Mining Big Data

Large-Scale File Systems and Map-Reduce

(Chapter 2)

Agenda

- What is Map-Reduce, and why need/want it?
- What can Map-Reduce do?
  - Simplest operation: Words Counting
  - Medium-level operation: Matrix-vector Multiplication (very common)
    - The vector fits in main memory
    - The vector can not fit in main memory
  - Relational-Algebra (relational database) operations
    - Selection, Projection, Union, Intersection, and Difference

Motivation

- Modern internet applications need to manage immense amounts of data.
- Often data is extremely regular and we can exploit parallelism.

Examples:

- Ranking of webpages by importance which involves interated matrix-vector multiplication (dimension tens of billion).
- Search in “friends” network at social-networking sites which involves graphs with hundreds of millions of nodes.

Need for a distributed file system that is fault-tolerant and allows parallel computations.

Distributed File Systems

- Large-scale Web services has lead to computing on installations with thousands of compute nodes operating more or less independently.
- This has lead to new programming systems which take advantage of the power of parallelism and tackle reliability problems cause by hardware failures.

Cluster Computing

- Compute nodes are stored on racks (8-64 per rack)
- Racks are connected by a network (usually gigabit ethernet)

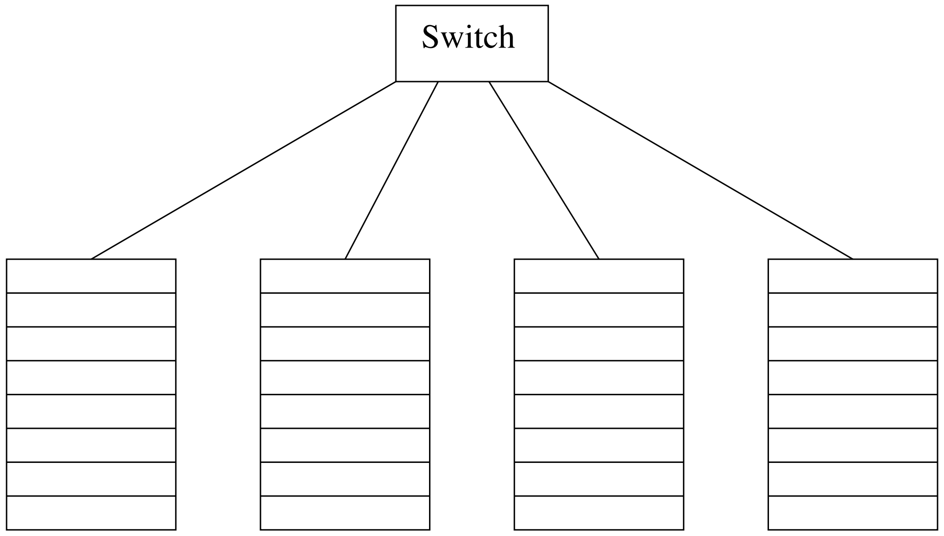


Figure 2.1 in Rajaramam/Ullman

Cluster Computing

- System with many components (compute nodes, interconnection networks) will frequently have components that fail.
- Principle failure modes:
  - loss of a single node
  - loss of an entire rack
- Important calculations might take minutes or even hours on thousands of compute nodes.
- We can not restart the computation every time a component fails.

Coping with Failures

Solution:

- Files must be stored redundantly and are available at different nodes.
- Computations must be divided into tasks such that if one task fails it can be restarted without affecting the other tasks.

Distributed File System (DFS)

Properties:

- Files can be enormous, possibly terabytes in size.
- Files are rarely updated, they are read as data and additional data is appended from time to time
- Files are divided into chunks (typically 64 megabytes in size)
- Chunks are replicated (let’s say 3 times) at different compute nodes and on different racks.
- Another small file for that file (called master node) allows to find chunks of that file. Master node is replicated as well.
- Directory for the whole file system knows where to find files (and copies).
- Directory might also be replicated (all participants of DFS know where directory copies are).

Map Reduce

- Map Reduce is a way of programming to handle large datasets in a fault tolerant way.
- Many different implementations including the one by Google and Hadoop (open source).
- You need to write two functions Map and Reduce.
- System manages then the execution including coordination of tasks and deals with failures.

Map-Reduce

Map:

- Map task is given chunks of data from DFS.
- Map task turns chunk into key-value pairs.
- Mapping is specified by user writing the Map function.

Master controller collects key-value pairs and sorts them by keys.

Reduce:

- Works on one key at a time and combines all the values associated with that key.
- The combination is determined by the function written by the user for the Reduce function

Scheme for Map-Reduce

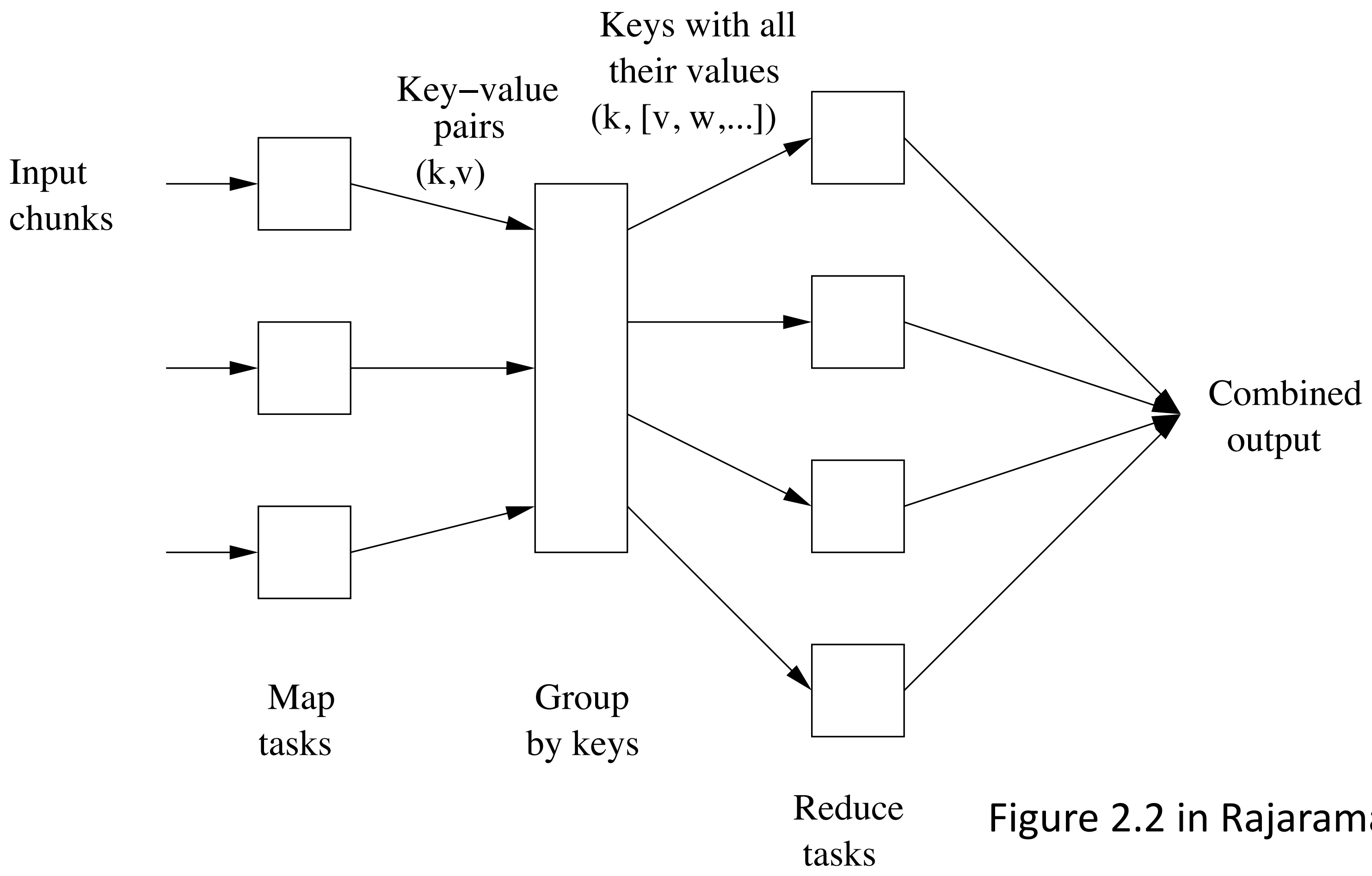


Figure 2.2 in Rajaramam/Ullman

Map Tasks

- Input files consist of elements which can be of any type.
- A chunk is a collection of elements (no element is stored across two chunks)
- All inputs of Map tasks and outputs from Reduce tasks are of key-value pair form.
- This enables composition of several map-reduce processes.
- Map function has to convert elements to key-value pairs.
- Types of keys and values can be arbitrary.
- Keys don’t have to be unique and a Map task can produce several key-value pairs with the same key.

Example: Word Count

- Input file is repository of documents
- Each document is an element
- Keys that are of type string.
- Values are integers
- Map task reads a document and breaks it into sequence of words

$$w_1, w_2, \ldots, w_n$$

- We have sequence of key-value pairs where the value is always 1
- Output of Map task for this document is:

$$(w_1, 1),\ (w_2, 1), \ldots, (w_n, 1)$$

- Map will usually process more than one document
- If a word w appears m time, then there will be m key-value pairs (w,1)
- Alternative option is to combine m pairs into a single pair (w, m).

Grouping and Aggregation

- Grouping and aggregation is independent of the implementation of Map and Reduce.
- Assume that there are r Reduce tasks. Then we have r local files.
- Master controller picks a hash function that is applied to the keys and maps to {0, …, r-1} and put in the corresponding local file.
- After Map tasks have been completed, the master controller merges the file from each Map task that belongs to a particular Reduce task i, 0 ≤ i ≤ r-1.
- Merged file is used by the Reduce task as a sequence of key-list-of-value $(k, [v_1, v_2, \ldots, v_n])$, where $(k, v_1), (k, v_2) \ldots, (k, v_n)$ are all key-value pairs with key k in all Map tasks.

Reduce Tasks

- Input of a Reduce function are pairs consisting of a key and its list of associated values.
- The output is a sequence of key-value pairs consisting of each input key k paired with the combined value constructed from the list of values for key k.
- Output from all Reduce tasks is merged into a single file.

Example: Word Count

- Reduce function adds up all values.
- Output of the Reduce tasks is a sequence of (w, m) pairs, where
  - w is a word that appears at least once among all input documents
  - m is the total number of occurrences of w in all documents

Combiners

- Reduce function is often associative and commutative.
- Values can be combined in any order leading to the same result.
- If Reduce is associative and commutative, some work of Reduce can be pushed to the Map task.
- In our example, the Reduce function could have been applied in the Map task leading to pairs (w,m).

Execution of Map-Reduce

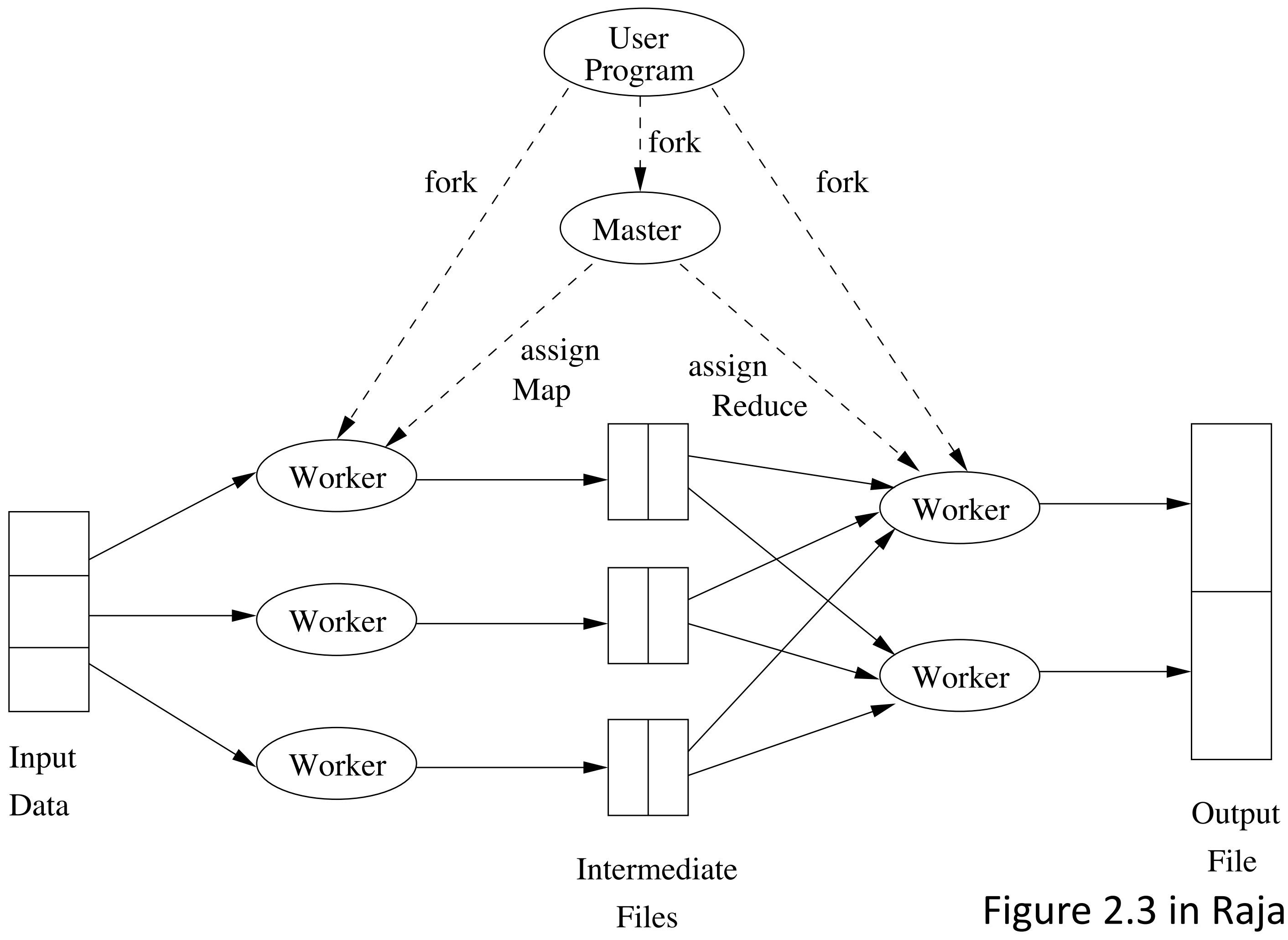


Figure 2.3 in Rajaramam/Ullman

Details

- User program forks a Master Controller to execute a number of Worker processes at different compute nodes.
- A Worker handles either Map or Reduce tasks (but not both)

Responsibilities of Master:

- Create number of Map and Reduce tasks
- Assign tasks to Worker processes
- Keeps track of each Map and Reduce task (idle, executed at Worker, or completed)
- Worker reports to Master when a task is finished, and Master schedules new task to Worker

Details

Map and Reduce tasks:

- Reasonable to create one Map task for every chunk of input file(s), but usually fewer Reduce tasks.
- Map creates for each Reduce task an intermediate file.
- Limiting the number of Reduce tasks implies limiting the number of such files.
- Each Map task is assigned one or more chunk(s) of input file(s) and executes the code written by user.
- Map task creates a file for each Reduce task on the local disk of the Worker executing the Map task.
- Master is informed on location and size of these files and the corresponding Reduce task.
- When Reduce task is assigned to Worker by Master, it will be given all the files as the input.
- Reduce task executes code written by user and writes its output to a file that is part of the distributed file system.

Coping with Node Failures

- If the compute node where the Master is located, the entire Map-Reduce job needs to be restarted (worst case that can happen)
- If a compute node where a Map job is located fails all Map tasks assigned to this Worker have to be redone as the output resides at that compute node.
- If a compute node of a Reduce job fails the Reduce job will simply be rescheduled.

Algorithms using Map-Reduce

- Entire DFS only makes sense when files are very large and updated rarely.
- Not suitable for managing online retail sales (such as Amazon) even though thousands of computing nodes are used to process requests (too frequent changes to database).
- Map-Reduce can be used to perform certain analytic queries on large amounts of data (such as finding buying patterns)

Matrix-Vector Multiplication

- Original purpose of Map-Reduce was to execute every large matrix multiplications (needed for PageRank by Google).
- Suppose we are given an n×n matrix M whose element in row i and column j is denoted by m_{ij}.
- Suppose v is a vector of length n and the jth element is denoted by v_j.
- Matrix-vector product is the vector x of length n whose ith element is given by

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

Matrix-Vector Multiplication

- DFS or Map-Reduce doesn’t make sense for small n.
- Calculation is for ranking of Web pages in search engines where n is in the tens of billions.

Matrix-Vector Multiplication

- First we assume that n is large, but v still fits into the main memory.
- Each the Matrix M and the vector v will be stored in a file of the DFS.
- We assume that the row-column coordinates of each matrix is discoverable either from its position in the file or because it's stored with explicit coordinates (i,j, m_{ij}). Same holds for v_j of the vector v.

Vector in Memory

Map Function:

- Each Map task takes the entire vector v and a chunk of the matrix M. From each matrix element m_{ij} it produces a key-value pair (i, m_{ij}, v_j).
- This means that all terms that make up component xi of the matrix-vector product get the same key.

Reduce Function:

- Sum all the values associated with a given key i. The result is a pair (i, x_i).

Vector not in Main Memory

- Assume that the vector is too large to fit into main memory.
- If we have to move pieces from disk to main memory frequently to multiply components, this will be slow.

Alternative:

- Divide matrix into vertical stripes of equal width
- Divide vector into horizontal stripes of same height
- Each stripe should fit in the main memory.
- The ith stripe of the matrix only multiplies components from the ith stripe of the vector.
- Divide matrix into one file for each stripe and do the same for the vector.

Matrix Multiplication

Illustration:

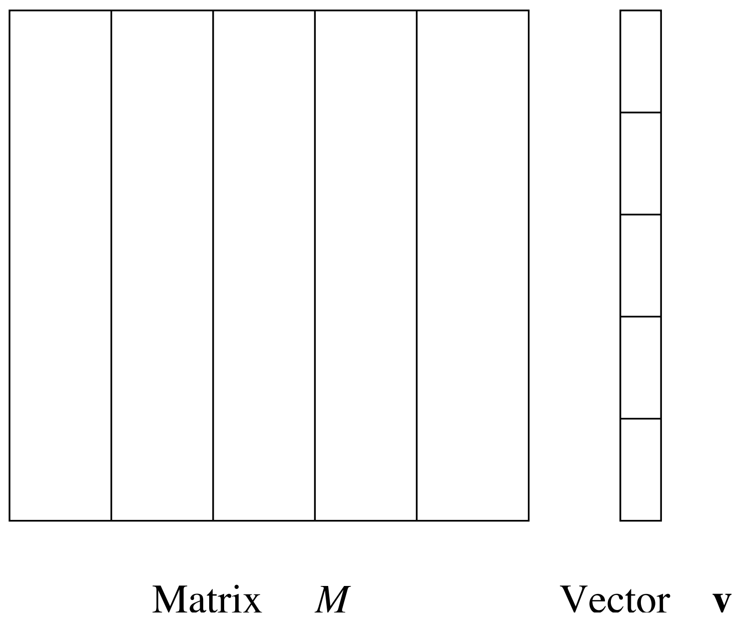


Figure 2.4 in Rajaramam/Ullman

Matrix-Vector Multiplication

Map Task:

- Each Map task is assigned a chunk from one of the stripes of the matrix and the corresponding stripe of the vector.

Map and Reduce tasks can act as for the case where Map tasks get the entire vector.

Relational-Algebra Operations

- Many operations on large scale data belong to database queries.
- Queries involve retrieval of small amounts of data, but the database may be larger (for example: ask for bank balance of a particular account)
- There are many operations on data that can be described easily in terms of common database-query primitives.
- We will now summarize such basic operations on relations which set the basis for several map-reduce applications.

Relations

- A relation is a table with column headers called attributes and rows of the relation are called tuples.
- The set of attributes of a relation is called its schema.

From	*To*
url1	url2
url1	url3
url2	url3
url2	url4
...	...

Example: Relation consisting of pairs of urls
such that the first has one or more links to the second
(Figure 2.5 in Rajaramam/Ullman)

- We write R(A_1, A_2, …, A_n) to say that the relation R has attributes A_1, A_2, …, A_n.

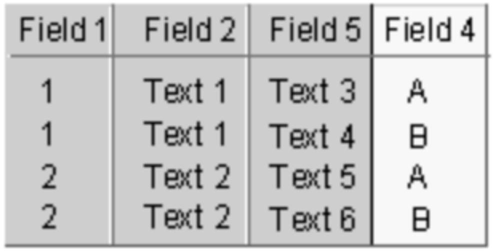
Standard Operations on Relations

There are several standard operations on relations (referred to as relational algebra):

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

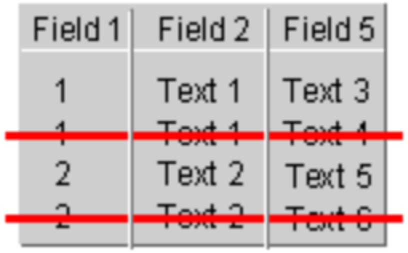
Selection

$\sigma_C(R)$: Apply a condition C to each tuple in the relation R and produce as output only those tuples that satisfy C.



Field 1	Field 2	Field 5	Field 4
1	Text 1	Text 3	A
1	Text 1	Text 4	B
2	Text 2	Text 5	A
2	Text 2	Text 6	B

Selection condition: TABB - Field 4 = 'A'.



Field 1	Field 2	Field 5
1	Text 1	Text 3
~~1~~	~~Text 1~~	~~Text 4~~
2	Text 2	Text 5
~~2~~	~~Text 2~~	~~Text 6~~

Map Function:

- For each tuple t in R, test if it satisfies C. If so, produce a key-value pair (t,t) having key and value t.

Reduce Function:

Is the identity which passes each key-value pair to the output.

Picture from https://help.sap.com/doc/saphelp_nw70/7.0.31/en-US/cf/21ec6a446011d189700000e8322d00/content.htm?no_cache=true

Projection

$\pi_S(R)$: For some subset S of attributes of the relation R, produce from each tuple only the components for the attributes in S.

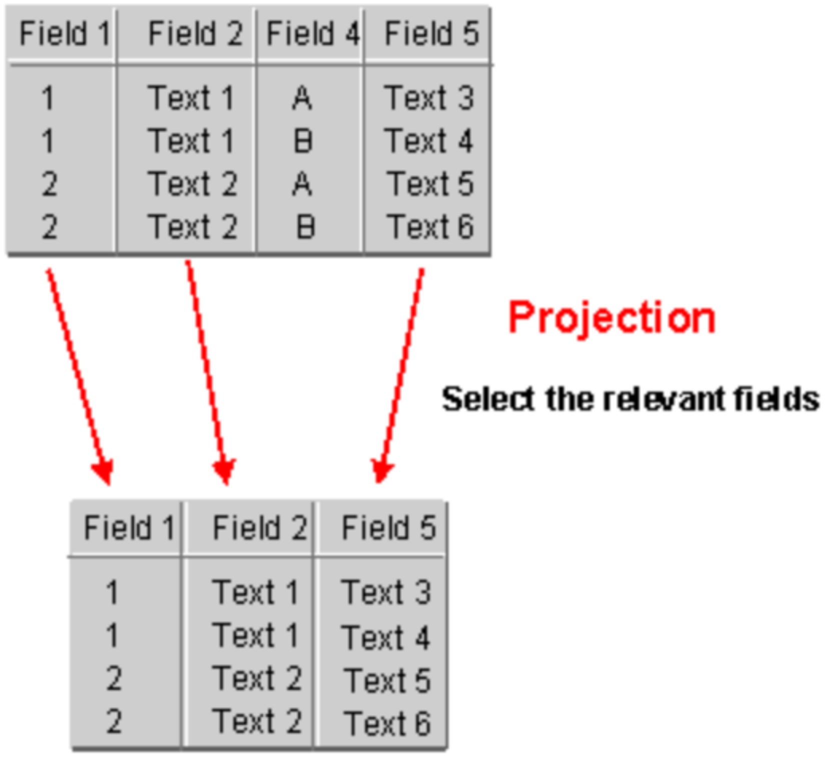
Similar to Selection, but Reduce has to remove duplicates .

Map Function:

- For each tuple t in R, construct a tuple t' by eliminating from t those components whose attributes are not in S. Output key-value pair (t',t') having key and value t'.

Reduce Function:

- Reduce function turns (t', [t',t', …, t']) into (t',t'), i.e. for each key t' produced by any of the Map tasks, we produce on key-value pair (t', t').



Field 1	Field 2	Field 4	Field 5
1	Text 1	A	Text 3
1	Text 1	B	Text 4
2	Text 2	A	Text 5
2	Text 2	B	Text 6

Field 1	Field 2	Field 5
1	Text 1	Text 3
1	Text 1	Text 4
2	Text 2	Text 5
2	Text 2	Text 6

Union, Intersection, Difference

These are the usual set operations applied to the set of tuples in two relations that have the same schema.

Consider union of two relations R and S having the same schema. Map task will be assigned chunks from either R or S.

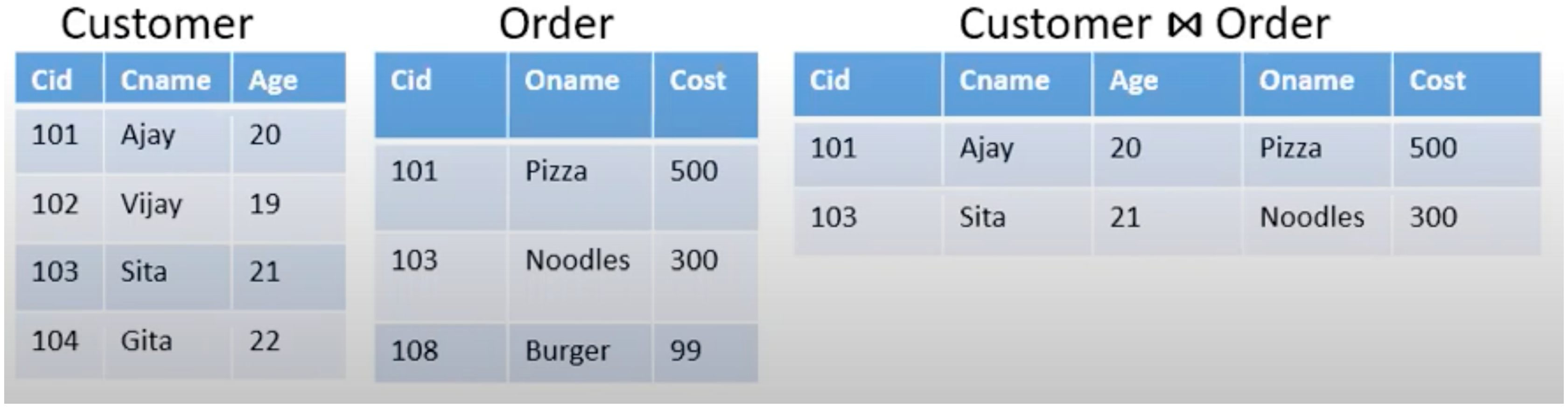
Map Function:

- Turn each input tuple t into a key-value pair (t,t)

Reduce Function:

- Associated with each key t there will be either one of two values. Produce output (t,t) in either case.

Natural Join



Customer

Cid	Cname	Age
101	Ajay	20
102	Vijay	19
103	Sita	21
104	Gita	22

Order

Cid	Oname	Cost
101	Pizza	500
103	Noodles	300
108	Burger	99

Customer ⋈ Order

Cid	Cname	Age	Oname	Cost
101	Ajay	20	Pizza	500
103	Sita	21	Noodles	300

R⋈S: Given two relations R and S, compare each pair of tuples (one from R and one from S). Take the tuples that agree on all attributes that are common to the two schemas (don't take pairs that disagree). Produce a corresponding tuple that has components for each of the attributes in either schema.

Map Function:

- For each tuple (a,b) of R, produce key-value pair (b, (R,a)). For each tuple (b,c) of S, produce key-value pair (b, (S,c))

Reduce Function:

- For each key b, the output 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 value b.

Picture from https://www.youtube.com/watch?v=jAcimWgWEmA

Example

We want to find the paths of length 2 using the relation links of Figure 2.5, i.e. the triples of URL's (u,v,w) such that there is a link from u to v and a link from v to w.

- Can be done by using the natural join.
- Assume that we got two copies of Links L1(U1, U2) and L2(U2, U3).
- If we compute $L1 \bowtie L2$, we get the result, that is for each tuple t1 of L1 and each tuple t2 of L2 we see whether the U2 component is the same.
- We might only want pairs (u,w) such that there is at least one path of length 2 from u to w.
- This can be done by computing $\pi_{U1,U3}(L1 \bowtie L2)$ (projecting out the second component)

Grouping and Aggregation

Given a relation R, partition its tuples according to their values in one set of attributes G (called grouping attributes). For each group aggregate the value. Usual aggregations are SUM, COUNT, AVG, MIN, and MAX.

We denote the grouping-and-aggregation on R by $\gamma_{X(R)}$ where X is a list of elements that are either

- A grouping attribute, or
- An expression Θ(A) where Θ is one of the aggregation operations and A an attribute not common among the grouping attributes.

Grouping and Aggregation

Consider case of one grouping attribute A and one aggregation $\Theta(B)$ and relation R(A,B,C) to which we apply $\gamma_{A,\Theta(B)}(R)$.

Map Function:

- Does grouping. For each tuple (a,b,c) produce key-value pair (a,b).

Reduce Function:

- Each key a represents a group. Apply the aggregation operator Θ to the list [b1,b2,…,bn] of B values associated with a. Output is the pair (a,x) where x is the result of applying Θ to the list. For example if Θ=SUM, then x = b1+b2+…+bn.