Data Mining:

① Discovering patterns, insights and knowledge from data.

② Process large amount of data quickly.

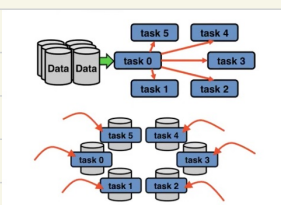
→ The data to be processed is very large and hence stored onto different locations.

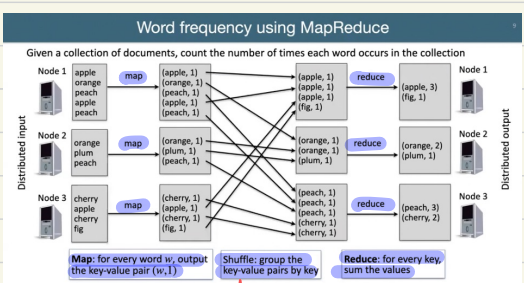
Traditional Parallelism

- Bring data to the computer

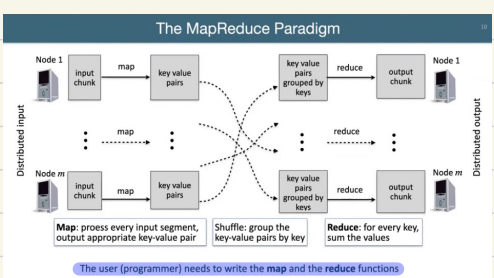
MapReduce

- Compute where data is present





GroupBy will be done by the system (No need to code)



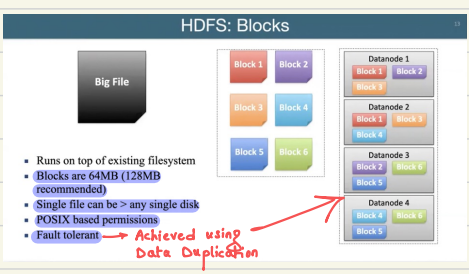
Note: HDFS runs on top of Operating System.
eg.: Implementing HDFS on Linux.

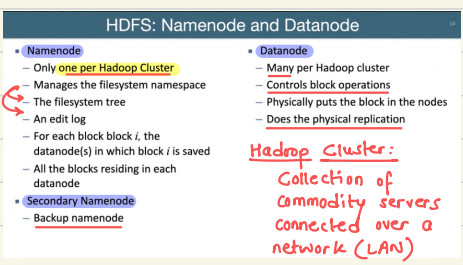
- **Hadoop:** Open Source MapReduce platform
  2 components:
  a) HDFS : Hadoop Distributed File System (used to store data)
  b) MapReduce engine : (to process data)
- It uses **Master - Slave** architecture using commodity servers.

HDFS
Master: NameNode (Metadata)
Slave : DataNode (Data)

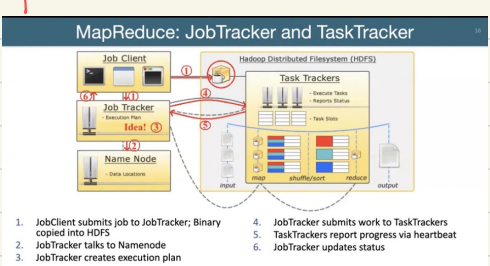
MapReduce
Master: JobTracker
Slave : TaskTracker

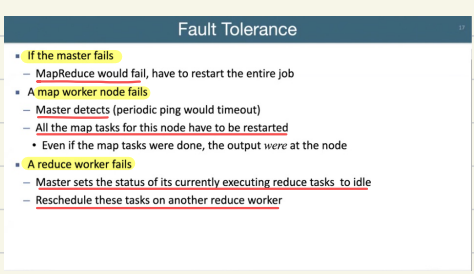
- Hadoop Distributed File System: (HDFS)





- MapReduce :





Example : Matrix – Vector Multiplication

Understanding : M is a matrix with dimensions n x n

v is a vector with n elements

∴ M x v = n x 1 vector

⇒ $Mv = \sum_{j=1}^{n} m_{ij} v_j$

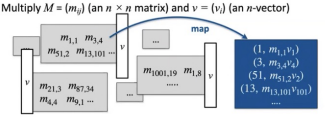
understand that n is very large (n>>1000)

↳ Hence cannot be stored on disk

↳ ∴ it is stored into parts onto diff. locations.

Matrix – Vector Multiplication

- Multiply $M = (m_{ij})$ (an $n \times n$ matrix) and $v = (v_i)$ (an n-vector)



$Mv = (x_i)$

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

Case 1: Large n, M does not fit into main memory, but v does

- Since v fits into main memory, all of v can be copied to every map worker
- ***Map***: for each matrix element m_{ij}, emit key value pair $(i, m_{ij} v_j)$
- *Shuffle*: groups all $m_{ij} v_j$ values together for the same i
- ***Reduce***: sum $m_{ij} v_j$ for all j for the same i

Need all $m_{ij} v_j$ values together for the same i

So, i needs to be the **key**

Case 1

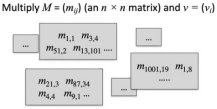
understand that n is very large (n>>1000)

↳ Hence cannot be stored on disk

↳ ∴ it is stored into parts onto diff. locations.

Matrix – Vector Multiplication

- Multiply $M = (m_{ij})$ (an $n \times n$ matrix) and $v = (v_i)$ (an n-vector)



$Mv = (x_i)$

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

Case 2: Very large n, even v does not fit into main memory of the nodes

- For every map, many accesses to disk (for parts of v) required!
- Solution:
  - Partition v so that each partition of v fits into memory
  - Take dot product of one partition of v and the corresponding partition of M
  - *Map* and *reduce* same as before, but several times to complete the computation

Case 2

Selection using MapReduce

- SQL: select * from R where C
- Trivial using MapReduce
- Map: For each tuple t in R, test if t satisfies condition C. If so, produce the key-value pair $(t, 1)$.
- Reduce: The identity function. It simply passes each key t to the output.

Selection is like "filtering"

MapReduce — Debapriyo Majumdar

Union using MapReduce

- Union of two relations R and S
- Suppose R and S have the same schema
- Map tasks are generated from chunks of both R and S
- Map: For each tuple t, produce the key-value pair $(t, 1)$
- Reduce: Only need to remove duplicates
  - For all key t, there would be either one (if the tuple is present in one relation) or two values (if the tuple is present in both)
  - Output t in either case

R

$Attr_1$	$Attr_2$	$Attr_3$	$Attr_4$
xyz	abc	1	true
abc	xyz	1	true
xyz	def	1	false
bcd	def	2	true

S

$Attr_1$	$Attr_2$	$Attr_3$	$Attr_4$
xyz	abc	1	true
abc	xyw	1	false
xyz	def	1	false
bcd	def	2	true

Reducing is simply emitting keys (after groupby)

MapReduce — Debapriyo Majumdar

Natural join using MapReduce

- Join $R(A, B)$ with $S(B, C)$ on attribute B
- Map:
  - For each tuple $t = (a, b)$ of R, emit key value pair $(b, (R, a))$
  - For each tuple $t = (b, c)$ of S, emit key value pair $(b, (S, c))$
- Reduce:
  - Each key b would be associated with a list of values that are 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

R

A	B
x	m
y	n
z	p
w	q

S

B	C
m	1
p	3
q	4
r	7

$R \bowtie S$

A	B	C
x	m	1
z	p	3
w	q	4

Grouping and Aggregation using MapReduce

- Group and aggregate on a relation $R(A, B)$ using aggregation function $\gamma(B)$, group by A
- Map:
  - Grouping comes for free with MapReduce, so the attribute to "group by" needs to be the key
  - Map is essentially the identity function
  - For each tuple $t = (a, b)$ of R, emit key value pair (a, b)
- Reduce:
  - Reduce does the aggregation part
  - For the group $\{(a, b_1), \ldots, (a, b_m)\}$ represented by a key a, apply γ to obtain $b_a = \gamma(b_1, \cdots, b_m)$

??

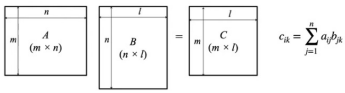
R

A	B
x	2
y	1
z	4
z	1
x	5

```
select A, sum(B)
from R group by A;
```

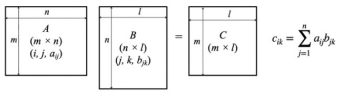
A	$sum(B)$
x	7
y	1
z	5

Matrix multiplication using MapReduce



- Think of a matrix as a relation with three attributes
- For example matrix A is represented by the relation $A(I, J, V)$
  - For every non-zero entry (i, j, a_{ij}), the row number is the value of I, column number is the value of J, the entry is the value in V
  - Also advantage: usually most large matrices would be sparse, the relation would have less number of entries
- The product is *analogous to* a natural join followed by a grouping with aggregation

Matrix multiplication using MapReduce



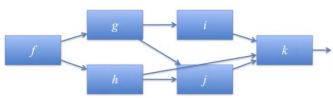
- First MapReduce process has produced key value pairs $((i, k), (a_{ij} b_{jk}))$
- Another MapReduce process to group and aggregate
  - Map: identity, just emit the key value pair $((i, k), (a_{ij} b_{jk}))$
  - Reduce:
    for each key (i, k)
    produce the sum of the all the values for the key:

Combiners

- If the aggregation operator in the reduce algorithm is associative and commutative
  - Associative: $(a * b) * c = a * (b * c)$
  - Commutative: $(a * b) = (b * a)$
  - It does not matter in which order we aggregate
- We can do some work for reduce in the map step
- Example: the word count problem
  - Instead of emitting $(w, 1)$ for each word
  - We can emit (w, n) for each document or each map task where n is the number of times the word w appears in the whole input chunk

Extensions of MapReduce

- MapReduce: two-step workflow
- Extension: any collection of functions: acyclic graph representing the workflow



- Each function of the workflow can be executed by many tasks
- A master controller divides the work among the tasks
- Each task's output goes to the successor as input
- Two experimental systems: Clustera (University of Wisconsin), Hyracks (University of California at Irvine)
- Similarly, recursive use of MapReduce