TinyGoogle Discussion

Main Task

- Develop and infrastructure to support two types of queries
  - IndexDoc(D)
    - Produce an Index of a D
      - Count the occurrence of each work in the document
    - Merge the index of d with the Main Index
  - Retrieve[<Di, (i=1, N)>, <Ki, (i=N>)]
    - Parse keyword set
    - For each key, k, identify all documents where k appears
    - Rank order all documents
    - Return query result

Indexing a document

I can not do everything, but still I can do something; and because I cannot do everything, I will not refuse to do something I can do

Word	Count
And	1
Because	1
But	1
Can	4
Do	4
Everything	2
I	5
Not	3
Refuse	1
Something	2
Still	1
To	1
Will	1

Doc Indexing – Program Structure

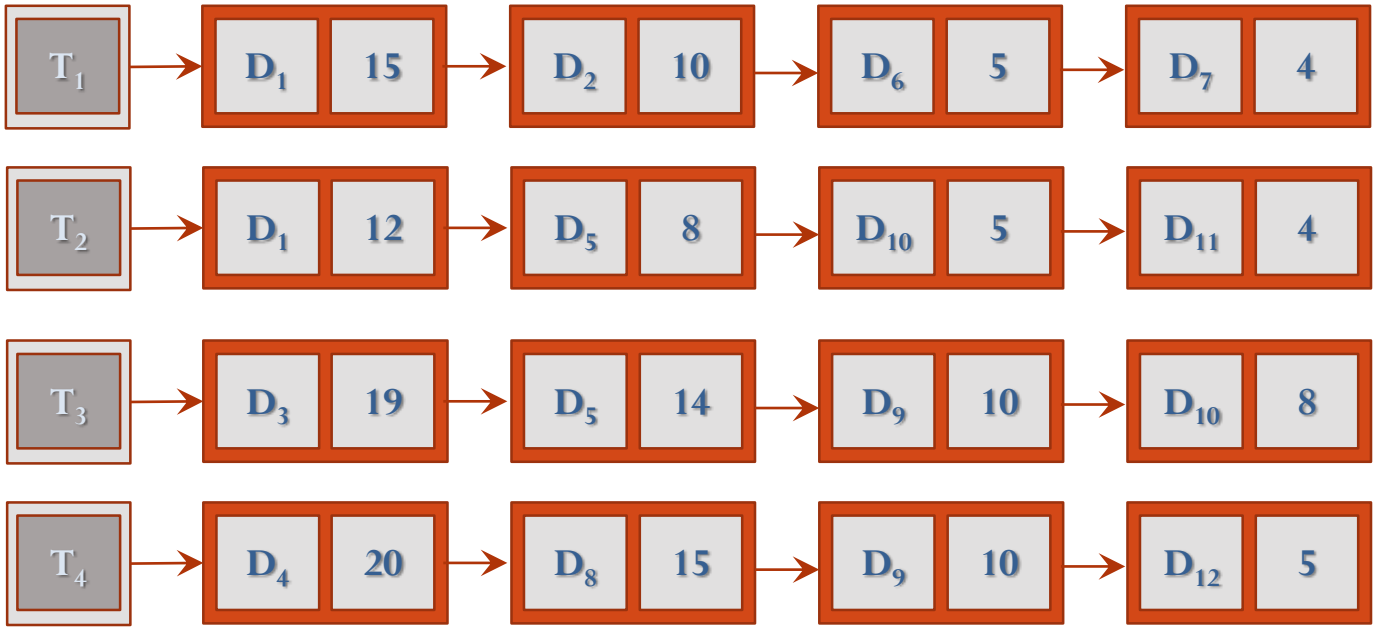
- A three-phased program can be used to distribute the work over several machines the final document index and merge it with main index
- Phase I – Document Processing
  - Each machine will process a fraction of the document set
- Phase II – Count Aggregation
  - Partial key word counts from individual machines are combined into the document index
- Phase III – Merge document index into main index
  - Divide document index into chunks and assign machines to merge checks into main index

Query Retrieval – Program Structure

- Assign keywords to different workers
  - Main index must be stored distributed for parallel access
- Produce documents
- Rank order document,

Inverted Index Structure

- Given a query, retrieval involves fetching posting with query terms and traversing the postings to compute the result set

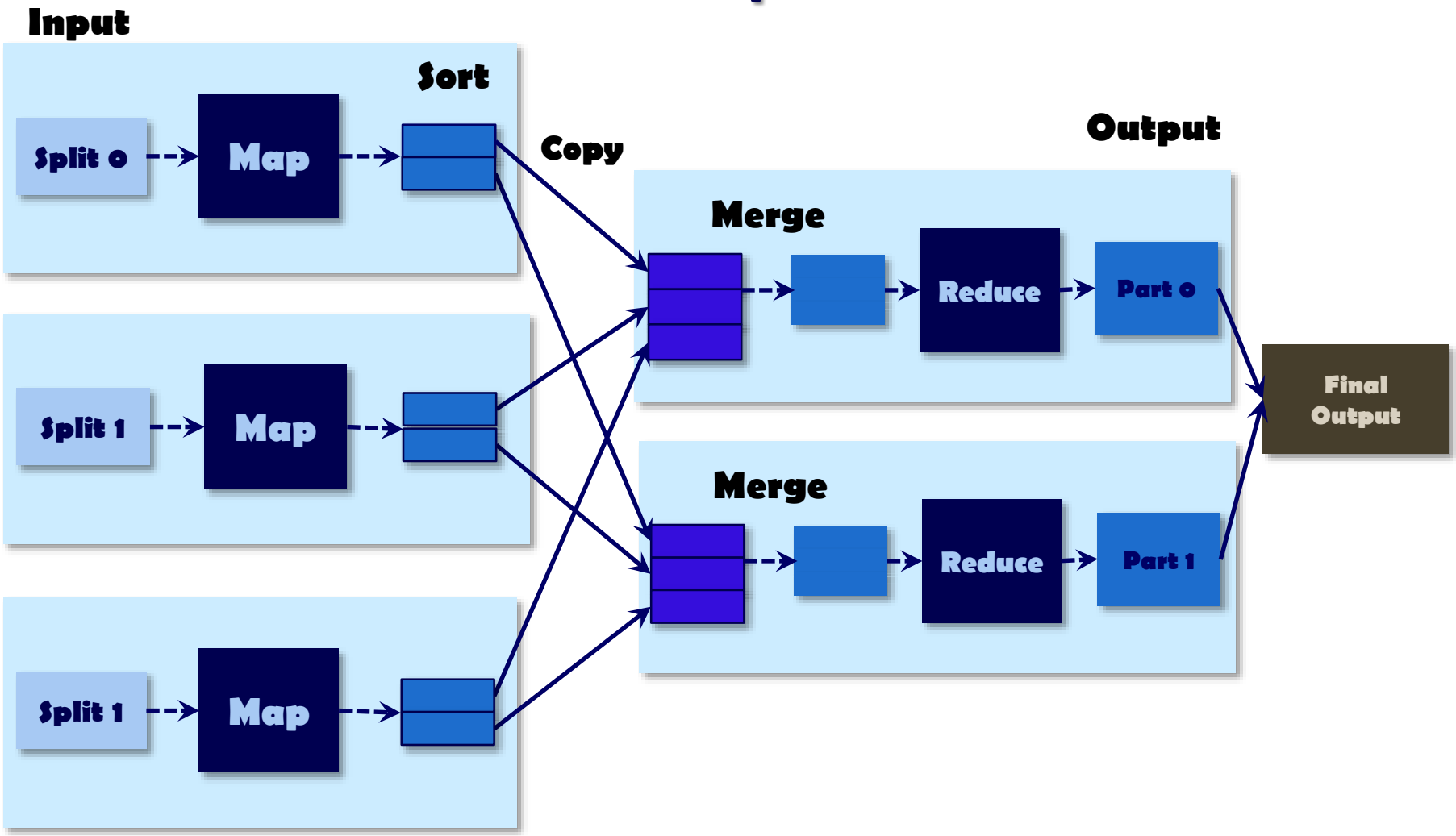


MapReduce

MapReduce – TinyGoogle Data Flow

- A MapReduce job is a unit of work to be performed
  - Job consists of the **"MapReduce Program"**, the **Input data** and the **Configuration Information and Runtime System**
- The MapReduce job is divided it into two types of tasks – map tasks and reduce tasks
- The Input data is divided into fixed-size pieces called **splits**
  - **One map task is created for each split**
- Configuration information and runtime system
  - Preprocessing of input data Location of input indicates where the input lies, and the output is stored

Data Flow – Multiple Reduce Tasks



Document Index

- Define WordCount as Multiset;
- For Each Document in DocumentSet, S
- {
- T {} = tokenize(d in S);
- For Each Token in T
- {
- WordCount[token]++;
- }
- DocIndex(<t, cont>, for t in T)
- }

WordCount Program – II

Phase I

- Define WordCount as Multiset;
- For Each D in DocSubset {
  - T = tokenize(D);
  - For Each Token in T {
    - WordCount[token]++;
- }
- }
- SendToPhaseII(wordCount);

Phase II

- Define TotalWordCount as Multiset;
- For each WordCount Received From firstPhase {
  - MultisetAdd (TotalWordCount, WordCount);
- }

Inverted Index Algorithm – Mapper

- Individual Documents are processed in parallel by the mappers
- Input to the mapper <key, value> pairs
  - Key = Document IDs,
  - Value = Actual document content
- Document is analyzed and tokenized
  - Term frequencies, **f**, are computed by iterating over all analyzed documents
- Mapper emits intermediate <key, value> pairs
  - Key = Term,
  - Value = Posting

Inverted Index Algorithm – Mapper

```
class MAPPER
    procedure MAP(did id, doc d)
    {

        forall term t in doc d do
            EMIT( term t, posting [id, H{t}] )
    }
```

Shuffle and Sort Phase

- In the shuffle and sort phase, MapReduce runtime performs a large distributed computation to group postings by term
  - With no additional effort by the programmer, execution framework brings together all postings that belong in the same posting list
- The outcome of the shuffle and sort phase simplifies significantly the task of the reducer

Inverted Index Algorithm – Reducer

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

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.
    - Further consolidation of the file is not needed
  - 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