SUMMARY:

Jeffrey Dean and Sanjay Ghemawat.
MapReduce: Simplified Data Processing on Large Clusters
In *Proc. 6th Symp. Operating System Design and Implementation(OSDI)*, pages 137-150, 2004.

DATE: 25 January 2010

The paper presents MapReduce, which provides a simple abstraction for processing large amounts of data in a distributed fashion. The users of MapReduce provide input in the form of key/value pairs, a map function to process each key/value pair into a set of intermediate key/value pairs and a reduce function to combine values associated with the same intermediate key/value pair. The MapReduce takes care of partitioning the data, allocating distributed resources to perform map and reduce tasks, tolerating faults and producing the output. Moreover, it shields the user from these details and complications. Such a model can be used to efficiently solve many common problems like sorting, inverting index, distributed grep etc. on a large data set.

The paper gives details of MapReduce implementation on a cluster of commodity machines on which, copies of the MapReduce program run. One copy is "master" and the rest are workers. The master is responsible for assigning map and reduce tasks to workers and checking their progress by pinging them periodically. The input is partitioned between M (specified by user) splits so that different inputs can be assigned to different map workers. The map worker parses input assigned to it and produces the intermediate key/value pairs using the map function provided. The output is written locally, split into R regions. The location of these regions is passed to reduce workers through the master. Each reduce task is responsible for one of these R regions. The reduce worker reads, sorts and groups contents of these regions by intermediate keys and passes all values associated with a key to the reduce function which produces an output. The output from each reduce task is written to a separate output file. MapReduce provides fault tolerance by rescheduling tasks once any worker has failed. It can also assign the same task to multiple workers to avoid problems arising from a few sluggish workers. However, if the master fails the MapReduce program fails.

The division of tasks allows for processing reduce tasks in parallel with completing map tasks, increasing the efficiency of the over all system. This high efficiency is confirmed by experiments based on TeraSort benchmark.

SUMMARIZED BY: Rehan Rauf