III. HADOOP MAPREDUCE

A. What is HadoopMapReduce

Hadoop MapReduce is a software framework for distributed processing of large data sets on compute clusters of commodity hardware. Hadoop consists of two components, the Hadoop Distributed File System (HDFS) and MapReduce, performing distributed processing by single-master and multiple-slave servers. MapReduce is the heart of Hadoop®. It is this programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster. The MapReduce concept is fairly simple to understand for those who are familiar with clustered scale-out data processing solutions [24].There are two elements of MapReduce, namely JobTracker and TaskTracker, and two elements of HDFS, namely DataNode and NameNode [25]. Fig. 2 shows the MapReduce framework which includes some sub-categories as the following [26]: (1) Map Processing: HDFS divides the massive input data set into minor data blocks (64 MB by default) controlled using the property dfs.block.size, (2) Spill: When the buffer size arrives to a threshold size controlled by io.sort.spill.percent (default 80%), a background thread begins to stumble the contents to disk,(3) Partitioning: Prior to writing to the disk, the background thread splits the data into multiple partitions,(4) Sorting: Memory sort is executed on key (i.e., method of key class),(5) Merging: Prior to finishing the map task, the spill files are unified as a single partition, and sorted as the output file,(6) Compression: The map output can be compressed before writing to the disk resulting in: quicker disk writing, lower disk space, and less amount of data transferred to the reducer, (7) Reduce Operations: the reducer has three phases as following: Copy, Sort and Reduce [7]. Hadoop splits the input to a MapReduce job into fixed-size segments called "input splits". Hadoop produces one map task for every split which launches the user-defined map function for every record in the split. Having numerous splits implies the fact that the time spent to process for each split is so slight, compared to the total time to process the whole input. Therefore, if the splits are processed simultaneously and in parallel, the processing is better load-balanced for the reason that the splits are slight and a quicker machine is capable of processing equivalently more splits over the course of the job than a sluggish machine. Even if the machines are the same, failed processes or other jobs running in parallel make load balancing much more appropriate, and thus the quality of the load balancing goes up as the splits get more fine-grained. Oppositely, if splits are too slight and tiny, then the overhead of managing the splits and of map task creation starts to overpass the overall job execution time. In most cases, a desirable split size is inclined to be the size of a HDFS block, 64 MB by default. Although this can be modified for the cluster (for whole the recently created files), or distinguished when each file is formed and created.

In general, Hadoop Distributed File System (HDFS) consists of many components and specific roles as follows [26]: (1) NameNode: HDFS cluster consists of a single NameNode for managing the file system namespace and regulates access to file by clients, (2) DataNodes: Each slave machine in the cluster will host a DataNode daemon to perform the grunt work of the distributed file system both reading and writing, (3) SecondaryNameNode (SNN): SNN is an assistant daemon for monitoring the state of the cluster HDFS, (4) Job Tracker: The Job Tracker daemon is the liaison between the application and Hadoop, (5) TaskTracker: TaskTracker manages the execution of individual tasks on each slave node. MapReduce programs are executed in two main phases called "mapping" and "reducing". Each phase is defined by a data processing function and these functions are called "mapper" and "reducer", respectively. In the mapping phase, MapReduce takes the input data and feeds each data element to the mapper. In the reducing phase, the reducer processes all the outputs from the mapper and arrives at a final result as shown in Fig. 2.

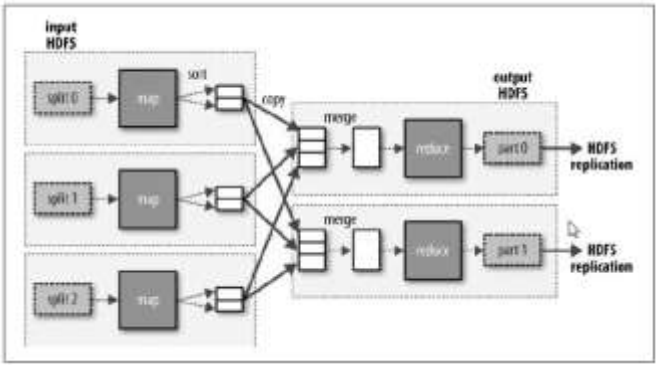


Figure 1. The basic concept of MapReduce[27].

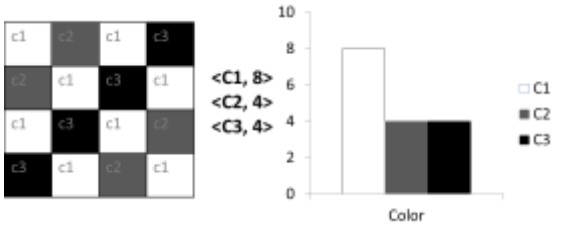
This diagram makes it clear why the data flow between map and reduce tasks is colloquially known as "the shuffle," as each reduce task is fed by many map tasks.When there are multiple reducers, the map tasks partition their output while each creating one partition for each reduce task. There can be many keys and their associated values in each partition, but the records for every key are all in a single partition. Typically, many MapReduce jobs are limited by the bandwidth available on the cluster, so it pays to minimize the data transferred between map and reduce tasks. Hadoop allows the user to specify a combiner function to be run on the map output. Since the combiner function is an optimization, Hadoop does not provide a guarantee of how many times it will call for a particular map output record. In other words, calling the combiner function zero, one, or many times should produce the same output from the reducer [26].

B. Hadoop MapReduce and Image Processing

In this section, an applying MapReduce scheme to image processing processes is presented. One example is the image histogram calculation that is easy to understand how Image Processing can be done using MapReduce. In programming MapReduce, it is possible to perform parallel distributed processing by writing programs involving the following three steps: Map, Shuffle, and Reduce [25]. Fig. 3 shows a 4x4 pixels image that we want to calculate as a histogram. The process should be done as follows:

1) The mapping phase of the map/reduce approach represented by ***map:*** *<key, value>⇒ list <key', value'>*. It

applies a function to each input value, producing a list of key/value pairs for each input. All these lists (each containing several key/value pairs) are gathered into another list to constitute the final output of the mapping phase as shown in Fig. 4.



(a) (b)

Figure 2. (a) 4x4 pixel of gray scale image, (b) a histogram of the example image

Figure 3. Data input and output from mapping phase.

2) The combine phase (Shuffle) can be represented by ***shuffle:*** *list <key', value'> ⇒{<key'', list(value'')>}*.It takes the output of the mapping phase and collects each key and associated values from the collection of lists of key/value pairs. The combined output is then essentially a map with unique keys created during the mapping process, with each associated value being a list of values from the mapping phase as shown in Fig. 5.

Figure 4. Data input and output from combine phase.

3) The reduce phase, the input to the reduce phase is the output of the combiner, which is a map, with keys being all the unique keys found in the mapping operation and the values being the collected values for each key from the mapping process. The output of the reduce phase can be any arbitrary value and can be represented by ***reduce:*** *{<key'', list(value'')>} ⇒ list(key''', value''') or list(value''')*,the input a <K′ , LIST V′ > pair, where "LIST V′ " is a list of all V′ values that are associated with a given key K′ as shown in Fig. 6.

Figure 5. Data input and output from reduce phase.