CSCI-GA.3033-005 Realtime and Big Data Analytics February 2, 2016

Summary

Name: Gowtham Goli (N17656180) *Due: Wednesday, February 3rd*

MapReduce is a programming model and an associated implementation for processing and generating large data sets. It consists of two parts. Map, written by the user, takes an input key/value 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, also written by the user, accepts an intermediate key and a set of values for that key. It merges together these values to form a possibly smaller set of values. Some of the examples which uses MapReduce are counting word frequency in a document, Distributed Grep, Count of URL Access Frequency, Reverse Web-Link Graph, Term-Vector per Host, Inverted Index, Distributed Sort. Many different implementations of MapReduce are possible. The most common implementation that is used at Google is described. The MapReduce library partitions the input file into M splits. There is a special node called Master and the rest are called Workers. The Master picks the idle workers and assigns tasks to them. These workers parse the key/value pair and pass them to the Map function. The intermediate key/value pairs produced by the Map function are buffered in the memory that are periodically written to the local disk, partitioned into R regions. The locations of these pairs on the disk are passed back to the Master. The Master forwards these locations to reduce workers. The reduce workers pass these intermediate key/value pairs to the reduce function whose output is appended to the final output file for this reduce partition. After the completion of all Map and Reduce tasks, the MapReduce call returns back to the user code. MapReduce is resilient to large scale worker failures. The Master simply re-executes the work done by the unreachable worker machines, and continues to make forward progress, eventually completing the MapReduce operation. The details of partitioning the input data, scheduling the programs execution across workers, handling failures and managing the required inter-machine communication is all taken care by the system. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. Though by simply defining map and reduce functions is sufficient for most of the cases, a few refinements can still be done, a few of which are described. For example, if the output keys are URLs, the partition function is *hash(Hostname(urlkey)) mod R* causes all URLs from the same host to end up in the same output file. There is an option for the users to specify an optional Combiner function that does partial merging of the data before it is sent over the network. Users can add support for a new input type by providing an implementation of a simple reader interface. There is an an optional mode of execution where the MapReduce library detects which records cause deterministic crashes and skips these records in order to make forward progress. To help facilitate debugging, profiling, and small-scale testing, there is an alternative implementation of the MapReduce library that sequentially executes all of the work for a MapReduce operation on the local machine. 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. The MapReduce programming model has been successfully used at Google for many different purposes because the model is easy to use, even for programmers without experience with parallel and distributed systems