Data-intensive computing systems

Relational Algebra with MapReduce

University of Verona
Computer Science Department

Damiano Carra

Acknowledgements

- *Credits*
  - *Part of the course material is based on slides provided by the following authors*
    - *Pietro Michiardi, Jimmy Lin*

Relational Algebra Operators

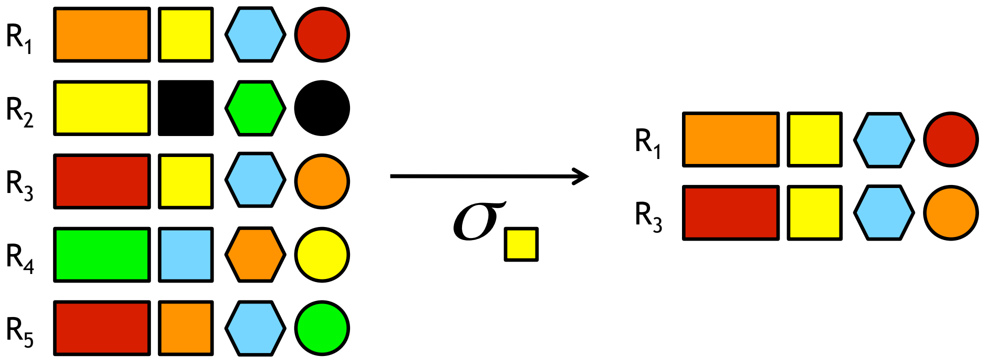
- There are a number of operations on data that fit well the relational algebra model
  - In traditional RDBMS, queries involve retrieval of small amounts of data
  - In this course, we should keep in mind the particular workload underlying MapReduce
  - → Full scans of large amounts of data
  - → Queries are not selective, they process all data
- A review of some terminology
  - A *relation* is a table
  - *Attributes* are the column headers of the table
  - The set of attributes of a relation is called a *schema*
  - Example: $R(A_1, A_2, \ldots, A_n)$ indicates a relation called R whose attributes are $A_1, A_2, \ldots, A_n$

Relational Algebra Operators

- Relations (however big) can be stored in a distributed filesystem
  - If they don't fit in a single machine, they're broken into pieces (think HDFS)
- Next, we review and describe a set of relational algebra operators
  - Intuitive explanation of what they do
  - "Pseudo-code" of their implementation in/by MapReduce

Selection

- Selection: $\sigma_C(R)$
  - Apply condition C to each tuple of relation R
  - Produce in output a relation containing only tuples that satisfy C



Selection in MapReduce

- A full-blown MapReduce implementation is not necessary in practice
  - It can be implemented in the map portion alone
  - Alternatively, it could also be implemented in the reduce portion

- A MapReduce implementation of $\sigma_C(R)$

  Map: For each tuple t in R, check if t satisfies C

  If so, emit a key/value pair (t, t)

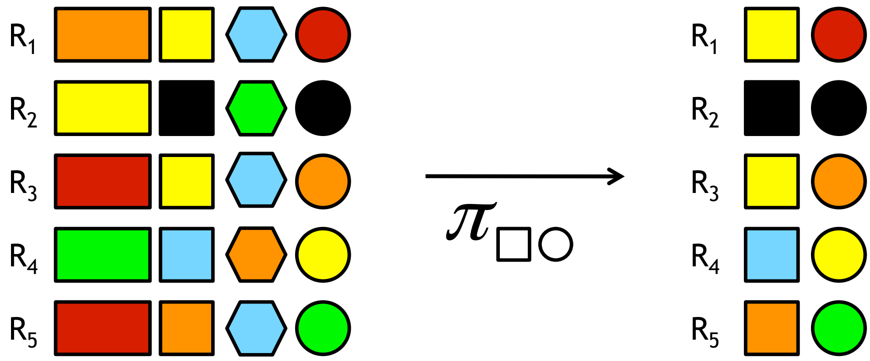
  Reduce: Identity reducer

  Question: single or multiple reducers?

- NOTE: the output is not exactly a relation
  - WHY?

Projections

- Projection: $\pi_S(R)$
  - Given a subset S of relation R attributes
  - Produce in output a relation containing only tuples for the attributes in S



Projections in MapReduce

- Similar process to selection
  - But, projection may cause same tuple to appear several times

- A MapReduce implementation of $\pi_S(R)$

  Map: - For each tuple *t* in R, construct a tuple *t'* by eliminating those components whose attributes are not in S

  - Emit a key/value pair (*t'*, *t'*)

  Reduce: - For each key produced by any of the Map tasks, fetch *t', [t', · · · , t']*

  - Emit a key/value pair (*t'*, *t'*)

- NOTE: the reduce operation is duplicate elimination
  - This operation is associative and commutative, so it is possible to optimize MapReduce by using a `Combiner` in each mapper

Union, Intersection and Difference

- Well known operators on sets
- Apply to the set of tuples in two relations that have the same schema
  - Variations on the theme: work on bags

Unions in MapReduce

- Suppose relations R and S have the same schema
  - Map tasks will be assigned chunks from either R or S
  - Mappers don't do much, just pass by to reducers
  - Reducers do duplicate elimination

- A MapReduce implementation of Union

Map: For each tuple t in R or S, emit a key/value pair (t, t)

Reduce: For each key t, emit a key/value pair (t, t)

Note: each key will have either one or two values

Intersection in MapReduce

- Very similar to computing Union
  - Suppose relations R and S have the same schema
  - The map function is the same (an identity mapper) as for union
  - The reduce function must produce a tuple only if both relations have that tuple

- A MapReduce implementation of Intersection

Map: For each tuple t in R or S, emit a key/value pair (t, t)

Reduce: If key t has value list [t,t], emit a key/value pair (t, t)

Otherwise, emit a key/value pair (t, *NULL*)

Difference in MapReduce

- Assume we have two relations R and S with the same schema
  - The only way a tuple t can appear in the output is if it is in R but not in S
  - The map function can pass tuples from R and S to the reducer
  - NOTE: it must inform the reducer whether the tuple came from R or S

- A MapReduce implementation of Difference

Map: For a tuple t in R emit a key/value pair (t, 'R')

For a tuple t in S, emit a key/value pair (t, 'S')

Reduce: If key t has value list [R], emit a key/value pair (t, t)

Otherwise, emit a key/value pair (t, *NULL*)

i.e., ['R', 'S'] or ['S', 'R'] or ['S']

Grouping and Aggregation

- Grouping and Aggregation: γ_X (R)
  - Given a relation R, partition its tuples according to their values in one set of attributes G
    - The set G is called the grouping attributes
  - Then, for each group, aggregate the values in certain other attributes
    - Aggregation functions: SUM, COUNT, AVG, MIN, MAX, ...

- In the notation, X is a list of elements that can be:
  - A grouping attribute
  - An expression θ(A), where θ is one of the (five) aggregation functions and A is an attribute NOT among the grouping attributes

Grouping and Aggregation

- Grouping and Aggregation: γ_X (R)
  - The result of this operation is a relation with one tuple for each group
  - That tuple has a component for each of the grouping attributes, with the value common to tuples of that group
  - That tuple has another component for each aggregation, with the aggregate value for that group

- Let's work with an example
  - Imagine that a social-networking site has a relation `Friends(User, Friend)`
  - The tuples are pairs (*a*, *b*) such that *b* is a friend of *a*
  - Question: compute the number of friends each member has

Grouping and Aggregation: Example

- How to satisfy the query $\gamma_{User,COUNT(Friend))}$(Friends)
  - This operation groups all the tuples by the value in their first component
  - → There is one group for each user
  - Then, for each group, it counts the number of friends

- Some details
  - The COUNT operation applied to an attribute does not consider the values of that attribute
  - In fact, it counts the number of tuples in the group
  - In SQL, there is a "count distinct" operator that counts the number of different values

Grouping and Aggregation in MapReduce

- Let R(A, B, C) be a relation to which we apply $\gamma_{A,\theta(B)}$(R)
  - The map operation prepares the grouping
  - The grouping is done by the framework
  - The reducer computes the aggregation
  - Simplifying assumptions: one grouping attribute and one aggregation function

- MapReduce implementation of $\gamma_{A,\theta(B)}$(R)

Map: For a tuple *(a,b,c)* emit a key/value pair (*a*, *b*)

Reduce: Each key a represents *a* group, with values [b_1, b_2, …, b_n]

Apply θ to the list [b_1, b_2, …, b_n]

Emit the key/value pair (*a*,*x*), where x = θ ([b_1, b_2, …, b_n])

Join

- ❑ Natural join R⋈S
  - Given two relations, compare each pair of tuples, one from each relation
  - If the tuples agree on all the attributes common to both schema → produce an output tuple that has components on each attribute
  - Otherwise produce nothing
  - Join condition can be on a subset of attributes

Join: Example

- ❑ Below, we have part of a relation called Links describing the structure of the Web
  - There are two attributes: From and To
  - A row, or tuple, of the relation is a pair of URLs, indicating the existence of a link between them
  - The number of tuples in a real dataset is in the order of billions (10^9)

From	To
url-1	url-2
url-1	url-3
url-2	url-3
…	…

- ❑ Question: find the paths of length two in the Web

Join: Example

- Informally, to satisfy the query we must:
  - find the triples of URLs in the form (u,v,w) such that there is a link from u to v and a link from v to w

- Using the join operator
  - Imagine we have two relations (with different schemas), and let's try to apply the natural join operator
  - There are two copies of Links: L1(U1, U2) and L2(U2, U3)
  - Let's compute L1$\bowtie$L2
    - For each tuple t1 of L1 and each tuple t2 of L2, see if their U2 component are the same
    - If yes, then produce a tuple in output, with the schema (U1,U2,U3)

Join in MapReduce (Reduce-side Join)

- Assume to have two relations: R(A, B) and S(B, C)
  - We must find tuples that agree on their B components
- A MapReduce implementation of Natural Join

Map: For a tuple *(a,b)* in R emit a key/value pair (*b*, ('R',a))

For a tuple *(b,c)* in S, emit a key/value pair (*b*, ('S',c))

Reduce: If key *b* has value list [('R',a),('S',c)], emit a key/value pair (*b*, *(a,b,c)*)

- NOTES
  - In general, for *n* tuples in relation R and *m* tuples in relation S all with a common B-value, then we end up with *nm* tuples in the result
  - If all tuples of both relations have the same B-value, then we're computing the *cartesian product*