Algorithms in MapReduce

Krzysztof Dembczyński

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

Intelligent Decision Support Systems
Master studies, second semester
Academic year 2015/16 (summer course)

Review of the previous lectures

- Mining of massive datasets.
- Evolution of database systems.
- Dimensional modeling.
- ETL and OLAP systems.
- Multidimensional queries.
- Processing of very large data.
- Nearest neighbor search.
- Data partitioning and MapReduce:
  - Data partitioning.
  - The overall idea of the MapReduce paradigm.
  - Hadoop implementation.

Outline

Outline

Algorithms in Map-Reduce

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

Outline

Matrix-vector Multiplication

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

Matrix-vector Multiplication

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

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{v},$$

where $\boldsymbol{v} = (v_1, \ldots, v_n)$ 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 $\boldsymbol{x}$ cannot fit in main memory, and be part of the input to every Map task.
- The matrix $\boldsymbol{A}$ is stored with explicit coordinates, as a triple (i, j, a_{ij}).
- We also assume the position of element x_j in the vector $\boldsymbol{x}$ will be stored in the analogous way.

Matrix-vector multiplication

- **Map**:

Matrix-vector multiplication

- **Map**: each map task will take the entire vector $\boldsymbol{x}$ and a chunk of the matrix $\boldsymbol{A}$. 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 $\boldsymbol{x}$ and a chunk of the matrix $\boldsymbol{A}$. 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

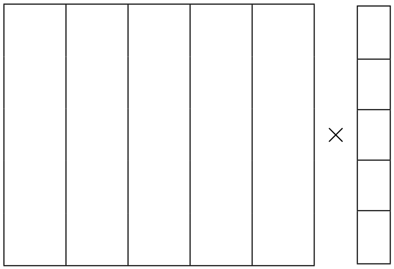
- **Map**: each map task will take the entire vector $\boldsymbol{x}$ and a chunk of the matrix $\boldsymbol{A}$. From each matrix element a_{ij} it produces the key-value pair $(i, a_{ij}x_j)$. Thus, all terms of the sum that make up the component v_i of the matrix-vector product will get the same key.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

Relational-Algebra Operations

Example (Relation **Links**)

From	To
`url1`	`url2`
`url1`	`url3`
`url2`	`url3`
`url2`	`url4`
...	...

Relational-Algebra Operations

- Selection
- Projection
- Union, Intersection, and Difference
- Natural Join
- Grouping and Aggregation

Relational-Algebra Operations

- 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

- **Map**:

Selection

- **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

- **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.

Projection

Projection

- **Map**:

Projection

- **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

- **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 $(t', [t', t', \ldots, t'])$ into (t', t'), so it produces exactly one pair (t', t') for this key t'.

Union

- **Map**:

Union

- **Map**: Turn each input tuple t either from relation R or S into a key-value pair (t, t).
- **Reduce**:

Union

- **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.

Intersection

- **Map**:

Intersection

- **Map**: Turn each input tuple t either from relation R or S into a key-value pair (t, t).
- **Reduce**:

Intersection

- **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.

Minus

- **Map**:

Minus

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

Minus

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

Natural Join

- Let us assume that we join relation $R(A, B)$ with relation $S(B, C)$ that share the same attribute B.
- **Map**:

Natural Join

- Let us 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 $(b, (\texttt{name}(R), a))$. For each tuple (b, c) of S, produce the key-value pair $(b, (\texttt{name}(S), c))$.
- **Reduce**:

Natural Join

- Let us 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 $(b, (\texttt{name}(R), a))$. For each tuple (b, c) of S, produce the key-value pair $(b, (\texttt{name}(S), c))$.
- **Reduce**: Each key value b will be associated with a list of pairs that are either of the form $(\texttt{name}(R), a)$ or $(\texttt{name}(S), c)$. Construct all pairs consisting of one with first component $\texttt{name}(R)$ and the other with first component S, say $(\texttt{name}(R), a)$ and $(\texttt{name}(S), c)$. The output for key b is $(b, [(a1, b, c1), (a2, b, c2), ...])$, that is, b associated with the list of tuples that can be formed from an R-tuple and an S-tuple with a common b value.

Grouping and Aggregation

- Let assume that we group a relation $R(A, B, C)$ by attributes A and aggregate values of B.
- **Map**:

Grouping and Aggregation

- Let assume that we group a relation $R(A, B, C)$ by attributes A and aggregate values of B.
- **Map**: For each tuple (a, b, c) produce the key-value pair (a, b).
- **Reduce**:

Grouping and Aggregation

- Let assume that we group a relation $R(A, B, C)$ by attributes A and aggregate values of B.
- **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 $[b_1, b_2, \ldots, b_n]$ 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 $x = b_1 + b_2 + \ldots + b_n$, and if θ is `MAX`, then x is the largest of $b_1, b_2, \ldots, b_n$.

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:

$$pik =$$

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:

$$pik = \sum_j m_{ij} n_{jk}$$

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:

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

Matrix Multiplication

Matrix Multiplication

- **Map**:

Matrix Multiplication

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

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

- **Reduce**:

Matrix Multiplication

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

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

- **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 $k = 1, 2, \ldots$, 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 $i = 1, 2, \ldots$, 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 jth 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

Extensions to MapReduce

Extensions to MapReduce

- Workflow systems, like Clustera or Hyracks, extend map-reduce from the simple two-step workflow (the Map function feeds the Reduce function) to any collection of functions, with an acyclic graph representing workflow among the functions.

Extensions to MapReduce

- Workflow systems, like Clustera or Hyracks, extend map-reduce from the simple two-step workflow (the Map function feeds the Reduce function) to any collection of functions, with an acyclic graph representing workflow among the functions.
- Pregel is a distributed programming framework for graph algorithms
  - Pregel views its data as a graph.
  - Each node of the graph corresponds roughly to a task
  - Each graph node generates output messages that are destined for other nodes of the graph, and each graph node processes the inputs it receives from other nodes.

Extensions to MapReduce

- Workflow systems, like Clustera or Hyracks, extend map-reduce from the simple two-step workflow (the Map function feeds the Reduce function) to any collection of functions, with an acyclic graph representing workflow among the functions.
- Pregel is a distributed programming framework for graph algorithms
    - Pregel views its data as a graph.
    - Each node of the graph corresponds roughly to a task
    - Each graph node generates output messages that are destined for other nodes of the graph, and each graph node processes the inputs it receives from other nodes.
- PIG and Hive: the former is an implementation of relational algebra on top of Hadoop, while the latter implements a restricted form of SQL on top of Hadoop.
- And many others . . .

Outline

Summary

- Algorithms in MapReduce:
  - Matrix-vector multiplication.
  - Relational-Algebra Operations.
  - Matrix multiplication.
- Extensions of MapReduce

Bibliography

- A. Rajaraman and J. D. Ullman. *Mining of Massive Datasets*.
  Cambridge University Press, 2011
  `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