Distribution model determines which nodes are responsible for request processing and it is also the way data is distributed to the nodes. We will be talking about the advantage and disadvantages of these two types of distribution model, Master to slave and peer to peer.

Master Slave model, in distribution models we first have to ask ourselves “who is in charge here?” and as you can \*point\* see; in the master-slave model, only one node, the master node is responsible of all request and data distribution. The advantage of the master-slave model is that the data is very consistent because all requests has to be made to the master node, therefore it is data is consistent unless the master node fails, and the standby master needs to be used, then that might cause a minor rollback on the data.

Disadvantages, As we can see, every node is connected to the master node and all requests has to be made to the master node, under stressful condition, one of the disadvantages is network and possibly hardware bottleneck, because everything is depending on the master node. Another disadvantages is the difficulty of testing the standby master, since the standby master has to be tested in a working environment, we have to shutdown the master node in order to test if the standby master can handle the master node’s job, but this would lead to very high risk of shutting down the whole distribution model if the standby master fails. Last thing, is single point failure, because of the difficulties of testing the standby master, has increased probability that the standby master wont be able to take the master node’s job and if the master node experience a system failure, the whole distribution model will collapse. From this, we can also assume that performing maintenance on the master node will also be very problematic.

Peer to Peer model, in this model, all nodes are in charge, as it is shown, all nodes are interconnected. The advantages of this model is that it is easier to test and it is single point failure proof, because if we remove a node from this cluster, the other node will still be able to do the missing node’s job, therefore providing the advantage of high data availability. In the other hand, the disadvantage of this model is that low data consistency and latency, \*point\* when we perform an update request from this node, due to latency, the data at the other \*point\* nodes to become affected by the update will be delayed, which is a data consistency problem, if requests are made before the nodes has been updated.

Choosing the right distribution model depends on the needs of the user. If it is either high availability or data consistency, if we are seeking for High Availability then Peer to peer is the distribution model of choice, since peer to peer can work even if one of the nodes experience failure, as well as all the nodes can take requests therefore providing faster response time. But if we are looking for data consistency, then master-slave model is the choice, since only the master is able to take request and manage the data of the cluster, the master-slave model will always be able to provide the newest data in the cluster and it can prevent data from conflicting as well.

Using MapReduce on distributed systems, can solve the single-processor bottleneck problem and provide linear scalability, But it characteristics is that it requires large amount of input and output, it is Disk-based, which means it reads from a disk and writes to a disk, it is also batch-oriented which means that the data is not processed in real time, but in batches.

When talking about using MapReduce on distributed systems, Hadoop distributed file system (HDFS) are the most popular in the market. HDFS is designed to write the data once, for the data to be read many times. HDFS provides fault-tolerance by storing files in three different locations and if one of the drive fails, the other two drives can replicate lost data and store it in another drive. Similar to RAID drives. HDFS also uses a key-value store to boost its performance. Some difference between HDFS and other ordinary systems, is that it uses 64Mb block size, instead of the default like windows that uses a 4Kb block size. The data in HDFS is also immutable meaning that it cannot be modified, in order to change data, the whole block as to be deleted and created again, and when performing MapReduce jobs, it takes input of a gigabyte or larger.

How does MapReduce solves Big data problems? First of we will talking about the Map operation, \*point\* Map operation takes input data and use several processors to transform the data and output data in the key-value structure. Second thing is the Reduce operation, which takes all the key-values created and perform the reduce operation and then output the values needed.

It might sound easy to create a MapReduce framework, but there are many things to take into account when creating one. Some questions that you can ask yourself are, what if your source data is replicated on three or more nodes? Do you move the data between nodes? How do you assign the right key to the right reduce processor? What happens if one of the map or reduce jobs fails in mid-operation? Etc.

It seems to be a lot of questions right? But the good thing is that \*point\* HDFS can solve all those questions by making sure the reduce node gets the input based on the keys, and making sure that the job finishes even if there is a failure. Therefore, you only have to focus on implementing the map and reduce operations.