# Engineering Cloud Computing

MapReduce

* MapReduce model is designed to handle two applications-
  + Word count
  + Inverted Index
* User interacts only with the master node by configuring the type of application the MapReduce should handle.
* Based on the type of application and number of mappers to be spawned, master partitions the input data and stores it in key-value server (datastore).
* Once the input data is stored, master spawns the mapper which reads the data from the key-value store and spawns the mapper tasks as a separate process.
* Each mapper tasks extracts the words from the input and does the necessary operations and writes their output to the key-value store.
* Once all mappers are done, master then shuffles the mapper output in the key-value store based on the number of reducer tasks to be spawned. It is implemented as an additional functionality within the key-value store.
* After the completion of shuffle task, master spawns the reducer which in turn spawn separate processes for each input from the key-value store.
* Finally, after completion of all the reducer tasks the master is notified.
* When the master receives the notification from the reducer, it reads the output of the reducer tasks from the key-value store and writes it on to the user specified location.

MapReduce for Word-Count:

* For word count example, MapReduce is designed with 10 mappers and 10 reducers. Master splits the input data into 10 parts and stores it within the key-value store.
* Once stored, master spawns the mapper which reads the data from the key value store and spawns 10 separate processes which appends 1 next to each word they receive and write back into the key-value store. Once all processes are completed, master is notified.
* As master receives mapper confirmation, it invokes shuffle task, in which same words are grouped together and clubbed into 10 groups in order to spawn 10 reducer processes and written back to the key-value store.
* After shuffling master invokes the reducer, which spawns separate processes corresponding to each input it receives and counts the number of occurrences of each word within the input and append the total count next to each unique word and writes them back to the key-value store.
* Once all reducer tasks are completed, master gets the output from the key-value store and writes them to the user specified location and moves key value store file to a different file in order to facilitate for the next inverted-index task.

MapReduce for inverted index:

* For inverted-index example MapReduce is designed to operate with 3 mappers and 2 reducers.
* Master splits the input data from 3 different files to 3 parts and stores within the key-value store.
* Once input is stored, mapper is spawned by the master similar to the word-count example, which creates 3 separate processes corresponding to each reduce task.
* Filename is stored in ‘document\_pid’ format for simplicity.
* Each map tasks joins words with their filenames & counts and stores it in key-value store.
* After completion of map tasks, master invokes shuffle task which combines same words within different files along with their corresponding number of occurrences in each file and stores it back to the key-value store in group of 2 in order to create 2 reducers in the future and notifies back to the master.
* Master then spawns reducer which gets the input from the store and simply writes them back to the store in 2 separate processes and notifies the master.
* Master then reads the output from the store and writes it onto the user specified location and replaces store file with different name (backup\_datastore.csv) to facilitate next MapReduce tasks.