MapReduce: Simplified Data Processing on Large Clusters (Summary)

MapReduce is a programming model for processing and generating large data sets which involves: a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values (passed from *map* via an iterator) associated with the same intermediate key. Programs written in this style are automatically parallelised, run on a large cluster of distributed commodity machines and highly scalable (processes terabytes of data on thousands of machines). Some examples include Distributed Grep, Count of URL Access Frequency, Reverse Web-Link Graph, Term-Vector per Host, Inverted Index and Distributed Sort.

Different implementations of the MapReduce in interface are possible depending on the environment. The implementation involving large clusters of commodity PCs connected together with switched Ethernet is considered here. The *Map* input data is distributed into partitions/splits across multiple machines using parallelisation while the *Reduce* data is distributed by partitioning the intermediate key space.

The MapReduce library must incorporate fault tolerance mechanisms as large amounts of data are being processed. Faults that may occur are master (special copy of program on a specific machines) failure, worker (the other copies on the rest of the cluster of machines) failure and semantics in the presence of failures. A straggler – a machine which takes an unusually long time to complete one of the last few map or reduce tasks in the computation – can lengthen the total execution time but this can be bypassed by the master scheduling backup executions of the remaining in-progress tasks when the MapReduce operation is close to completion.

Network bandwidth is conserved by storing the input data on the local disks of the cluster of machines – most input data is read locally and consumes no network bandwidth when large MapReduce operations are run on a majority of the workers in a cluster. Task granularity is achieved by subdividing both phases into a number of pieces (having practical bounds), ideally larger than the number of workers.

The writing of the map and reduce functions is generally sufficient for most needs but some extensions may prove to be useful. Examples of these include: the partitioning function, ordering guarantees, the combiner function, input and output types, side-effects, skipping bad records, local execution, status information and counters. The performance of MapReduce was measured over cluster configuration, Grep, sort, effect of backup tasks and machine failures.

The MapReduce library has been used in multiple domains including: machine-learning, clustering, extraction of data and graph computations. Its most significant use is for large-scale indexing.

With MapReduce, a simple program may be written and run efficiently on a thousand machines in the course of half an hour, greatly speeding up the development and prototyping cycle. It is easy to use as it hides the details of parallelization, fault-tolerance, locality optimization and load balancing. A large variety of problems are easily expressible as MapReduce computations. It also scales to large clusters of machines comprising thousands of machines.

Related work includes: Bulk Synchronous Programming, MPI primitives, active disks, the eager scheduling mechanism used in the Charlotte System, the cluster management system used in Condor, the sorting facility used in NOW-Sort, the communication system in River implemented by sending data over distributed queues and TACC.