MapReduce: Simplified Data Processing on Large Clusters

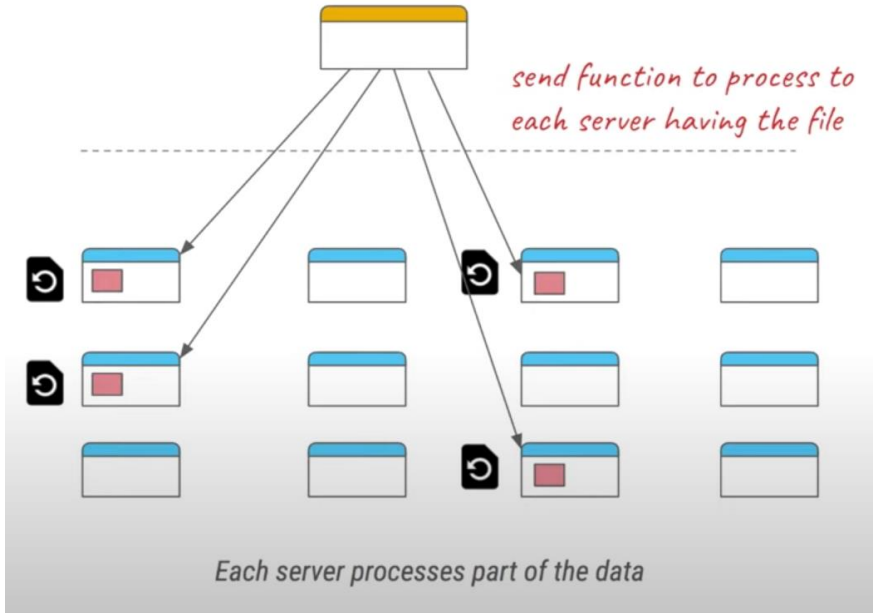
Jeff Dean, Sanjay Ghemawat
Google, Inc.

Presented by Wenyan Li and Hao Feng

Motivation

- Problems
  - High network bandwidth
  - Multi-TB files(s)
  - Slow to process

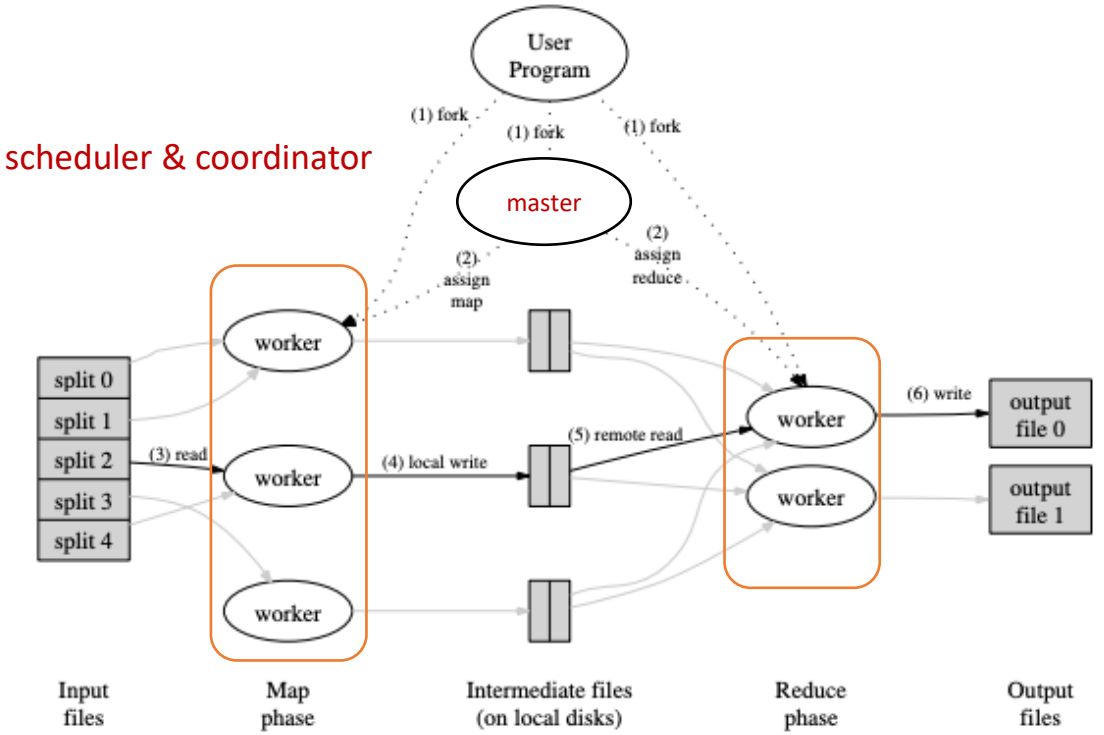
- MapReduce provides:
  - Automatic parallelization and distribution
  - Simple API for programmers
  - Fault-tolerance
  - I/O scheduling
  - Status and monitoring



img source:https://www.youtube.com/watch?v=MAJ0aW5g17c

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



Programming model

Map = processing part of data

Map (in_key, in_value) -> list(out_key, intermediate_value)

Input key/value pair

Produces set of intermediate pairs

Reduce= Aggregation

Reduce (out_key, list(intermediate_value)) -> list(out_value)

Combines all intermediate values for a particular key

Produce merged output value

Example --- count word

Input words

“YouTube”
“Netflix”
“CNN”
“NBC”
“NBC”
“Netflix”
….

Map (word, 1)

Parse data
output each word and a count (1)

Map worker-1
- “YouTube”: 1
- “Netflix”:1
- “CNN”:1
-“NBC”:1

Map worker-2
- “NBC”: 1
- “Netflix”:1

Reduce (word, total_count)

Sort: sort by keys (words)
Reduce: Sum together counts each key (word)

Reduce worker-1
- “YouTube”: 1
- “Netflix”:2

Reduce worker-2
- “CNN”: 1
- “NBC”:2

Detail --- Fault Tolerance

- Worker failure
  - Master detect failure periodically
  - Re-execute Map tasks
  - Re-execute in progress Reduce tasks
- Master failure
  - Single master -> Unlikely
  - Abort

Detail --- Locality

- Network bandwidth is a scarce resource
- Runs on GFS (64MB blocks, several replica)
- Map tasks scheduled so GFS input block replica are on same machine or same rack

Detail --- Combiner function

- Network bandwidth is a scarce resource
- Word counting example
  - Hundreds or thousands of records of the form <the, 1>
  - Merging the data before sent over the network <the, 100>

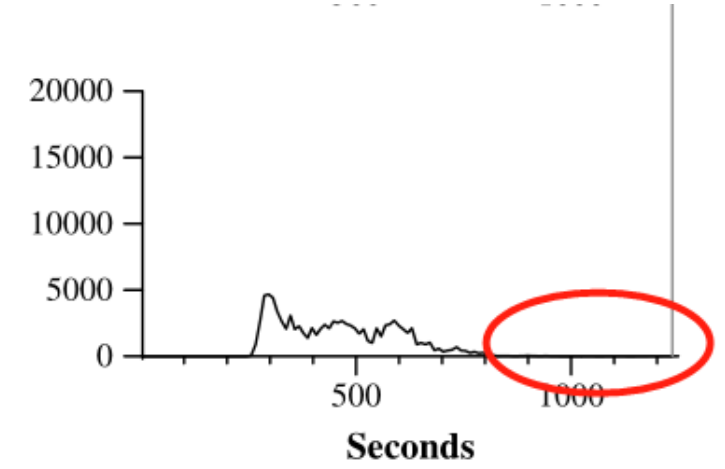
Detail --- Task Granularity

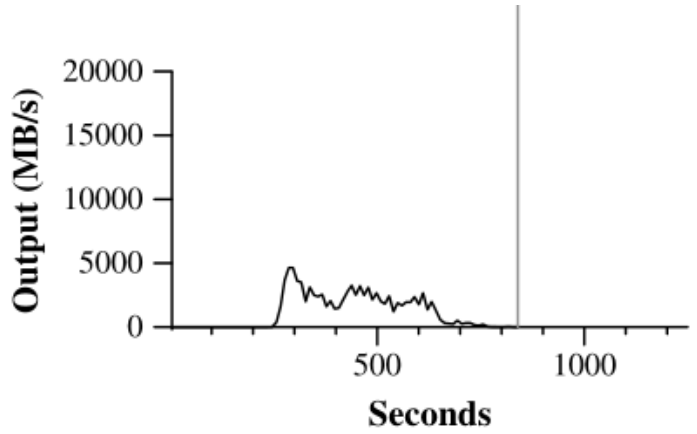
- How many Maps? How many Reduces?
- The more, the better
  - Minimizes time for fault recovery
  - Can pipeline shuffling with map execution
  - Dynamic load balancing

- In practice
  - Choose Map: task 16MB – 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

- “Straggler” --- slow workers
  - Bad disk
  - Other jobs consuming resources
  - Weird things: cache disabled ?
- Solution: Near end of phase, backup tasks
  - Whichever one finishes first "wins“





Detail --- Skipping Bad Records

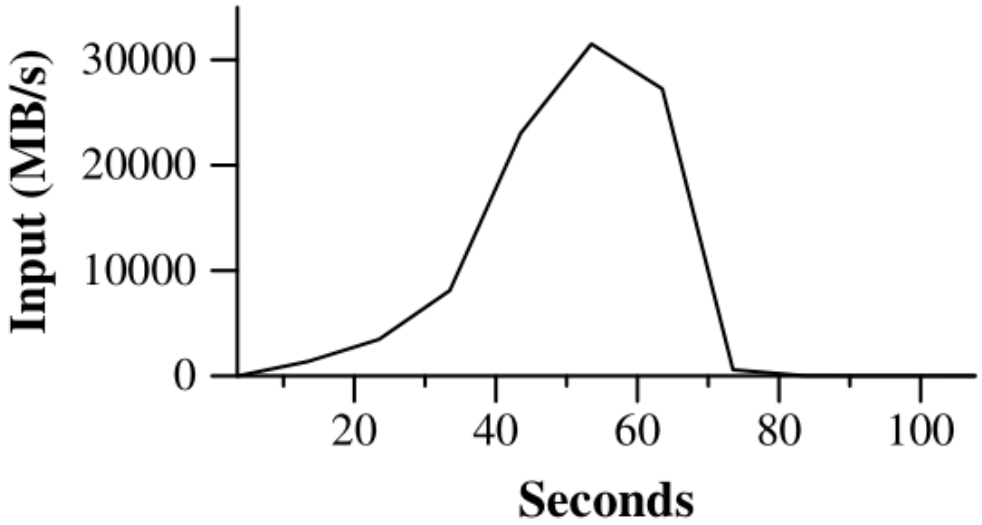
- Records cause deterministic crashes
  - Best solution is to debug & fix, but not always possible

- Solution: Detect and skip
  - If master sees two failures for same record

Experiments

- Grep
  - Scan 10^{10} 100-byte records to extract records matching a rare pattern (92K matching records)
- Sort
  - Sort 10^{10} 100-byte records (TB)

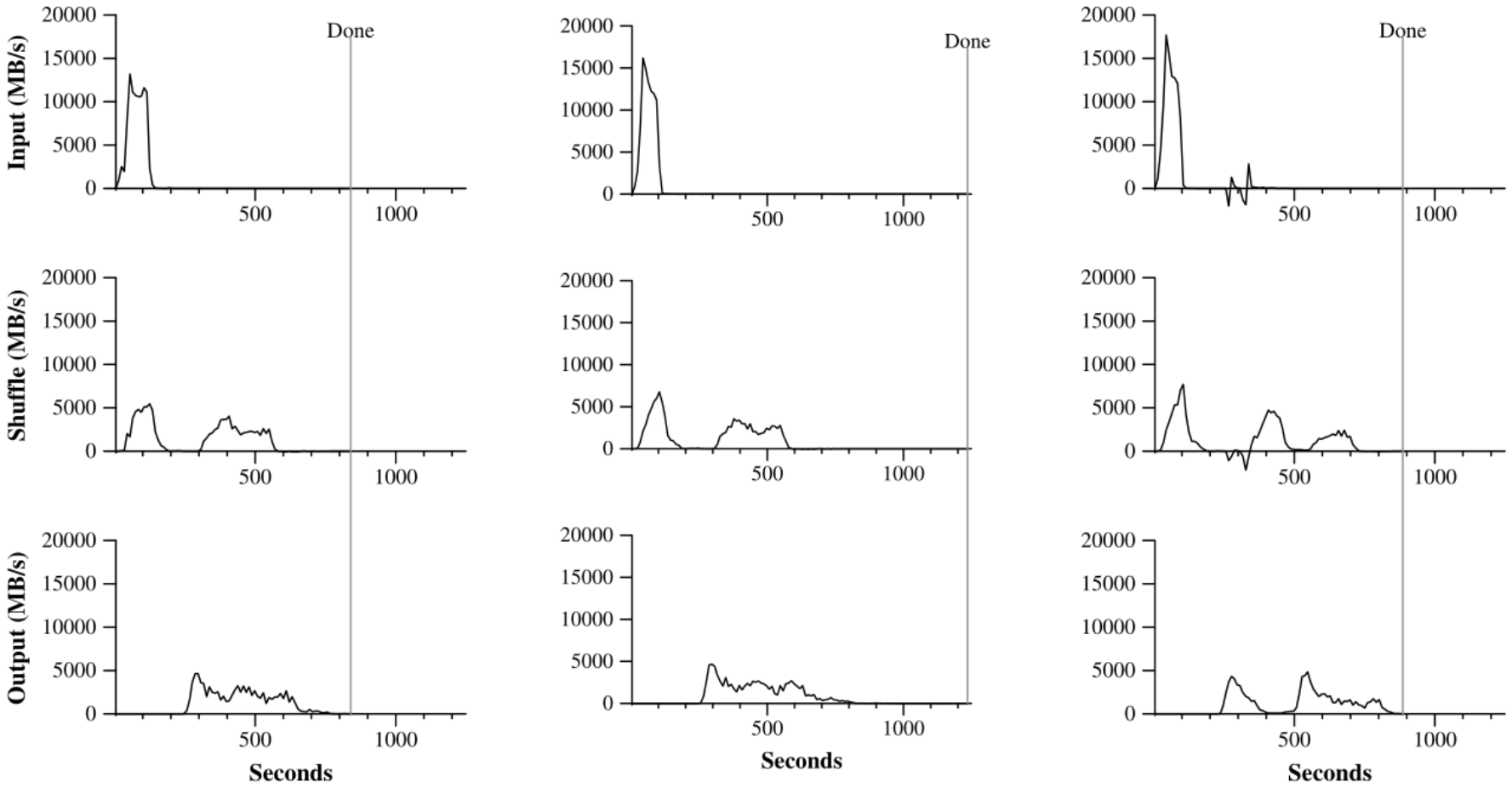
MR_Grep



- 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 machines

MR_Sort

- Backup tasks
- Failures



(a) Normal execution

(b) No backup tasks

(c) 200 tasks killed

Authors' Conclusions

- restricting the programming model makes it easy
- network bandwidth is a scarce resource
- redundant execution for slow machines, failures and data loss.