Motivation

- Large-Scale Data Processing (petabytes of data)
    - E.g., build search index, or sort, or analyze structure of web
- Want seamless scalability: scale "out", not "up"
    - Large cluster of commodity PCs vs small cluster of high-end servers
    - Scale to thousands of CPU cores on clusters of commodity servers
    - The power of scaling out:
        - 1 HDD: 80 MB/sec
        - 1000 HDDs: 80 GB/sec
        - The Web: 20 billion web pages x 20KB ≈ 400 TB
            - 1 HDD: 2 months to read the web
            - 1000 HDDs: 1.5 hours to read the web
- Applications not written by distributed systems experts
    - Distributing the computations, fault tolerance, recovery, etc.
- Failures:
    - Failures are very common due to the scale
    - One server may stay up 3 years (approx. 1000 days)
    - For 1000 servers: 1 fail per day
    - Google had 1M machines in 2011: 1000 fails per day!
- Sharing a global state is difficult: synchronization, deadlocks

Map-Reduce

- System for automatic parallelizing the computation (batch) of large volume of data across multiple machines
- Provides an API for expressing computations using two operations: Map and Reduce
    - The Map task takes an input file and outputs a set of intermediate (key, value) pairs
    - The intermediate values with the same key are then grouped together and processed in the Reduce task for each distinct key
- Designed to run on clusters of commodity hardware
- Balancing load over servers
    - Full scans of datasets stored in GFS
    - Huge files (100s of GB to TB)
    - Data is rarely updated in place
    - Reads and appends are common
- Scalability
        - n "worker" computers get you nx throughput
            - Maps()s can run in parallel, since they don't interact
            - Same for Reduce()s

Note: There is a shuffle and sort phase in between map and reduce

- The programming model is inspired by functional programming languages
    - Makes easy to distribute computations across nodes
    - Many data parallel problems can be phrased as "embarrassingly" parallel problems
- Transparency
    - Sending application code to servers
    - Tracking which tasks are done
    - Moving data from Maps to Reduces
- Provides automatic fault tolerance

Execution overview

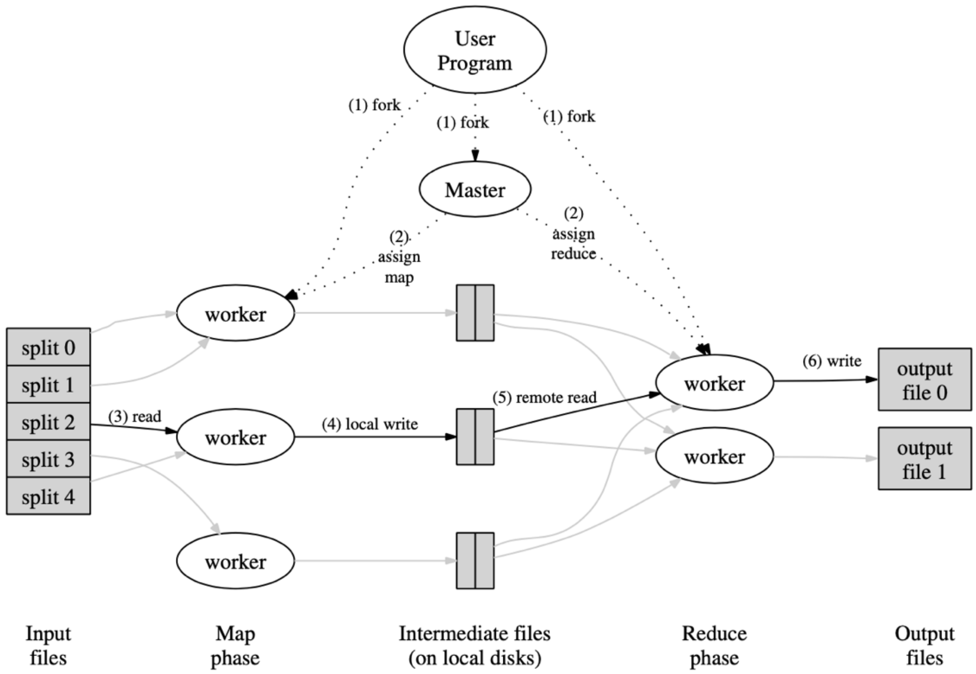


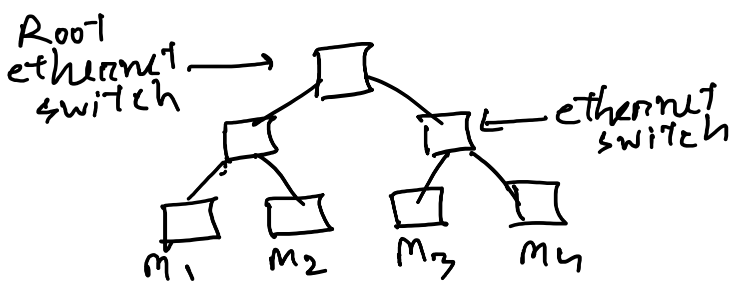
Figure 1: Execution overview

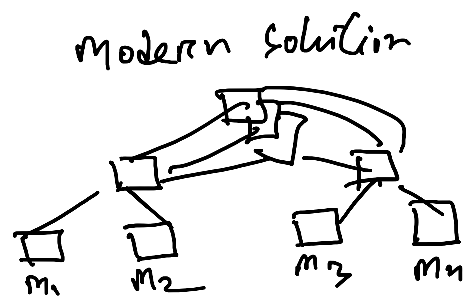
- The user program forks a coordinator process and some number of worker processes at different compute nodes
    - Usually, a worker handles either map tasks (a map worker) or reduce tasks (a reduce worker), but not both
- The coordinator responsibilities include:
    - Create some number of map tasks and some number of reduce tasks
    - These tasks will be assigned to worker processes by the coordinator
    - The coordinator keeps track of the status of each map and reduce task (idle, executing at a particular worker, or completed)
- A worker process reports to the coordinator when it finishes a task, and a new task is scheduled by the coordinator for that worker process
    - Each map task is assigned one or more chunks of the input file(s) and executes the code written by the user
    - The map task creates a file for each reduce task on the local disk of the worker
    - The coordinator is informed of the location and sizes of each of these files
    - When a reduce task is assigned by the coordinator to a worker process, that task is given all the files that form its input
    - The reduce task executes code written by the user and writes its output to a distributed file system

Issues

- No interaction or state (other than via intermediate output)
- Iteration, multi-stage pipelines are slow
- No real-time or streaming processing
- In 2004, authors were limited by network capacity

    - Maps read input from GFS
    - Reducer read Map output
        - Can be as large as input
        - Reducers write output files to GFS





    - In MR's all-to-all shuffle, *half of traffic* goes through root switch
    - Paper's root switch: 100 to 200 gigabits/second, total
        - 1800 machines, so 55 megabits/second/machine
        - 55 is small, e.g., much less than disk or RAM speed
- Today: networks and root switches are much faster relative to CPU/disk

Network usage:

- Master tries to run each Map task on GFS server that stores its input
    - All computers run both GFS and MR workers
    - So input is read from local disk (via GFS), not over network
- Intermediate data goes over network just once
    - Map worker writes to local disk
    - Reduce workers read directly from Map workers, not via GFS
- Intermediate data partitioned into files holding many keys

Load balance

- Wasteful and slow if $n-1$ servers must wait for 1 slow server to finish
- But some tasks likely take longer than others

Solution:

- Many more tasks than workers
- Coordinator hands out new tasks to workers who finish previous tasks
    - So, no task is so big it dominates completion time (hopefully)
    - So, faster servers do more tasks than slower ones, finish about the same time

Fault Tolerance and Crash Recovery

- What if the coordinator crashes?
    - As coordinator failures were rare, their implementation simply aborted the whole execution if coordinator failed, for the execution to be retried from the start
- Map worker crashes
    - Coordinator notices worker no longer responds to pings

    - Coordinator knows which Map tasks it ran on that worker
        - Those tasks' intermediate output is now lost
            - Must be recreated
        - Coordinator tells other workers to run those tasks
    - Can omit re-running if Reducer already fetched the intermediate data
    - What if the Coordinator gives two workers the same Map() task? Perhaps the Coordinator incorrectly thinks one worker died
        - It will tell Reduce workers about only one of them
- Reduce worker crashes
    - *Finished tasks are OK*
        - *stored in GFS*, with replicas
    - Coordinator re-starts worker's unfinished tasks on other workers
    - What if the Coordinator gives two workers the same Reduce() task?
        - They will both try to write the same output file on GFS!
        - *Atomic GFS rename* prevents mixing; one complete file will be visible

Other failures/problems

- What if a single worker is very slow
    - a "straggler"?
    - Perhaps due to flakey hardware
    - Coordinator starts a second copy of last few tasks
    - When a MapReduce operation is close to completion, the master schedules backup executions of the remaining in-progress tasks
- What if a worker computes incorrect output, due to broken h/w or s/w?
    - MR assumes "fail-stop"(after a node crashes, it never recovers) CPUs and software

Optional Read:

- "Simplified Data Processing on Large Clusters", by Dean and Ghemawat
    - https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf