MapReduce

ANISHA JOSEPH

MapReduce

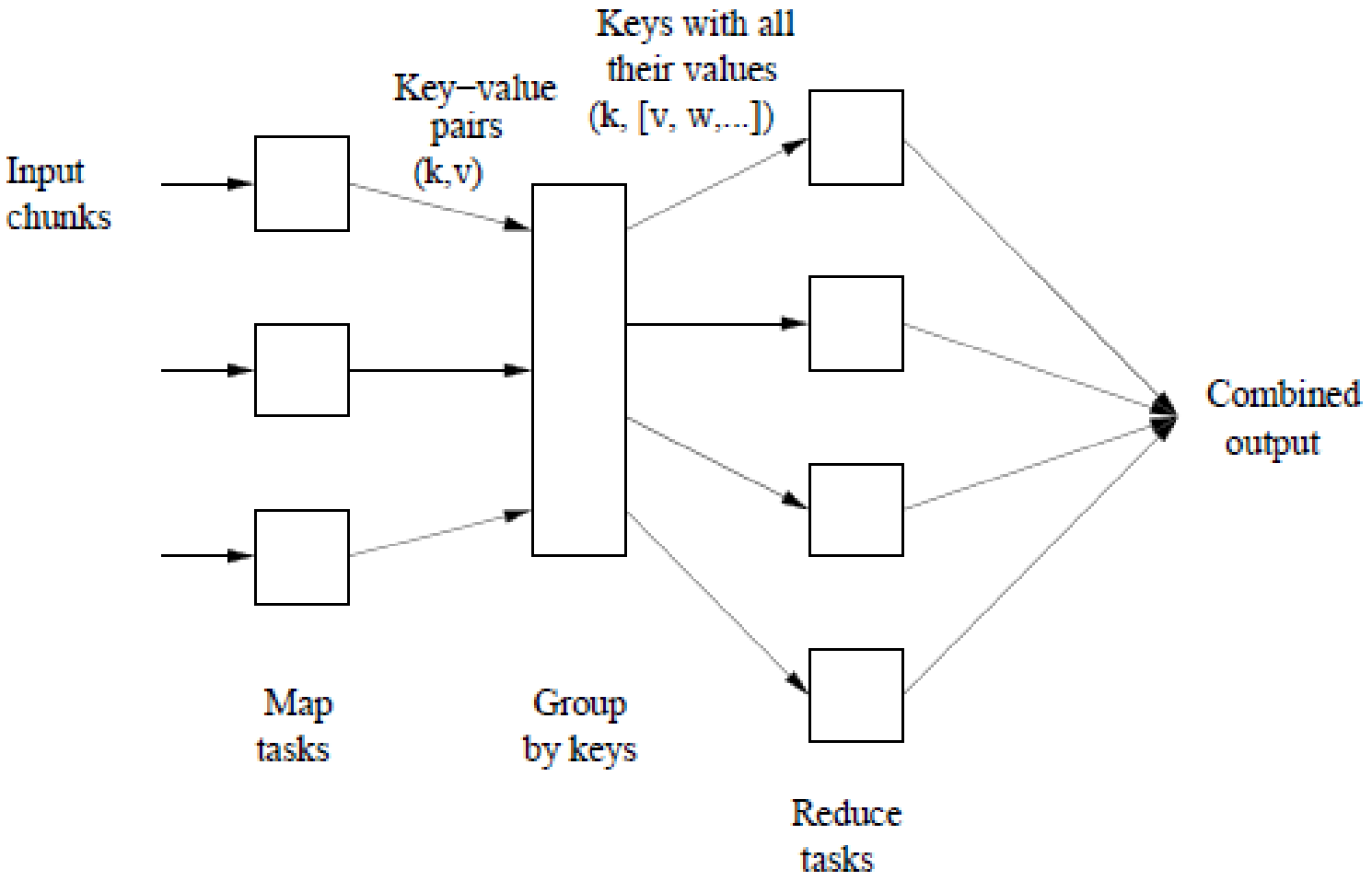
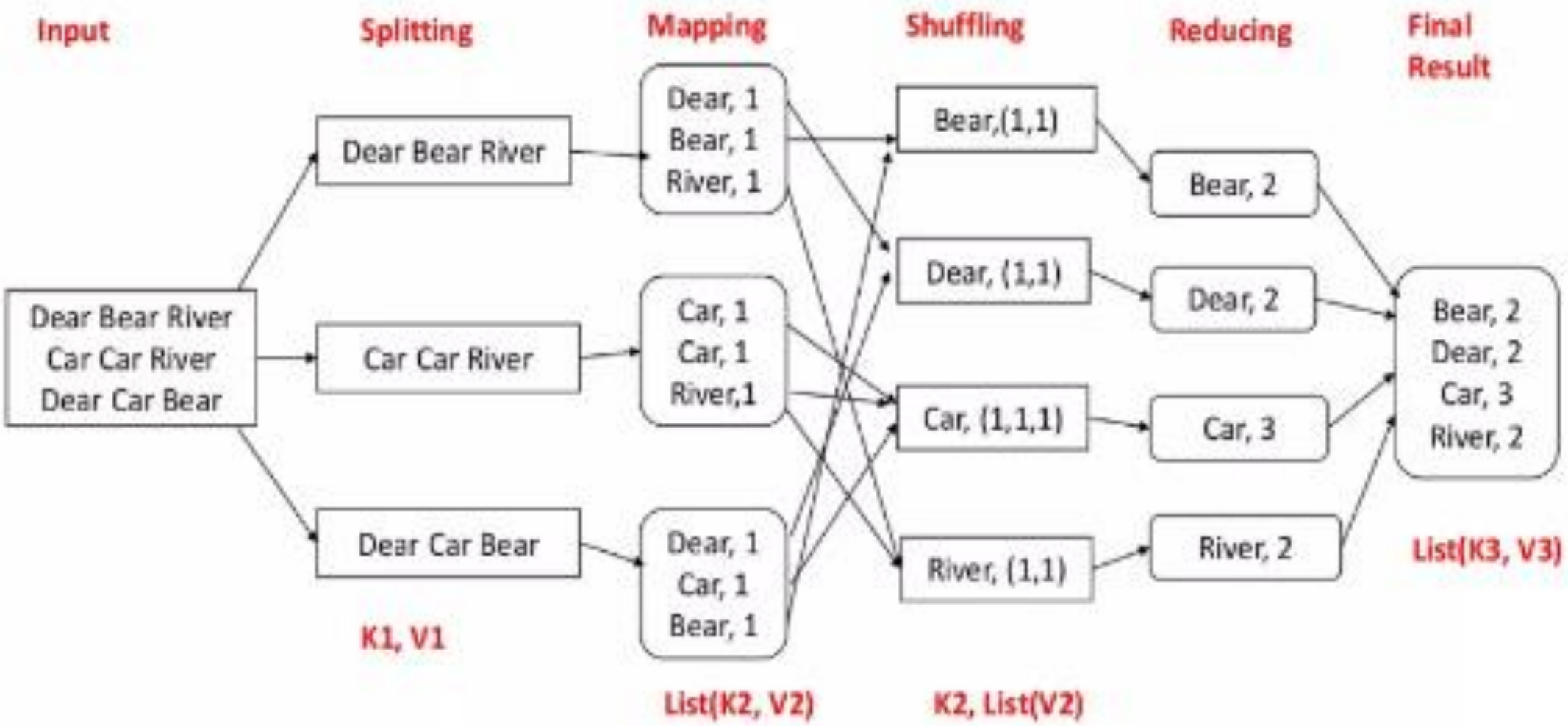


Figure 2.2: Schematic of a MapReduce computation

Map Reduce Word count program

Over all Map Reduce word count process



MapReduce

- Some number of Map tasks each are given one or more chunks from distributed file system. These Map tasks turn the chunk into a sequence of key-value pairs.
- The key-value pairs from each Map task are collected by a master controller and sorted by key.
- The Reduce tasks work on one key at a time, and combine all the values associated with that key in some way.

The Map Tasks

- Input files for a Map task as consisting of elements, which can be any type: Eg: a tuple or a document.
- A chunk is a collection of elements, and no element is stored across two chunks.
- Technically, all inputs to Map tasks and outputs from Reduce tasks are of the key-value-pair form.
- Eg: Word count task using MapReduce.

Grouping by Key

- As soon as the Map tasks have all completed successfully, the key-value pairs are grouped by key, and the values associated with each key are formed into a list of values.

- That is, for each key k, the input to the Reduce task that handles key k is a pair of the form (k, [v1, v2, . . . , vn]), where (k, v1), (k, v2), . . . , (k, vn) are all the key-value pairs with key k coming from all the Map tasks.

The Reduce Tasks

- The output of the Reduce function is a sequence of zero or more key-value pairs.
- A Reduce task receives one or more keys and their associated value lists. That is, a Reduce task executes one or more reducers.

Combiners

- These key-value pairs would thus be replaced by one pair with key w and value equal to the sum of all the 1's in all those pairs.
- That is, the pairs with key w generated by a single Map task would be replaced by a pair (w, m), where m is the number of times that w appears among the documents handled by this Map task.

MapReduce Execution

- Taking advantage of a library provided by a MapReduce system such as Hadoop, the user program forks a Master controller process and some number of Worker processes at different compute nodes.
- Normally, a Worker handles either Map tasks (a Map worker) or Reduce tasks (a Reduce worker), but not both.
- Master responsibility: One is to create some number of Map tasks and some number of Reduce tasks, these numbers being selected by the user program.
- These tasks will be assigned to Worker processes by the Master.

MapReduce Execution

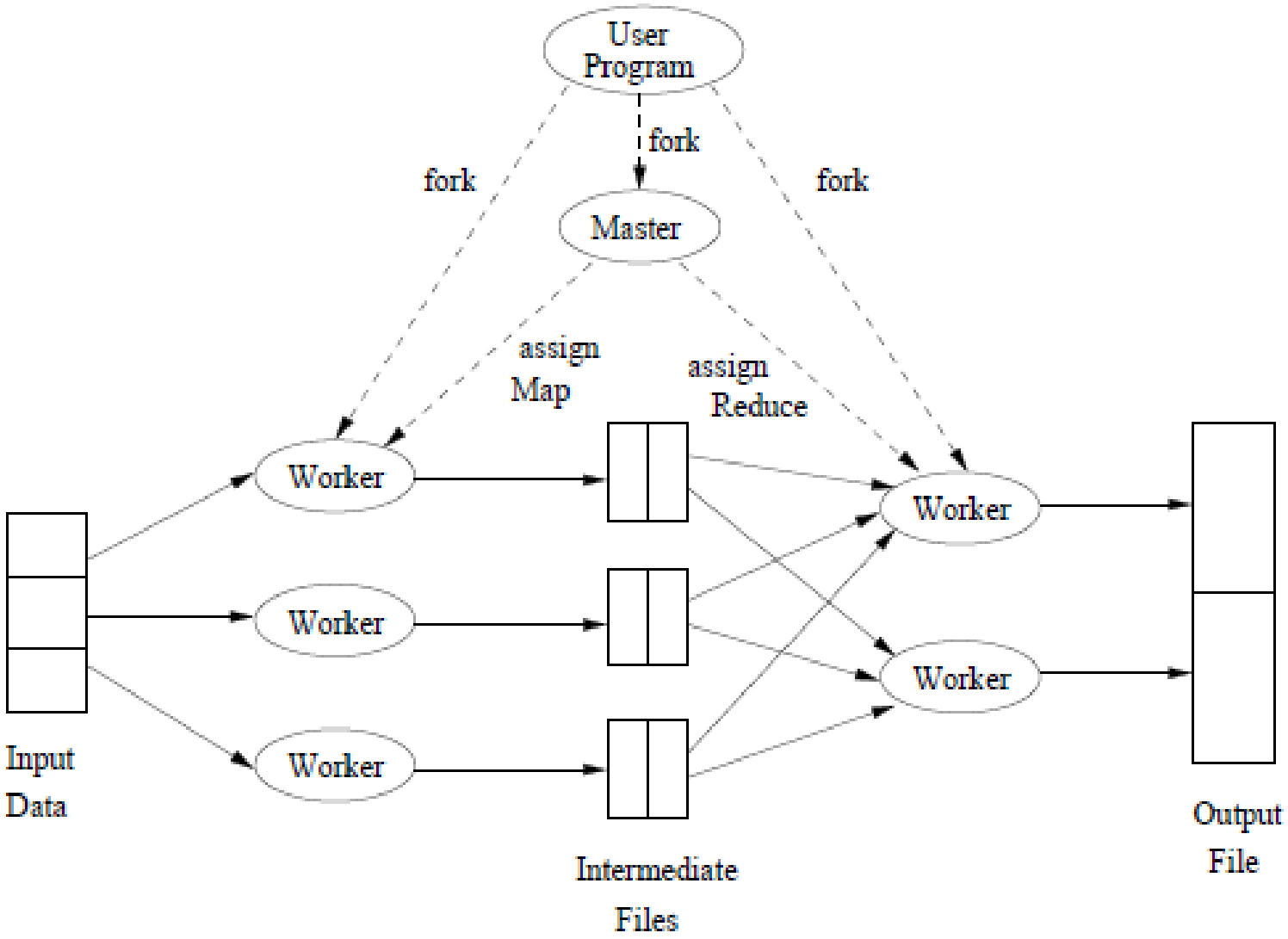


Figure 2.3: Overview of the execution of a MapReduce program

Coping With Node Failures

- The worst thing that can happen is that the compute node at which the Master is executing fails. The entire MapReduce job must be restarted.
- But only this one node can bring the entire process down; other failures will be managed by the Master, and the MapReduce job will complete eventually.
- Suppose the compute node at which a Map worker resides fails. This failure will be detected by the Master.
- All the Map tasks that were assigned to this Worker will have to be redone, even if they had completed.

Coping With Node Failures

- The Master sets the status of each of these Map tasks to idle and will schedule them on a Worker when one becomes available.
- The Master must also inform each Reduce task that the location of its input from that Map task has changed.
- Dealing with a failure at the node of a Reduce worker is simpler. The Master simply sets the status of its currently executing Reduce tasks to idle. These will be rescheduled on another reduce worker later.

Implementation

The MapReduce framework in Hadoop has native support for running Java applications. It also supports running non-Java applications in Ruby, Python, C++ and a few other programming languages, via two frameworks, namely the Streaming framework and the Pipes framework.

MapReduce Example in Java

https://www.youtube.com/watch?v=qgBu8Go1SyM

Install Hadoop: https://www.youtube.com/watch?v=Slbi-uzPtnw

Algorithms Using MapReduce

- The entire distributed-file-system milieu makes sense only when files are very large and are rarely updated in place.
- Thus, we would not expect to use either a DFS or an implementation of MapReduce for managing online retail sales, even though a large on-line retailer such as Amazon.com uses thousands of compute nodes when processing requests over the Web.
- The reason is that the principal operations on Amazon data involve responding to searches for products, recording sales, and so on, processes that involve relatively little calculation and that change the database.

Algorithms Using MapReduce

- Amazon might use MapReduce to perform certain analytic queries on large amounts of data, such as finding for each user those users whose buying patterns were most similar.
- Matrix-vector and matrix-matrix calculations fit nicely into the MapReduce style of computing.
- Another important class of operations that can use MapReduce effectively are the relational-algebra operations.

Matrix-Vector Multiplication by MapReduce

we have an n×n matrix M, whose element in row i and column j will be denoted mij . Suppose we also have a vector v of length n, whose jth element is vj .

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

If n = 100, we do not want to use a DFS or MapReduce for this calculation.

Matrix-Vector Multiplication by MapReduce

n is large, but vector v can fit in main memory and thus be available to every Map task.

The matrix M and the vector v each will be stored in a file of the DFS.
We assume that the row-column coordinates of each matrix element will be discoverable, either from its position in the file, or because it is stored with explicit coordinates, as a triple (i, j,mij).

Matrix-Vector Multiplication by MapReduce

The Map Function:

Each Map task will operate on a chunk of the matrix M. From each matrix element mij it produces the key-value pair (i,mijvj). Thus, all terms of the sum that make up the component xi of the matrix-vector product will get the same key, i.

The Reduce Function: The Reduce function simply sums all the values associated with a given key i. The result will be a pair (i, xi).

If the Vector v Cannot Fit in Main Memory

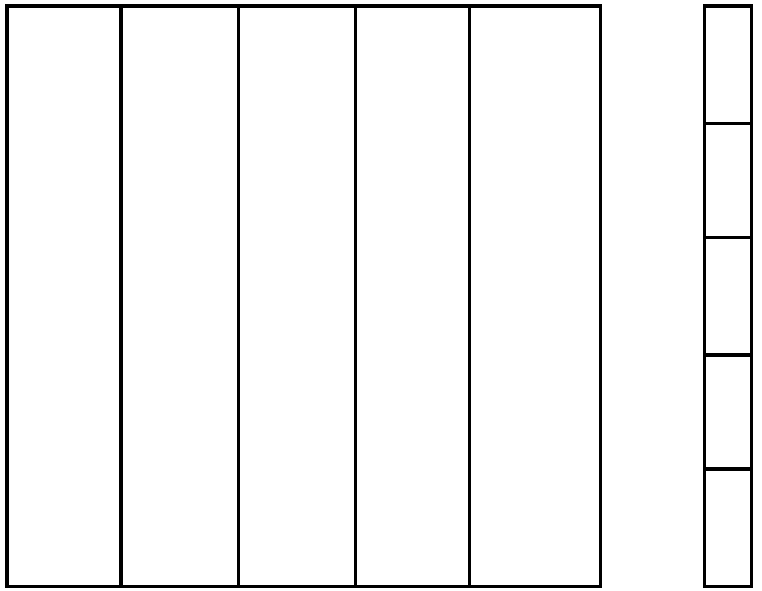


Figure 2.4: Division of a matrix and vector into five stripes

If the Vector v Cannot Fit in Main Memory

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.

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

The Map and Reduce tasks can then act exactly as was described above for the case where Map tasks get the entire vector

Relational-Algebra Operations

Many traditional database applications involve retrieval of small amounts of data, even though the database itself may be large.

For example, a query may ask for the bank balance of one particular account. Such queries are not useful applications of MapReduce.

Relational-Algebra Operations

Eg.1: Relation Links consists of the set of pairs of URL's, such that the first has one or more links to the second.

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

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

Relational-Algebra Operations

- To find 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.
- There are two copies of Links, namely L1(U1,U2) and L2(U2,U3). Now, if we compute L1 ⋈ L2.
- For each tuple t1 of L1 (i.e., each tuple of Links) and each tuple t2 of L2 (another tuple of Links, possibly even the same tuple), see if their U2 components are the same.
- If these two components agree, then produce a tuple for the result, with schema (U1,U2,U3).

Relational-Algebra Operations

We may not want the entire path of length two, but only want the pairs (u, w) of URL's such that there is at least one path from u to w of length two. If so, we can project out the middle components by computing $\pi_{U1,U3}(L1 \bowtie L2)$.

Eg.2. Imagine that a social-networking site has a relation.

Friends(User, Friend)

This relation has tuples that are pairs (a, b) such that b is a friend of a. The site might want to develop statistics about the number of friends members have.

Relational-Algebra Operations

- Their first step would be to compute a count of the number of friends of each user.
- We denote a grouping-and-aggregation operation on a relation R by γX(R).

$$\gamma_{User,\ COUNT(Friend)}(Friends)$$

- This operation groups all the tuples by the value in their first component, so there is one group for each user.
- The result will be one tuple for each group, and a typical tuple would look like (Sally, 300), if user “Sally” has 300 friends.

Computing Selections by MapReduce

- Selections really do not need the full power of MapReduce.
- They can be done most conveniently in the map portion alone, although they could also be done in the reduce portion alone.

MapReduce implementation of selection σC(R)

The Map Function: For each tuple *t* in *R*, test if it satisfies *C*. If so, produce the key-value pair *(t, t)*. That is, both the key and value are *t*.

The Reduce Function: The Reduce function is the identity. It simply passes each key-value pair to the output.

Computing Selections by MapReduce

Note that the output is not exactly a relation, because it has key value pairs.

However, a relation can be obtained by using only the value components (or only the key components) of the output.

Computing Projections by MapReduce

Projection is performed similarly to selection, because projection may cause the same tuple to appear several times, the Reduce function must eliminate duplicates.

We may compute $\pi_S(R)$ as follows.

The 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 the key-value pair (t′, t′).

The Reduce Function: 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′, . . . , t′]) into (t′, t′), so it produces exactly one pair (t′, t′) for this key t′.

Union

Suppose relations R and S have the same schema. Map tasks will be assigned chunks from either R or S; it doesn't matter which.

The Map tasks don't really do anything except pass their input tuples as key-value pairs to the Reduce tasks.

The latter need only eliminate duplicates as for projection.

The Map Function: Turn each input tuple t into a key-value pair (t, t).

The Reduce Function: Associated with each key t there will be either one or two values. Produce output (t, t) in either case.

Intersection,

- However, the Reduce function must produce a tuple only if both relations have the tuple.
- If the key *t* has a list of two values [t, t] associated with it, then the Reduce task for *t* should produce (t, t).
- However, if the value-list associated with key t is just [t], then one of R and S is missing t, so we don't want to produce a tuple for the intersection.

Intersection,

The Map Function: Turn each tuple t into a key-value pair (t, t).

The Reduce Function: If key t has value list [t, t], then produce (t, t). Otherwise, produce nothing.

Difference

(R − S).The only way a tuple t can appear in the output is if it is in R but not in S.

The Map Function: For a tuple t in R, produce key-value pair (t, R), and for a tuple t in S, produce key-value pair (t, S). Note that the intent is that the value is the name of R or S (or better, a single bit indicating whether the relation is R or S), not the entire relation.

The Reduce Function: For each key t, if the associated value list is [R], then produce (t, t). Otherwise, produce nothing.

Computing Natural Join by MapReduce

The Map Function: For each tuple (a, b) of R, produce the key-value pair (b, (R, a)). For each tuple (b, c) of S, produce the key-value pair (b, (S, c)).

The Reduce Function: Each key value b will be associated with a list of pairs that are either of the form (R, a) or (S, c).

Construct all pairs consisting of one with first component R and the other with first component S, say (R, a) and (S, c). The output from this key and value list is a sequence of key-value pairs.

The key is irrelevant. Each value is one of the triples (a, b, c) such that (R, a) and (S, c) are on the input list of values.

Grouping and Aggregation by MapReduce

Let R(A,B,C) be a relation to which we apply the operator $\gamma_{A,\theta(B)}(R)$. Map will perform the grouping, while Reduce does the aggregation.

The Map Function: For each tuple (a, b, c) produce the key-value pair (a, b).

The Reduce Function: Each key a represents a group. Apply the aggregation operator θ to the list [b1, b2, . . . , bn] 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 $x = b1 + b2 + \cdots + bn$, and if θ is MAX, then x is the largest of b1, b2, . . . , bn.

Matrix Multiplication

- If M is a matrix with element mij in row i and column j, and N is a matrix with element njk in row j and column k, then the product P = MN is the matrix P with element pik in row i and column k, where,

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

- The product MN is almost a natural join followed by grouping and aggregation.
- Four-component tuple (i, j, k, v × w), because that represents the product mijnjk.
- Once we have this relation as the result of one MapReduce operation, we can perform grouping and aggregation, with I and K as the grouping attributes and the sum of V ×W as the aggregation.

Matrix Multiplication

We can implement matrix multiplication as the c**ascade of two MapReduce operations**, as follows. **First:**

The Map Function: For each matrix element mij , produce the key value pair (j, (M, i,mij)). Likewise, for each matrix element njk, produce the key value pair (j,(N, k, njk)).

The Reduce Function: For each key j, examine its list of associated values.

For each value that comes from M, say (M, i,mij), and each value that comes from N, say (N, k, njk), produce a key-value pair with key equal to (i, k) and value equal to the product of these elements, mijnjk.

Matrix Multiplication

Second: We perform a grouping and aggregation by another MapReduce operation.

The Map Function: This function is just the identity. That is, for every input element with key (i, k) and value v, produce exactly this key-value pair.

The Reduce Function: 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 MapReduce Step

The Map Function: For each element mij of M, produce all the key-value pairs ((i, k), (M, j,mij)) for k = 1, 2, . . ., up to the number of columns of N.

Similarly, for each element njk of N, produce all the key-value pairs ((i, k), (N, j, njk)) for i = 1, 2, . . ., up to the number of rows of M.

As before, M and N are really bits to tell which of the two matrices a value comes from.

Matrix Multiplication with One MapReduce Step

- **The Reduce Function:** Each key (i, k) will have an associated list with all the values (M, j,mij) and (N, j, njk), for all possible values of j.
- The Reduce function needs to connect the two values on the list that have the same value of j, for each j.
- An easy way to do this step is to 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, mij and njk extracted and multiplied.
- Then, these products are summed and the result is paired with (i, k) in the output of the Reduce function.

Exercise 2.3.1: Design MapReduce algorithms to take a very large file of integers and produce as output:

(a) The largest integer.

(b) The average of all the integers.

(c) The same set of integers, but with each integer appearing only once.

(d) The count of the number of distinct integers in the input.

Complexity Theory for MapReduce

For many problems there is a spectrum of MapReduce algorithms requiring different amounts of communication.

Moreover, the less communication an algorithm uses, the worse it may be in other respects, including wall-clock time and the amount of main memory it requires.

Reducer Size and Replication Rate

- **reducer size:**
- The first is the **reducer size**, which we denote by q.
- This parameter is the upper bound on the number of values that are allowed to appear in the list associated with a single key.
- Reducer size can be selected with at least two goals in mind.
  - By making the reducer size small, we can force there to be many reducers. Then there will be a high degree of parallelism, and we can look forward to a low wall-clock time.
  - We can choose a reducer size sufficiently small that we are certain the computation associated with a single reducer can be executed entirely in the main memory of the compute node where its Reduce task is located. Running time will be greatly reduced if we can avoid having to move data repeatedly between main memory and disk.

replication rate

- The second parameter is the replication rate, denoted r.
- r to be the number of **key-value pairs produced by all the Map tasks** on all the inputs, divided by the number of inputs.
- That is, the replication rate is the average communication from Map tasks to Reduce tasks (measured by counting key-value pairs) per input.

Eg:one-pass matrix-multiplication algorithm

Suppose that all the matrices involved are n × n matrices. Then the replication rate **r is equal to n.**

That fact is easy to see, since for each element mij , there are n key-value pairs produced; these have all keys of the form (i, k), for $1 \leq k \leq n$.

Likewise, for each element of the other matrix, say njk, we produce n key-value pairs, each having one of the keys (i, k), for $1 \leq i \leq n$.

one-pass matrix-multiplication algorithm

- **q, the required reducer size, is 2n.**
- That is, for each key (i, k), there are n key-value pairs representing elements mij of the first matrix and another n key-value pairs derived from the elements njk of the second matrix.
- More generally, there is a tradeoff between r and q, that can be expressed as $qr \geq 2n^2$.

An Example: Similarity Joins

Are given a large set of elements X and a similarity measure s(x, y) that tells how similar two elements x and y of set X are.

We assume that s is symmetric, so s(x, y) = s(y, x), but we assume nothing else about s. The output of the algorithm is those pairs whose similarity exceeds a given threshold t.

Similarity Joins

The total number of bytes communicated from Map tasks to Reduce tasks is 1,000,000 (for the pictures) times 999,999 (for the replication), times 1,000,000 (for the size of each picture). That's 1018 bytes, or one exabyte.

That's 10 ^18 bytes, or one exabyte. To communicate this amount of data over gigabit Ethernet would take 10^10 seconds, or about 300 years.

Similarity Joins

- Alas, this algorithm will fail completely. The reducer size is small, since no list has more than two values, or a total of 2MB of input.
- Although we don't know exactly how the similarity functions operates, we can reasonably expect that it will not require more than the available main memory.
- However, the replication rate is 999,999, since for each picture we generate that number of key-value pairs, one for each of the other pictures in the dataset.

Similarity Joins

The Map Function: Take an input element (i, Pi) and generate g − 1 key value pairs.

For each, the key is one of the sets {u, v}, where u is the group to which picture i belongs, and v is one of the other groups.

The associated value is the pair (i, Pi).

Similarity Joins

The Reduce Function: Consider the key {u, v}. The associated value list will have the 2 × 10^6/g elements (j, Pj), where j belongs to either group u or group v.

The Reduce function takes each (i, Pi) and (j, Pj) on this list, where i and j belong to different groups, and applies the similarity function s(Pi, Pj).

Similarity Joins

replication rate and reducer size as a function of the number of groups g.

Each input element is turned into g − 1 key-value pairs. That is, the replication rate is g − 1, or approximately r = g, since we suppose that the number of groups is still fairly large.

The reducer size is 2×10^6/g, since that is the number of values on the list for each reducer.

Each value is about a megabyte, so the number of bytes needed to store the input is 2 × 10^12/g.

A Graph Model for MapReduce Problems

- Study of a technique that will enable us to prove lower bounds on the replication rate, as a function of reducer size for a number of problems.
- First step is to introduce a graph model of problems.
- For each problem solvable by a MapReduce algorithm there is:
  - 1. A set of inputs.
  - 2. A set of outputs.
  - 3. A many-many relationship between the inputs and outputs, which describes which inputs are necessary to produce which outputs.

Example: similarity-join problem

Graph for the similarity-join problem four pictures rather than a million.

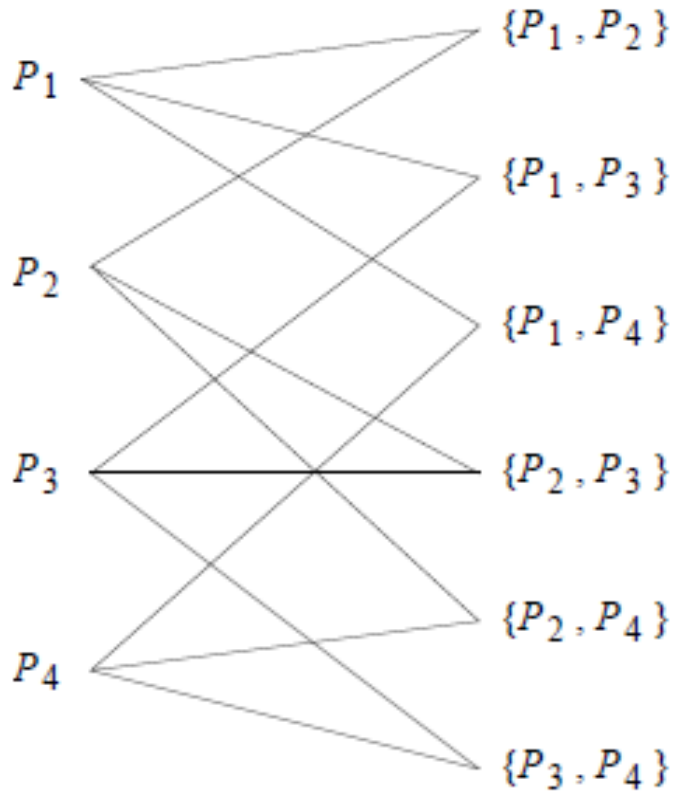


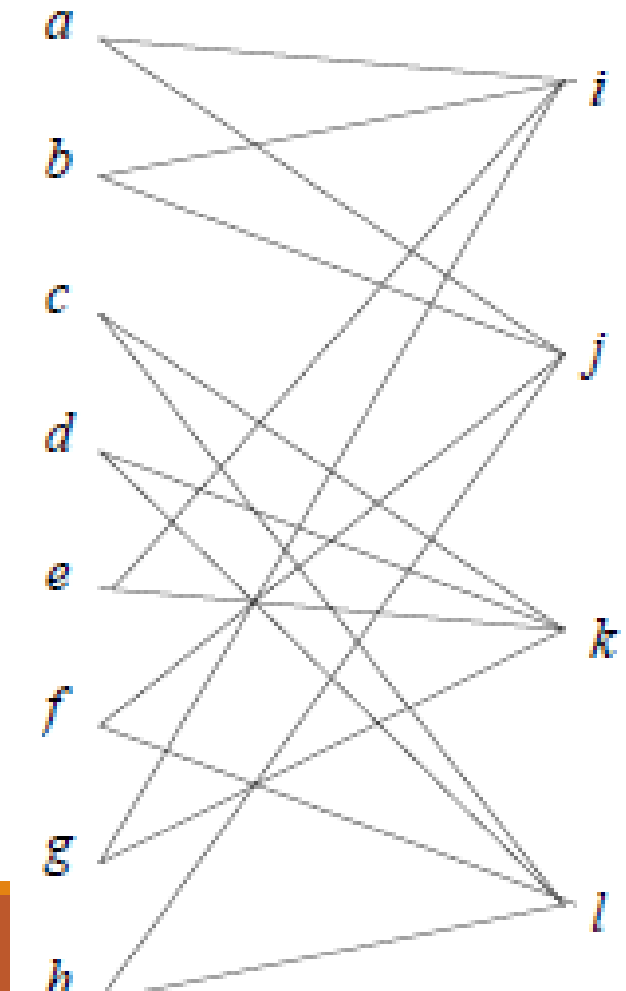
Figure 2.9: Input-output relationship for a similarity join

Example: Matrix multiplication

- If we multiply n × n matrices M and N to get matrix P, then there are 2n^2 inputs, mij and njk, and there are n^2 outputs pik.
- Each output pik is related to 2n inputs: mi1,mi2, . . . ,min and n1k, n2k, . . . , nnk.
- Moreover, each input is related to n outputs.
- For example, mij is related to pi1, pi2, . . . , pin.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} i & j \\ k & l \end{bmatrix}$$

Input-output relationship for matrix multiplication



Mapping Schemas

- Each such algorithm must have a mapping schema, which expresses how outputs are produced by the various reducers used by the algorithm.
- That is, a mapping schema for a given problem with a given reducer

Size q is an assignment of inputs to one or more reducers, such that:

- 1. No reducer is assigned more than q inputs.
- 2. For every output of the problem, there is at least one reducer that is assigned all the inputs that are related to that output. We say this reducer covers the output.

Example: Similarity Join

- Reconsider the “grouping” strategy.
- To generalize the problem, suppose the input is p pictures, which we place in g equal-sized groups of p/g inputs each.
- The number of outputs is pC2, or approximately p^2/2 outputs.
- A reducer will get the inputs from two groups – that is 2p/g inputs. so the reducer size we need is q = 2p/g.
- Each picture is sent to the reducers corresponding to the pairs consisting of its group and any of the g − 1 other groups. Thus, the replication rate is g − 1, or approximately g.

Example: Similarity Join

- If we replace g by the replication rate r in q = 2p/g, we conclude that r = 2p/q.
- That is, the replication rate is inversely proportional to the reducer size.
- This family of algorithms is described by a family of mapping schemas, one for each possible q.
- In the mapping schema for q = 2p/g, there are gC2, or approximately g^2/2 reducers.

Example: Similarity Join

- Each reducer corresponds to a pair of groups, and an input P is assigned to all the reducers whose pair includes the group of P.
- Thus, no reducer is assigned more than 2p/g inputs.
- Every output is covered by some reducer. Specifically, if the output is a pair from two different groups u and v, then this output is covered by the reducer for the pair of groups {u, v}.