http://dx.doi.org/10.5339/qfarf.2012.CSO10

A flexible and concurrent MapReduce programming model for shared-data applications

Fan Zhang, Qutaibah M Malluhi

Carnegie Mellon University in Qatar, Doha, QATAR; Qatar University, Doha, QATAR

fanzhan1@qatar.cmu.edu

The rapid growth of large data processing has meant the implementation of the MapReduce programming model as a widely accepted solution. The simple map and reduce stages have introduced convenience to programmers in order that they may quickly compose and design complex solutions for large-scale problems.

Due to the ever-increasing complexity of execution logic in real-life applications, more and more MapReduce applications involve multiple correlated jobs encapsulated and executed according to a defined order. For example, a PageRank job involves two iterative MapReduce sub-jobs; the first job joins the rank and linkage table and the second one calculates the aggregated rank of each URL. Two non-iterative MapReduce sub-jobs for counting out-going URLs and assigning initial ranks are also included in the PageRank job.

Besides this, MapReduce programming model lacks built-in support and optimization when the input data are shared. The performance will benefit when the shared and frequently accessed files are read from local instead of from distributed file system.

This paper presents Concurrent MapReduce; a new programming model built on top of MapReduce while maintaining optimization and scheduling for big data applications that are composed of large number of shared data items. Concurrent MapReduce has three major characteristics:

(1) Unlike traditional homogeneous map and reduce functions, it provides a flexible framework, which supports and manages multiple yet heterogeneous map and reduce functions. In other words, programmers are able to write many different map and reduce functions in a MapReduce job.

(2) It launches multiple jobs in a task-level concurrency and a job-level concurrency manner. For job-level concurrency, the framework manages the shared data by replicating them from HDFS to the local file system to ensure data locality. For task-level concurrency, it is the programmers' responsibility to define the data to be shared.

(3) We have evaluated the framework using two benchmarks: Single Source Shortest Path and String Matching. Results have demonstrated up to 4X performance speedup compared to traditional non-concurrent MapReduce.