Claire Chemutai -12102019  
Operating System

Summary of Map Reduce by Jeffrey Dean and Sanjay Ghemawat

**MAP REDUCE**

The paper, “*MapReduce: Simplified Data Processing on Large Clusters by Jeffrey Dean and Sanjay Ghemawat*”, presents MapReduce, a programming model and associated implementation for processing and generating large datasets. It is assumed that in an environment processing and generating large datasets, you may have hundreds or thousands of machines and the machines may experience failures. The MapReduce framework hides the details of parallelizing work, fault tolerance, data distribution to workers and load balancing behind the abstractions.

The user of MapReduce is responsible for writing the map and reduce functions while the MapReduce library is responsible for executing that program in a distributed environment. The **map function** is used to process a key/value pair to generate a set of immediate key/value pairs. The **reduce function** is used to merge all the intermediate values associated with the same intermediate key.

**Programming model**

The computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The MapReduce library expresses the computation as two functions: Map and Reduce functions. The map function is written by the user. It takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the Reduce function. The Reduce function accepts an intermediate key and a set of values for that key and merges together these values to form a possibly smaller set of values. Zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the user’s reduce function via an iterator. The MapReduce framework therefore allows us to handle lists of values that are too large to fit in memory.

**Implementation**

There are different possible implementations of the MapReduce interface. The right implementation depends on the environment. For example, one may be suitable for a small shared-memory machine, another for a large NUMA multi-processor, and another for an even larger collection of networked machines.

**Execution**

Given a map reduce framework with a user-defined map and reduce functions, the program is executed as below. The framework splits the input data into M pieces of typically 16 megabytes to 64 megabytes (MB) per piece. The megabytes are controllable by user via optional parameters. It then starts up many copies of the program on a cluster of machines.

The master copy of the program is special and the rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task.

A worker assigned a map task reads the contents of the corresponding input split. It then parses key/value pairs out of the input data and passes each pair to the user-defined Map function. The intermediate key/value pairs produced by the Map function are buffered in memory

The buffered pairs are periodically written to local disk and partitioned into R regions by the partitioning function. The locations of the buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.

A reduce worker is notified by the master about the locations and it uses remote procedure calls to read the buffered data from the local disks of the map workers. After reading all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together. External sort is used in the case where all the amount of intermediate data is too large to fit in memory.

The reduce worker then iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user’s Reduce function. The output of the Reduce function is appended to a final output file.

After all the map tasks and reduce tasks have been completed, the master wakes up the user program. The MapReduce call in the user program at this point returns back to the user code

**Functions provided by the Map Reduce**

Partitioning function

The users of Map Reduce specify the number of reduce tasks/output files that they desire (R). Data gets partitioned across these tasks using a partitioning function on the immediate key.

Ordering Guarantees

Map Reduce guarantee that within a given partition, the intermediate key/value pairs are processed in increasing key order. This ordering guarantee makes it easy to generate a sorted output file per partition.

Combiner Function

The Combiner function is executed on each machine that performs a map task. The output of a combiner function is written to an intermediate file that will be sent to a reduce task. Partial combining significantly speeds up certain classes of Map Reduce operations

Input and output types

The Map Reduce library provides support for reading input data in several different formats. In a similar fashion, it support a set of output types for producing data in different formats and it is easy for user code to add support for new output types.

Side effects

Map reduce do not provide support for atomic two-phase commits of multiple output files produced by a single task. Therefore, tasks that produce multiple output files with cross-file consistency requirements should be deterministic.

Skipping Bad Records

Map reduce provide an optional mode of execution where the Map Reduce library detects which records cause deterministic crashes and skips these records in order to make forward progress

Local Execution

To help facilitate debugging, profiling, and small-scale testing, an alternative implementation of the Map Reduce library that sequentially executes all of the work for a Map Reduce operation on the local machine has been developed

Status Information

The master runs an internal HTTP server and exports a set of status pages for human consumption. The status pages show the progress of the computation.

Counters

The Map Reduce library provides a counter facility to count occurrences of various events.

**Applications**

The map reduce has been used in a wide range of domains within Google. This include:

1. Large-scale machine learning problems
2. Clustering problems for the Google News and Froogle products
3. Extraction of data used to produce reports of popular queries (e.g. Google Zeitgeist)
4. Extraction of properties of web pages for new experiments and products (e.g. extraction of geographical locations from a large corpus of web pages for localized search)
5. Large-scale graph computations.

REFERENCE

Dean, J., & Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters.