“Join Algorithms in MapReduce Framework”

AGENDA

- Introduction
- Objectives
- Join Algorithm Selection
- Discussions (Comparison, Advantages, Issues , Applications)
- References

INTRODUCTION

MapReduce:

- Large scale data processing in parallel

- Two phases in MapReduce
    - Read a lot of Data
    - **Map**: Extract something you care about from each record
    - Shuffle and Sort
    - **Reduce**: Aggregate, Summarize, Filter or Transform
    - Write the results

- Outline stays the same, map and reduce change to fit the problem

INTRODUCTION

MapReduce A Brief

Map Phase

- map(in_key, in_value) → list(out_key, intermediate_value)
- Processes input key value pairs
- Produces set of intermediate pairs

Reduce Phase

- reduce(out_key,list(intermediate_value)) → list(out_value)
- Combines all intermediate value for a particular key.
- Produces a set of merged output values(usually just one)

INTRODUCTION

Hadoop:

- Apache Hadoop is an open-source software framework that allows for the distributed processing of large datasets across clusters of commodity computers using a simple programming model.

- Hadoop is an open source implementation of Google MapReduce, GFS (distributed file system)

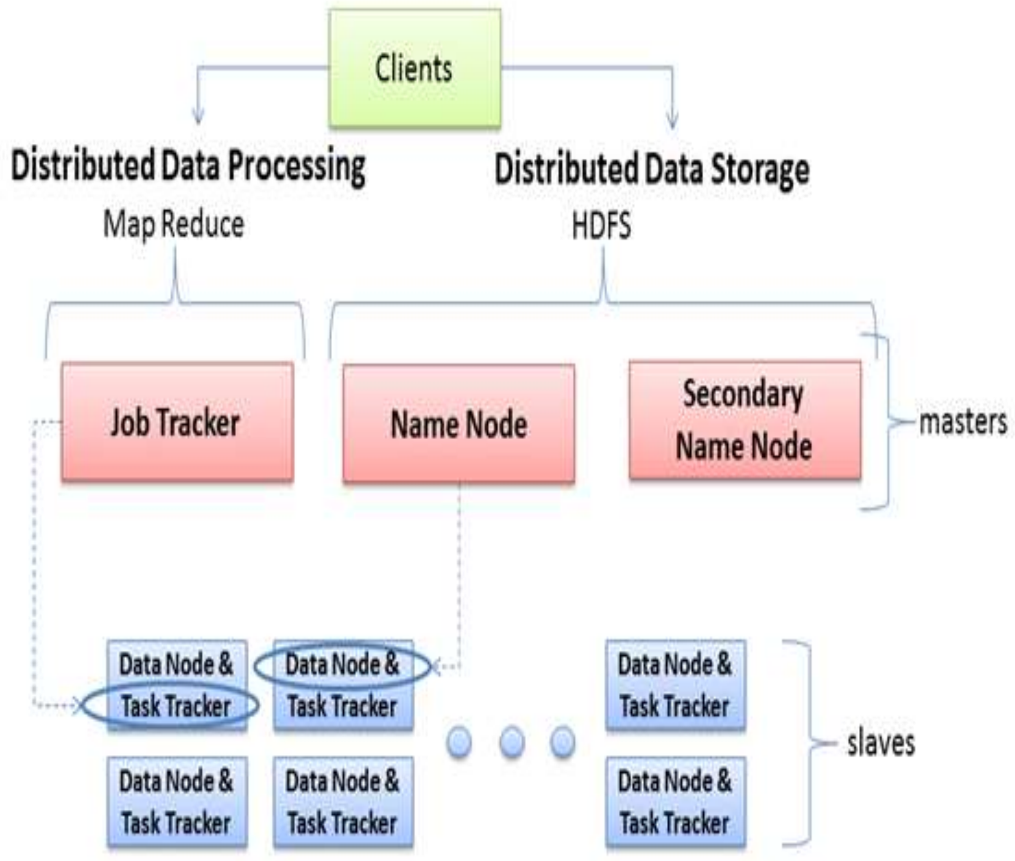
- Supports running of applications on large clusters of commodity hardware.

- Task are divided into Map-Reduce framework

- Provides a distributed file system that stores data on the compute nodes.

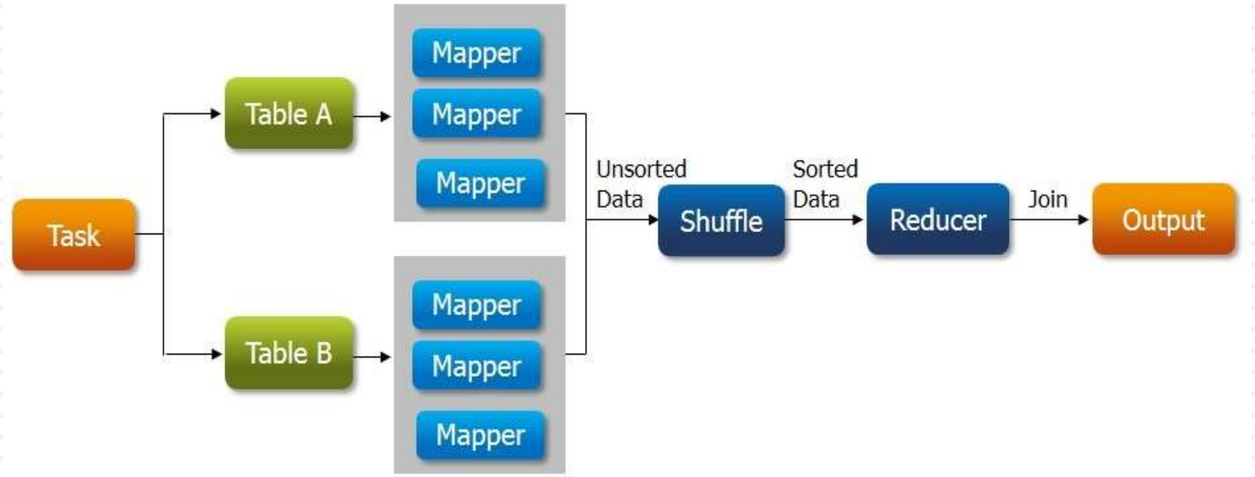
INTRODUCTION

- The Master nodes oversee the two key functional pieces that make up Hadoop: storing lots of data (HDFS), and running parallel computations on all that data (Map Reduce).
- The Name Node oversees & coordinates the data storage function (HDFS), while the Job Tracker oversees & coordinates the parallel processing of data using Map Reduce.
- Slave Nodes do all the dirty work of storing the data and running the computations. Each slave runs both a Data Node and Task Tracker daemon that communicate with and receive instructions from their master nodes.
- The Task Tracker daemon is a slave to the Job Tracker, the Data Node daemon & Name Node.



INTRODUCTION

Join Operation:



- Two stages- a ‘**Map stage**’ and a ‘**Reduce stage**’.
- A mapper’s job during Map Stage is to *“read”* the data from join tables and to *“return”* the **‘join key’** and **‘join value’** pair into an intermediate file.
- Further, in the shuffle stage, this intermediate file is then sorted and merged.
- The reducer’s job during reduce stage is to take this sorted result as input & complete the task of join.

OBJECTIVE

Objectives:

The contributions of this seminar can be summarized as follows:

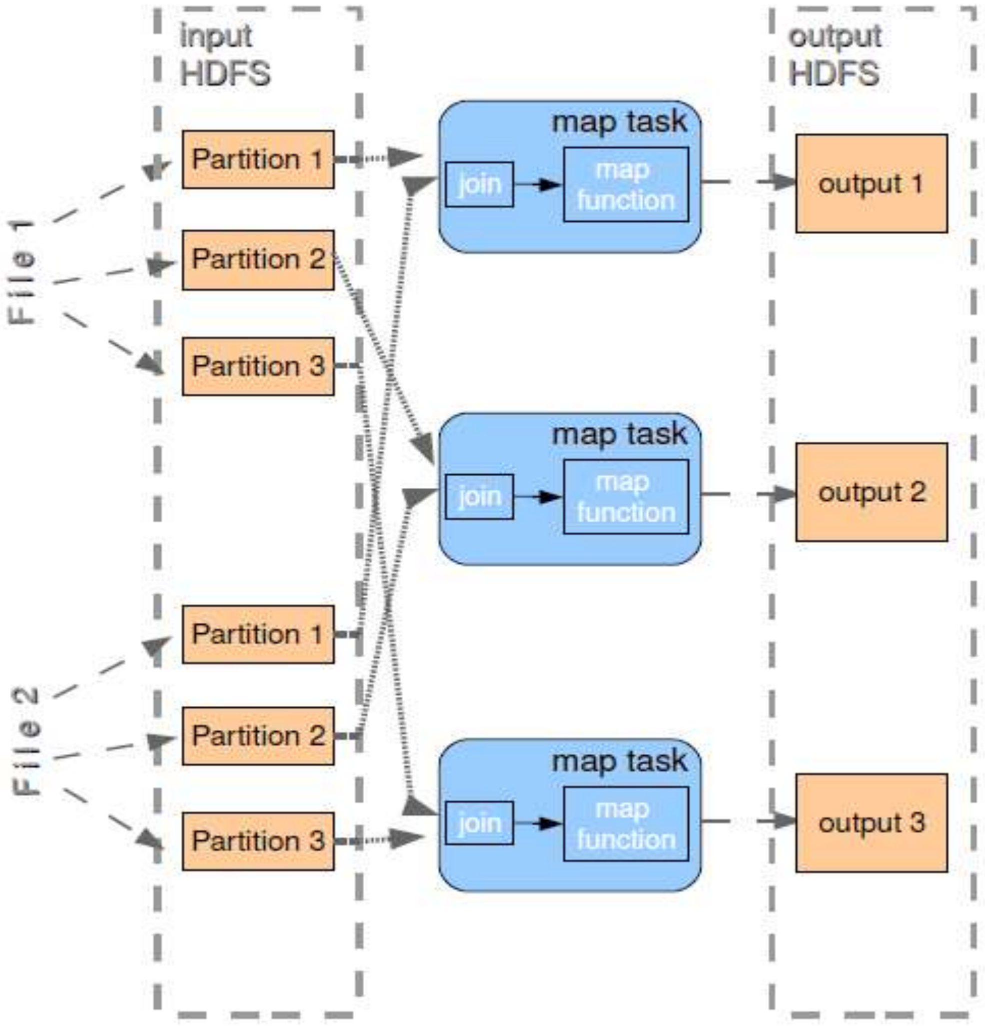
- There is a great controversy over which join implementation the user should choose.
- Analysis of MapReduce framework, make a comparison between different ways to implement join in MapReduce. Examine the existing algorithms for joining datasets using Map/Reduce.
- Analysis of affecting factors to MapReduce, and study the new influence of these factors when implementing join in MapReduce
- This lec describes various methods for Joining datasets using Hadoop MapReduce & proposes a strategy to find out best suitable join processing algorithm.

JOIN ALGORITHMS

Map Side Join

A map-side join takes place when the data is joined before it reaches the map function.

- A given key has to be in the same partition in each dataset so that all partitions that can hold a certain key are joined together. For this to work, all datasets should be partitioned using the same partitioner and moreover, the number of partitions in each dataset should be identical.
- The sort order of the data in each dataset must be identical. This requires that all datasets must be sorted using the same comparator.



JOIN ALGORITHMS

Reduce Side Join

Map Phase

The ‘map’ phase only pre-processes the tuples of the two datasets to organize them in terms of the join key.

Partitioning and Grouping Phase

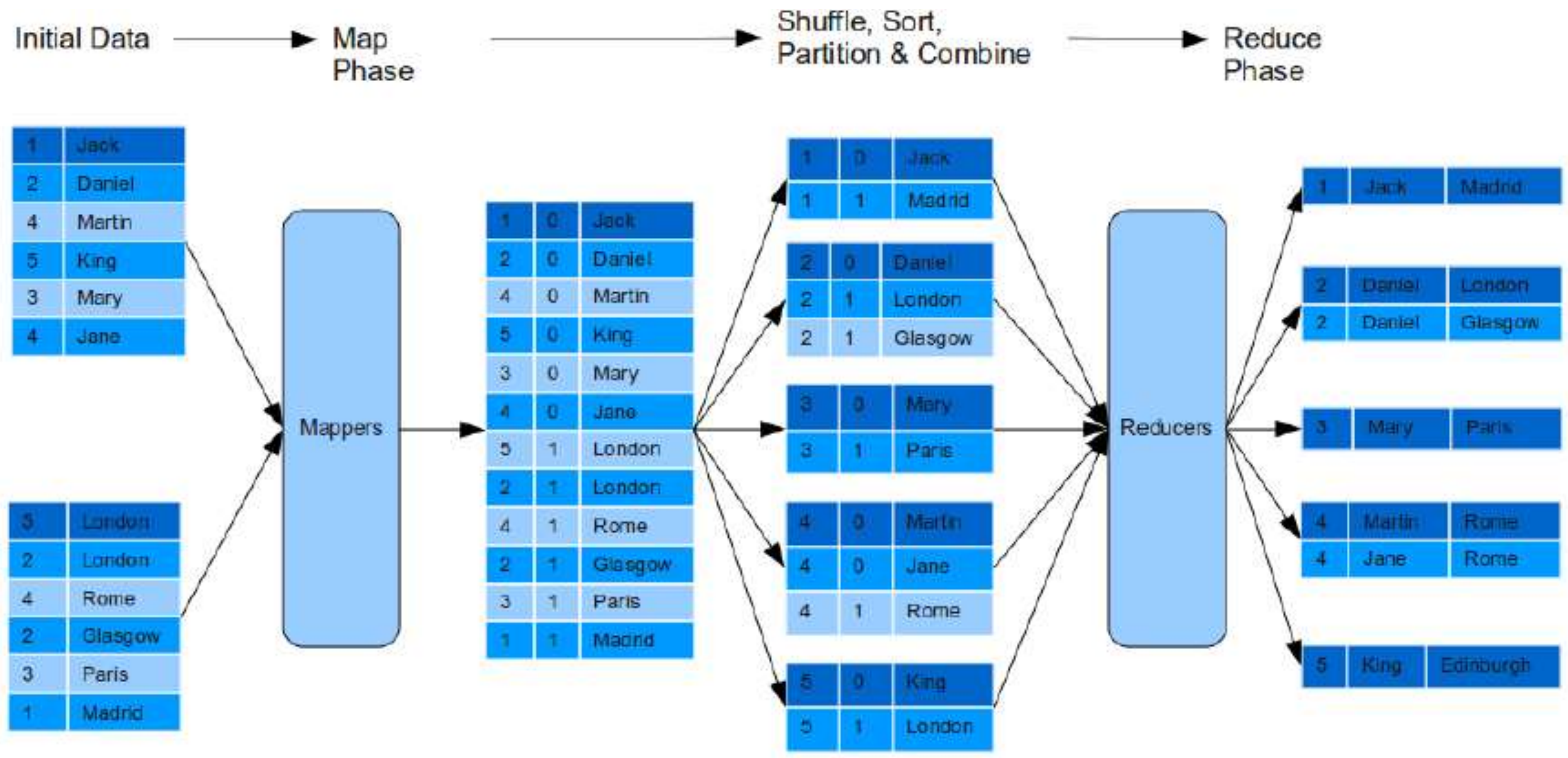
The partitioner partitions the tuples among the reducers based on the join key such that all tuples from both datasets having the same key go to the same reducer. The **reduce** function is called once for a key and the list of values associated with it. This list of values is generated by grouping together all the tuples associated with the same key. We are sending a composite TextPair key and hence the Reducer will consider (key, tag) as a key

Reduce Phase

The framework sorts the keys (in this case, the composite TextPair key) and passes them on with the corresponding values to the Reducer. Since the sorting is done on the composite key (primary sort on Key and secondary sort on Tag), tuples from one table will all come before the other table.

JOIN ALGORITHMS

Reduce Side Join



1	Jack
2	Daniel
4	Martin
5	King
3	Mary
4	Jane

5	London
2	London
4	Rome
2	Glasgow
3	Paris
1	Madrid

1	0	Jack
2	0	Daniel
4	0	Martin
5	0	King
3	0	Mary
4	0	Jane
5	1	London
2	1	London
4	1	Rome
2	1	Glasgow
3	1	Paris
1	1	Madrid

1	0	Jack
1	1	Madrid

2	0	Daniel
2	1	London
2	1	Glasgow

3	0	Mary
3	1	Paris

4	0	Martin
4	0	Jane
4	1	Rome

5	0	King
5	1	London

1	Jack	Madrid

2	Daniel	London
2	Daniel	Glasgow

3	Mary	Paris

4	Martin	Rome
4	Jane	Rome

5	King	Edinburgh

JOIN ALGORITHMS

Repartition Join [8]

Map Phase :

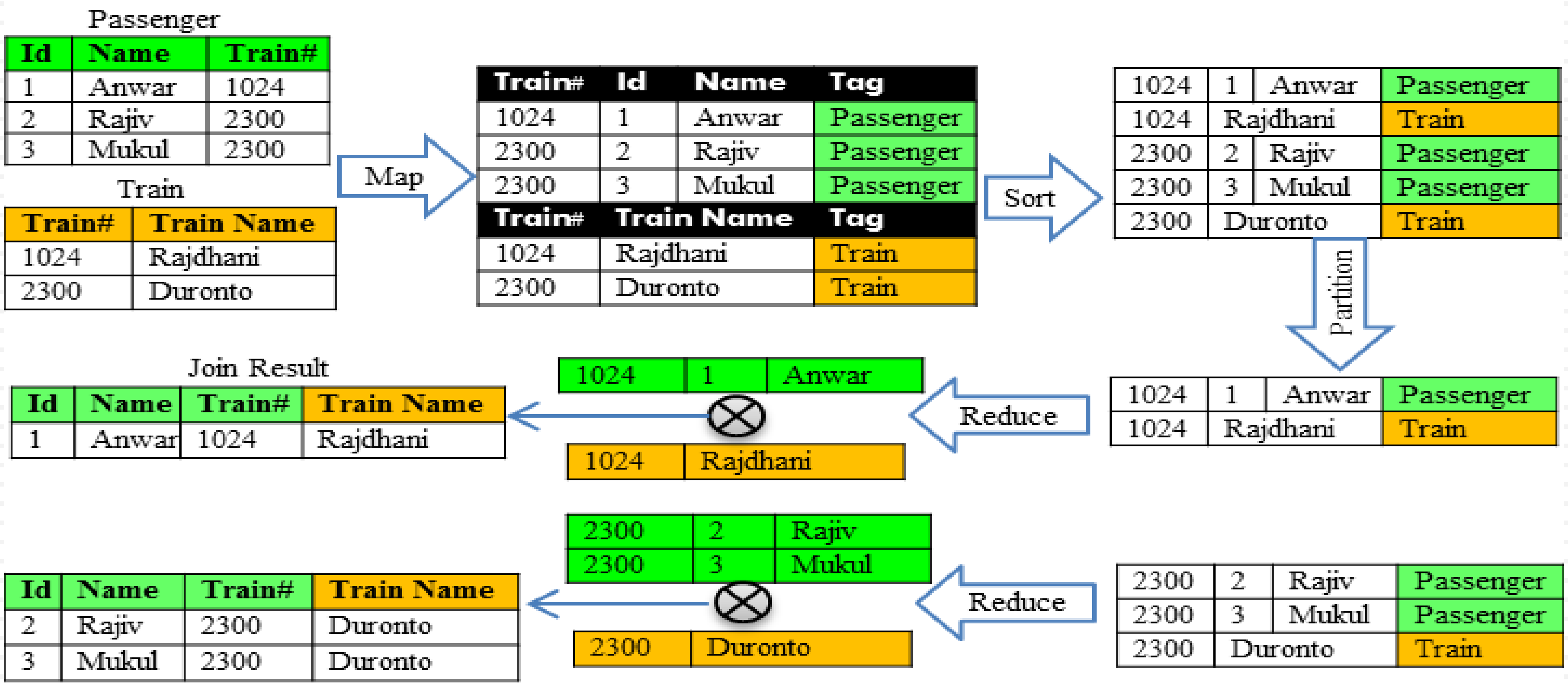
- Each map task works on a split of either R or L.
- Each map task tags the record with its originating table.
- Outputs the extracted join key and the tagged record as a (key, value) pair.
- The outputs are then partitioned, sorted and merged by the framework.

Reducer Phase :

- All the records for each join key are grouped together and eventually fed to a reducer.
- For each join key, the reduce function first separates and buffers the input records into two sets according to the table tag.
- Performs a cross-product between records in the above sets.

JOIN ALGORITHMS

Repartition Join [8]



Passenger

Id	Name	Train#
1	Anwar	1024
2	Rajiv	2300
3	Mukul	2300

Train

Train#	Train Name
1024	Rajdhani
2300	Duronto

Train#	Id	Name	Tag
1024	1	Anwar	Passenger
2300	2	Rajiv	Passenger
2300	3	Mukul	Passenger
Train#	**Train Name**		**Tag**
1024	Rajdhani		Train
2300	Duronto		Train

1024	1	Anwar	Passenger
1024	Rajdhani		Train
2300	2	Rajiv	Passenger
2300	3	Mukul	Passenger
2300	Duronto		Train

1024	1	Anwar	Passenger
1024	Rajdhani		Train

1024	1	Anwar

$\otimes$

1024	Rajdhani

Join Result

Id	Name	Train#	Train Name
1	Anwar	1024	Rajdhani

2300	2	Rajiv	Passenger
2300	3	Mukul	Passenger
2300	Duronto		Train

2300	2	Rajiv
2300	3	Mukul

$\otimes$

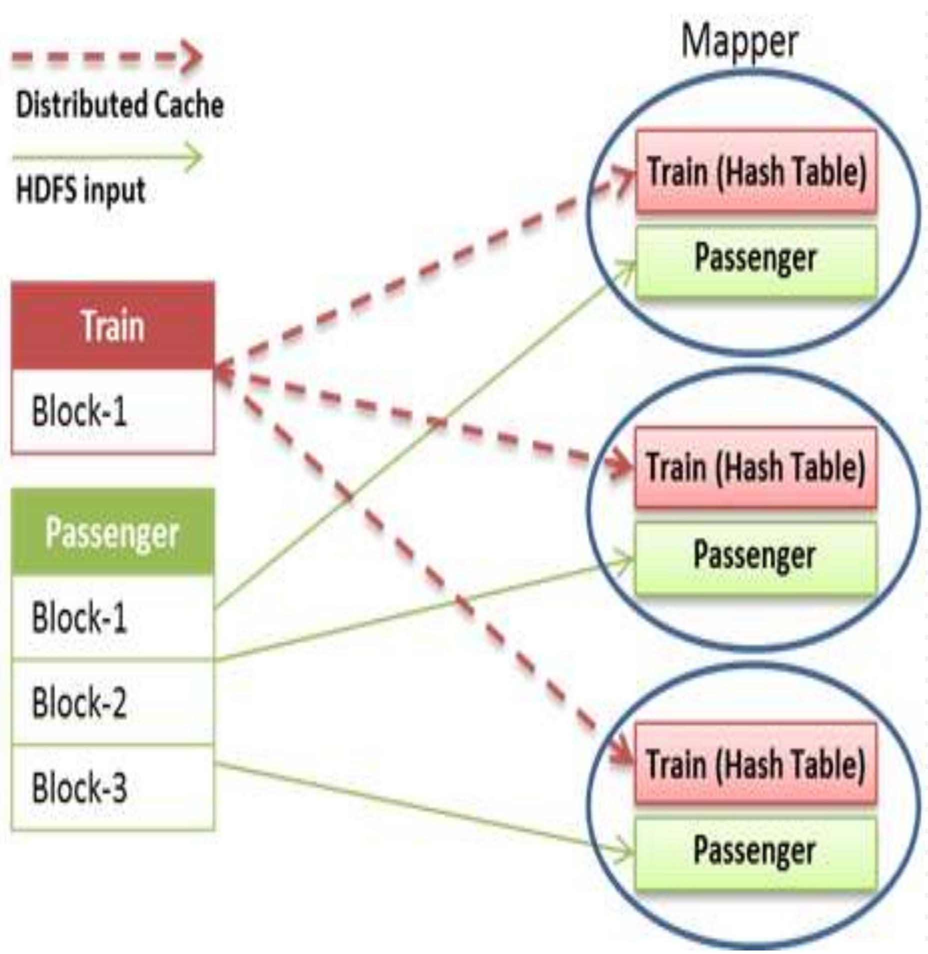
2300	Duronto

Id	Name	Train#	Train Name
2	Rajiv	2300	Duronto
3	Mukul	2300	Duronto

JOIN ALGORITHMS

Broadcast Join [8]

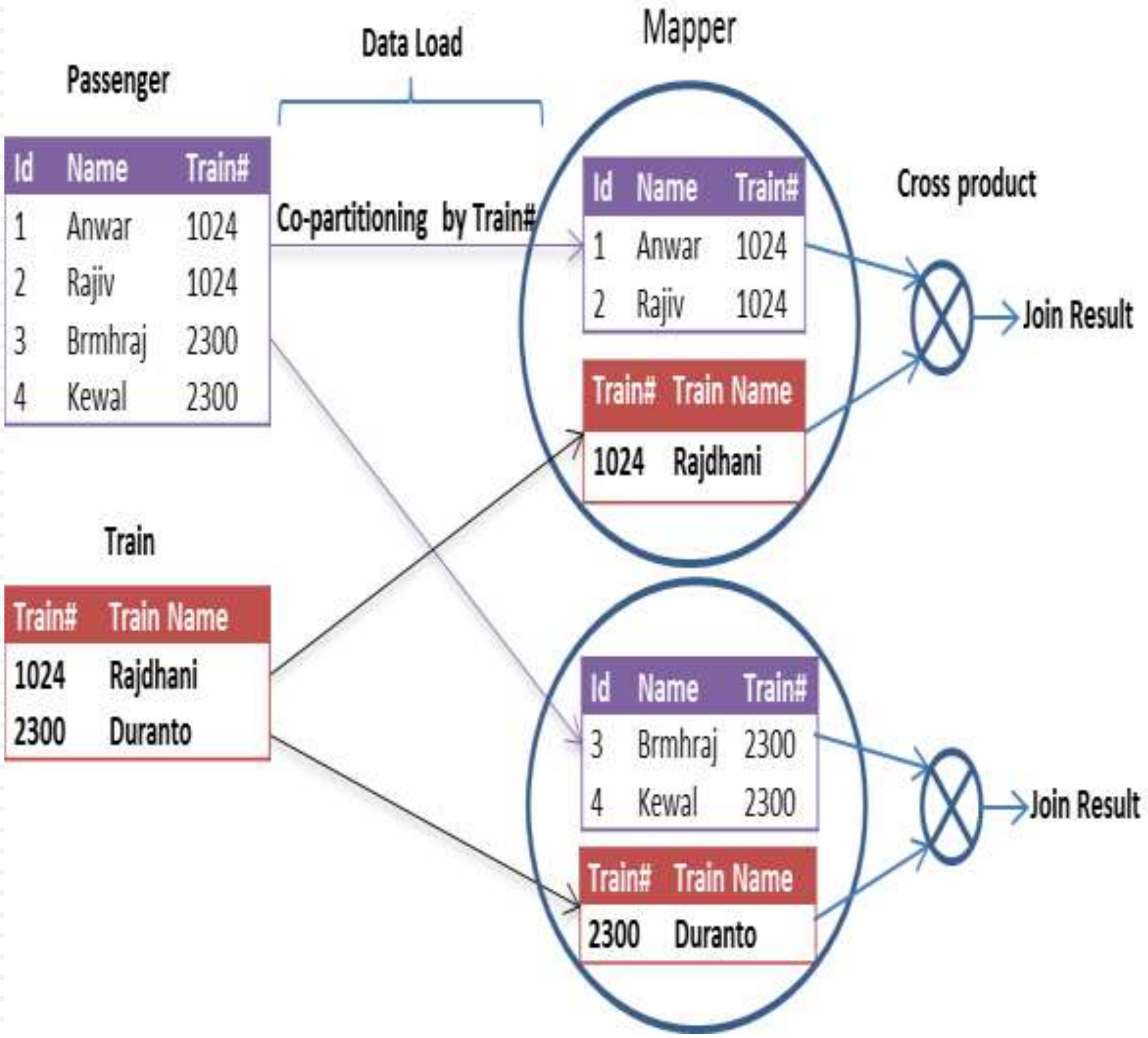
- Broadcast join is used there need to join the smaller data set with large data set.
- With the assumption that smaller data set will fit into memory easily.
- The smaller data set is pushed into distributed cache and replicated among the cluster nodes.
- In init method of mapper a hash map(or other associative array) is built from smaller data set.
- And join operation is performed in map method and output is emitted.



JOIN ALGORITHMS

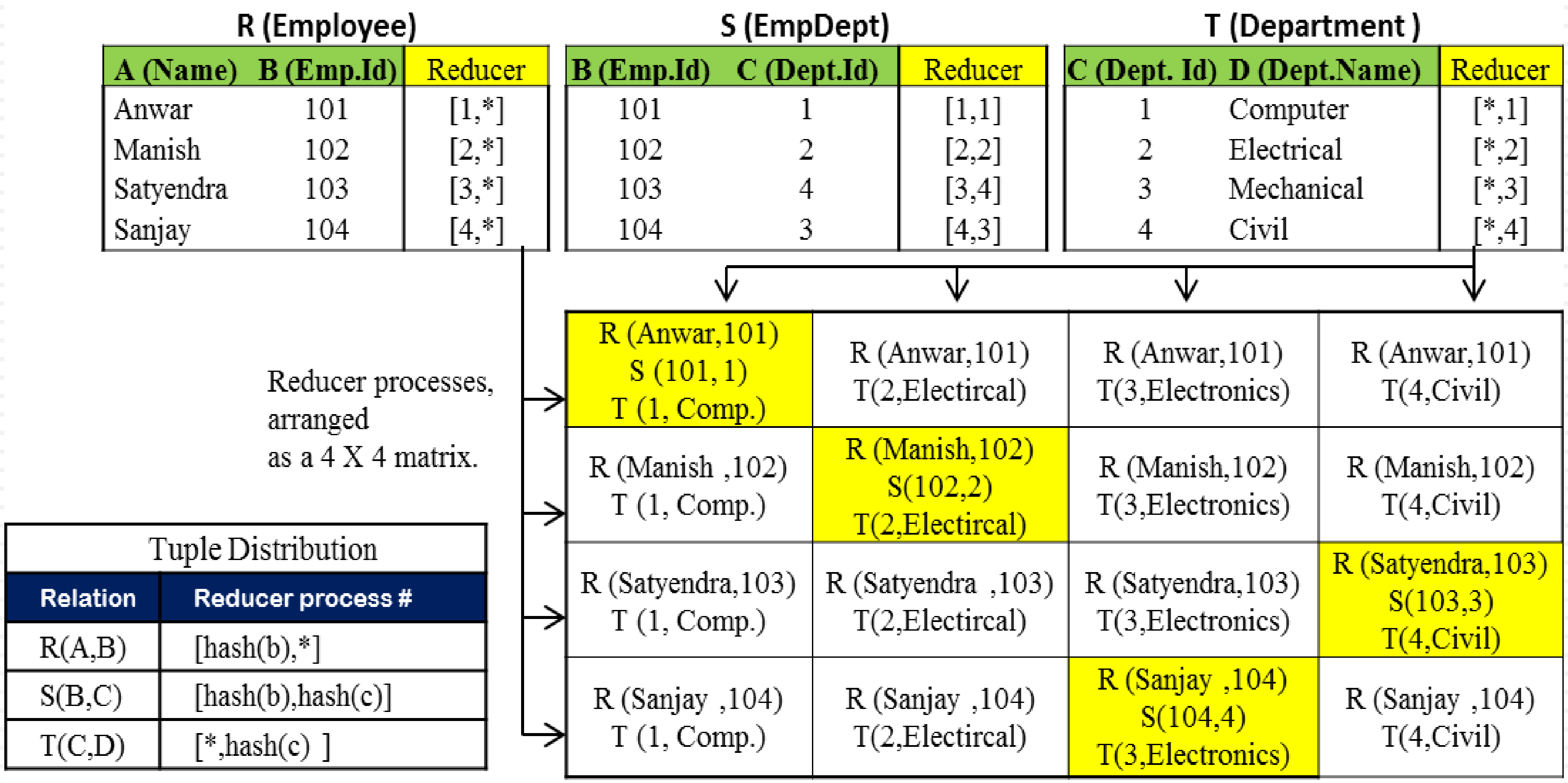
Trojan Join [8]

- Trojan Join supports more effective join by assuming we know the schema and workload
- Idea is to co-partition the data at load time.
- Joins are locally processed within each node at query time, but we are free to group the data on any attribute other than the join attribute in same mapreduce job



JOIN ALGORITHMS

Replicated Join [8]



R (Employee)

A (Name)	B (Emp.Id)	Reducer
Anwar	101	[1,*]
Manish	102	[2,*]
Satyendra	103	[3,*]
Sanjay	104	[4,*]

S (EmpDept)

B (Emp.Id)	C (Dept.Id)	Reducer
101	1	[1,1]
102	2	[2,2]
103	4	[3,4]
104	3	[4,3]

T (Department)

C (Dept. Id)	D (Dept.Name)	Reducer
1	Computer	[*,1]
2	Electrical	[*,2]
3	Mechanical	[*,3]
4	Civil	[*,4]

Reducer processes, arranged as a 4 X 4 matrix.

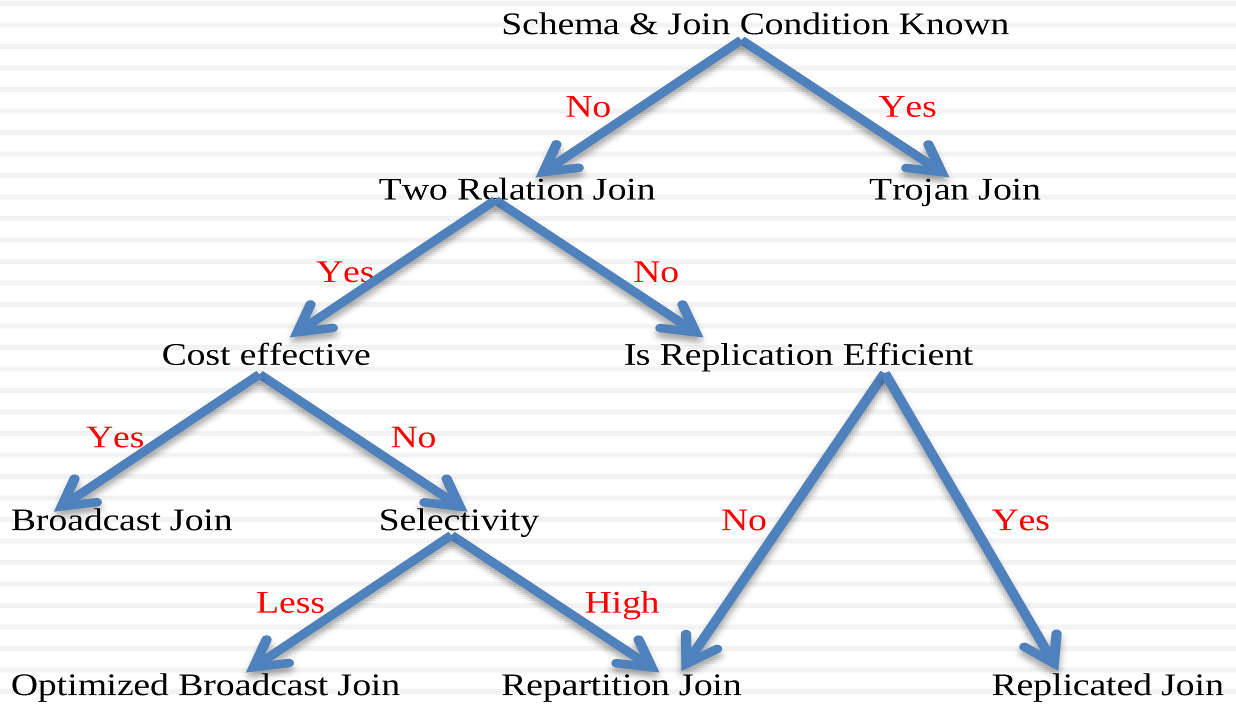
R (Anwar,101) S (101, 1) T (1, Comp.)	R (Anwar,101) T(2,Electircal)	R (Anwar,101) T(3,Electronics)	R (Anwar,101) T(4,Civil)
R (Manish ,102) T (1, Comp.)	R (Manish,102) S(102,2) T(2,Electircal)	R (Manish,102) T(3,Electronics)	R (Manish,102) T(4,Civil)
R (Satyendra,103) T (1, Comp.)	R (Satyendra ,103) T(2,Electircal)	R (Satyendra,103) T(3,Electronics)	R (Satyendra,103) S(103,3) T(4,Civil)
R (Sanjay ,104) T (1, Comp.)	R (Sanjay ,104) T(2,Electircal)	R (Sanjay ,104) S(104,4) T(3,Electronics)	R (Sanjay ,104) T(4,Civil)

Tuple Distribution

Relation	Reducer process #
R(A,B)	[hash(b),*]
S(B,C)	[hash(b),hash(c)]
T(C,D)	[*,hash(c)]

JOIN ALGORITHM SELECTION

Decision Tree for Join Algorithm Selection [8]



DISSCUSSIONS

Criteria for Join Algorithm Selection

- Prior knowledge of schema and join condition
- Transmission over network
- Number of tuples

DISSCUSSIONS

Comparison of Join processing methods

Join Type	MapReduce jobs	Advantages	Issues
Repartition	1 MapReduce job	Simple implementation of Reduce phase	Sorting and movement of tuples over network
Broadcast	1 Map phase	No sorting and movement of tuples	Useful only if one relation is small.
Trojan	1 Map phase	Uses schema knowledge	Useful, if join conditions are known
Replicated	1 MapReduce job	Efficient for Star join and Chain join	For large relations more number of reducers / replicas are required

REFERENCES

[1] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Commun. ACM, 51(1):107–113, 2008.

[2] Fuhui Wu et al. “Comparison & Performance Analysis of Join Approach in MapReduce” ISCTCS 2012, CCIS 320, pp. 629–636[8]

[3] Marko Lalić et al. “Comparison of a Sequential & a MapReduce Approach to Joining Large Datasets” MIPRO 2013, pp.1289-1291[3]

[4] Spyros, B., Jignesh, M.P., Vuk, E., Jun, R., Eugene, J., Yuanyuan, T.: A Comparison of Join Algorithms for Log Processing in MapReduce. In: SIGMOD 2010, June 6–11. ACM, Indianapolis (2010)

[5] Foto N. Afrati et al. “Optimizing Multiway Joins in a Map-Reduce Environment” IEEE Transactions On Knowledge And Data Engineering, pp. 1282- 1298[17], 2011

[6] Alper Okcan et al. “Processing Theta-Joins using MapReduce” SIGMOD‟11, June 12–16, 2011 pp. 949-951[12]

[7] Xiaofei Zhang et al. “Efficient Multiway Theta Join Processing Using MapReduce” Proceedings of the VLDB Endowment, Vol. 5, No. 11, pp.1184-1196[12]

[8] Anwar Shaikh et al. “Join Query Processing in MapReduce Environment” CNC 2012, LNICST , pp.275-281[7]

THANK YOU !