

MapReduce

Krzysztof Dembczyński

Intelligent Decision Support Systems Laboratory (IDSS)
Poznań University of Technology, Poland



Software Development Technologies
Master studies, second semester
Academic year 2018/19 (winter course)

Review of previous lectures

- Mining of massive datasets.
- Classification and regression.
- Evolution of database systems:
 - ▶ Evolution of database processing systems,
 - ▶ Analytical database systems,
 - ▶ Star schema,
 - ▶ Processing of massive datasets.

Outline

- 1 Motivation
- 2 MapReduce
- 3 Algorithms in MapReduce
- 4 Summary

Outline

- 1 Motivation
- 2 MapReduce
- 3 Algorithms in MapReduce
- 4 Summary

Motivation

- Computational burden \rightarrow divide and conquer

Motivation

- Computational burden \rightarrow divide and conquer
 - ▶ Data partitioning

Motivation

- Computational burden → divide and conquer
 - ▶ Data partitioning
 - ▶ Distributed systems

Motivation

- Traditional DBMS vs. NoSQL

Motivation

- Traditional DBMS vs. NoSQL
- New emerging applications: search engines, social networks, online shopping, online advertising, recommender systems, etc.

Motivation

- Traditional DBMS vs. NoSQL
- New emerging applications: search engines, social networks, online shopping, online advertising, recommender systems, etc.
- New computational challenges: WordCount, PageRank, etc.

Motivation

- Traditional DBMS vs. NoSQL
- New emerging applications: search engines, social networks, online shopping, online advertising, recommender systems, etc.
- New computational challenges: WordCount, PageRank, etc.
- Computational burden → distributed systems

Motivation

- Traditional DBMS vs. NoSQL
- New emerging applications: search engines, social networks, online shopping, online advertising, recommender systems, etc.
- New computational challenges: WordCount, PageRank, etc.
- Computational burden → distributed systems
 - ▶ Scaling-out instead of scaling-up

Motivation

- Traditional DBMS vs. NoSQL
- New emerging applications: search engines, social networks, online shopping, online advertising, recommender systems, etc.
- New computational challenges: WordCount, PageRank, etc.
- Computational burden → distributed systems
 - ▶ Scaling-out instead of scaling-up
 - ▶ Move-code-to-data

MapReduce-based systems

- Accessible – run on large clusters of commodity machines or on cloud computing services such as AWS (Amazon Web Services).

MapReduce-based systems

- Accessible – run on large clusters of commodity machines or on cloud computing services such as AWS (Amazon Web Services).
- Robust – are intended to run on commodity hardware; designed with the assumption of frequent hardware malfunctions; they can gracefully handle most such failures.

MapReduce-based systems

- Accessible – run on large clusters of commodity machines or on cloud computing services such as AWS (Amazon Web Services).
- Robust – are intended to run on commodity hardware; designed with the assumption of frequent hardware malfunctions; they can gracefully handle most such failures.
- Scalable – scales linearly to handle larger data by adding more nodes to the cluster.

MapReduce-based systems

- Accessible – run on large clusters of commodity machines or on cloud computing services such as AWS (Amazon Web Services).
- Robust – are intended to run on commodity hardware; designed with the assumption of frequent hardware malfunctions; they can gracefully handle most such failures.
- Scalable – scales linearly to handle larger data by adding more nodes to the cluster.
- Simple – allow users to quickly write efficient parallel code.

MapReduce: Two simple procedures

- Word count: A basic operation for every search engine.
- Matrix-vector multiplication: A fundamental step in many algorithms, for example, in PageRank.

MapReduce: Two simple procedures

- Word count: A basic operation for every search engine.
- Matrix-vector multiplication: A fundamental step in many algorithms, for example, in PageRank.
- How to implement these procedures for efficient execution in a distributed system?

MapReduce: Two simple procedures

- Word count: A basic operation for every search engine.
- Matrix-vector multiplication: A fundamental step in many algorithms, for example, in PageRank.
- How to implement these procedures for efficient execution in a distributed system?
- How much can we gain by such implementation?

MapReduce: Two simple procedures

- Word count: A basic operation for every search engine.
- Matrix-vector multiplication: A fundamental step in many algorithms, for example, in PageRank.
- How to implement these procedures for efficient execution in a distributed system?
- How much can we gain by such implementation?
- Let us focus on the word count problem ...

Word count

- Count the number of times each word occurs in a set of documents:

Do as I say, not as I do.

| Word | Count |
|------|-------|
| as | 2 |
| do | 2 |
| i | 2 |
| not | 1 |
| say | 1 |

Word count

- Let us write the procedure in pseudo-code for a single machine:

Word count

- Let us write the procedure in pseudo-code for a single machine:

```
define wordCount as Multiset;  
  
for each document in documentSet {  
    T = tokenize(document);  
  
    for each token in T {  
        wordCount[token]++;  
    }  
}  
  
display(wordCount);
```


Word count

- Let us write the procedure in pseudo-code for many machines:

Word count

- Let us write the procedure in pseudo-code for many machines:
 - ▶ First step:

```
define wordCount as Multiset;  
  
for each document in documentSubset {  
    T = tokenize(document);  
    for each token in T {  
        wordCount[token]++;  
    }  
}  
  
sendToSecondPhase(wordCount);
```

Word count

- Let us write the procedure in pseudo-code for many machines:

- ▶ First step:

```
define wordCount as Multiset;  
  
for each document in documentSubset {  
    T = tokenize(document);  
    for each token in T {  
        wordCount[token]++;  
    }  
}  
  
sendToSecondPhase(wordCount);
```

- ▶ Second step:

```
define totalWordCount as Multiset;  
  
for each wordCount received from firstPhase {  
    multisetAdd (totalWordCount, wordCount);  
}
```

- Should be there one or many workers running the totalWordCount procedure?

Word count

- To make the procedure work properly across a cluster of distributed machines, we need to add a number of functionalities:

Word count

- To make the procedure work properly across a cluster of distributed machines, we need to add a number of functionalities:
 - ▶ Store files over many processing machines (of phase one).

Word count

- To make the procedure work properly across a cluster of distributed machines, we need to add a number of functionalities:
 - ▶ Store files over many processing machines (of phase one).
 - ▶ Write a disk-based hash table permitting processing without being limited by RAM capacity.

Word count

- To make the procedure work properly across a cluster of distributed machines, we need to add a number of functionalities:
 - ▶ Store files over many processing machines (of phase one).
 - ▶ Write a disk-based hash table permitting processing without being limited by RAM capacity.
 - ▶ Partition the intermediate data (that is, wordCount) from phase one.

Word count

- To make the procedure work properly across a cluster of distributed machines, we need to add a number of functionalities:
 - ▶ Store files over many processing machines (of phase one).
 - ▶ Write a disk-based hash table permitting processing without being limited by RAM capacity.
 - ▶ Partition the intermediate data (that is, wordCount) from phase one.
 - ▶ Shuffle the partitions to the appropriate machines in phase two.

Word count

- To make the procedure work properly across a cluster of distributed machines, we need to add a number of functionalities:
 - ▶ Store files over many processing machines (of phase one).
 - ▶ Write a disk-based hash table permitting processing without being limited by RAM capacity.
 - ▶ Partition the intermediate data (that is, `wordCount`) from phase one.
 - ▶ Shuffle the partitions to the appropriate machines in phase two.
 - ▶ Ensure fault tolerance.

Outline

- ① Motivation
- ② MapReduce
- ③ Algorithms in MapReduce
- ④ Summary

MapReduce

- MapReduce programs are executed in two main phases, called mapping and reducing:

MapReduce

- MapReduce programs are executed in two main phases, called mapping and reducing:
 - ▶ Map: the map function is written to convert input elements to key-value pairs.

MapReduce

- MapReduce programs are executed in two main phases, called mapping and reducing:
 - ▶ Map: the map function is written to convert input elements to key-value pairs.
 - ▶ Reduce: the reduce function is written to take pairs consisting of a key and its list of associated values and combine those values in some way.

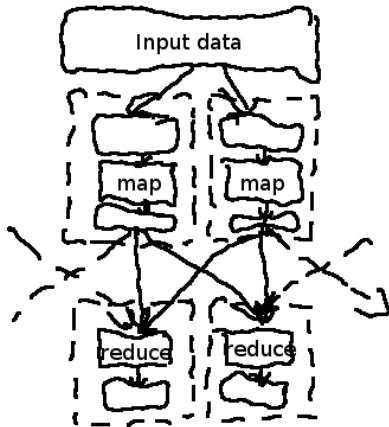
MapReduce

- The complete data flow:

| | Input | Output |
|--------|---|-----------------------------------|
| map | <code>(<k1, v1>)</code> | <code>list(<k2, v2>)</code> |
| reduce | <code>(<k2, list(<v2>))</code> | <code>list(<k3, v3>)</code> |

MapReduce

Figure: The complete data flow



MapReduce

- The complete data flow:

MapReduce

- The complete data flow:
 - ▶ The input is structured as a list of key-value pairs: `list(<k1,v1>)`.

MapReduce

- The complete data flow:
 - ▶ The input is structured as a list of key-value pairs: `list(<k1,v1>)`.
 - ▶ The list of key-value pairs is broken up and each individual key-value pair, `<k1,v1>`, is processed by calling the map function of the mapper (the key `k1` is often ignored by the mapper).

MapReduce

- The complete data flow:
 - ▶ The input is structured as a list of key-value pairs: `list(<k1,v1>)`.
 - ▶ The list of key-value pairs is broken up and each individual key-value pair, `<k1,v1>`, is processed by calling the map function of the mapper (the key `k1` is often ignored by the mapper).
 - ▶ The mapper transforms each `<k1,v1>` pair into a list of `<k2,v2>` pairs.

MapReduce

- The complete data flow:
 - ▶ The input is structured as a list of key-value pairs: `list(<k1,v1>)`.
 - ▶ The list of key-value pairs is broken up and each individual key-value pair, `<k1,v1>`, is processed by calling the map function of the mapper (the key `k1` is often ignored by the mapper).
 - ▶ The mapper transforms each `<k1,v1>` pair into a list of `<k2,v2>` pairs.
 - ▶ The key-value pairs are processed in arbitrary order.

MapReduce

- The complete data flow:
 - ▶ The input is structured as a list of key-value pairs: `list(<k1,v1>)`.
 - ▶ The list of key-value pairs is broken up and each individual key-value pair, `<k1,v1>`, is processed by calling the map function of the mapper (the key `k1` is often ignored by the mapper).
 - ▶ The mapper transforms each `<k1,v1>` pair into a list of `<k2,v2>` pairs.
 - ▶ The key-value pairs are processed in arbitrary order.
 - ▶ The output of all the mappers are (conceptually) aggregated into one giant list of `<k2,v2>` pairs. All pairs sharing the same `k2` are grouped together into a new aggregated key-value pair: `<k2,list(v2)>`.

MapReduce

- The complete data flow:
 - ▶ The input is structured as a list of key-value pairs: `list(<k1,v1>)`.
 - ▶ The list of key-value pairs is broken up and each individual key-value pair, `<k1,v1>`, is processed by calling the map function of the mapper (the key `k1` is often ignored by the mapper).
 - ▶ The mapper transforms each `<k1,v1>` pair into a list of `<k2,v2>` pairs.
 - ▶ The key-value pairs are processed in arbitrary order.
 - ▶ The output of all the mappers are (conceptually) aggregated into one giant list of `<k2,v2>` pairs. All pairs sharing the same `k2` are grouped together into a new aggregated key-value pair: `<k2,list(v2)>`.
 - ▶ The framework asks the reducer to process each one of these aggregated key-value pairs individually.

Combiner and partitioner

- Beside map and reduce there are two other important elements that can be implemented within the MapReduce framework to control the data flow.

Combiner and partitioner

- Beside map and reduce there are two other important elements that can be implemented within the MapReduce framework to control the data flow.
- **Combiner** – perform local aggregation (the reduce step) on the map node.

Combiner and partitioner

- Beside map and reduce there are two other important elements that can be implemented within the MapReduce framework to control the data flow.
- **Combiner** – perform local aggregation (the reduce step) on the map node.
- **Partitioner** – divide the key space of the map output and assign the key-value pairs to reducers.

WordCount in MapReduce

- Map:
 - ▶ For a pair $\langle k1, \text{document} \rangle$ produce a sequence of pairs $\langle \text{token}, 1 \rangle$, where `token` is a token/word found in the document.

```
map(String filename, String document) {  
    List<String> T = tokenize(document);  
  
    for each token in T {  
        emit ((String)token, (Integer) 1);  
    }  
}
```

WordCount in MapReduce

- Reduce

- ▶ For a pair $\langle \text{word}, \text{list}(1, 1, \dots, 1) \rangle$ sum up all ones appearing in the list and return $\langle \text{word}, \text{sum} \rangle$, where sum is the sum of ones.

```
reduce(String token, List<Integer> values) {  
    Integer sum = 0;  
  
    for each value in values {  
        sum = sum + value;  
    }  
  
    emit ((String)token, (Integer) sum);  
}
```

Matrix-vector Multiplication

- Let A to be large $n \times m$ matrix, and x a long vector of size m .
- The matrix-vector multiplication is defined as:

Matrix-vector Multiplication

- Let \mathbf{A} to be large $n \times m$ matrix, and \mathbf{x} a long vector of size m .
- The matrix-vector multiplication is defined as:

$$\mathbf{Ax} = \mathbf{v},$$

where $\mathbf{v} = (v_1, \dots, v_n)$ and

$$v_i = \sum_{j=1}^m a_{ij}x_j.$$

Matrix-vector multiplication

- Let us first assume that m is large, but not so large that vector \mathbf{x} cannot fit in main memory, and be part of the input to every Map task.
- The matrix \mathbf{A} is stored with explicit coordinates, as a triple (i, j, a_{ij}) .
- We also assume the position of element x_j in the vector \mathbf{x} will be stored in the analogous way.

Matrix-vector multiplication

- **Map:**

Matrix-vector multiplication

- **Map:** Each map task will take the entire vector x and a chunk of the matrix A . From each matrix element a_{ij} it produces the key-value pair $(i, a_{ij}x_j)$. Thus, all terms of the sum that make up the component v_i of the matrix-vector product will get the same key.

Matrix-vector multiplication

- **Map:** Each map task will take the entire vector x and a chunk of the matrix A . From each matrix element a_{ij} it produces the key-value pair $(i, a_{ij}x_j)$. Thus, all terms of the sum that make up the component v_i of the matrix-vector product will get the same key.
- **Reduce:**

Matrix-vector multiplication

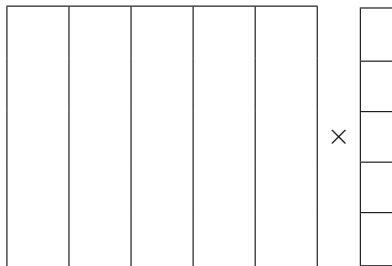
- **Map:** Each map task will take the entire vector \mathbf{x} and a chunk of the matrix \mathbf{A} . From each matrix element a_{ij} it produces the key-value pair $(i, a_{ij}x_j)$. Thus, all terms of the sum that make up the component v_i of the matrix-vector product will get the same key.
- **Reduce:** A reduce task has simply to sum all the values associated with a given key i . The result will be a pair (i, v_i) where:

$$v_i = \sum_{j=1}^m a_{ij}x_j.$$

Matrix-Vector Multiplication with Large Vector v

Matrix-Vector Multiplication with Large Vector v

- Divide the matrix into vertical stripes of equal width and divide the vector into an equal number of horizontal stripes, of the same height.



- The i th stripe of the matrix multiplies only components from the i th stripe of the vector.
- Thus, we can divide the matrix into one file for each stripe, and do the same for the vector.

Matrix-Vector Multiplication with Large Vector v

- Each Map task is assigned a chunk from one the stripes of the matrix and gets the entire corresponding stripe of the vector.
- The Map and Reduce tasks can then act exactly as in the case where Map tasks get the entire vector.

Outline

- 1 Motivation
- 2 MapReduce
- 3 Algorithms in MapReduce**
- 4 Summary

Algorithms in MapReduce

- How to implement fundamental algorithms in MapReduce?
 - ▶ Relational-Algebra Operations.
 - ▶ Matrix multiplication.

Relational-algebra operations

Example (Relation **Links**)

| From | To |
|------|------|
| url1 | url2 |
| url1 | url3 |
| url2 | url3 |
| url2 | url4 |
| ... | ... |

Relational-algebra operations

- We assume that input and output are *real* relations (no duplicated rows)

Relational-algebra operations

- We assume that input and output are *real* relations (no duplicated rows)
- Operations:
 - ▶ Selection
 - ▶ Projection
 - ▶ Union, intersection, and difference
 - ▶ Natural join
 - ▶ Grouping and aggregation

Relational-algebra operations

- We assume that input and output are *real* relations (no duplicated rows)
- Operations:
 - ▶ Selection
 - ▶ Projection
 - ▶ Union, intersection, and difference
 - ▶ Natural join
 - ▶ Grouping and aggregation
- Notation:
 - ▶ R, S - relation
 - ▶ t, t' - a tuple
 - ▶ \mathcal{C} - a condition of selection
 - ▶ A, B, C - subset of attributes
 - ▶ a, b, c - attribute values for a given subset of attributes

Selection

- **Operation:** $\text{Select}_{\mathcal{C}}(R)$

Selection

- **Operation:** $\text{Select}_{\mathcal{C}}(R)$
- **Map:**

Selection

- **Operation:** $\text{Select}_{\mathcal{C}}(R)$
- **Map:** For each tuple t in R , test if it satisfies \mathcal{C} . If so, produce the key-value pair (t, t) . That is, both the key and value are t .
- **Reduce:**

Selection

- **Operation:** $\text{Select}_{\mathcal{C}}(R)$
- **Map:** For each tuple t in R , test if it satisfies \mathcal{C} . If so, produce the key-value pair (t, t) . That is, both the key and value are t .
- **Reduce:** The Reduce function is the identity. It simply passes each key-value pair to the output.

Selection

- **Operation:** $\text{Select}_{\mathcal{C}}(R)$
- **Map:** For each tuple t in R , test if it satisfies \mathcal{C} . If so, produce the key-value pair (t, t) . That is, both the key and value are t .
- **Reduce:** The Reduce function is the identity. It simply passes each key-value pair to the output.

| | Input | Output |
|--------|---------------------------------------|-------------------------------------|
| map | $\langle k1, t \rangle$ | $\text{list}(\langle t, t \rangle)$ |
| reduce | $(\langle t, \text{list}(t) \rangle)$ | $\text{list}(\langle t, t \rangle)$ |

Projection

- **Operation:** $\text{Project}_A(R)$

Projection

- **Operation:** $\text{Project}_A(R)$
- **Map:**

Projection

- **Operation:** $\text{Project}_A(R)$
- **Map:** For each tuple t in R , construct a tuple t' by eliminating from t those components whose attributes are not in A . Output the key-value pair (t', t') .
- **Reduce:**

Projection

- **Operation:** $\text{Project}_A(R)$
- **Map:** For each tuple t in R , construct a tuple t' by eliminating from t those components whose attributes are not in A . Output the key-value pair (t', t') .
- **Reduce:** For each key t' produced by any of the Map tasks, there will be one or more key-value pairs (t', t') . The Reduce function turns $(t', [t', t', \dots, t'])$ into (t', t') , so it produces exactly one pair (t', t') for this key t' .

Projection

- **Operation:** $\text{Project}_A(R)$
- **Map:** For each tuple t in R , construct a tuple t' by eliminating from t those components whose attributes are not in A . Output the key-value pair (t', t') .
- **Reduce:** For each key t' produced by any of the Map tasks, there will be one or more key-value pairs (t', t') . The Reduce function turns $(t', [t', t', \dots, t'])$ into (t', t') , so it produces exactly one pair (t', t') for this key t' .

| | Input | Output |
|--------|--|---------------------------------------|
| map | $\langle k1, t \rangle$ | $\text{list}(\langle t', t' \rangle)$ |
| reduce | $(\langle t', \text{list}(t', \dots, t') \rangle)$ | $\text{list}(\langle t', t' \rangle)$ |

Union

- **Operation:** $\text{Union}(R, S)$

Union

- **Operation:** $\text{Union}(R, S)$
- **Map:**

Union

- **Operation:** $\text{Union}(R, S)$
- **Map:** Turn each input tuple t either from relation R or S into a key-value pair (t, t) .
- **Reduce:**

Union

- **Operation:** $\text{Union}(R, S)$
- **Map:** Turn each input tuple t either from relation R or S into a key-value pair (t, t) .
- **Reduce:** Associated with each key t there will be either one or two values. Produce output (t, t) in either case.

Union

- **Operation:** $\text{Union}(R, S)$
- **Map:** Turn each input tuple t either from relation R or S into a key-value pair (t, t) .
- **Reduce:** Associated with each key t there will be either one or two values. Produce output (t, t) in either case.

| | Input | Output |
|--------|--|-------------------------------------|
| map | $\langle k1, t \rangle$ | $\text{list}(\langle t, t \rangle)$ |
| reduce | $(\langle t, \text{list}(t) \rangle)$ or $(\langle t, \text{list}(t, t) \rangle)$ | $\text{list}(\langle t, t \rangle)$ |

Intersection

- **Operation:** $\text{Intersection}(R, S)$

Intersection

- **Operation:** $\text{Intersection}(R, S)$
- **Map:**

Intersection

- **Operation:** $\text{Intersection}(R, S)$
- **Map:** Turn each input tuple t either from relation R or S into a key-value pair (t, t) .
- **Reduce:**

Intersection

- **Operation:** $\text{Intersection}(R, S)$
- **Map:** Turn each input tuple t either from relation R or S into a key-value pair (t, t) .
- **Reduce:** If key t has value list $[t, t]$, then produce (t, t) . Otherwise, produce nothing.

Intersection

- **Operation:** $\text{Intersection}(R, S)$
- **Map:** Turn each input tuple t either from relation R or S into a key-value pair (t, t) .
- **Reduce:** If key t has value list $[t, t]$, then produce (t, t) . Otherwise, produce nothing.

| | Input | Output |
|--------|--|--|
| map | $\langle k1, t \rangle$ | $\text{list}(\langle t, t \rangle)$ |
| reduce | $\langle t, \text{list}(t) \rangle$ or $\langle t, \text{list}(t, t) \rangle$ | $\text{list}(\langle t, t \rangle)$ if $\langle t, \text{list}(t, t) \rangle$ |

Minus

- **Operation:** $\text{Minus}(R, S)$

Minus

- **Operation:** $\text{Minus}(R, S)$
- **Map:**

Minus

- **Operation:** $\text{Minus}(R, S)$
- **Map:** For a tuple t in R , produce key-value pair $(t, \text{name}(R))$, and for a tuple t in S , produce key-value pair $(t, \text{name}(S))$.
- **Reduce:**

Minus

- **Operation:** $\text{Minus}(R, S)$
- **Map:** For a tuple t in R , produce key-value pair $(t, \text{name}(R))$, and for a tuple t in S , produce key-value pair $(t, \text{name}(S))$.
- **Reduce:** For each key t , do the following.
 - 1 If the associated value list is $[\text{name}(R)]$, then produce (t, t) .
 - 2 If the associated value list is anything else, which could only be $[\text{name}(R), \text{name}(S)]$, $[\text{name}(S), \text{name}(R)]$, or $[\text{name}(S)]$, produce nothing.

Minus

- **Operation:** $\text{Minus}(R, S)$
- **Map:** For a tuple t in R , produce key-value pair $(t, \text{name}(R))$, and for a tuple t in S , produce key-value pair $(t, \text{name}(S))$.
- **Reduce:** For each key t , do the following.
 - 1 If the associated value list is $[\text{name}(R)]$, then produce (t, t) .
 - 2 If the associated value list is anything else, which could only be $[\text{name}(R), \text{name}(S)]$, $[\text{name}(S), \text{name}(R)]$, or $[\text{name}(S)]$, produce nothing.

| | Input | Output |
|--------|---|---|
| map | $\langle k1, (t, R) \rangle$ or $\langle k1, (t, S) \rangle$ or | $\text{list}(\langle t, R \rangle)$ or $\text{list}(\langle t, S \rangle)$ |
| reduce | $\langle t, \text{list}(R) \rangle$ or $\langle t, \text{list}(S) \rangle$ or $\langle t, \text{list}(R, S) \rangle$ or $\langle t, \text{list}(S, R) \rangle$ | $\text{list}(\langle t, t \rangle)$ if $\langle t, \text{list}(R) \rangle$ |

Natural Join

- **Operation:** $\text{Join}_B(R, S)$

Natural Join

- **Operation:** $\text{Join}_B(R, S)$
- Assume that we join relation $R(A, B)$ with relation $S(B, C)$ that share the same attribute B .
- **Map:**

Natural Join

- **Operation:** $\text{Join}_B(R, S)$
- Assume that we join relation $R(A, B)$ with relation $S(B, C)$ that share the same attribute B .
- **Map:** For each tuple (a, b) of R , produce the key-value pair $(b, (\text{name}(R), a))$. For each tuple (b, c) of S , produce the key-value pair $(b, (\text{name}(S), c))$.
- **Reduce:**

Natural Join

- **Operation:** $\text{Join}_B(R, S)$
- Assume that we join relation $R(A, B)$ with relation $S(B, C)$ that share the same attribute B .
- **Map:** For each tuple (a, b) of R , produce the key-value pair $(b, (\text{name}(R), a))$. For each tuple (b, c) of S , produce the key-value pair $(b, (\text{name}(S), c))$.
- **Reduce:** Each key value b will be associated with a list of pairs that are either of the form $(\text{name}(R), a)$ or $(\text{name}(S), c)$. Construct all pairs consisting of one with first component $\text{name}(R)$ and the other with first component S , say $(\text{name}(R), a)$ and $(\text{name}(S), c)$. The output for key b is a list $(b, (a1, b, c1)), (b, (a2, b, c2)), \dots$

Natural Join

- **Operation:** $\text{Join}_B(R, S)$
- Assume that we join relation $R(A, B)$ with relation $S(B, C)$ that share the same attribute B .
- **Map:** For each tuple (a, b) of R , produce the key-value pair $(b, (\text{name}(R), a))$. For each tuple (b, c) of S , produce the key-value pair $(b, (\text{name}(S), c))$.
- **Reduce:** Each key value b will be associated with a list of pairs that are either of the form $(\text{name}(R), a)$ or $(\text{name}(S), c)$. Construct all pairs consisting of one with first component $\text{name}(R)$ and the other with first component S , say $(\text{name}(R), a)$ and $(\text{name}(S), c)$. The output for key b is a list $(b, (a1, b, c1)), (b, (a2, b, c2)), \dots$

| | Input | Output |
|--------|--|---|
| map | $\langle k1, (t, R) \rangle$ or $\langle k1, (t, S) \rangle$ or | $\text{list}(\langle b, (a, R) \rangle)$ or $\text{list}(\langle b, (c, S) \rangle)$ |
| reduce | $\langle b, \text{list}((a1, R), \dots, (c1, S), \dots) \rangle$ | $\text{list}(\langle b, (a1, b, c1) \rangle, \dots)$ |

Grouping and Aggregation

- **Operation:** $\text{Aggregate}_{(\theta, A, B)}(R)$

Grouping and Aggregation

- **Operation:** $\text{Aggregate}_{(\theta, A, B)}(R)$
- Assume that we group a relation $R(A, B, C)$ by attributes A and aggregate values of B by using function θ .
- **Map:**

Grouping and Aggregation

- **Operation:** $\text{Aggregate}_{(\theta, A, B)}(R)$
- Assume that we group a relation $R(A, B, C)$ by attributes A and aggregate values of B by using function θ .
- **Map:** For each tuple (a, b, c) produce the key-value pair (a, b) .
- **Reduce:**

Grouping and Aggregation

- **Operation:** $\text{Aggregate}_{(\theta, A, B)}(R)$
- Assume that we group a relation $R(A, B, C)$ by attributes A and aggregate values of B by using function θ .
- **Map:** For each tuple (a, b, c) produce the key-value pair (a, b) .
- **Reduce:** Each key a represents a group. Apply the aggregation operator θ to the list $[b_1, b_2, \dots, b_n]$ of B -values associated with key a . The output is the pair (a, x) , where x is the result of applying θ to the list. For example, if θ is SUM, then $x = b_1 + b_2 + \dots + b_n$, and if θ is MAX, then x is the largest of b_1, b_2, \dots, b_n .

Grouping and Aggregation

- **Operation:** $\text{Aggregate}_{(\theta, A, B)}(R)$
- Assume that we group a relation $R(A, B, C)$ by attributes A and aggregate values of B by using function θ .
- **Map:** For each tuple (a, b, c) produce the key-value pair (a, b) .
- **Reduce:** Each key a represents a group. Apply the aggregation operator θ to the list $[b_1, b_2, \dots, b_n]$ of B -values associated with key a . The output is the pair (a, x) , where x is the result of applying θ to the list. For example, if θ is SUM, then $x = b_1 + b_2 + \dots + b_n$, and if θ is MAX, then x is the largest of b_1, b_2, \dots, b_n .

| | Input | Output |
|--------|---|--|
| map | $\langle k1, t \rangle$ | $\text{list}(\langle a, b \rangle)$ |
| reduce | $\langle a, \text{list}((b1, b2, \dots)) \rangle$ | $\text{list}(\langle a, f(b1, b2, \dots) \rangle)$ |

Matrix Multiplication

- If M is a matrix with element m_{ij} in row i and column j , and N is a matrix with element n_{jk} in row j and column k , then the product:

$$P = MN$$

is the matrix P with element p_{ik} in row i and column k , where:

$$p_{ik} =$$

Matrix Multiplication

- If M is a matrix with element m_{ij} in row i and column j , and N is a matrix with element n_{jk} in row j and column k , then the product:

$$P = MN$$

is the matrix P with element p_{ik} in row i and column k , where:

$$p_{ik} = \sum_j m_{ij}n_{jk}$$

Matrix Multiplication

- We can think of a matrix M and N as a relation with three attributes: the row number, the column number, and the value in that row and column, i.e.,:

$$M(I, J, V) \quad \text{and} \quad N(J, K, W)$$

with the following tuples, respectively:

$$(i, j, m_{ij}) \quad \text{and} \quad (j, k, n_{jk}).$$

- In case of sparsity of M and N , this relational representation is very efficient in terms of space.
- The product MN is almost a natural join followed by grouping and aggregation.

Matrix Multiplication

Matrix Multiplication

- **Map:**

Matrix Multiplication

- **Map:** Send each matrix element m_{ij} to the key value pair:

$$(j, (M, i, m_{ij})).$$

Analogously, send each matrix element n_{jk} to the key value pair:

$$(j, (N, k, n_{jk})).$$

- **Reduce:**

Matrix Multiplication

- **Map:** Send each matrix element m_{ij} to the key value pair:

$$(j, (M, i, m_{ij})).$$

Analogously, send each matrix element n_{jk} to the key value pair:

$$(j, (N, k, n_{jk})).$$

- **Reduce:** For each key j , examine its list of associated values. For each value that comes from M , say (M, i, m_{ij}) , and each value that comes from N , say (N, k, n_{jk}) , produce the tuple

$$(i, k, v = m_{ij}n_{jk}),$$

The output of the Reduce function is a key j paired with the list of all the tuples of this form that we get from j :

$$(j, [(i_1, k_1, v_1), (i_2, k_2, v_2), \dots, (i_p, k_p, v_p)]).$$

Matrix Multiplication

Matrix Multiplication

- **Map:**

Matrix Multiplication

- **Map:** From the pairs that are output from the previous Reduce function produce p key-value pairs:

$$((i_1, k_1), v_1), ((i_2, k_2), v_2), \dots, ((i_p, k_p), v_p).$$

- **Reduce:**

Matrix Multiplication

- **Map:** From the pairs that are output from the previous Reduce function produce p key-value pairs:

$$((i_1, k_1), v_1), ((i_2, k_2), v_2), \dots, ((i_p, k_p), v_p).$$

- **Reduce:** For each key (i, k) , produce the sum of the list of values associated with this key. The result is a pair

$$((i, k), v),$$

where v is the value of the element in row i and column k of the matrix

$$P = MN.$$

Matrix Multiplication with One Map-Reduce Step

- **Map:**

Matrix Multiplication with One Map-Reduce Step

- **Map:** For each element m_{ij} of M , produce a key-value pair

$$((i, k), (M, j, m_{ij})),$$

for $k = 1, 2, \dots$, up to the number of columns of N .

Also, for each element n_{jk} of N , produce a key-value pair

$$((i, k), (N, j, n_{jk})),$$

for $i = 1, 2, \dots$, up to the number of rows of M .

Matrix Multiplication with One Map-Reduce Step

- **Reduce:**

Matrix Multiplication with One Map-Reduce Step

- **Reduce:** Each key (i, k) will have an associated list with all the values

$$(M, j, m_{ij}) \quad \text{and} \quad (N, j, n_{jk}),$$

for all possible values of j . We connect the two values on the list that have the same value of j , for each j :

- ▶ We sort by j the values that begin with M and sort by j the values that begin with N , in separate lists,
- ▶ The j th values on each list must have their third components, m_{ij} and n_{jk} extracted and multiplied,
- ▶ Then, these products are summed and the result is paired with (i, k) in the output of the Reduce function.

Outline

- 1 Motivation
- 2 MapReduce
- 3 Algorithms in MapReduce
- 4 Summary

Summary

- Computational burden → data partitioning, distributed systems.
- New data-intensive challenges like search engines.
- MapReduce: The overall idea and simple algorithms.
- Algorithms in MapReduce.

Bibliography

- J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2014
<http://infolab.stanford.edu/~ullman/mmds.html>
- J.Lin and Ch. Dyer. *Data-Intensive Text Processing with MapReduce*. Morgan and Claypool Publishers, 2010
<http://lintool.github.com/MapReduceAlgorithms/>
- Ch. Lam. *Hadoop in Action*. Manning Publications Co., 2011