MapReduce I

Big Data Management

Knowledge objectives

1. Enumerate several use cases of MapReduce
2. Explain 6 benefits of using MapReduce
3. Describe what the MapReduce is in the context of a DDBMS
4. Recognize the signature of Map and Reduce functions
5. Justify to which extent MapReduce is generic

Understanding objectives

1. Simulate the execution of a simple MapReduce algorithm from the user (agnostic of implementation details) perspective

Application objectives

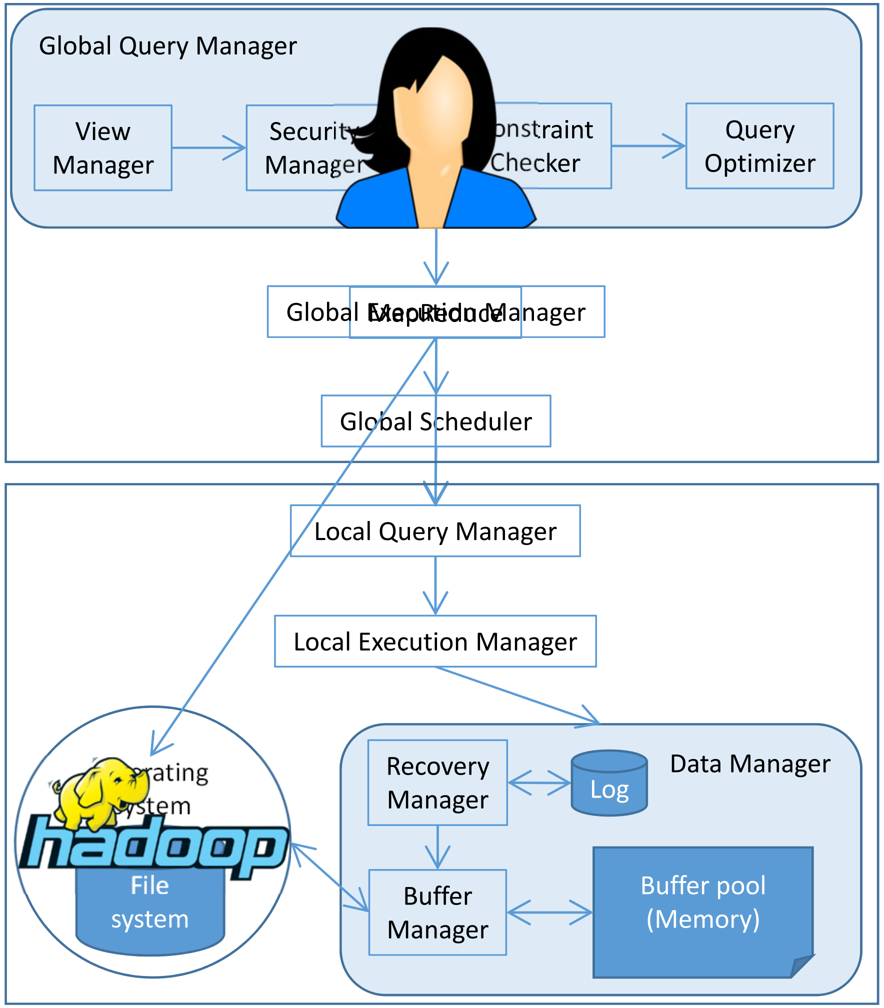
1. Identify the usefulness of MapReduce in a given use case
2. Define the key in the output of the map for a simple problem
3. Provide the pseudo-code of map and reduce functions for a simple problem

Distributed processing framework

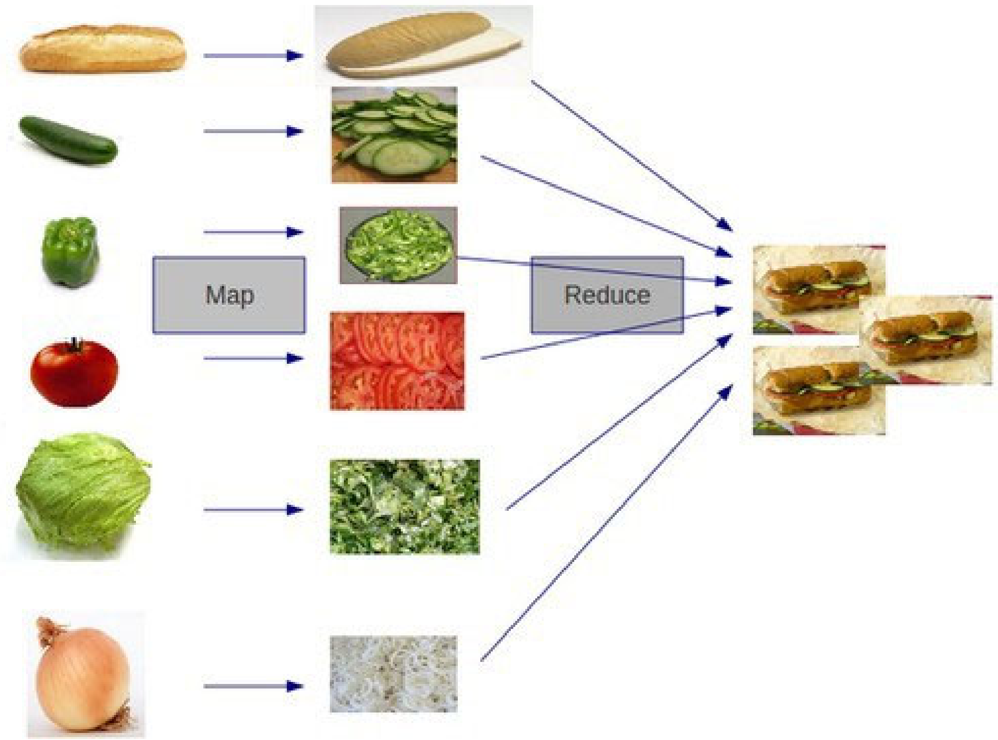
Origins

- Based on Google development
  - Conceived to compute the page Rank
- Data processing framework
  - Facilitate scalability
  - Hidden parallelism
  - Transparent distribution
    - Exploit data locality
    - Balance workload
  - Resilience to failure
    - Fine grained fault tolerance
- Useful in any domain

MapReduce as a DDBMS component

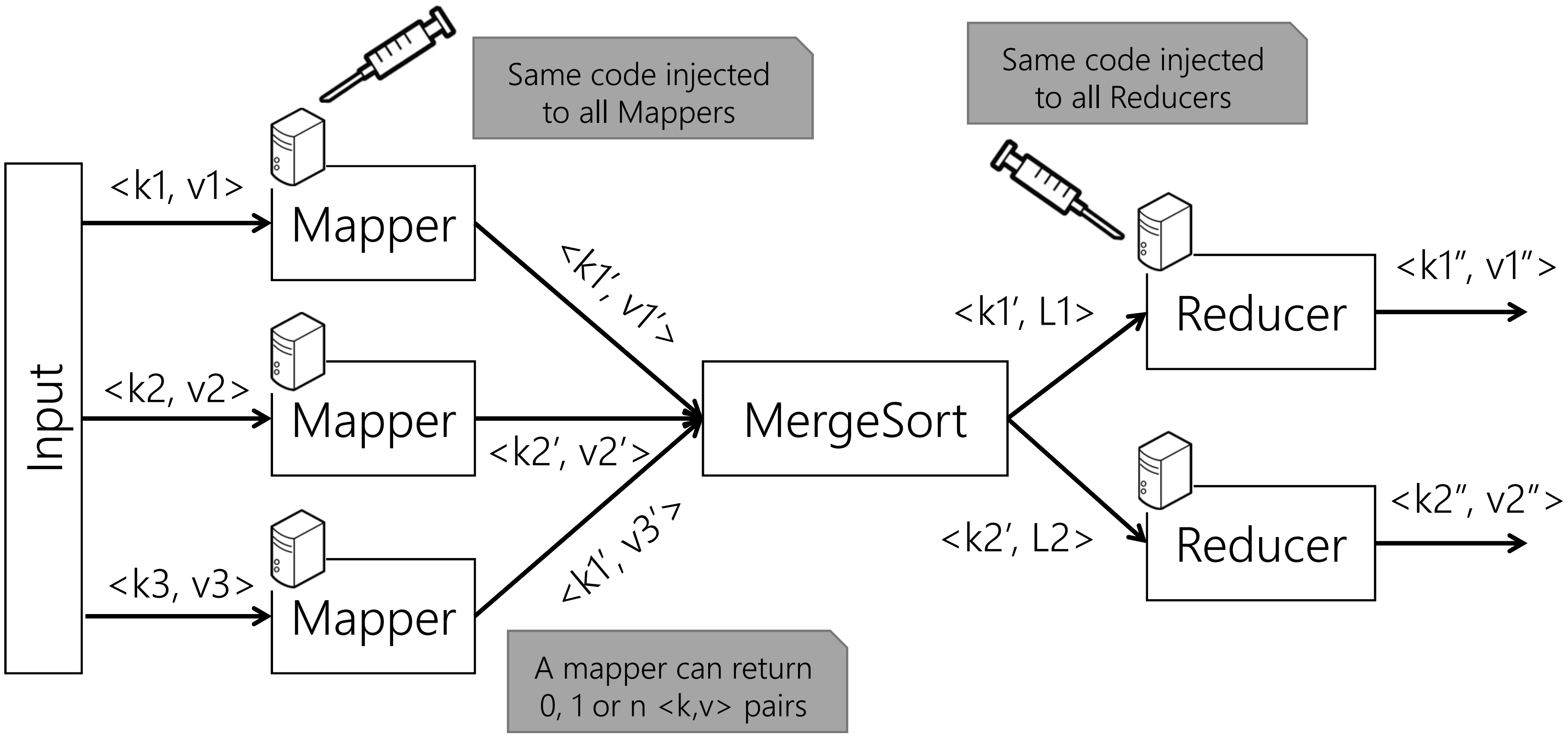


Chain production



Components and use

The MapReduce framework



The MapReduce framework in detail

1. Input: read input from a DFS
2. Map: for each input <key_{in}, $value_{in}$>
   - generate zero-to-many <key_{map}, $value_{map}$>
3. Partition: assign sets of <key_{map}, $value_{map}$> to reducer machines
4. Shuffle: data are shipped to reducer machines using a DFS
5. Sort&Merge: reducers sort their input data by key
6. Reduce: for each key_{map}
   - the set $value_{map}$ is processed to produce zero-to-many <key_{red}, $value_{red}$>
7. Output: writes the result of reducers to the DFS

Formal definition

- Single input
  - Data are represented as <key, value> pairs
    - Value can be anything (structured or not)
- Functional programming
  - Map phase, for each input <key, value> a function *f* is applied that returns a multiset of new <key, value> pairs:

$$f(\langle k, v\rangle) \mapsto \{\langle k_1, v_1\rangle, \ldots, \langle k_n, v_n\rangle\}$$

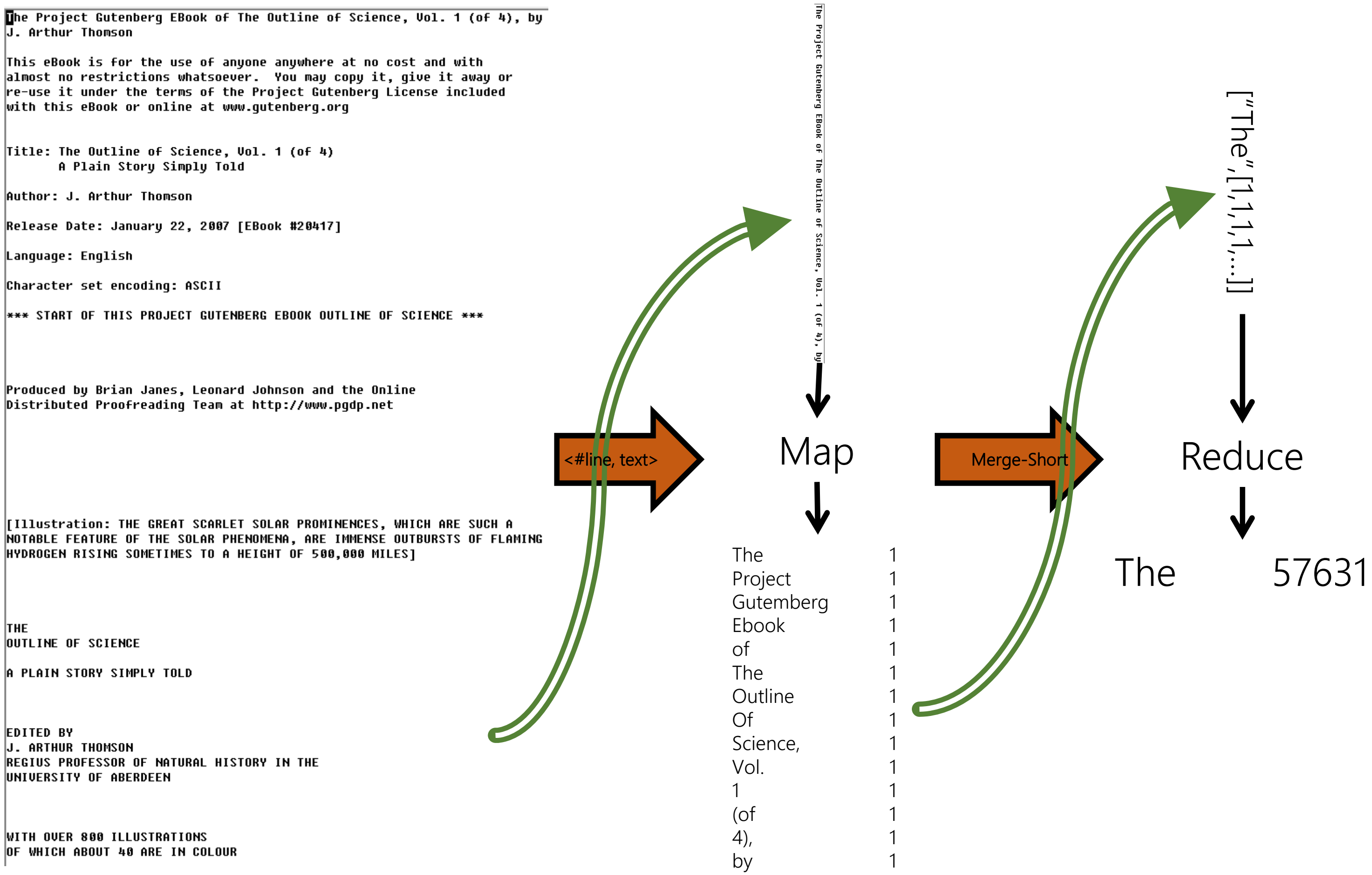
  - Reduce phase, all pairs with the same key are grouped and a function *g* is applied, which returns also a multiset of new <key, value> pairs:

$$g(\langle k, \{v_1, \ldots, v_n\}\rangle) \mapsto \{\langle k_1, v_1\rangle, \ldots, \langle k_m, v_m\rangle\}$$

MapReduce examples

Word count

Word count example



WordCount Code Example

```
public void map(LongWritable key, Text value) {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
        write(new Text(tokenizer.nextToken()), new IntWritable(1));
    }
}

public void reduce(Text key, Iterable<IntWritable> values) {
    int sum = 0;
    for (IntWritable val : values) {
        sum += val.get();
    }
    write(key, new IntWritable(sum));
}
```

WordCount Code Example

```
public void map( Key , Value ) {
Blackbox
    write( Key , Value );
  }
}

public void reduce( Key , Values ) {
Blackbox
  write( Key , Value );
}
```

MapReduce examples

Common friends

Friends in common example

- In a social network (e.g., Facebook), we aim to compute the friends in common for every pair of users
  - This is a value that does not frequently change, so it can be precomputed
- Friends are stored as

Person -> [List of friends]

  - A → B C D
  - B → A C D E
  - C → A B D E
  - D → A B C E
  - E → B C D

Friends in common – Map task

- For every friend in the list, the mapper will generate a <k,v>
  - Key: the input key concatenated with one friend in alphabetical order
  - Value: the whole list of friends
- Keys will be sorted, a pair of friends go to the same reducer

A → B C D
    (A B) → B C D
    (A C) → B C D
    (A D) → B C D

B → A C D E
    (A B) → A C D E
    (B C) → A C D E
    (B D) → A C D E
    (B E) → A C D E

C → A B D E
    (A C) → A B D E
    (B C) → A B D E
    (C D) → A B D E
    (C E) → A B D E

. . .

Friends in common – Reduce task

- Reducers receive two lists of friends per pair of people

(A B) → (B C D) (A C D E)
(A C) → (B C D) (A B D E)
(A D) → (B C D) (A B C E)
…

- The reduce function intersects the lists of values and generates the same key

(A B) → (C D)
(A C) → (B D)
(A D) → (B C)
…

- Now, when D visits A's profile we can lookup (A D) to see their common friends

Relational algebra in MapReduce

MapReduce Genericity

- Supported in many store systems
  - HBase, MongoDB, CouchDB, etc.
- Programming paradigm is computationally complete
  - Any data process can be adapted to it
    - Some tasks better adapt to it than others
    - Not necessarily efficient
      - Optimization is very limited because of lack of expressivity
- Signature is closed
  - Iterations can be chained
    - Fault tolerance is not guaranteed in between
    - Resources are released to be just requested again
- Criticized for being too low-level
  - APIs for Ruby, Python, Java, C++, etc.
  - Attempts to build declarative languages on top
    - SQL-like
      - HiveQL
      - Cassandra Query Language (CQL)

Relational operations: Projection

$$\pi_{a_{i_1},\ldots,a_{i_n}}(T) \Rightarrow \begin{cases} \texttt{map}(\text{key } k, \text{value } v) \mapsto [(\texttt{prj}_{a_{i_1},\ldots,a_{i_n}}(k \oplus v), 1)] \\ \texttt{reduce}(\text{key } ik, \text{vset } ivs) \mapsto [(ik)] \end{cases}$$

Relational operations: Cross Product

$$T \times S \Rrightarrow \begin{cases} \texttt{map}(\text{key } k, \text{value } v) \mapsto \\ \begin{cases} [(\mathrm{h}_T(k) \bmod D, k \oplus v)] & \text{if input}(k \oplus v) = T, \\ [(0, k \oplus v), .., (D-1, k \oplus v)] & \text{if input}(k \oplus v) = S. \end{cases} \\ \texttt{reduce}(\text{key } ik, \text{vset } ivs) \mapsto \\ \Big[\texttt{crossproduct}(T_{ik}, S) \mid \\ \quad T_{ik} = \{iv \mid iv \in ivs \land \text{input}(iv) = T\}, \\ \quad S = \{iv \mid iv \in ivs \land \text{input}(iv) = S\}\Big] \end{cases}$$

Closing

Summary

- MapReduce usefulness and benefits
- MapReduce programming model
  - Expressivity
- Relational algebra in MapReduce

References

- J. Dean et al. *MapReduce: Simplified Data Processing on Large Clusters*. OSDI'04
- A. Pavlo et al. *A Comparison of Approaches to Large-Scale Data Analysis*. SIGMOD, 2009
- J. Dittrich et al. *Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing)*. Proc. VLDB Endow. 3(1-2), 2010
- M. Stonebraker et al. *MapReduce and parallel DBMSs: friends or foes?* Communication of ACM 53(1), 2010
- S. Abiteboul et al. *Web data management*. Cambridge University Press, 2011
- A. Rajaraman et al. *Mining massive data sets*. Cambridge University Press, 2012
- P. Sadagale and M. Fowler. *NoSQL distilled*. Addison-Wesley, 2013