“Map Reduce” Computing Paradigm

pm jat @ daiict

Map Reduce

- Problem Context
- Map Reduce as Solution
- Map Reduce Infrastructure
- Map Reduce Program execution flow
- Creating Map Reduce Programs
- Examples

Map Reduce - Problem Context

- You have huge file say in terms of 100’s of TBs, and
- We need to process the whole file – say scan and extract some information
- Computation is nearly infeasible on single computer, whatever powerful it is.

Map Reduce - Problem Context

- Application scenario:
  - data generated by online services like google maps, facebook
  - sensors data

  These data are very large – Terabytes is a small unit here!
- An example task
  - Suppose we have temperature data by sensors at different locations in a city (or even multiple cities) for last 10 years. Say each record in the file contains values for two attributes(LocationId, TimeStamp, Temperature).
  - Let our goal be to compute Average temperature of each month, i.e. January to December averaged over 10 years.

Map Reduce - Problem Context

Here are some numbers

- Suppose a computer has a read speed of 35MB/sec[1];
- Reading a file of 1TB should take approx. 8 hours
- Reading a file of 100TB should take approx. 800 hours (33 days)
- This is just read time, computation time is extra.
- Challenge here is File Size !
- Map reduce is cluster based architecture that addresses this issue!

[1] http://mmds.org/mmds/v2.1/ch02-mapreduce.pdf

“Map Reduce” Computing Paradigm

- Map Reduce is primarily a very large file processing framework
- It is highly scalable approach; computation is done in parallel on several computers, typically called computing nodes
- A typical Map Reduce computation processes terabytes of data on hundreds (or even thousands) of machines.
- It was introduced by Google in 2004 through article “MapReduce: Simplified data processing on large clusters“ after successfully using it for their internal use.

“Map Reduce” Computing Paradigm

- Here are few quotes from their ACM Communication article in 2010 [2]

“We built a system around this programming model in 2003 to simplify construction of the inverted index for handling searches at Google.com.”

“…. more than 10,000 distinct programs have been implemented using MapReduce at Google, including algorithms for large-scale graph processing, text processing, machine learning, and statistical machine translation”

“Map Reduce” Computing Paradigm

- Creating parallel and distributed computing programs are inherently hard.
- Maps reduce comes as a very simple programming abstraction that hides all underlying communications.
- Programmers only needs to write map and reduce functions while are underlying communications (and many more things) are performed and managed by map-reduce infrastructure.
- Map Reduce said to be a “**simple programming abstraction**” over “**cluster based computing**”

“Map Reduce” Computing Paradigm

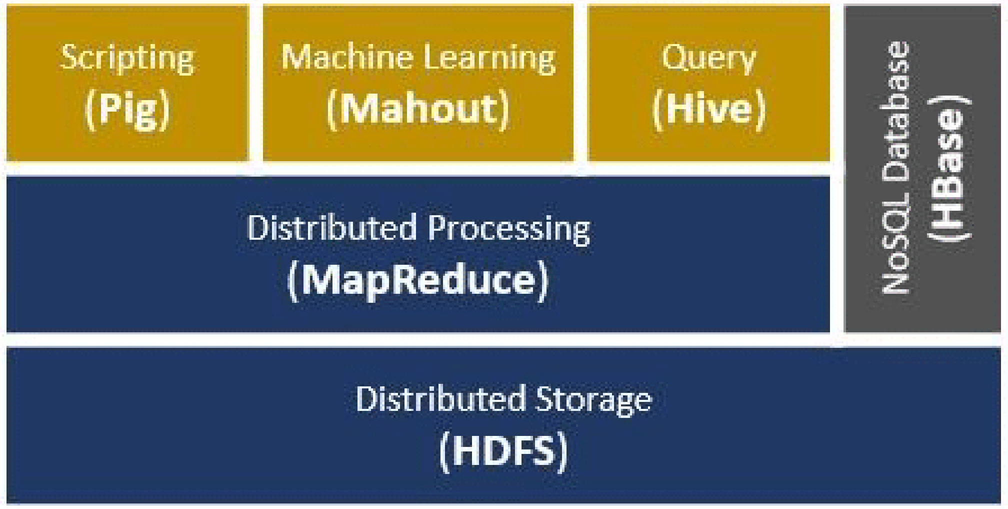
- Execution of map-reduce programs are automatically parallelized and executed on a large number of “commodity machines”.

Map-Reduce infrastructure

- Map-Reduce infrastructure includes
  - Networked Computers (“Commodity Computers”)
  - A Distributed File Systems (example: Google File System, Hadoop Distributed File System)
  - Map Reduce Library
- Distributed File Systems and Map reduce library transparently take care of
  - **Parallelization**
  - **Fault Tolerance** [replicated data]
  - **Data Distribution**
  - **Load Balancing** [overloading/under-loading of nodes]

Map-Reduce implementations

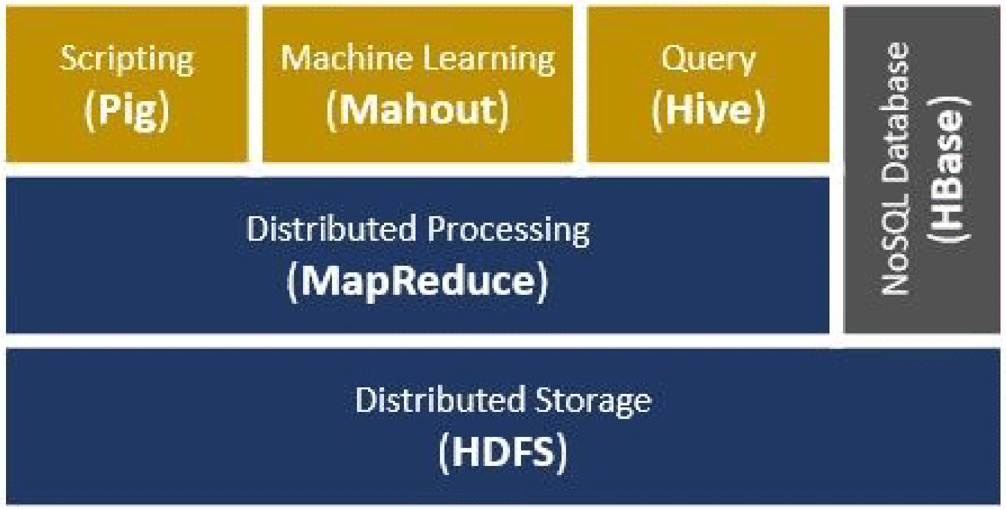
- First Map Reduce implementation was done by Google but that was not available to world!
- Apache Hadoop is open source Map Reduce implementation, and provides all necessary infrastructure to perform map reduce based computations!
- Diagram here depicts hadoop system.


Scripting (Pig)
Machine Learning (Mahout)
Query (Hive)
NoSQL Database (HBase)
Distributed Processing (MapReduce)
Distributed Storage (HDFS)

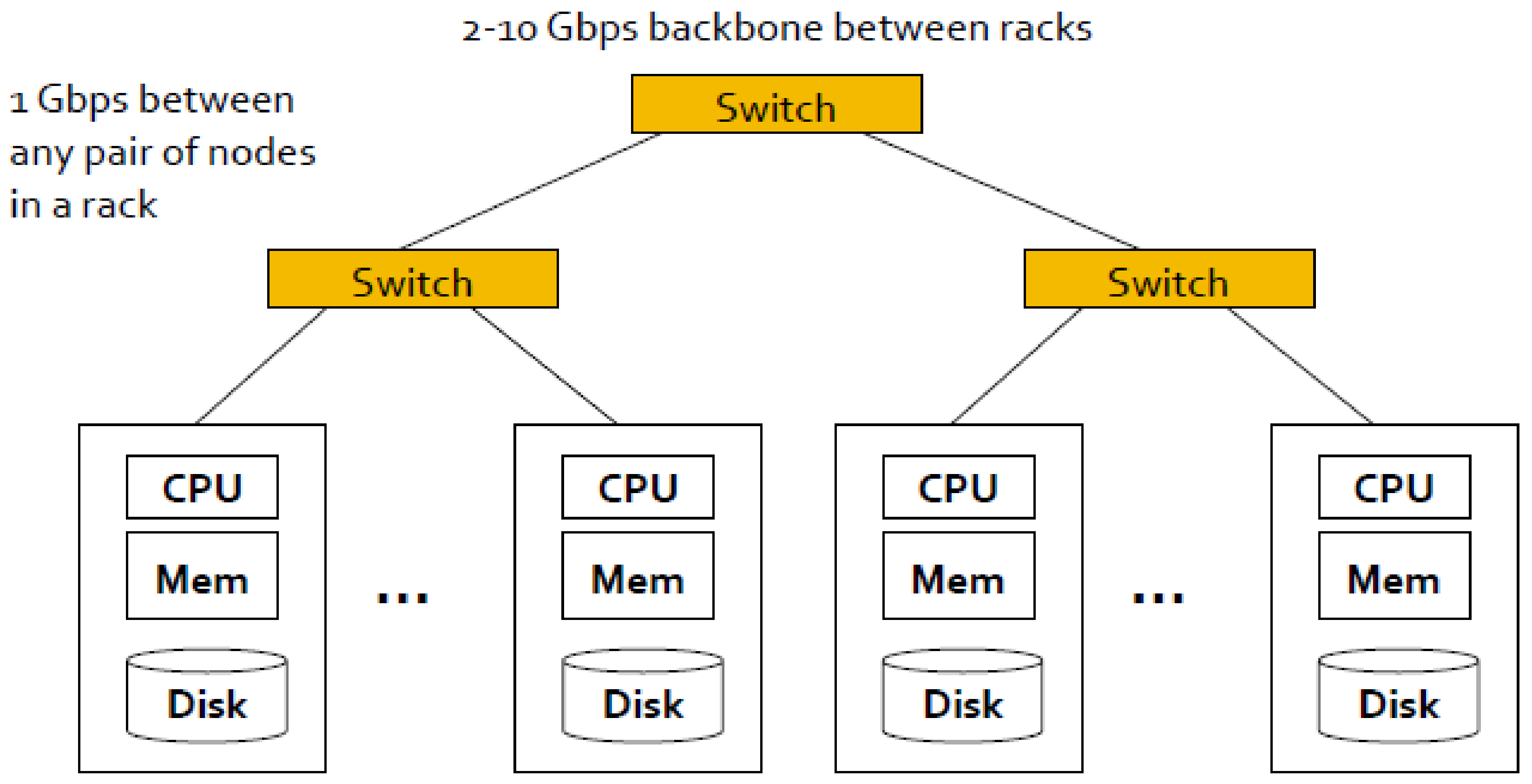
- Hadoop is considering a versatile large scale computing ecosystem.

Map-Reduce implementations

- Various higher layer tools are built on top HDFS and Map Reduce
- **Pig** (Manage the data),
- **Hive** (SQL like interface over raw files),
- **Mahout** (ML Library),
- **HBase** (a column family database)



Cluster Architecture


2-10 Gbps backbone between racks
1 Gbps between any pair of nodes in a rack
Switch
Switch
Switch
CPU
Mem
Disk
...
CPU
Mem
Disk
CPU
Mem
Disk
...
CPU
Mem
Disk

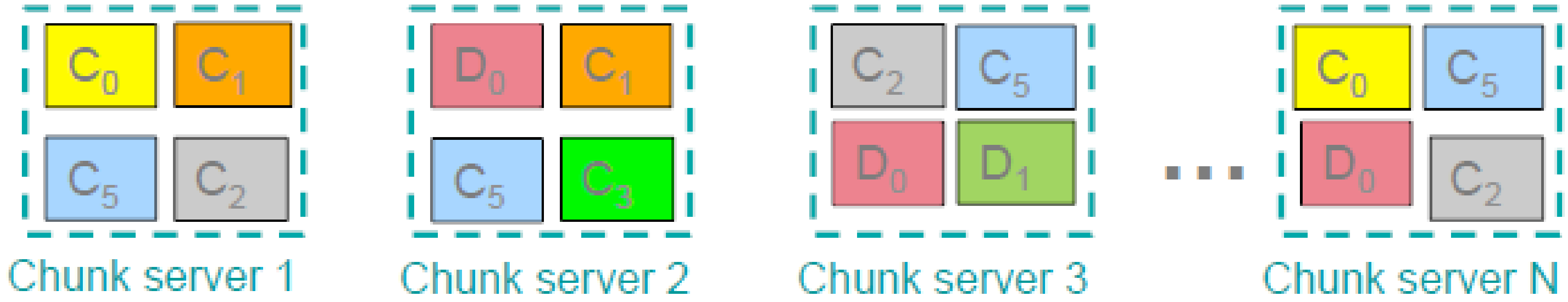
Each rack contains 16-64 nodes

In 2011 it was guestimated that Google had 1M machines, http://bit.ly/Shh0RO

http://mmds.org/mmds/v2.1/ch02-mapreduce.pdf

How typically data are stored on DFS

- Data files are broken into small-small “data chunks” spread over several machines.
- Each data chunk is replicated on multiple machines (with a replication factor), called as chunk servers.

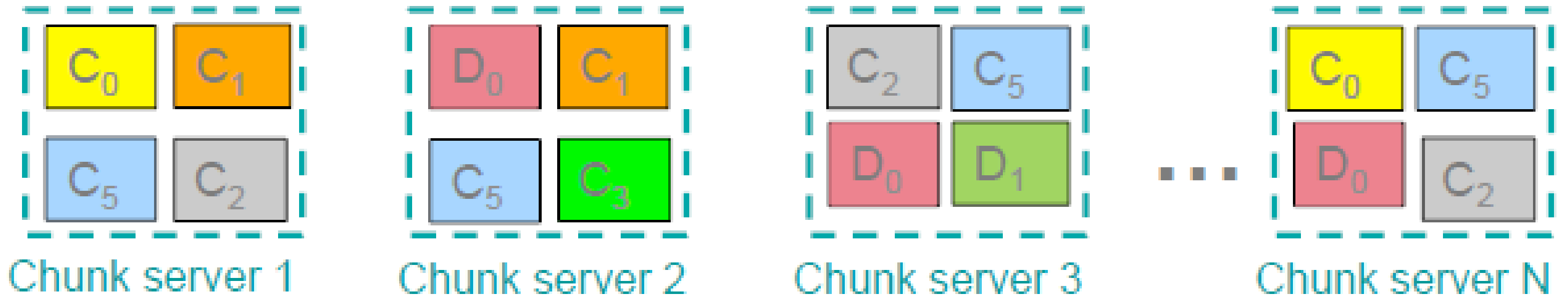


- Suppose a file C has been split into data chunks: C_0, C_1, C_2, C_3, and so on. Figure here show that how these chunks are distributed and replicated (x2) on four different computing nodes.

http://mmds.org/mmds/v2.1/ch02-mapreduce.pdf

How typically data are stored on DFS

- C_0 is stored on (CS1, CS-N), C_1 on (CS1, CS2) , C_2 on (CS1, CS 3), and so forth.
- Similarly chunks of file D are distributed and replicated (x2)



- Computers that store data are more often referred as "Compute Nodes" or "Data Nodes"

http://mmds.org/mmds/v2.1/ch02-mapreduce.pdf

Distributed File System

- Distributed File System is a important component of a map reduce system.
- A system setup lets us specify number of computers to work, data chunk size, replication factor or so.
- We call this task as "cluster setup"
- One of computer work as master node (name node in Hadoop)
- DFS should provide a global file system interface, where you have hierarchical directory structure and files within.
- Under the hood files are of course distributed over computers in the cluster.

MR Computation in nutshell

- A map reduce program has just two functions:
  - **Map function** and **Reduce function**
- Map function processes data on the same machine where data are. (Recall data are distributed over a number of computers)
- Computers (chunk servers) on which map function runs are called **Mappers**
- Map functions generates some output on each node where it run.
- Reduce function supposed to process output of Map functions.

MR Computation in nutshell

- But reduce function runs on different computers. Computers where reduce function run are called **Reducers**.
- **Master Node** decides, what computers are going to be Mappers and what computers are going to be Reducers for a Map Reduce Job.
- When all Mappers are done, output of mappers are **shuffled to reducers**. Here data moves from mappers to reducers over the network.
- Map reduce always reads and processes data in Key-Value form. Where every data record is a key, and have a key.
- Output of map function is also in Key-Value form

MR Computation in nutshell

- Output of map functions are shuffled as following: we apply a hash function on key of a data record and compute target reducer
- All data records of a key reaches to a single reducer node.
- When data records arrive to the reducer, they are sorted on their key.
- It is important to note here is that programmer just need to provide map and reduce functions, rest of things are done by map reduce system, like hadoop.

MR Computation in nutshell

- Typically set of tasks done by system
  - Selection of Mappers and Reducers,
  - Master node dispatches map and reduce function to mappers and reducers respectively.
  - Shuffling of mapper outputs to reducers, programmer however can specify the “shuffling hash function”
  - Sorting of shuffled data before delivering to reducers.
  - Note that there will be data records of multiple keys to a reducer.

MR paradigm – Example First[5]

Here is an example from book "Hadoop: The definitive guide"[5]

Weather Dataset (raw NCDC data)

- Consider a huge log file contains weather records of a city.
- Each row has many values but, we will use two values - year and temperature.
- **Computing Goal**: find out maximum temperature for each year.
- In SQL terms, we basically are attempting compute following on a plain text data file (not a DB table)

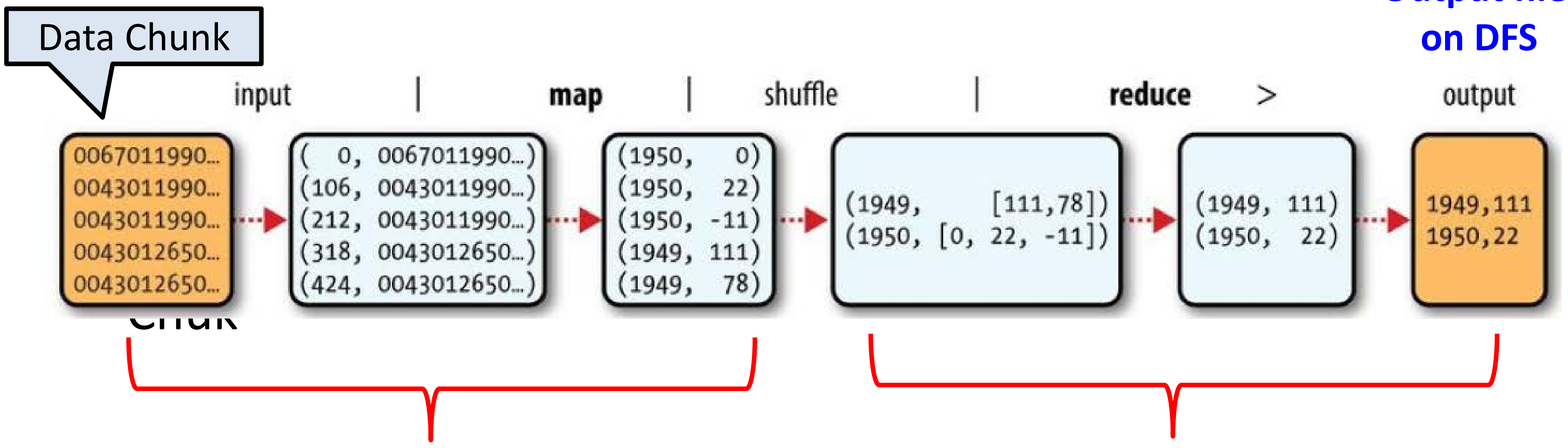
```
"select year, max(temperature)
        from weather group by year"
```

Compute maximum temperature[5]

- Computation is performed by Map and Reduce tasks and in a pipeline. Here is how it goes-

Input file on DFS

Output file on DFS



MAP Tasks, typically on data nodes

Mappers

REDUCE Tasks

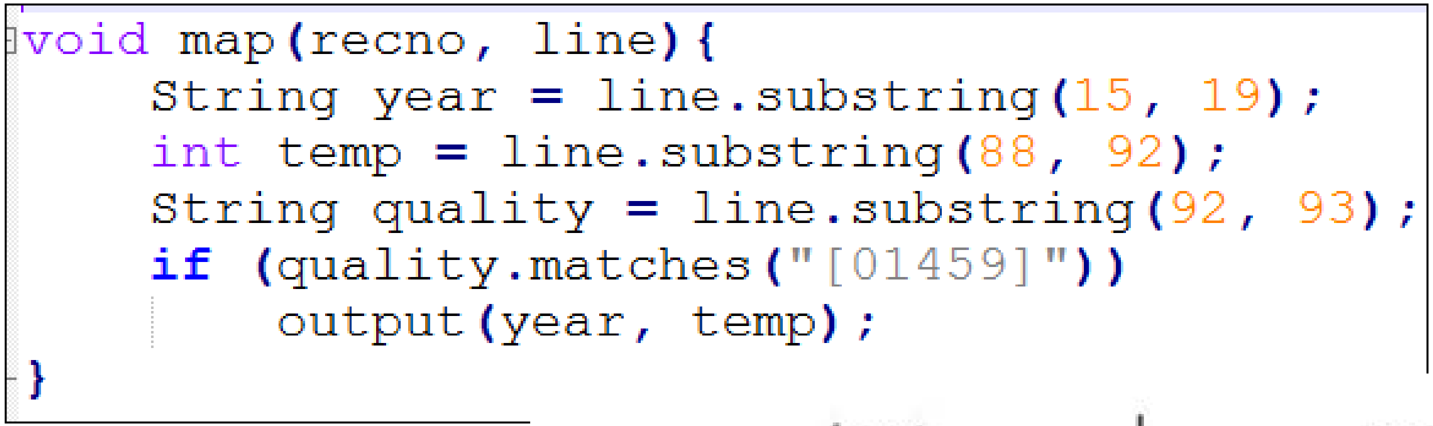
Reducers

Map Task

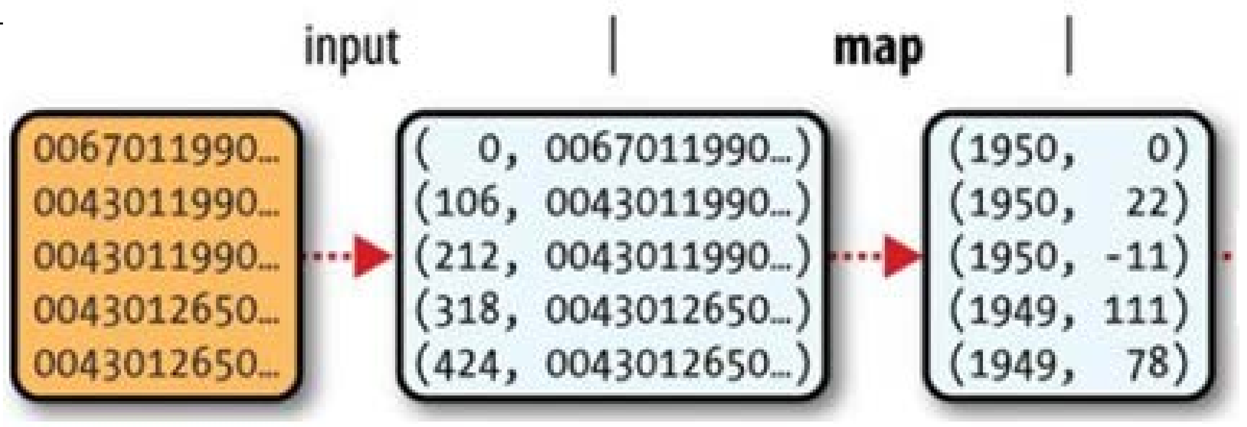
- Data file is put on distributed file system – distributed
- MR Job tracker iterates through data records on a mapper, and calls map function for every data record.
- Map extracts (and outputs) Year and Temperature as Key-Value Pair
- In a mapper’s output, we will have same number of records as of input.

Map Task

- The Map function in pseudo terms and figure should be sketching the input and output



```
void map(recno, line){
    String year = line.substring(15, 19);
    int temp = line.substring(88, 92);
    String quality = line.substring(92, 93);
    if (quality.matches("[01459]"))
        output(year, temp);
}
```



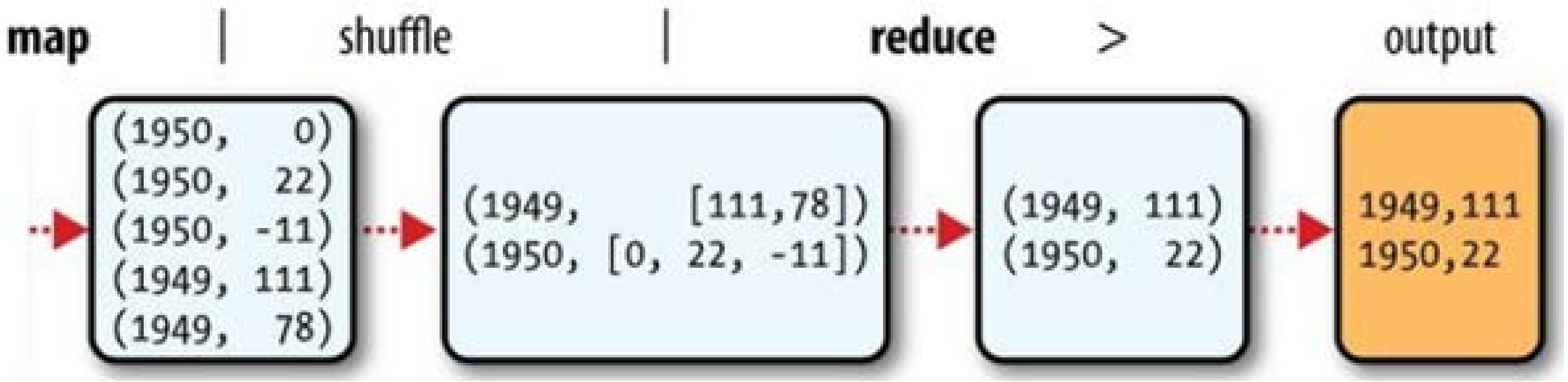
Reduce Task

- Output of all Mappers are shuffled to Reducers
- Let us say there are two reducers.
- Once all mappers done, output from all mappers are shuffled such that all output records of certain set of years go to one reducer and rest to other reducer – using a hash function!
- A hash function is used for identifying the Reducer for a Map output.
- Reducer then applies “aggregation” (i.e. MAX in this case) – recall aggregation from your database course.
- Reducer outputs one record (Key-Value per Key)

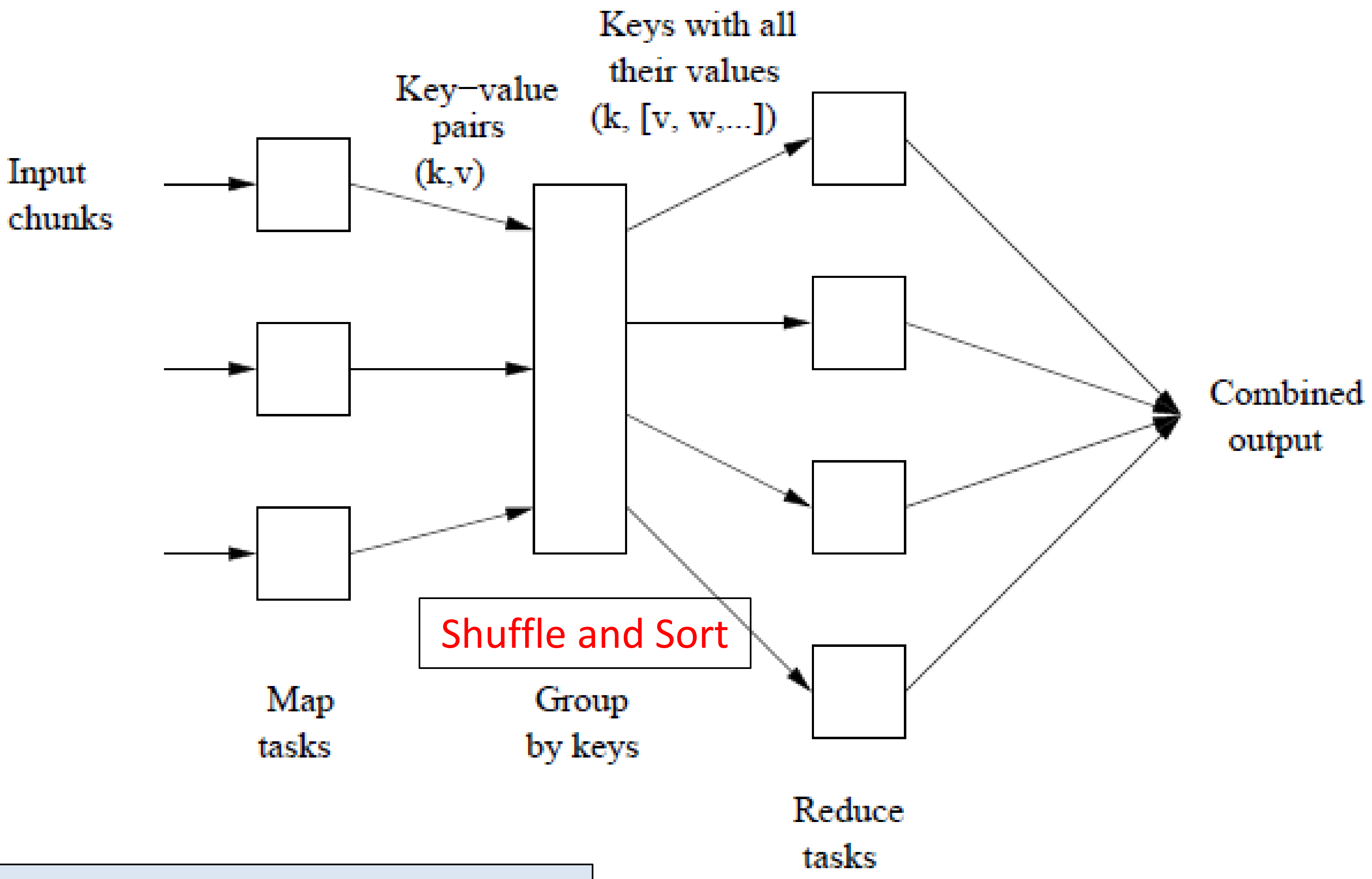
Reduce Task

- Here is user defined Reduce function for finding max temperature. Following figure should depict its functionality

```
void reduce(year, values) {
    int maxValue = MIN_VALUE;
    for (value : values)
        maxValue = max(maxValue, value);
    output(year, maxValue);
}
```



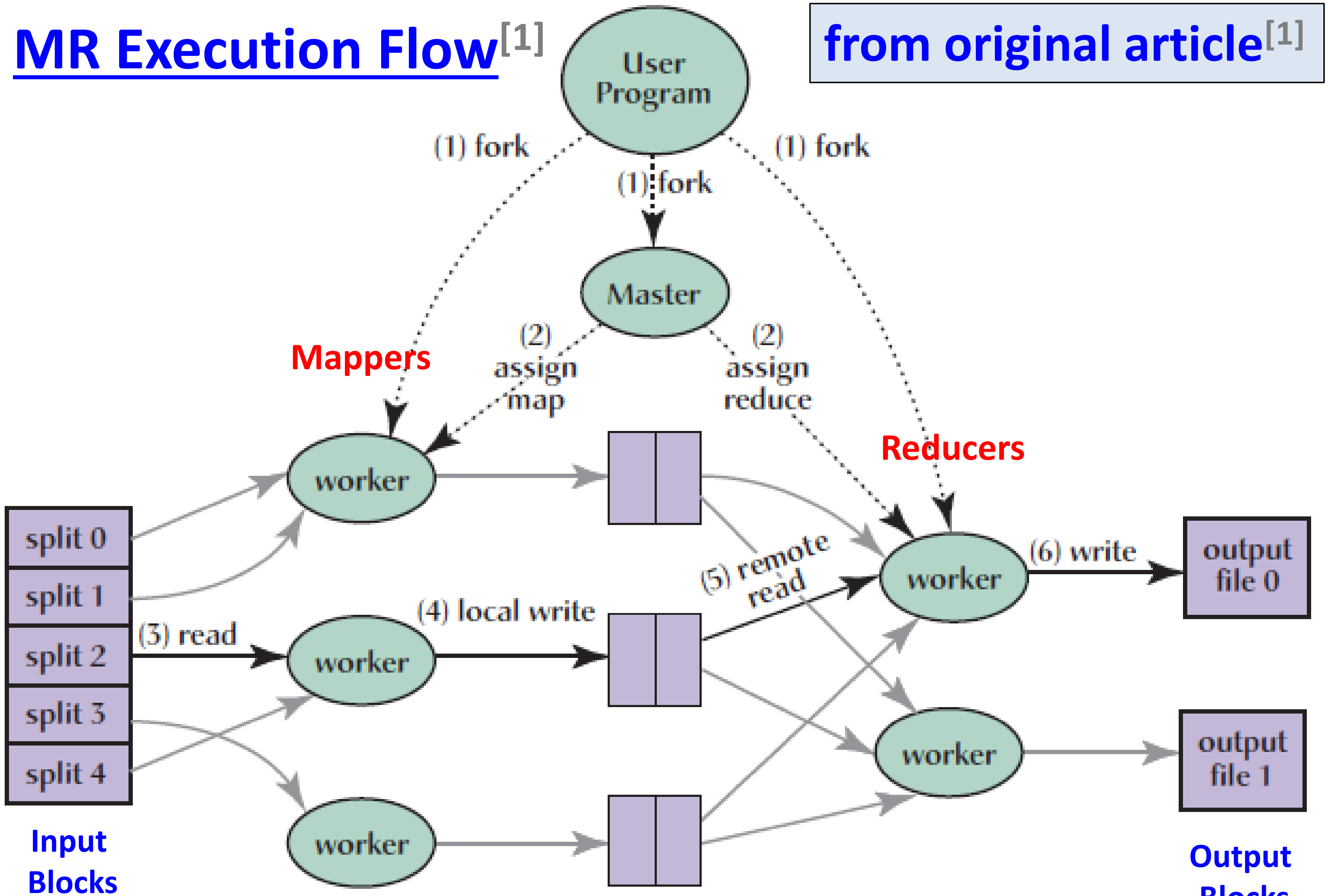
Schematic of a Map Reduce computation



http://infolab.stanford.edu/~ullman/mmds/ch2.pdf

MR Execution Flow[1]

from original article[1]



Mappers output on local File System

Map Reduce execution - summarized

- Client submits a job with (at least) following inputs: Input File Name, Output File Name, Map function, Reduce function
- Job is taken by “Master Node” that acts as a coordinator
- Master node typically creates M mapper tasks and assigns to compute nodes that are “near to data” monitors their execution, ensures successful completion, etc
- “Mapper Node” actually does the computation; produces intermediate results and makes it available to the “shuffle” that are taken by reducer.
- Coordinator designates some nodes as “reducer node” and they finally aggregate the result and output on DFS.

Map Reduce Programming

- User require writing Map and Reduce functions!
- Rest of the things are taken care by Map Reduce System!
  - Assign tasks to Mappers and Reducers
  - Shuffling of intermediate results to Reducers
  - Ensuring efficient parallelism with minimum data movement, all necessary communications, like RPC, or so, load balancing, deal with failures, etc
  - Final Output is made available on DFS

Sources/References

[1] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: Simplified data processing on large clusters." (2004)

[2] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: a flexible data processing tool." *Communications of the ACM* 53.1 (2010): 72-77.

[3] Mining of Massive Datasets, 2nd ed, Jure Leskovec, Anand Rajaraman, Jeffrey David Ullman, Cambridge University Press

[4] Video lessons by Anand Rajaraman on Map-Reduce at mmds.org [strongly recommended to go through]

[5] White, Tom. “Hadoop: The definitive guide“, 4th ed, O'Reilly Media, Inc., 2015.