Date Day

"MapReduce"

→ Map reduce performs the processing and generation of large data sets in a distributed and parallel manner.

→ MapReduce consist of two distinct task - Map and Reduce.

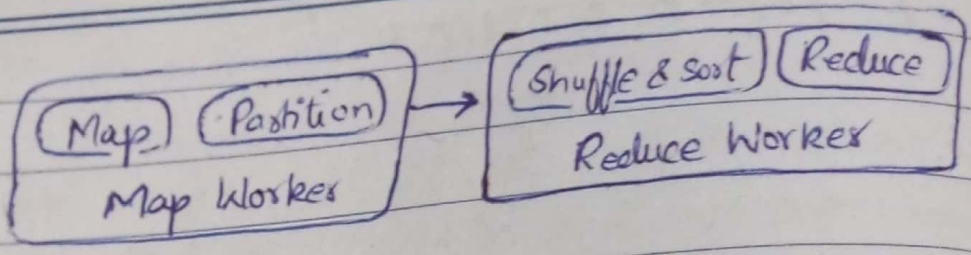
→ Two essential daemons of MapReduce: Job Tracker & Task Tracker
↳ it is like SlaveNode
it is like a Master

→ Map and Reduce work on divide & conquer approach.
↳ divide ↳ combines
↳ divides the data into key & value pairs

"MapReduce Framework"

→ It consist of two key functions
1. Map: Filters and sorts data into intermediate key-val pairs
2. Reduce: Aggregates the intermediate data into smaller key-val pairs or final output.

Date _____ Day _____



- Map:
  - ↳ Grab the relevant data from the source
  - ↳ Parse into key-val
  - ↳ Write it into an itermediate file
- Partition:
  - ↳ Partioning: identify ~~each~~ which of R reducer will hand which keys
  - ↳ Map partitions data to target it to one of R Reduce workers based on a partitioning functio (both R & Partioning function uses defined)

Shuffle & Sort:

↳ Shuffle ~~& Sort~~ : Fetch the relevant partition of the output from all mappers

↳ Sort by keys (different mappers may have sent data with the same key)

- Reduce:
  - ↳ Input is the sorted output of mappers
  - ↳ Call the uses Reduce function per key with the list of values for that key to aggregate the results.

Day

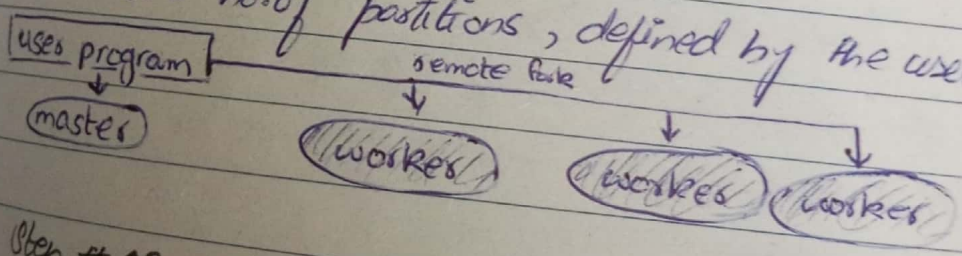
Date

Full Process of the Framework:

- Step #01:
- Split input files into chunks (shards)

shard 0	shard 1	shard 2	.....	shard M-1

- Step #02: Fork Processes
- Start up many copies of the program on a cluster of machines
  - One Master: scheduler & coordinator
  - Lots Workers
- Idle workers are assigned either:
  - map tasks (each works on shard) - there are ~~M Map tasks~~
  - reduce tasks (each works on intermediate files) - there are R
    - R = no. of partitions, defined by the user



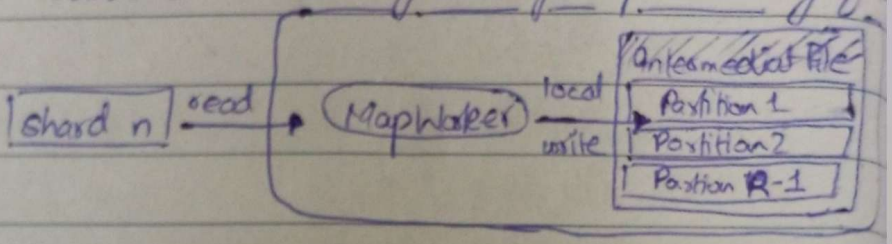
- Step #03: Run Map Tasks
- Reads contents of the input shard assigned to it
- Parses key-val pairs out of input data
- Passes each pair to the user defined map function
  - Produces intermediate key-val pairs

Date _____ Day _____

- These are buffer in memory

Shard 2 → read → Map Workers

- Step # 04: Create Intermediate Files
  - Intermediate key-val pairs produced by the user's map function buffered in memory and are periodically written to the local disk.
    - Partitioned into R regions by a partitioning fun



- Step # 04 (a): Partitioning:
  - Map data will be processed by Reduce workers.
    - User's reduce function will be called once per unique key generated by Map
  - We need to 1st sort all the key-val data by keys and decide which Reduce worker processes which keys
    - The reduce worker will do sorting
  - Partition Function

Decide which of R reduce workers will work on which key

- Default function: hash(key) mod R
- Map workers partitions the data by keys.

Date

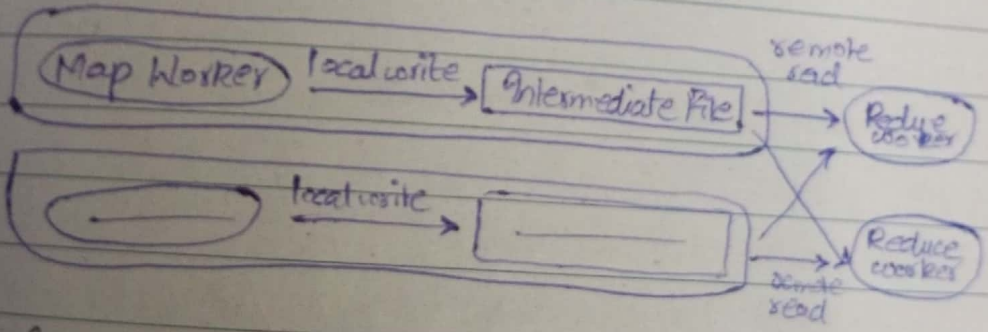
Day

- Each Reduce worker will later read their partition from every map worker.

- Step # 05 : Reduce Task Sorting
  - Reduce workers gets notified by the Master about the location of intermediate files for its partition

Shuffle : Uses RPCs to read the data from the local disks of the map workers.

Sort : When the reduce worker reads intermediate data for its partition
- It sorts the data by the intermediate keys
- All occurences of the same key are grouped together.

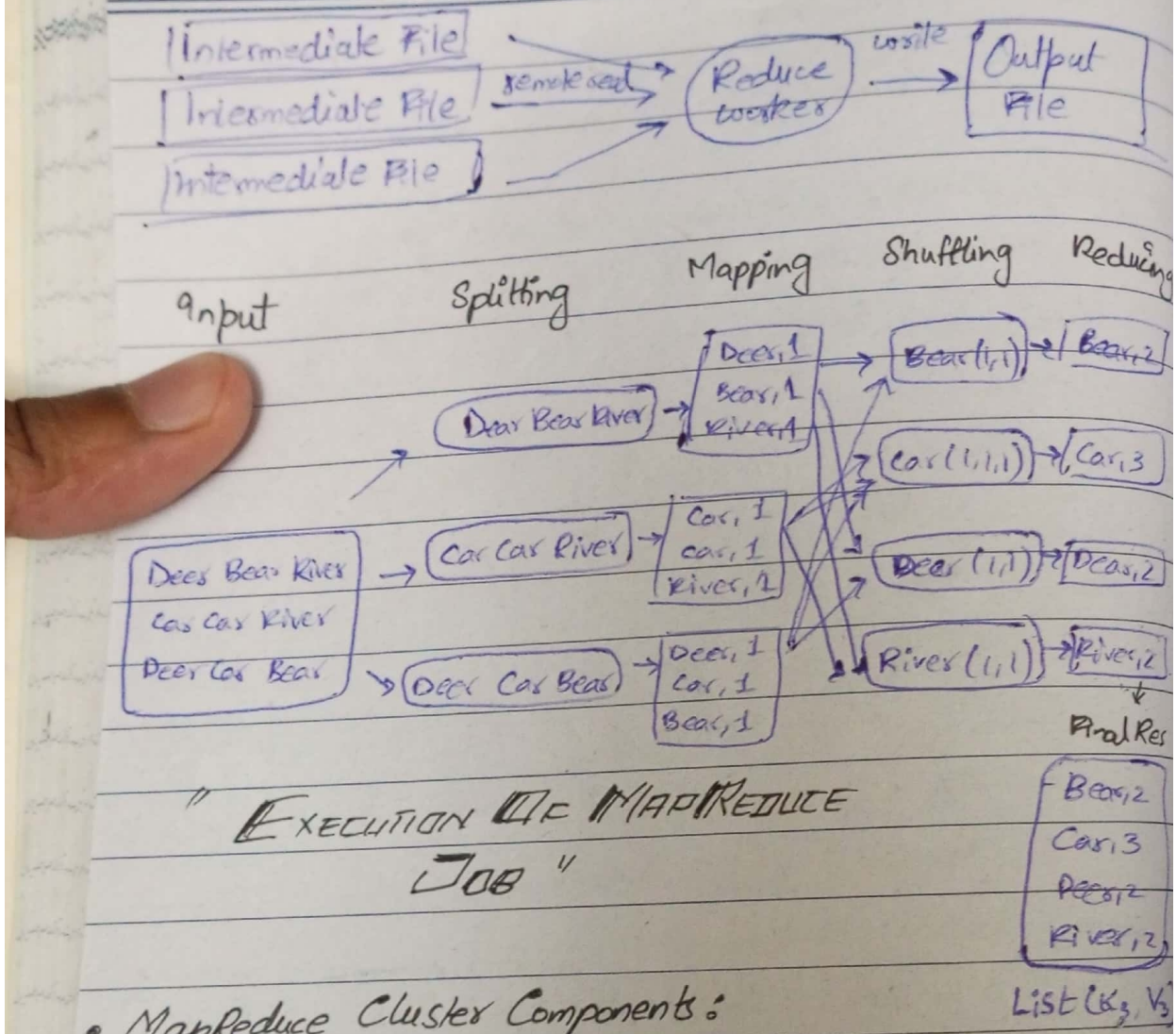


- Step # 06 : Reduce Task - Reduce
  - The sort phase grouped data with a unique intermediate key
  - User's Reduce func is given the key and the set of intermediate values for that key.

< key, (value 1, value 2, ...) >

  - The output is appended to output file.

Date _____ Day _____



"Execution of MapReduce Job"

- MapReduce Cluster Components:

a) Job Tracker (Master Node)

- Serves as the master node in a MapReduce cluster.
- Responsibilities:
  - Initiates and assigns Map and Reduce tasks to worker nodes
  - Monitors task progress and handles task failures.

Date Day

b) Task Tracker (Workers Nodes)

- Each worker node runs a Task Tracker process
- Responsibilities:
  - Communicates with Job tracker to recieve tasks and send status updates.
  - Executes tasks in an isolated environment using Java Virtual Machine (JVM) to ensure process isolation.

- Execution of a MapReduce Job

a) Job Submission:

1. The job tracker recieves the job and reads input data from HDFS
2. The data is split into smaller chunks called splits (or shards)

b) Data Localization: → enhances performance by processing data close to where it is stored.

- The Job Tracker assigns shards to workes nodes, prioritizing data locality - preferably selecting nodes that already hold data.
- Benefits of data localization:
  - Conserves network bandwidth
  - Reduces data transfer time, improving job efficiency.

Date _____ Day _____

c) Task Execution:

- Worker Nodes execute tasks assigned by JT and periodically send progress updates via the task tracker
- Tasks are executed indep by each worker in isolated JVM's to prevent conflicts.

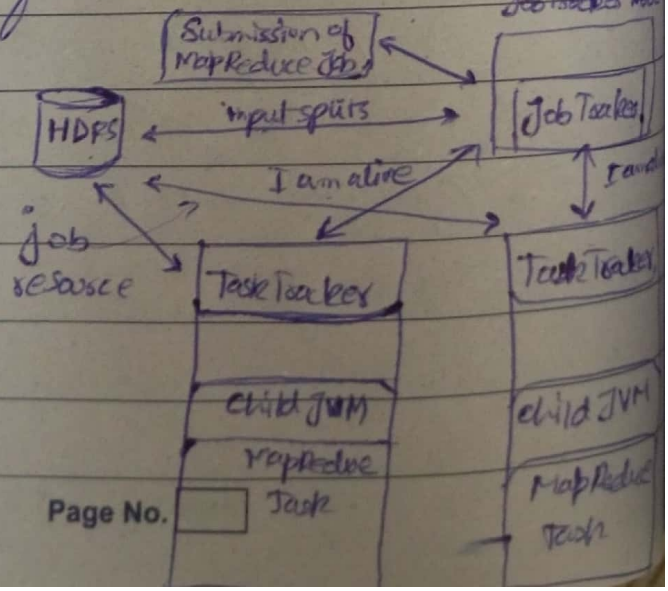
- Staggler Detection and Task Reassignment

a) Staggler Detection

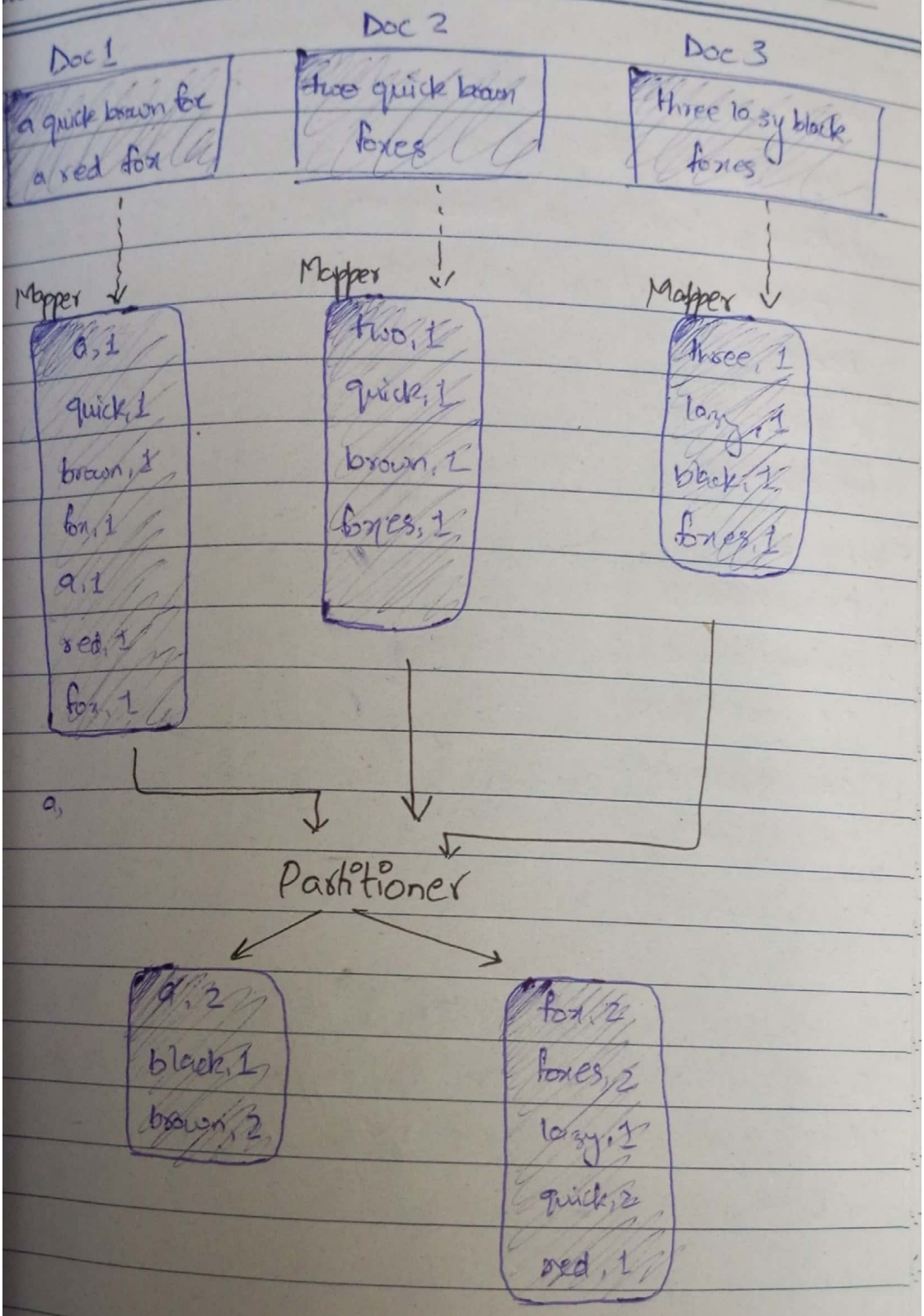
- Worker nodes regulary send updates to the JT
- If no update is received from a Task Tracker within a specified period, the JT assumes the worker has failed.

b) Task Reassignment:

- The job tracker reassigns the task to another avail node, ensuring the task is completed without delays
- This process ensures fault tolerance and maintains job progress.



Day



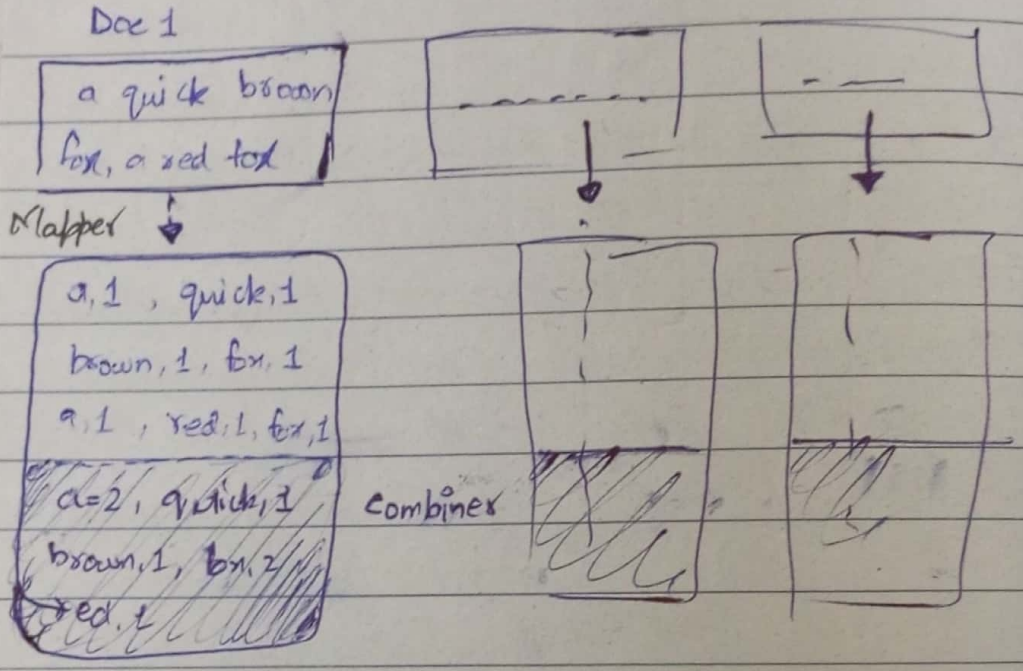
- Partitioner function for MapReduce

Date _____ Day _____

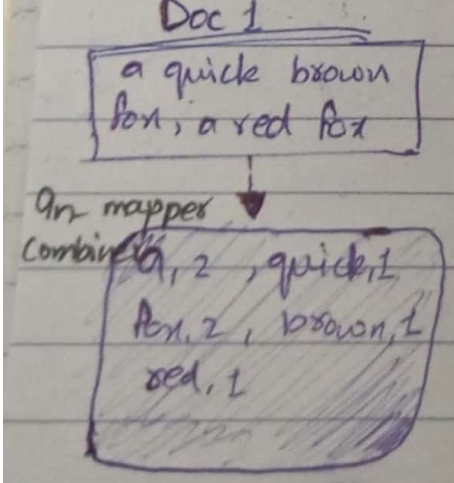
Combiner Function in MapReduce Job :

"The task of combiner func is to perform local reduction. That is reduce values with similar keys, locally at each mapper."

- Executed locally at each mapper



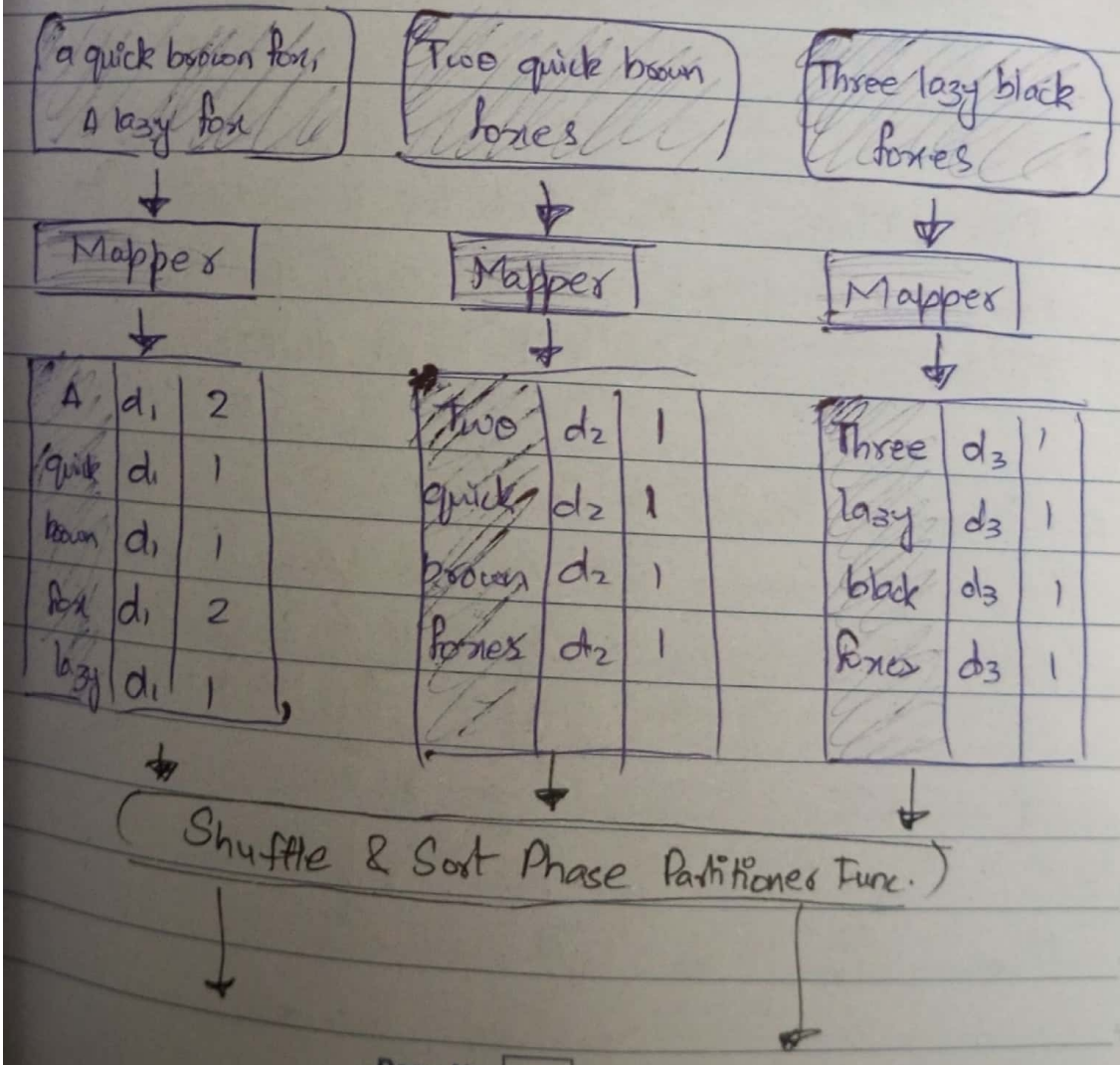
In-Mapper Combiner: It is different than the conventional combiner as the mapper function performs local aggregation before spilling the data.



Date Day

INVERTED INDEX :

- An inverted index is commonly used data structure for MapReduce Analytics. Using this we can make a map of some terms to a list of items.
- It creates an index from the data set to search for items in less time.
- It maps terms (words) to the locations (eg. documents or files) where they appear, enabling efficient full text searches.



Date _____ Day _____

Reducer

A	d_1	2		
black	d_3	1		
brown	d_1	1	d_2	1
fox	d_1	2		
foxes	d_2	1	d_3	2

Reducer

dog	d_1	1	d_3	1
quick	d_1	1	d_2	1
three	d_3	1		
two	d_2	1		

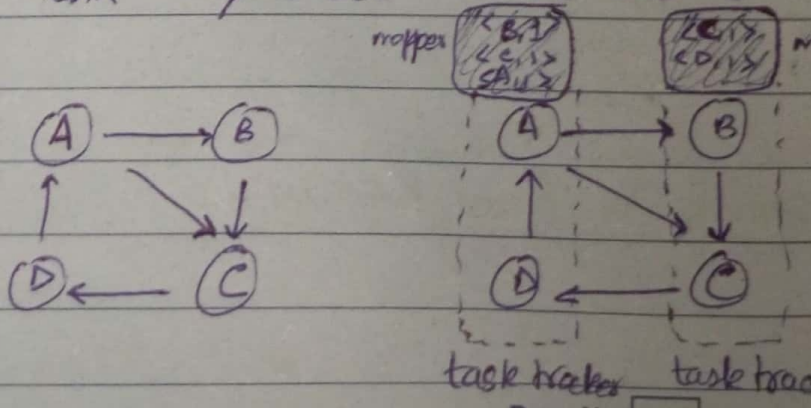
↳ Each term is associated with a list of documents identifies where it appears.

Applications:

1. Google, Bing, Elastic Search use inverted indexes to retrieve web pages matching search queries.
2. For indexing field in text-based datasets.

- Computing Inlinks & Outlink:

The no. of incoming links to a webpage is called inlink. MapReduce can be used to compute inlinks.

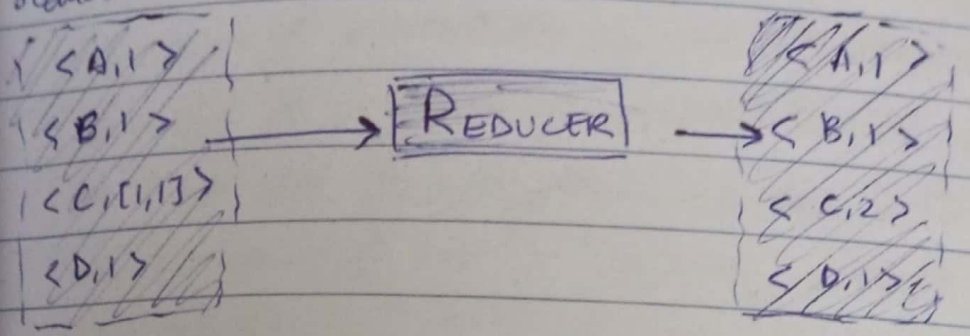


Page No. ☐

Date _____ Day _____

Two mapper are created. Each mapper computes ~~intrinks~~ outlinks for two pages.

After that the <key, value> pairs are sent to the reducer.



▷ MapReduce For Iterative Jobs :

1. Iterative Computation in MapReduce :
   - Iterative algo (eg: PageRank, Clustering) involve repeating a computation process until a convergence point is reached.
   - These algo req the output of a Reducer to be sent back as the input to the mapper in the next iteration.

2. Workflow for Iterative Jobs :
   - Output of the MapReduce phase is written to HDFS.
   - The Mapper in the next iteration reads the Reducer output from HDFS.

Page No. ☐

Date _____ Day _____

3. Inefficiency of MapReduce for Iterative Tasks:

- HDFS Write Overheads:
  - During the Reduce phase, output is written to HDFS, which involves
    - ↳ Transferring data over the network
    - ↳ Writing replicas for fault tolerance.
  - These operations significantly slow down the iterative process.

- HDFS Read Overhead:
  - Each iteration requires reading back data from HDFS, introducing further delays.

4. Result:

① HDFS read/write opes are expensive, as they involve heavy I/O and network usage.

② This repeated read/write make MR unsuitable for iterative computations.

Date Day

MapReduce – Fault Tolerance :

→ WORKER FAILURE

a) Worker Failure Detection :

- The master node pings workers periodically.
- If a worker fails to respond within a certain timeout, it is marked as failed by the Master.

b) Task Rescheduling :

- Map Tasks :
  - Completed map tasks on the failed worker are re-executed, as their output is stored on the local disk of the failed node & is no longer accessible
  - In-progress map tasks are reset to the idle state and rescheduled on other workers.
- Reduce Tasks :
  - In progress reduce tasks are also reset to the idle state and rescheduled
  - "Completed reduce tasks do not req re-exec" as their output is stored in the "global file system" eg: HDFS.

NOTE Page No.

Date _____ Day

→ MASTER FAILURE :

- if the master task fail:
- The system can restart the master from the checkpoint state " if such a mechanism is
- In the absence of checkpointing, the curre " MapReduce computation is aborted " if on single

- Client responsibility :
  : The client can detect master failure and ch to retry the entire MapReduce oper. .

→ SEMANTICS IN PRESENCE OF FAILURE :

a) Atomic Output Commits :

- To ensure fault tolerance, all outputs are ha atomically during map and reduce task comple

b) Output Handling by Map Tasks :

- Each map task writes its output to R private temporary files (one file per reducer).
- On completion the worker sends a message to the master with the names of these temporary file
- If the master recieves duplicate completion messages for the same task, it ignores them.

c) Output Handling by Reduce Task :

- Each reduce task writes to a private tempor file during execution.

ate _____ Day _____

- Once completed, the task atomically renames the temporary file to the final output file.
- If multiple workers execute the same reduce tasks, multiple ~~non~~ rename calls for the same file occur. However the atomic oper. ensures that only one final output file is created.

MapReduce - Data Locality

- Network bandwidth is a limited resource in distributed computing enviornments.
- Conserving bandwidth is critical to ensure efficent data processing and scalability.
- Data Locality:
  The idea is to move the computation to the data rathes than transfessing large amounts of data across the network.
- How it works?
  1. GFS (Google File System)
     - Files are divided into 64MB blocks and replicated 3 times accross diff machines.
     - This replication ensures redundancy and availibility of data.
  2. Sheduling MapTasks:
     - The masterNode uses the file location info from GFS.

NOTE

Date _____ Day _____

- It tries to shedule map tasks:
  - ↳ On a machine with a local copy of the data (Ha
  - ↳ Or near the machine (eg: on the same rack or network switch)

3. Result:
   - A significant portion of input data is read locally avoiding unnecessary network transfers

- Local reads consume "no network bandwidth."

- REFINMENTS:

- PARTITIONING FUNCTION:
  - Purpose:
    - → Divides intermediate data (key:val) into partition corresponding to reduce tasks.
    - → Ensures all data for a specific key or logical group ends up in the same reducer.

- Default Partitioning:
  - → Typically uses hash(key) mod R, where R is the no. of reduce tasks.
- Custom Partitioning:
  - → Users can provide a special partitioning func for specific needs.

Date _____ Day _____

→ Example:
↳ For URLs, using hash(hostname(urlKey)) mod R ensures all URLs from the same host go to the same partition/output file.

- ORDERING GUARANTEES:
  - Within-Partition Ordering:
    → MapReduce ensures intermediate key-val pairs within a partition are processed in increasing key order.
  - Benefits:
    → Enables the generation of sorted output files:
    → Useful for:
      ↳ Formats that require efficient random access lookups by key
      ↳ Users who find sorted data more convinient to process.

- COMBINER FUNCTION:
  - Performs local aggregation of intermediate data on the map worker before transferring it to reducer.
  - Act as a mini-Reduce function on each mapWorker
  - Combines data like key counts <word,1> to reduce the volume sent over the network.
  - Benefits:
    ↳ Reduces network traffic

Date _____ Day _____

↳ Speeds up operations by minimizing data transfer.

- Implementation
  - ↳ The same code is used for the Reduce & Combiner function
  - ↳ Difference
    - ↳ Combiner output is temporary and sent to reducer
    - ↳ Reducer output is final & written to HDFS.

① SIDE EFFECTS:

→ In MapReduce side effects refer to additional outputs like:

- ↳ Logs for debugging
- ↳ Metadata, etc

generated by a task beyond the main key-val output.

- Ensuring Atomicity:

→ Atomic writes prevent partially written or corrupted files from being visible to the system.

→ How it works

- ↳ While side effects to a temporary file
- ↳ Rename the temp file to its final name only after the operation is complete

→ This ensures that only fully generated outputs are visible to other parts of system.

Date _____ Day _____

⇒ Limitations:

1. No-Two Phase Comitt:

Map Reduce doesnot support atomic two phase comitts, means it cannot gurantee consistency between multiple files written by the same task.

2. Deterministic Behaviour Needed:

Task producing multiple outputs must ensure that their execution is deterministic means,

given the same input, the task must produce the same output each time.

This is essential when tasks are retried due to failures.

⇒ SKIPPING BAD RECORDS:

- Some records/data may cause deterministic crashes in user-defined Map or Reduce Functions
- When a task crashes on a specific record:
  - ↳ A signal handler catches the crash and logs the record's sequence no.
  - ↳ The MapReduce master tracks records that repeatedly cause crashes
  - ↳ After multiple failures, the master instructs workers to skip the problematic record.
- Ensures progress even if certain records cannot be processed.

Day

Date

▷ STATUS INFORMATION:

- Master Node's HTTP Server
  - Provides live status pages for monitoring job progress and diagnosing issues, debugging user code.
- Displayed Information
  - ↳ No. of completed & in-progress tasks
  - ↳ Data processed
    - ↳ bytes of input
    - ↳ bytes of intermediate data
    - ↳ bytes of output
    - ↳ processing rates
  - ↳ Worker Failures and tasks they were processing
  - ↳ Links to logs
    - ↳ Standard error files
    - ↳ Standard output files

▷ COUNTERS:

- Provides facility to count occurences of various events (eg: total words processed, no. of failed tasks).
- User define counters are updated in the Map or Reduce func
- Counter values are periodically sent to the master for aggregation.

Date _____ Day _____

- Current counter values are shown on the status page for live monitoring.
- Final aggregated values are returned to the user after job completion.
- Duplicate task executions are handled, ensuring no double counting of counters.

▷ PERFORMANCE

1. CLUSTER CONFIGURATION:

- 1800 machines, each with:
  - Two 2GHz Intel Xeon processor
  - 4 GB memory (1-1.5GB reserved for other tasks)
  - Two 160GB IDE disks
  - Gigabit Ethernet link
- Machines arranged in a two-level tree shaped network with 100-200 Gbps aggregate bandwidth at the root.
- Round-trip time b/w machines < 1 millisecond.
- Data splits into 64MB chunks for input.

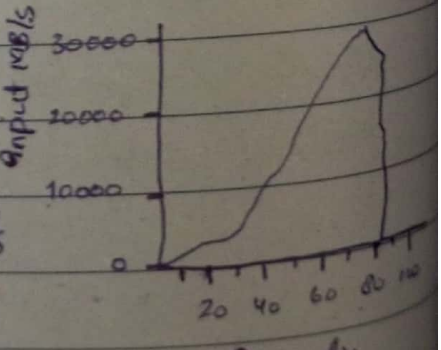
Date _____ Day _____

2. GREP :

- Scan 1 trillion 100 byte records to find a rare three characters pattern.
- Input Data :
  - ↳ Split into 64 MB chunks
  - ↳ 15,000 map tasks & 1 reduce task.
- Output Data : All result stored in 1 file
- EXECUTION OVERVIEW

1. Start up Phase :
   - ↳ takes 60 sec due to
     - ↳ Propagating the program to all worker machines
     - ↳ Interacting with GFS (Google File System)
       - ↳ Opening the set of 1,000 input files
       - ↳ Retrieving information for data locality optimization.

2. Map Phase :
   - ↳ Rate of data scanning increases as more machine (workers) are assigned to the job
   - ↳ Peak Input Rate : Over 30 GB/s with 1,764 workers



Data transfer rate over time

3. Completion Phase :
   - ↳ As map task finishes, the rate of data scanning declines
   - ↳ Hit zero around 80 seconds.

Date _____ Day _____

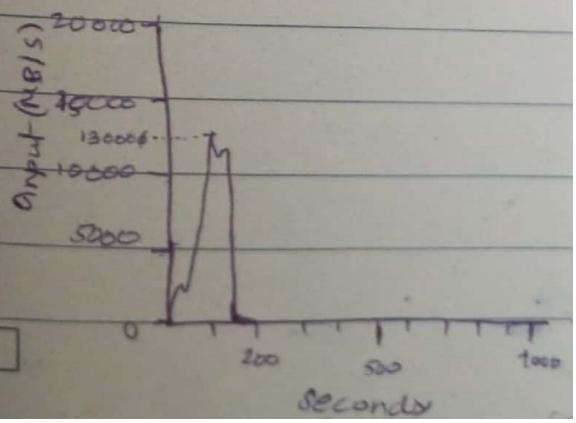
→ Entire computation takes 150 seconds.
  ↳ 60 sec for startup
  ↳ 90 sec for actual map & reduce task execution.

3. SORT :
- Sort 1 billion 100-byte records
- Input Data is divided into 15,000 map tasks each processing 64MB piece
- Output is partitioned int 4,000 files
- Map Function:
  ↳ Extracts a 10-byte key from each text line
  ↳ Emits the key as the intermediate key and entire line as the value
- Reduce Function:
  ↳ Uses an Identity Func., which passes the intermediate key-val pair unchanged as the final output.
- A custom partitioning function segregates data into 4000 partitions based on the initial bytes of the sorting key.

1. Input Read Rate

Peaks at 13 GB/s due to data locality optimization

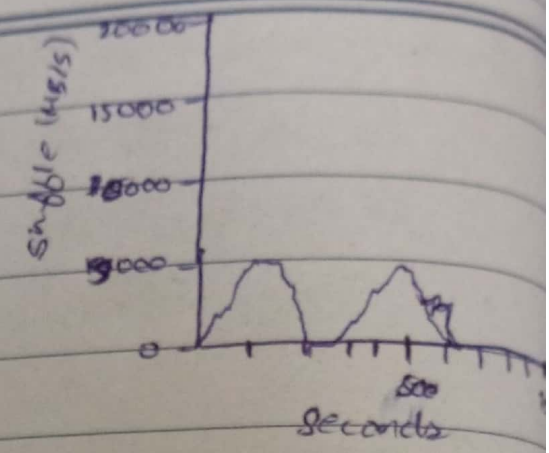
Drops off quickly once all map tasks complete



Date _____ Day _____

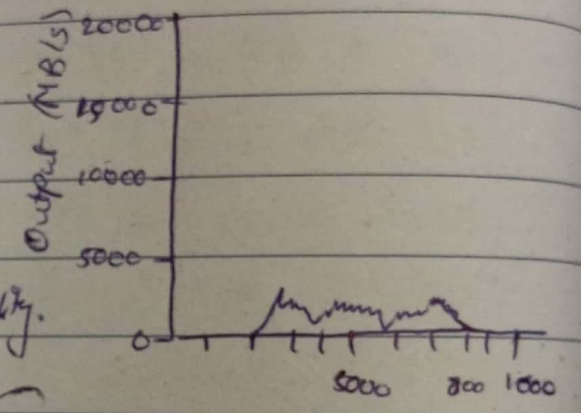
2. Shuffle Rate:

- Data transfer b/w map & reduce begins as soon as the 1st map task completes



3. Output Write Rate

- Lower than Shuffle rate due to replication.
- Two copies of output are written for reliability & availability.



Input Rate > Shuffle Rate > Output rate

↳ due to data locality

↳ It is slow bcz of replication.

Erasure coding instead of replication can solve this problem.

→ Backup task & Machine Failure From Slides!