When dividing the work into chunks, each processed parallel in different machines, the runtime is dominated by the longest process in any of the computers. Overhead is incurred when independent processes have to be recombined and again processed as a whole. Datasets grow beyond the capability of a single machine if processed in parallel. Synchronization problem may arise, reliability often comes into play and can’t be sacrificed at any cost.

So it is better to overcome these factors of parallel programming by using well-structured framework like Hadoop, and help ease our computation.

Map function helps remove bad data, erroneous result and filter the dataset for further processing. Map and Reduce functions in key-value sets, with input as key and output as value. The output of the map function is processed by the MapReduce framework before being sent to the reduce function for further processing.

Just like the performance of quicksort program is depended on the split, similarly the split of the MapReduce tasks by Hadoop determines the performance of the task. Too small splits leads to overhead in maintaining them in multiple machines and in synchronising their functions in parallel.

Data locality optimization is done for efficient processing of the tasks in the node where the data is stored as bandwidth is saved. Optimal split size is the size of the block where the input data resides, as it the largest possible size that can be used.

Principle of locality is used by the map tasks as the output of the map tasks are stored in the local disks and not fed into the HDFS. This is because the output of the map task is not the final and it must be first fed to the reduce task and then the final output is obtained.

MapReduce jobs are limited in their performance due to the bandwidth which is limited, and we need to pay to increase the bandwidth of the cluster.

In order to optimize, Hadoop enables a combiner function to be run on the map output which in turn is the input to the reducer function. The combiner function may be called multiple times and in turn it gives the output of the reducer function thus optimizing the work of the reducer function and hence in turn it optimizes the bandwidth requirement. But it does not replace the work of the reducer in all times. The main motto of the combiner function is to cut the amount of data transferred between the nodes in the cluster.

When the dataset outgrows the storage capacity of a single physical machine, data are stored across separate filesystems maintaining a network among them.

HDFS block size is 64MB by default, files are broken into block sized chunks and stored independently.

Namenode failure and its recovery:

Self-adaptive MapReduce Algorithm in heterogeneous environment:

Introduction to MapReduce and where it is used. Hadoop is the most celebrated MapReduce framework as it is fast and it is an open source. Scheduling algorithms are used to schedule tasks for map and reduce. Different algorithms have been devised in order to improve the efficiency of the tasks. LATE scheduling algorithm is similar to shortest job first algorithm used to schedule processes in the operating systems. It selects the task for execution that has the shortest remaining time and asserts to have found out the tasks that are slow by calculating the remaining time and labelling those with longer remaining time as slow tasks. Inspite of it appearing to be optimal as in operating system scheduling algorithm, it fails to be the optimal in Hadoop as the algorithm calculates the remaining time wrongly.

Resources are scarce. Slow tasks reduces the performance of MapReduce jobs as the output will depend on the slowest task, increasing the time required for execution as well as system resources.

System hardware and similar constraints such as architecture, memory, power etc also affect the performance of the MapReduce job.

Although Hadoop has mechanism to handle slow tasks by generating backup tasks for them, but it fails to correctly find out the tasks that are slowing the execution.

Map functions produce key-value outputs which are in turn fed to the reduce functions to retrieve the final required output which is also in key-value pairs. SAMR overcomes the drawback of Hadoop by finding accurately the slow tasks and separating the slow nodes into map slow nodes and reduce slow nodes. It thus creates backup task for the slow nodes.