyzed documents
- Mapper emits intermediate <key, value> pairs
  - Key = Term
  - Value = Posting list

INVERTED INDEX ALGORITHM – MAPPER

```
class MAPPER
    procedure MAP(did id, doc d)
    {
        F ← new ASSOCIATIVEARRAY
        forall term t in doc d do
            F{t}←F{t}+1
        forall term t in F do
            EMIT( term t, posting [id, F{t}] )
    }
```

- **Document Analysis**
- **Term Frequencies Produced**

INVERTED INDEX ALGORITHM – REDUCER

- First Step
  - Producer creates an empty list and appends all postings associated with the same key to the list
    - Key = term
- Second Step
  - Posting are sorted, based on document identifiers
- Final Step
  - Entire posting list is emitted as value, with term as key
  - The final <key, value> pairs are written to disk and comprise the inverted index

Inverted Index Algorithm – Reducer

class Reducer

procedure Reduce(term t, postings [<id_1, f_1>, <id_2, f_2>, …])

{

P <- new List

forall postings <a, f> **in** postings [<id_1, f_1>, <id_2, f_2>, …] **do**

Append(P, <a, f>)

Sort(P)

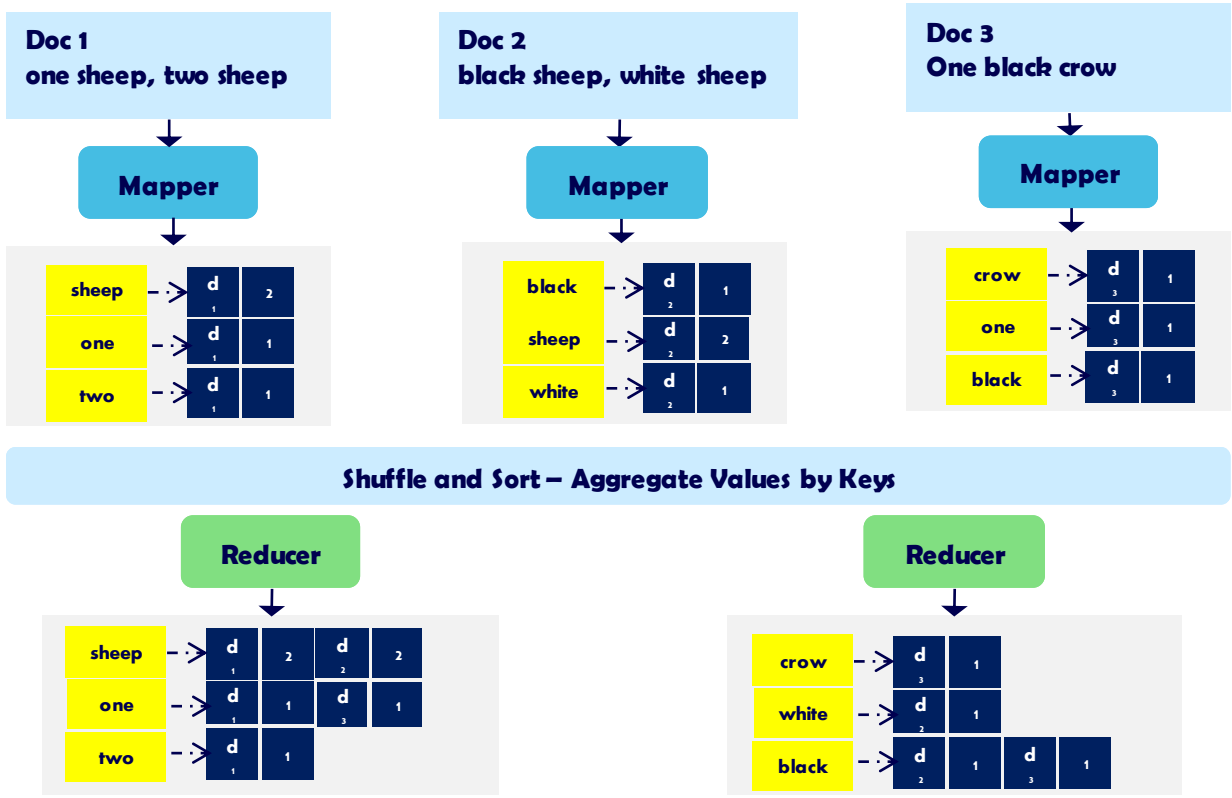
Emit(term t, posting P)

}

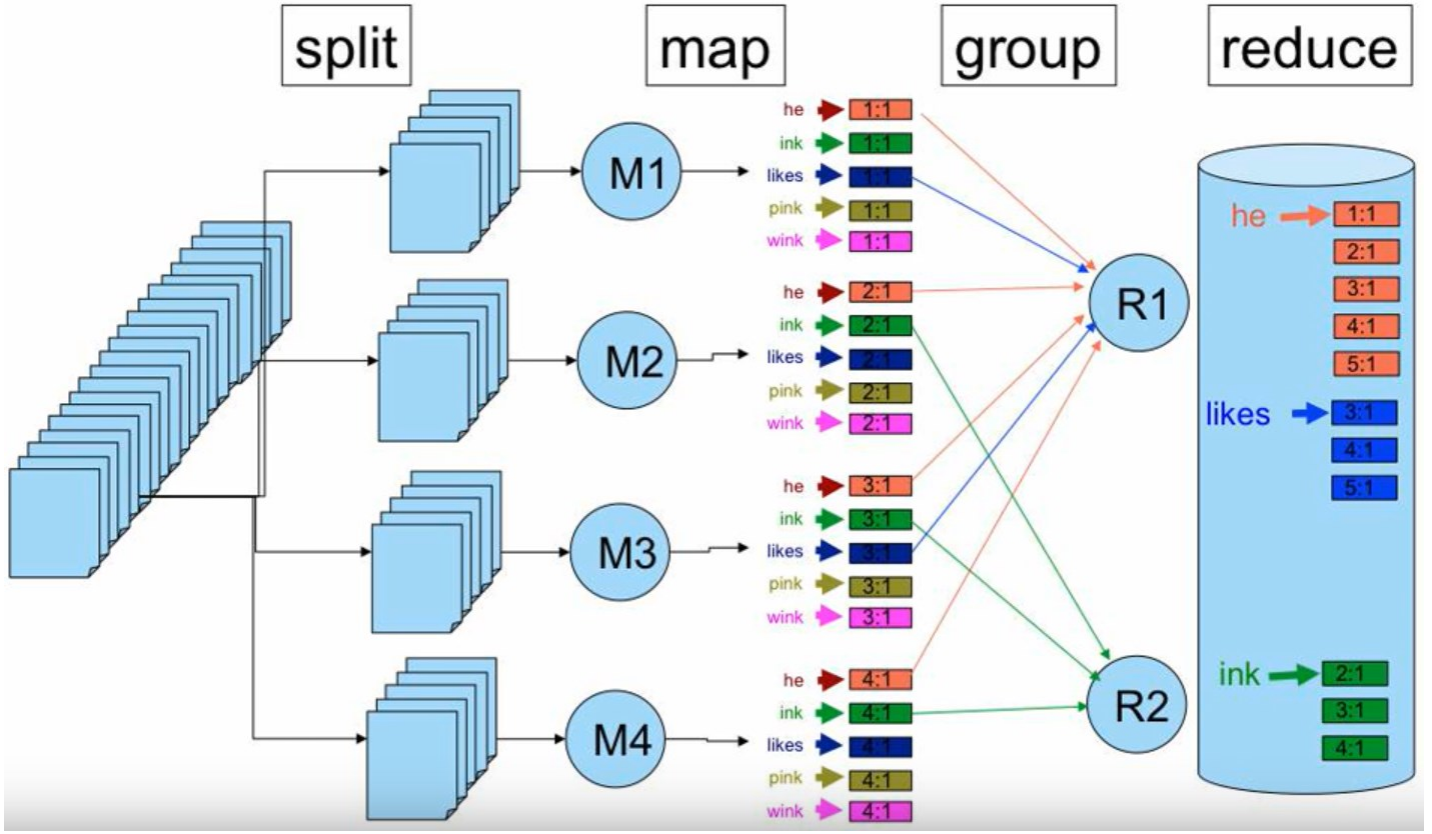
INVERTED INDEXING – FINAL OUTCOME

- Each reducer writes its output in a separate file in the distributed file system,
  - Final index is split across R files, where R is the number of reducers.
  - Separately, an index to the postings lists must be built
    - Index allows retrieval engine to fetch the postings list, for a given term, by opening the appropriate file and seeking to the correct byte offset position in the file
    - Typically in the form of mappings from term to (file, byte offset) pairs

INVERTED INDEXING – FINAL OUTPUT



INVERTED INDEXING – FINAL OUTPUT



IS THIS MAPREDUCE PROGRAM SCALABLE?

```
class MAPPER
    procedure MAP(did id, doc d)
    {
       F ← new ASSOCIATIVEARRAY
       forall term t in doc d do
         F{t}←F{t}+1
      forall term t in F do
       EMIT( term t, posting [id, F{t}] )
    }
```

```
class REDUCER
    procedure REDUCE(term t, postings [<id1, f1>, <id2, f2>, …])
    {
     P <- new LIST
     forall postings <a, f> in postings [<id1, f1>, <id2, f2>, …] do
       APPEND(P, <a, f>)
      SORT(P)
      EMIT( term t, posting P )
    }
```

INVERTED INDEXING
BASIC ALGORITHM - MEMORY BOTTLENECK

- MapReduce execution framework does not guarantee the ordering of values associated with the same key, the reducer must perform in-memory sort before emitting the postings
- The basic inverted indexing algorithm assumes sufficient memory to hold all postings associated with the same term
- Correct execution depends on memory size

OPTIONAL
INVERTED INDEX AS MAPREDUCE DESIGN PATTERN

INVERTED INDEX PATTERN – GENERAL INFORMATION

Pattern Description

- The inverted index pattern is commonly used as an example for MapReduce analytics.

Intent

- Generate an index from a data set to allow for faster searches or data enrichment capabilities.

Applicability

- Inverted indexes should be used when quick search query responses are required.The results of such a query can be preprocessed and ingested into a database.

INVERTED INDEX PATTERN - MOTIVATION

- It is often convenient to index large data sets on keywords, so that searches can trace terms back to records that contain specific values. While building an inverted index does require extra processing up front, taking the time to do so can greatly reduce the amount of time it takes to find something. Search engines build indexes to improve search performance. Imagine entering a key- word and letting the engine crawl the Internet and build a list of pages to return to you. Such a query would take an extremely long amount of time to complete. By building an inverted index, the search engine knows all the web pages related to a keyword ahead of time and these results are simply displayed to the user. These indexes are often ingested into a database for fast query responses. Building an inverted index is a fairly straight-forward application of MapReduce because the framework handles a majority of the work.

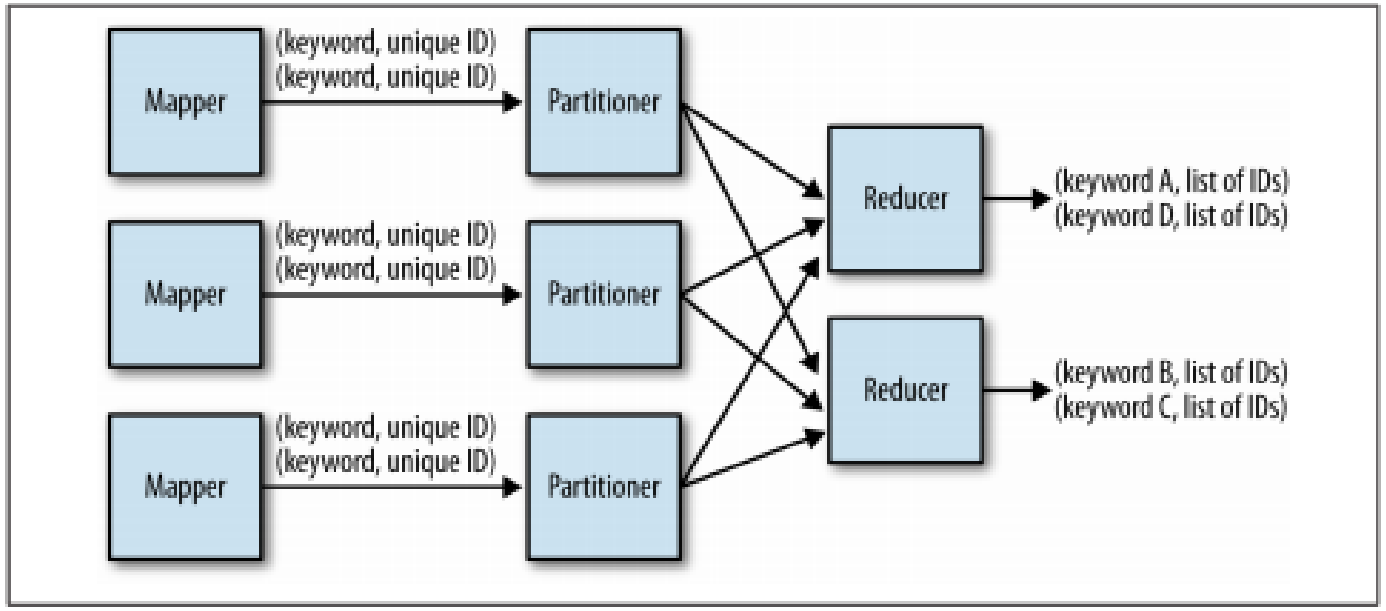
INVERTED INDEX PATTERN - STRUCTURE

- The mapper outputs the desired fields for the index as the key and the unique identifier as the value.
- The combiner can be omitted if you are just using the identity reducer, because under those circumstances a combiner would just create unnecessary processing. Some implementations concatenate the values associated with a group before outputting them to the file system. In this case, a combiner can be used. It won't have as beneficial an impact on byte count as the combiners in other patterns, but there will be an improvement.

INVERTED INDEX PATTERN - STRUCTURE - CONT'D

- The partitioner is responsible for determining where values with the same key will eventually be copied by a reducer for final output. It can be customized for more efficient load balancing if the intermediate keys are not evenly distributed.
- The reducer will receive a set of unique record identifiers to map back to the input key. The identifiers can either be concatenated by some unique delimiter, leading to the output of one key/value pair per group, or each input value can be written with the input key, known as the identity reducer.

INVERTED INDEX PATTERN - STRUCTURE - CONT'D



INVERTED INDEX PATTERN - STRUCTURE - CONT'D

- **Combiner optimization:** The combiner can be used to do some concatenation prior to the reduce phase. Because all row IDs are simply concatenated together, the number of bytes that need to be copied by the reducer is more than in a numerical summarization pattern.
- The same code for the reducer class is used as the combiner.

INVERTED INDEX PATTERN – OUTPUT/CONSEQUENCES

- The final output of is a set of part files that contain a mapping of field value to a set of unique IDs of records containing the associated field value

INVERTED INDEX PATTERN – PERFORMANCE ANALYSIS

- The performance of building an inverted index depends mostly on the computational cost of parsing the content in the mapper, the cardinality of the index keys, and the number of content identifiers per key. Parsing text or other types of content in the mapper can sometimes be the most computationally intense operation in a MapReduce job.
- This is especially true for semi-structured data, such as XML or JSON, since these typically require parsing arbitrary quantities of information into usable objects. It's important to parse the incoming re- cords as efficiently as possible to improve your overall job performance. If the number of unique keys and the number of identifiers is large, more data will be sent to the reducers. If more data is going to the reducers, you should increase the number of reducers to increase parallelism during the reduce phase.

REFERENCES

- **Inverted Indexing:** http://www.dcs.bbk.ac.uk/~dell/teaching/cc/book/ditp/ditp_ch4.pdf
- Hadoop The Definitive Guide 4th Edition, Tom White.
- Data-Intensive Text Processing with MapReduce, Jimmy Lin and Chris Dyer.