Map Reduce Program Model

Problem with Processing and Analyzing Big Data with typical programming model

* Lets say we want to find wordcount of 10 billion documents
* Each doc size is about 20kb -> 200TB
* Typical solution is to iterate through each document and count each word
* Will take FOREVERRRRRR

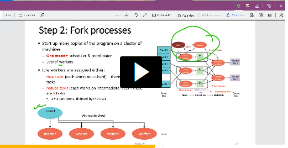
Traditional programming is serial.

Parallel programming breaks processing into parts that can be executed concurrently on mult processors

We want to identify tasks that can run concurrently and groups of data that can be processed concurrently (not all problems can be parallelized)

MAP REDUCE IN BIG DATA CONTEXT

* Framework for parallel computing
* Programmers get simple API
* Don’t have to worry about parallelization, data distr, load balancing, fault tolerance
* Can process HUGE amounts of data (terabytes and petabytes) on thousands of processors

Step 1: Split input file into chunks (shards) break data into M pieces (65 mb typical)

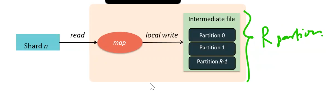
Step 2: Start up many copies of program on cluster of machines

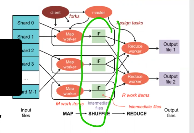
* Have one master : scheduler and coordinator, and lots of workers
* Idle workers assigned either map tasks (each workds on a shard) or reduce tasks( each work on intermediate files)
* R=#partitions defined by user

Step 3: Run Map Tasks

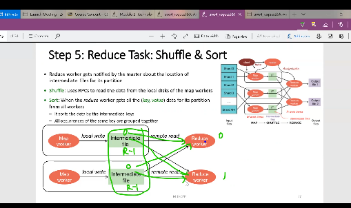
* Read content of input shard assigned to it
* Parse key/value pairs out of input data
* Pass each pair to user defined map function
  + Produces intermediate key/val pairs
  + Buffered in memory
* Map workers run map tasks

Step 4: Create intermediate Files

* Intermediate key/val pairs produced by user map function buffered in memory and periodically written to local disk
* Notifies master when complete
  + Passes locations of intermediate data to master, master forward to a reduce worker
* So the map workers partition their output from 0 to R-1 partitions where R is #of reduce workers

Step 4a: Partitioning

* Map data processed by Reduce Workers
  + User reduce function called once per unique key gen by map
* First group all key value data by keys and decide which reduce worker processes which keys
* Partition Function
  + Decides which of R reduce workers will work on which key
    - DEFAULT: hash(key)%R
    - Map worker partitions data by keys

Step 5: reduce Task: Shuffle and Sort

* Reduce worker gets notified by master about location of intermediate files for its partition
* Shuffle – Uses RPCs to read data from local disk of map workers
* Sort- when reduce worker gets all (key,value) data for partition from all workers
  + Will sort data by intermediate keys
  + All occurrences of same key grouped

Step 6: Reduce Task

* Sort phase grouped data by keys
* User reduce function given the key and set of intermediate values for key. Output of reduce function appended to output file

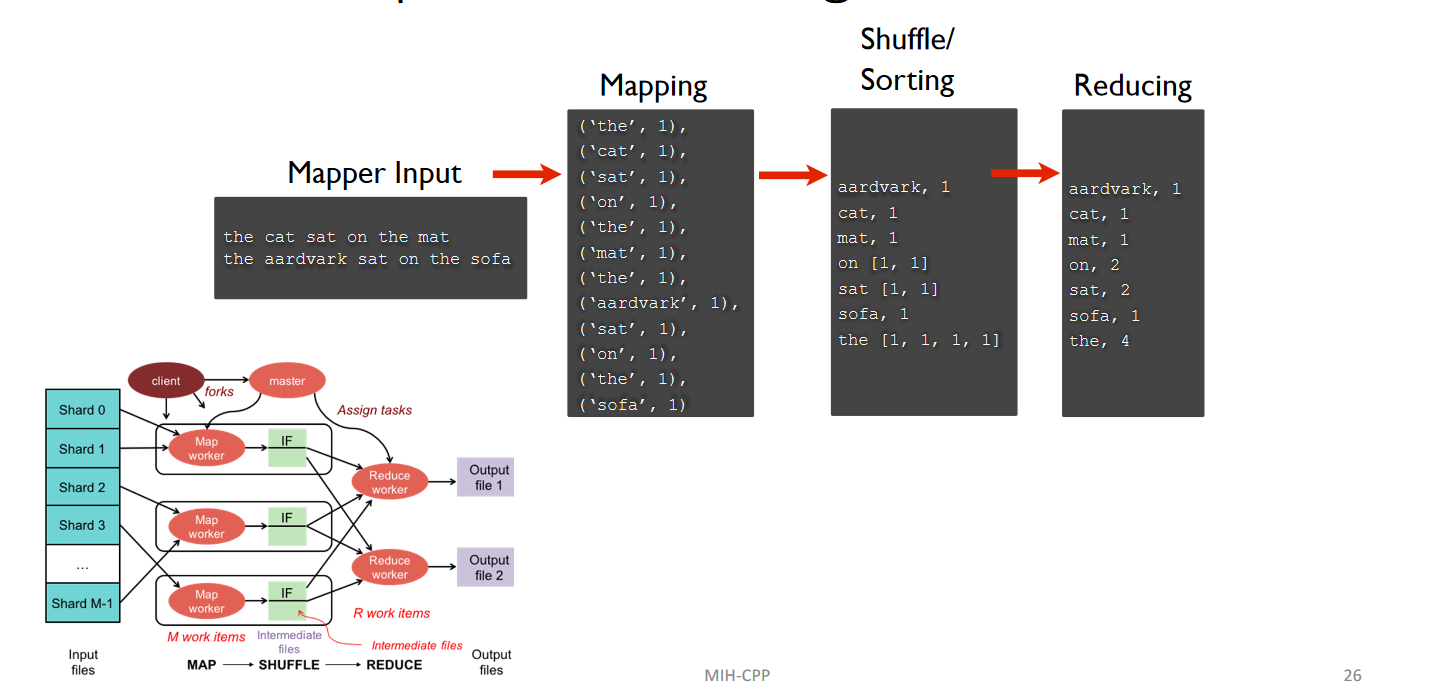
Step 7: Return to User

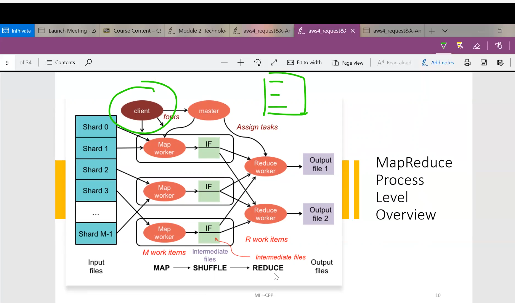
When map and reduce task finished, master wakes up user program

* MapReduce call in user P returns and P can resume exe
* Output of MapReduce is aavailable in R Output files

GREAT for word count for example:

* Word count hard over MASSIVE amounts of data
* MAP REDUCE AT A GLANCE





What happens when a client runs a map reduce program

So the map reduce program that you run, essentially gets broken down into smaller chunks that eventually all report back to the main parent process.

Part of the code gets copied and ran individually. So this pic goes from process level how it works

The copies keep working on map operation, some on reduce, some does operation on intermediate files…. All these things happen in parallel because at beginning we supply each of them with dif parts of input. SO huge file with 1 trillion values, can decide which part send to which… (32:00)

39:33