TECHNISCHE UNIVERSITÄT DRESDEN

Dresden Database Systems Group

Exercise 5

Information Retrieval

Dresden Database
Systems Group

7. Implementing IR-Systems II
Index Construction

Warm-Up

Dresden Database Systems Group

Exercise 7.1

- Are the following statements true or false? Give reasons for your answer.

a) External sorting algorithms are used to sort lists which do not fit in main memory.

b) The term-at-a-time approach to query answering requires more run-time memory than the document-at-a-time approach.

c) It is impossible to calculate the top-k documents for a query without completely reading the postings list of each query term.

d) When distributing a large index across a cluster of computers, we can partition the index by term or by document

e) In IR, we do not consider the case of evolving indexes, since document collections (almost) never change.

Efficient Top-k for One-Term-Queries

Dresden Database Systems Group

Exercise 7.2

Exercise 0.99

- Assume
  - we only have one-term queries
  - we order postings lists according to weight, instead of docId
  - we truncate every postings list at position k
  - the weight w stored for doc. d in the postings list of t is the cosine-normalized weight of t for d

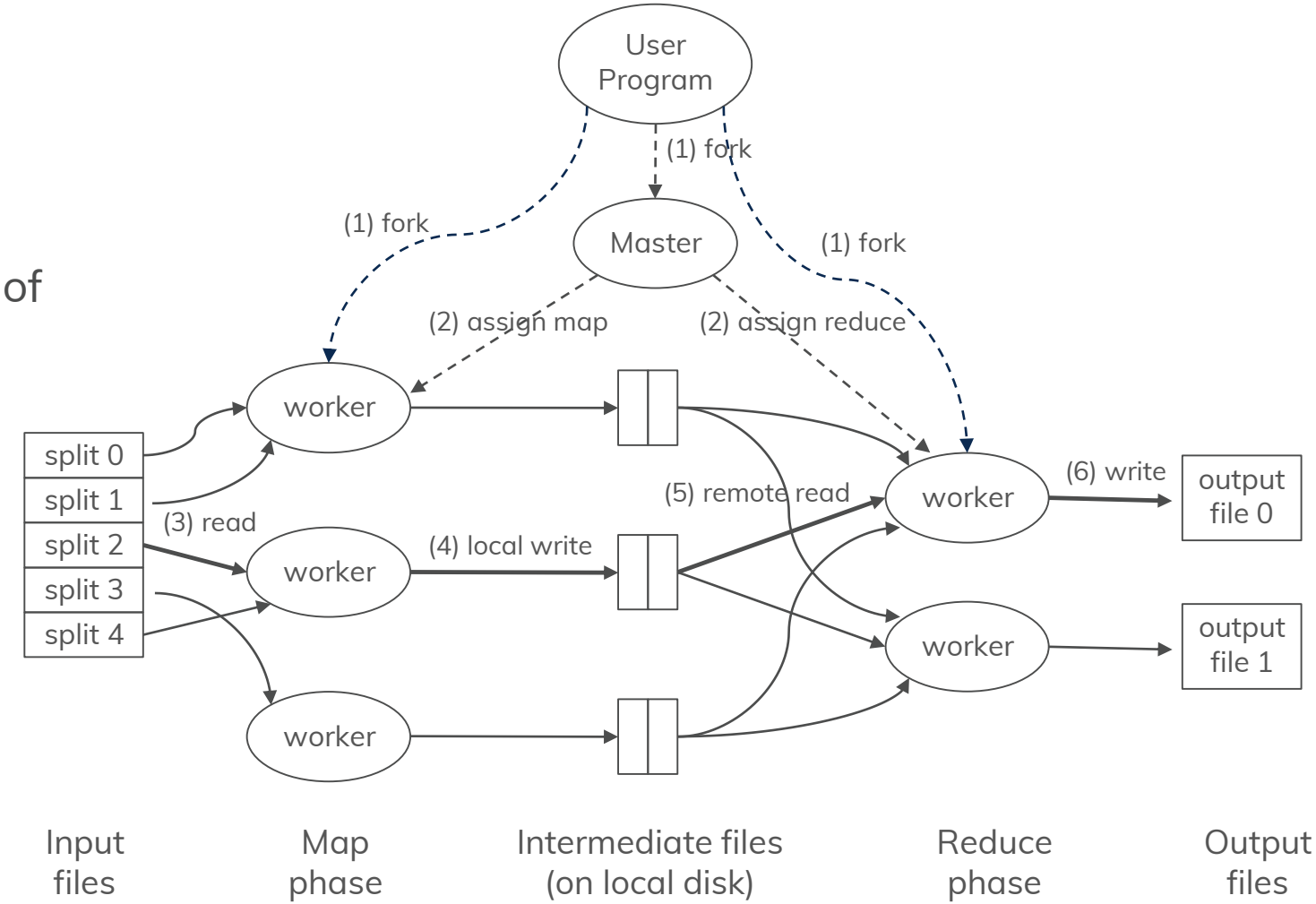
- Explain why this approach suffices for identifying the k highest scoring documents for a query

Distributed Indexing: MapReduce

Dresden Database Systems Group

Key Facts

- MapReduce is a programming model
- Programmer implements a `map` function and a `reduce` function
- Underlying runtime system takes care of
  - Parallelization across large-scale cluster
  - Machine failures
  - Communication
  - Performance
  - etc.
- see lecture slides for details



Index Construction in MapReduce

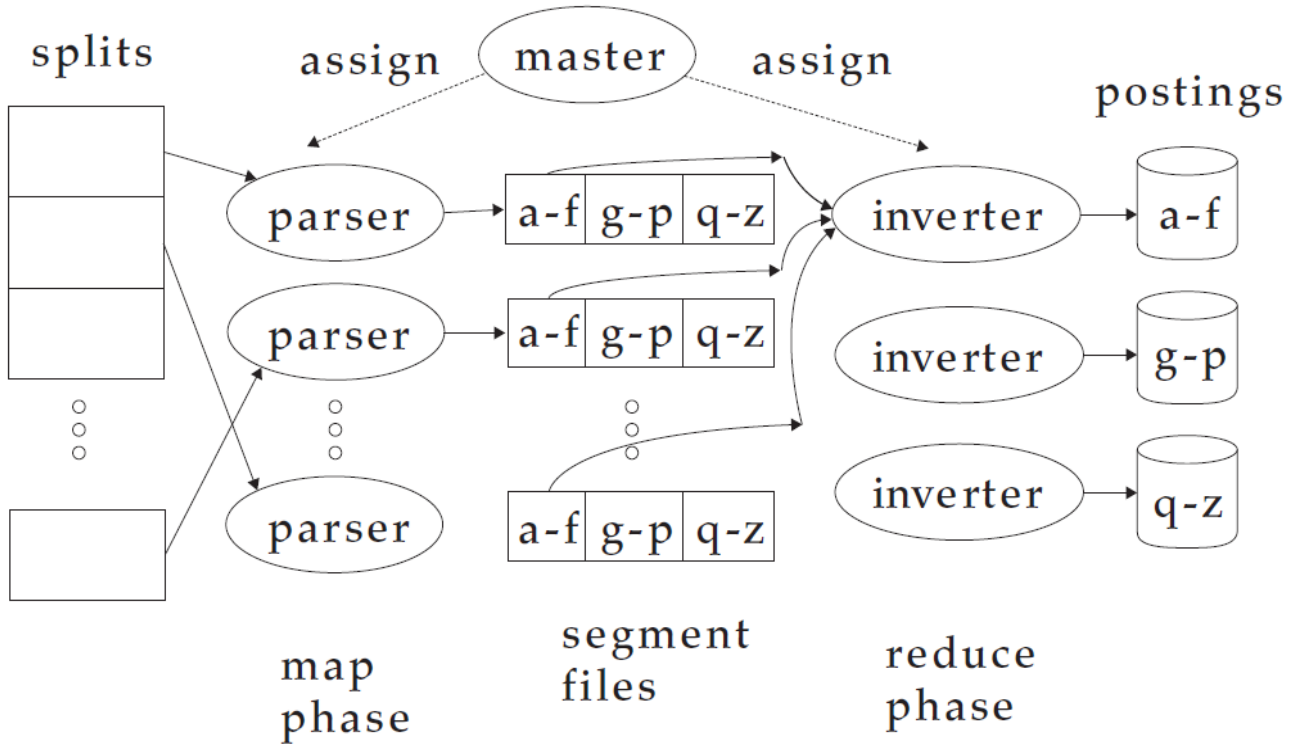
Dresden Database Systems Group

Parsers

- Master assigns a split to an idle parser machine
- Parser reads a document at a time and emits (term, docID)-pairs
- Parser writes pairs into j term-partitions
- Each for a range of terms' first letters
- E.g., a-f, g-p, q-z (here: $j = 3$)

Inverters

- Inverter collects all (term, docID)-pairs for one term-partition (e.g., for a-f)
- Sorts and writes to postings lists



`map` and `reduce` functions

Dresden Database Systems Group

Schema of *`map`* and *`reduce`* functions

- `map`: input → list(k, v)
- `reduce`: list(k, list(v)) → output

Example: Index Construction

- Parser (`map`): web collection → list(term, docID)
- Inverter (`reduce`): list(⟨term, list(docID)⟩) → postings_list1, postings_list2, …

Exercise 7.3

- Apply MapReduce to the problem of counting how often each term occurs in a set of files by specifying `map` and `reduce` operations for this task

Exercise 0.53

Distributed Indexing: MapReduce

Dresden Database Systems Group

Exercise 0.51

Exercise 7.4

- Assume that machines in MapReduce have 100 GB of disk space each
- Assume further that the postings list of the term `the` has a size of 200 GB
- Why can the MapReduce algorithm as described in the lecture not be run in this case?
- How would you modify MapReduce so that it can handle this case?

Exercise 0.52

Exercise 7.5

- For optimal load balancing, the inverters in MapReduce must get segmented postings files of similar sizes
- For a new collection, the distribution of key-value pairs may not be known a-priori
- How would you solve this problem?