MapReduce: Simplified Data Processing on Large Clusters

Summary:

Introduction

MapReduce is a programming model to process large amount of data. In these two functions namely Map and Reduce is used to process the data parallelly on large number of clusters. User uses map to map key value pair of a logical record and then use the reduce function to process this record over a large file on thousands of cluster machines. It includes parallelism, load balancing and checks for system failures.

Programming Model

The user writes a Map program which takes a key/ value pair and then emits the results. This result is passed to the reduce function also written by user which the combines the result for a particular key value pair and emits the final result.

This can be understood by the example of word counter in an document.

Map: I will take each word and generate a key value pair with word as key and count (1 in starting phase) and will emit this result.

Reduce: It will take all the Map key/value pairs and will then combine to get the final count of all the words present in the document.

Implementation

The MapReduce implementation is based on environment available. The given task is broken into M map functions and R reduce functions. The tasks are then sent over to the cluster machines. The machine then sends multiple copies of M and R to cluster. One the cluster acts as Master while others work as Worker. The master assigns work to the worker cluster who first work on M map functions and after completion of the map function they send the result to R reduce functions which is then read by workers.

The master distributes the task to workers depending on the state of workers which is *idle, in process or completed*.

Failure Tolerance

The master keeps on pinging all the workers to check if they are active and working properly. In case any worker does not ping back after some set amount of time them master labels that worker as *failed.* Any Map task assigned to that worker is then passed to another worker. Even is the task if in process or completed. It is marked idle and given to another worker. In case of reduce task if the worker fails the it is passed to another worker only if it is not completed.

In case of master fails the system can start another master from the last checkpoint or in case there is only one master then user need to reboot the program and start whole MapReduce program from beginning again.

Task Granularity: The number of Map tasks (M) and number of Reduce tasks (R) should be greater than the number of master and worker available. This acts as a great load balancing and the exact ratio depends on the type of task that is running on the system. Master must make O(M+R) scheduling decisions and must keep O(M*R) states in the memory.

Refinements

Although MapReduce function is sufficient in itself but it can use some refinements like user can write an optional combiner function after reduce function to speed up the process. User can also skip bad records if it has seen more failures on that record. The MapReduce library provides a counter facility to count occurrences of various events. For example, user code may want to count total number of words processed
or the number of German documents indexed, etc.

Performance

The performance of MapReduce function was good practically when tried over a large cluster of machines. The MapReduce gas been used at Google for various tasks:

- large-scale machine learning problems,
- clustering problems for the Google News and Froogle products,

- extraction of data used to produce reports of popular queries (e.g. Google Zeitgeist),
- 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), and
- large-scale graph computations.

Conclusion

The MapReduce programming model has been successful at Goggle due to various reasons.

First, the model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, fault-tolerance, locality optimization, and load balancing. Second, a large variety of problems are easily expressible as MapReduce computations. Third, they have developed an implementation of MapReduce that scales to large clusters of machines comprising thousands of machines. The implementation makes efficient use of these machine resources and therefore is suitable for use on many of the large computational problems encountered at Google.