Systems for Data Science - HW5

MapReduce - Paper Review

Shubham Shetty

February 13 2021

1 Summary

Processing data stored in a distributed fashion faces several challenges. For example, in GFS the files are stored as large object chunks over several chunkservers. Collecting all these chunks into a single server to process it would run into network bandwidth limitations. Also the amount of time to process large amounts of data also increases exponentially, and the server can become a single point of failure. It becomes imperative to adopt distributed processing methods in order to optimize the work loads.

Distributed data processing faces issues of how to effectively parallelize the computations, partition the data, and deal with server faults. MapReduce framework was developed in order to solve these issues and optimize distributed processing. It was developed internally at Google to help in processing their distributed data stored in GFS, with use cases such as Web indexing and reverse search indexing.

MapReduce is a programming model and associated implementations which allows programmers to write easily parallelizable code. MapReduce is used to process key-value data stored on a distributed system. The core idea behind MapReduce are the two functions - map and reduce. Map function processes a key-value pair to return an intermediate key-value pair, which is then assigned to other servers which then perform the reduce function, which generally aggregates (or performs any other function) on the intermediate output to generate the final output. In this way, the main master server only has to communicate the mapping and reducing functions to the servers, with only minimal transfer of data between nodes, i.e. MapReduce takes the processing to the data. This makes it easier to manage and optimize.

The map and reduce functions are user defined. The MapReduce framework takes care of all the distributed processing overheads such as data distribution, fault tolerance, load-balancing etc. so that the programmer only has to deal with the programming abstractions. Initially the framework first partitions the data into splits, which are divided across the system. These splits can be processed in parallel by different servers. After splitting, the map function is run on the servers called workers, which are coordinated and synchronised by a single master node. Master maps the tasks to the workers.

When a worker receives a map task, it processes the input key-value pair and returns the intermediate key-value output. This intermediate data is kept in memory and may be written to disk. The worker sends the location of intermediate output to the master, which then communicates this location to other nodes which can then continue processing. When a worker receives a reduce task, it can read the intermediate outputs on the location shared by the master using RPCs to the remote servers, and can then perform the reduce function on that data to get final output.

In case a worker node fails, MapReduce restarts the machine and reschedules that

workers task on a new machine. As the intermediate data is stored on disk, the worker would have to recreate that output. In case a master node fails, MapReduce aborts the program completely. In order to avoid complete loss, MapReduce regularly checkpoints the master state so that it can be recreated from that state after failure.

2 Strengths

The following are the main advantages of using MapReduce framework -

- MapReduce is highly scalable.
- MapReduce hides distributed processing issues such as parallelization, load-balancing, fault tolerance etc. under programming abstractions. This makes it easier to use for new programmers/engineers.
- A large number of use-cases can be easily solved using MapReduce.

3 Unclear Aspects

The following points are unclear with respect to MapReduce -

- There is some overhead involved in MapReduce during data shuffle step. How much overhead is this usually and how can it be minimised?
- Will MapReduce work with complex algorithms? Most of the use-cases mentioned were simple to implement algorithms. Would MapReduce help for complex algorithms such as ML or Neural Network programs?
- Would MapReduce produce similar results on a standalone system?

4 Limitations

The following are the limitations and areas of improvement for MapReduce -

- Multiple shuffles may affect the performance of the program. MapReduce may not be the most optimum solution for all distributed processing problems.
- Jobs in MapReduce run in isolation. In cases where jobs need to communicate with each other or transfer data, MapReduce will not be suitable.
- MapReduce is slow - for use cases which require real-time processing and immediated results MapReduce will fail.