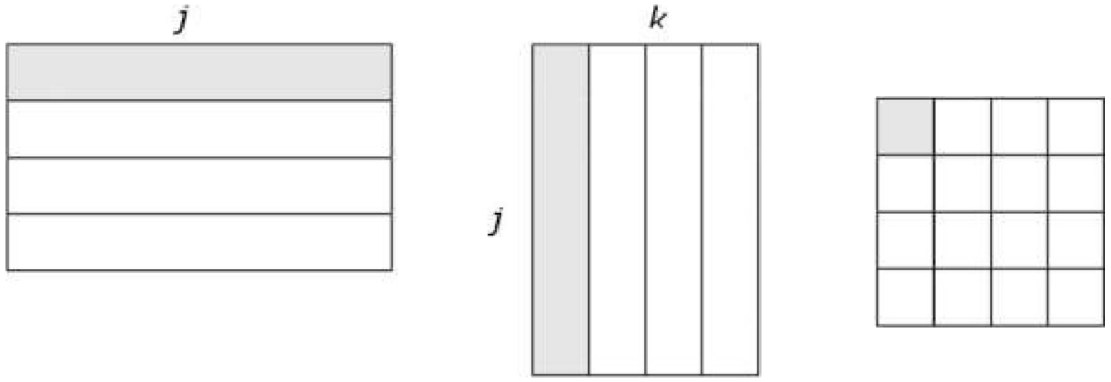
Experiment No.5
Implement simple sorting algorithm in Map reduce Matrix Multiplications.
Date of Performance:
Date of Submission:

Aim: Implement simple sorting algorithm in Map reduce- Matrix Multiplications.

Theory:
Matrix-vector and matrix-matrix calculations fit nicely into the MapReduce style of computing. In this post I will only examine matrix-matrix calculation as described in [l, ch.2]. Suppose we have a pxq matrix M, whose element in row i and column j will be denoted and a qxr matrix N whose element in row j and column k is donated by then the product P = MN will be pxr matrix P whose element in row i and column k will be donated by Pew, where P(it k) * n_{jk}.



Matrix Data Model for MapReduce

We represent matrix M as a relation with tuples and matrix N as a relation $N(J, K, W)$, with tuples . Most matrices are sparse so large amount of cells have value zero. When we represent matrices in this form, we do not need to keep entries for the cells that have values of zero to save large amount of disk space. As input data files, we store matrix M and N on HDFS in following format:

.11, it j. mo
M,0,0,10.0
M,0,2,9.0
M,0,3,9.0
M,1,0,1.0
M,1,1,3.0
M,1,2,18.0
M,1,3,25.2
....
N, j, k, n_{jk}
N,0,0,1.0
N,0,2,3.0
N,0,4,2.0
N,1,0,2.0
N,3,2,-1.0
N,3,6,4.0

We will write Map and Reduce functions to process input files. Map function will produce key, value pairs from the input data as it is described in Algorithm l. Reduce function uses the output of the Map function and performs the calculations and produces key value pairs as described in Algorithm 2. All outputs are written to HDFS.

Algorithm 1: The Map Function

```
1 for each element $m_{ij}$ of $M$ do
2     produce $(key, value)$ pairs as $((i,k),(M,j,m_{ij}))$, for $k = 1,2,3,..$ up
      to the number of columns of $N$
3 for each element $n_{jk}$ of $N$ do
4     produce $(key, value)$ pairs as $((i,k),(N,j,n_{jk}))$, for $i = 1,2,3,...$ up
      to the number of rows of $M$
5 return Set of (key, value) pairs that each key, $(i,k)$, has a list with
  values $(M,j,m_{ij})$ and $(N,j,n_{jk})$ for all possible values of $j$
```

Algorithm 2: The Reduce Function

```
1 for each key (i,k) do
2     sort values begin with $M$ by $j$ in $list_M$
3     sort values begin with $N$ by $j$ in $list_N$
4     multiply $m_{ij}$ and $n_{jk}$ for $j_{th}$ value of each list
5     sum up $m_{ij} * n_{jk}$
6 return $(i,k), \sum_{j=1} m_{ij} * n_{jk}$
```

Example:

We have an employee dataset with the following structure:

Employee	Name	department	Salary
1	John	HR	60000
2	Bob	IT	80000
3	Alice	HR	50000
4	Charlie	IT	70000
5	David	Marketing	55000

Map Phase:

In this phase, each record is processed to extract the department and salary information.

For the above dataset, the output of the Map Phase will be:

(HR, 60000) (IT, 80000) (HR, 50000) (IT, 70000) (Marketing, 55000)

Shuffle and Sort:

The next step is to group all the salary values by department, which will look like this:

HR: [60000, 50000] IT: [80000, 70000] Marketing: [55000]

Reduce Phase:

In this phase, we calculate the average salary for each department by summing up the salaries and dividing by the number of employees in that department.

For each department:

- HR:
  - sum_salary = 60000 + 50000 = 110000
  - employee_count = 2
  - average_salary = 110000 / 2 = 55000
- IT:
  - sum_salary = 80000 + 70000 = 150000
  - employee_count = 2
  - average_salary = 150000 / 2 = 75000
- Marketing:
  - sum_salary = 55000
  - employee_count = 1
  - average_salary = 55000 / 1 = 55000

Final Output:

The result after the Reduce Phase will be:

HR: Average Salary = 55000 IT: Average Salary = 75000 Marketing: Average Salary = 55000

Conclusion:

In this experiment, MapReduce successfully helps in distributed computation of average salaries for each department. By dividing the task into smaller subtasks (map phase) and then aggregating the results (reduce phase), this method efficiently processes large datasets across multiple nodes.