

Execution overview

* User program tells master it wants to run a map reduce job
* Master assign workers based on where the files are stored
* Apply map functions to the file chunks (store results on local disk)
* Call the user reduce function per key with the list of values for that key to aggregate the results

Detail Fault Tolerance

Worker failure

Master detect failure periodically

Re-executre Map tasks

Re-execute in progress Reduce tasks

Master failure

Single master->unlikely

Abort

Detail ---Locality

Network bandwidths is a scare resource

Run on GFS(64 MB blocks, several replica)

Map tasks scheduled so GFS input block replica are on same machine or same rack

Detail --- Combine function

Network bandwidth is a scare resource

Word Counting example

Hundreds or thousands of records of the form<the, 1>

Merging the data before sent over the network <the, 100>

Detail--- Task Grandularity

How many Maps? How many Reduces?

The more, the better

Minimizes time for fault recovery

Can pipeline shuffling with map execution

Dynamic load balancing

In practice

Choose Map : task 16 MB – 64 MB (GFS block size)

Choose Reduce : a small multiple of the number of worker

200,000 map /5000 reduce tasks w/ 2000 machines

Detail---Backup tasks

“Staggler”--- slow workers

Bad disk

Other jobs consulting resources

Weird things : cache disabled ?

Solution : Near end of phase, backup tasks

Whichever one finishes first “wins”

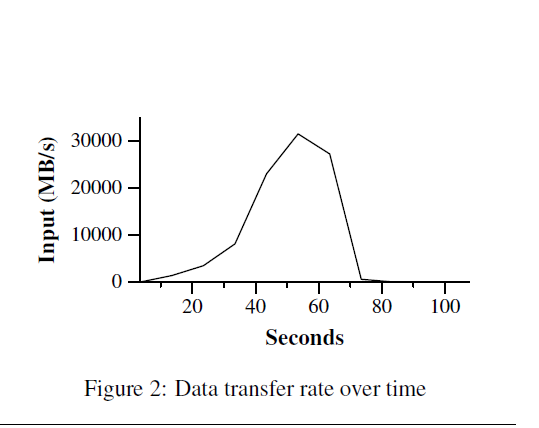
Detail- Skipping Bad Records

Records cause deterministic crashes

Best solutions to debug & fix, but not always possible

Solution : Detect and skip

If master sees two failures for same record



MR\_Grep

Locality optimization helps:

1800 machines read 1 TB of data at peak of ~31 GB / s

Without this, rack switches would limit to 10 GB / s

Startup overhead is significant for short jobs

Propagation of the program to all worker machine

MR\_Sort

Backup tasks

Failures

