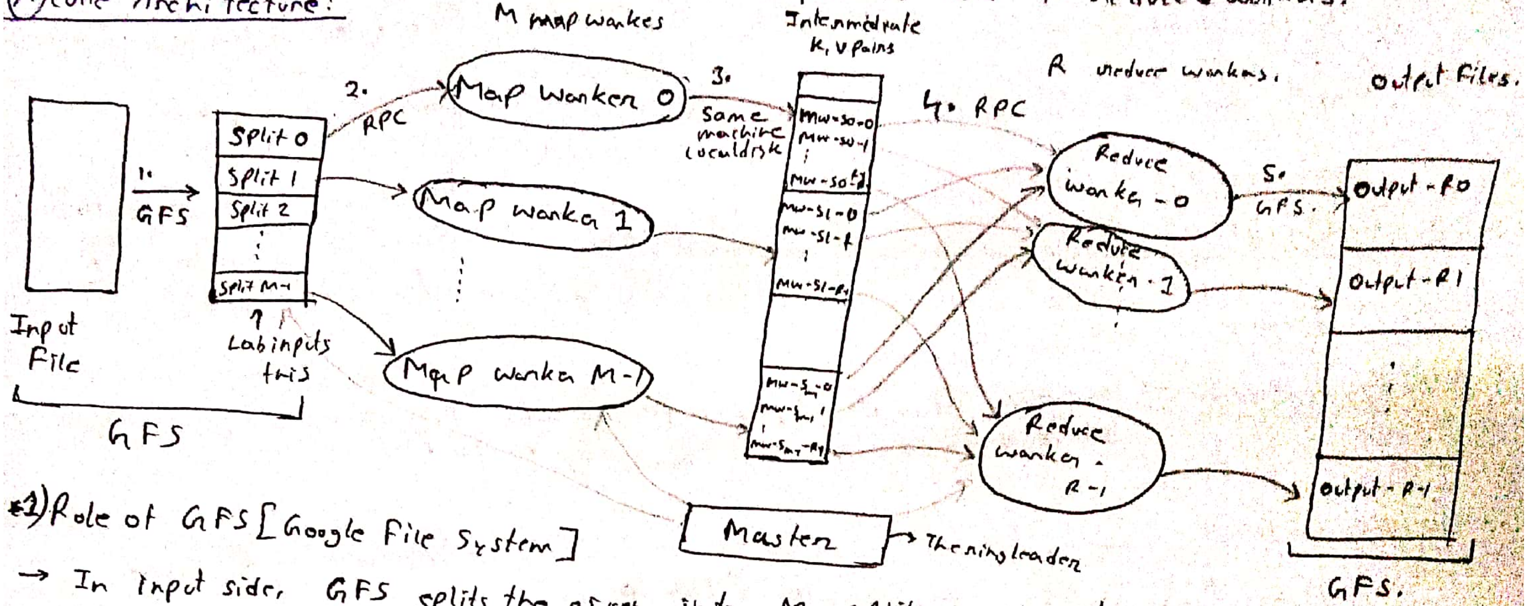
Map Reduce.

→ It is a batch processing model. [All Maps must be done before reduce starts]

→ Used to compute large amounts of data parallely.

(*) Core Architecture:

→ Here there are M mapworkers and R reducer workers.



1) Role of GFS [Google File System]

→ In input side, GFS splits the given into M splits each of size 64MB. It also replicates a split into 3 more machines for fault tolerance.

→ In output side, ~~GFS~~ the ~~map~~ reduce workers writes its output to R separate files manged by GFS [1 output for each Reduce worker]

2) The master knows the location of the splits. When a map worker is idle it gives the map worker a map task. In a map task, the worker take ~~a~~ single input ~~split~~ split parses it and passes it to the ~~k~~ user defined Map function ~~wai~~ where K = { Split Number } not imp V = { The contents of the file }.

3. Each map task produces R ~~out~~ intermediate files. The logic is to use a partition key ~~too~~ Ex: hash(intermediate key) % R → $\boxed{0}$ → goes to mw-s0-$\boxed{0}$ / → $\boxed{2}$ → goes to mw-s0-$\boxed{2}$

~~The files~~ The key is not written to file everytime as it is inefficient insted it is written to buffers and buffers are periodically ~~wa~~ flushed to files. It then informs the master the location of R intermediate output files. { The file is on the same computer where the worker runs.

4. The master informs the reduce workers about the locations so they can start working on it. Ex: Since it is already partitoned, Reduce worker 0 take the ~~item~~ intermediate files mw-*-0.txt and process it. Then it ~~scans~~ sorts the data ~~using stream map~~ and user given reduce fn is run where for each key k.1 value is the cumulative sum ~~for~~ of all intermediate values. [for word counting]

5. The output files from Reduce workers are initially temporary files.
Example ~~[illegible]~~ output-0-temp.txt and when processing is completed it is renamed to output-0.txt.

* Master Data Structure:

→ For Each map & reduce task, Stores the
1. State (idle, inprogress, completed)
2. Identity of worker [for non idle tasks]
3. For completed map tasks stores the Location of R intermediate files.

→ we need to detect which task failed & reschedule it.

* Fault Tolerance:

→ Master failure: → we can do checkpoints periodically but since this is only one ~~task~~ process we can manually restart it.

→ worker failure:
↳ Master Pings worker periodically [Lab 05] → Response → All Good.
↳ If no response is received from a worker, ~~master~~ worker fails.

→ The map tasks completed or inprogress by the worker are reset to 'initial' state and are eligible for rescheduling on other worker.

→ The reduce tasks that are in progress by the worker is reset to initial state and is eligible for rescheduling.

Why: Completed map tasks are stored in same machine' [local disk] as the worker so we need to redo that since disk is inaccessible.
Completed reduce tasks do not need rescheduling since the output is on file system.

→ When a map task is executed first by worker A and later by worker B [cuz A failed] all workers executing the reduce tasks are notified of re execution. Any reduce task that has not already read the data from worker A will now read from worker B.

→ Our system should produce same output as would have been produced by a sequential execution.

→ we rely on atomic commits to achieve this property.

→ Each in progress task writes to its temp files. A map task produces R such files and Reduce tasks one such file.
(per reduce task.)

→ when map task completes it sends its ~~message~~ R output file location & name. to the master. If master ~~when~~ receives completion message for a already complete task, ~~th~~ it ignores the message else it stores the location in its DS.

→ When a reduce task is complete, the worker renames its temporary output to final output file. [Rename is atomic]

* How Network Bandwidth problem is solved?

→ In 1 we said that GFS stores 3 copies for each split so we run our map task on the machines where the copies are present or if it fails, on same network. This minimizes the Network Bandwidth. [Read is local]

* Task Granularity :

→ # of Map task & # of Reduce task should be larger than # of worker machines.

why : i) Faster worker perform more tasks, preventing idle state.
ii) If a worker fails, its tasks can be spread across all other workers, speeding up re execution.

* StragglerProblem:

→ when the job is almost done, schedule "backups" copies of remaining ~~in-progr~~ in-progress tasks. Mark the task done when either the primary or backup finishes.

→ Stragglers (machines with bad disk or high CPU load) can delay completion so backup reduces completion time.

* ImplementationNote:

We have written master assigns task, which is push model → Hard to do
Instead we will do pull model ~~[illegible]~~ workers ask for task periodically → By RPC.