Twister: A new approach to MapReduce Programming

VASANTH METHKUPALLI[1,*]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: mvasanthiiit@gmail.com

Paper-1, March 6, 2017

MapReduce is a method to process vast sums of data in parallel without requiring the developer to write any other code other than the mapper and reduce functions. Starting with Google in 2004 there has been a lot of research going on in this particular field since the rate at which data is increasing is exponential. The need to store the data and analyze has become of paramount importance in the current situation. This has lead the researcher's to look for various parallel processing programming models to saturate the needs. Of these MPI, MapReduce are some of the examples which has produced good results to the scientific community. Here in this paper we examine an implementation of MapReduce programming model, Twister, which exhibits some improvements over the current model.

Keywords: MapReduce, Twister, Iterative, reduction, combine

https://github.com/cloudmesh/sp17-i524/blob/master/paper1/S17-IR-2019/report.pdf

This review document is provided for you to achieve your best. We have listed a number of obvious opportunities for improvement. When improving it, please keep this copy untouched and instead focus on improving report.tex. The review does not include all possible improvement suggestions and if you sea comment you may want to check if this comment applies elsewhere in the document.

In the Abstract, "without requiring the developer to write any..." is a very strong statement, any citations to support it?

1. INTRODUCTION

Emergence of massive data sets in many areas and settings have presented many challenges and opportunities in data storage and analysis[1][2]. Traditional analytic tools many a times cannot live up to the needs and demands at the ongoing rate at which data is produced to store and analyze it[3]. However, domain knowledge and research have given rise to new tools and technologies which has made many of these tasks easier[4]. Google's MapReduce falls in to one of these categories. The MapReduce programming framework uses two tasks common in functional programming: Map and Reduce, Map() procedure (method) that performs filtering and sorting, Reduce() method that performs a summary operation(merging the results)[5][6]. MapReduce is a new parallel processing framework and Hadoop is its open-source implementation on a single computing node or on clusters. Compared with existing parallel processing paradigms (e.g. grid computing and graphical processing unit (GPU)), MapReduce and Hadoop have two advantages:1) Fault-tolerant storage resulting in reliable data processing by replicating the computing tasks, and cloning the data chunks on different computing nodes across the computing cluster. 2) High-throughput data processing via a batch processing framework and the Hadoop distributed file system (HDFS)[7][8][3].

Data are stored in the HDFS and made available to the slave nodes for computation. A MapReduce program consists of the "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance[6]. MapReduce programming model has simplified the implementations of many data parallel applications[3]. The simplicity of the programming model and the quality of services provided by many implementations of MapReduce attract a lot of enthusiasm among parallel computing communities[8][9]. It has been identified that MapReduce can be extended to many other applications by improving on the programming model and the architecture. In this regard, Twister attempts to extend the Google's MapReduce application to more class of applications by including more features[10][11].

2. TWISTER:

remove the colon in the section title

Twister was developed as a part of Ph.D. research and is an going research project by the SALSA team @IU[9][1].

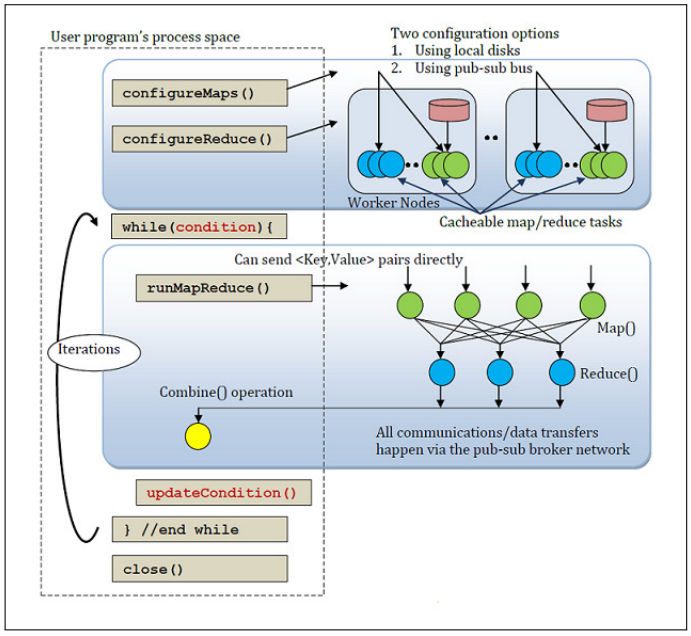

Fig. 1. Twister Programming model.

how about "at Indiana University"?

Identifying many problems in MapReduce programming model, they have envisioned to develop a better version to apply it to various scientific applications[4]. A set of extensions to the programming model and improvements to its architecture that will expand the applicability of MapReduce to more classes of applications[7]. Twister is a lightweight MapReduce runtime that has been developed by incorporating these enhancements[8][3].

reduce the above repeative description about how good it is

Twister provides the following features to support MapReduce computations. 1) Distinction on static and variable data 2) Configurable long running (cacheable) map/reduce tasks 3) Pub/sub messaging based communication/data transfers 4) Efficient support for Iterative MapReduce computations (extremely faster than Hadoop or Dryad/DryadLINQ) 5) Combine phase to collect all reduce outputs 6) Data access via local disks 7) Lightweight (5600 lines of Java code) 8) Support for typical MapReduce computations 9) Tools to manage data[10][11].

3. TWISTER IMPROVEMENTS OVER MAPREDUCE PROGRAMMING MODEL:

3.1. Static vs. Dynamic Data

In all the iterative applications which were observed, they showed two types of data formats, one-static data, two-dynamic data[9]. Static data is data which is fixed throughout the computation whereas the variable data(Dynamic Data) is the computed results in each iteration and typically consumed in the next iteration in other applications, one of which example is an expectation minimization algorithms[11].

good, they are different, but how the performance is improved?

3.2. Canceable Mappers/Reducers

Although some of the typical MapReduce computations such as distributed sorting and information retrieval consume very large data sets, many iterative applications we encounter operate on moderately sized data sets which can fit into the distributed memory of the computation clusters[7]. This observation led us to explore the idea of using long running map/reduce tasks similar to the long running parallel processes in many MPI applications which last throughout the life of the computation. The long running (cacheable) map/reduce tasks allow map/reduce tasks to be configured with static data and use them without loading again and again in each iteration[3]. Current MapReduce implementations such as Hadoop and DryadLINQ do not support this behavior and hence they initiate new map/reduce tasks and load static data in each iteration incurring considerable performance overheads[10].

3.3. Supports "side-effect-free" Programming

Twister has long running map-reduce takes by which it does not encourage users to store state information in the map/reduce tasks. Thereby, achieving the side-effect-free nature of MapReduce. Caching the static data across map/reduce tasks helps in achieving better thoroughput[3]. This framework does not ensure the use of same set of map/reduce tasks throughout the life of an iterative computation[10].

3.4. Combine Step as Further Reduction

Twister also introduce an optional reduction phase named "combine", which is another reduction phase that can be used to combine the results of the reduce phase into a single value. The user program and the combine operation run on a single process space allowing its output directly accessible to the user program[9]. This enables the user to check conditions based on the output of the MapReduce computations[10].

3.5. Uses Pub/sub messaging

Twister uses pub/sub messaging for all the communication/data transfer requirements which eliminates the overhead in transferring data via file systems as in Hadoop orDryadLINQ[9]. The output <Key,Value> pairs produced during the map stage get transferred directly to the reduce stage and the output of the reduce stage get transferred directly to the combined stage via the pub-sub broker network[3]. Currently Twister uses publish-subscribe messaging capabilities of NaradaBrokering messaging infrastructure, but the framework is extensible to support any other publish-subscribe messaging infrastructure such as Active MQ [10].

3.6. Data Access via local disks

Two mechanisms about data access have been proposed in Twister; (i) Directly from the local disk of computer nodes, (ii) Directly from the pub-sub infrastructure[7]. For the simplicity of the implementation, Twister provided a file based data access mechanism for the map/reduce tasks. Unlike Hadoop, twister does not come with the built in file system. Instead it provides a tool to manage the data across these distributed disks. Apart from the above the use of streaming enables Twister to support features such as directly sending input <Key,Value> pairs for the map stage from the user program and configuring map/reduce stages using the data sent from the user program[6].

3.7. Fault Tolerance

Providing fault tolerance support for iterative computations with Twister is currently under development.

4. APPLICATION OF TWISTER TO PRESENT PROBLEMS:

this section needs to be improved in this way: you don't need to illustrate in detail each typical problem, focus on showing some results to prove Twister has better scores, as there are many aspects being listed in previous section claiming Twister's advantages.

4.1. K-Means Clustering

Kmeans clustering is a well-known clustering algorithm aiming to cluster a set of data points to a predefined number of clusters. In that each map function gets a portion of the data, and it needs to access this data split in each iteration. These data items do not change over the iterations, and it is loaded once for the entire set of iterations. The variable data is the current cluster centers calculated during the previous iteration and hence used as the input value for the map function[10][1].

All the map functions get this same input data (current cluster centers) at each iteration and computes a partial cluster centers by going through its data set. A reduce function computes the average of the partial cluster centers and produce the cluster centers for the next step. Main program, once it gets these new cluster centers, calculates the difference between the new cluster centers and the previous cluster centers and determine if it needs to execute another cycle of MapReduce computation[12][10].

4.2. Matrix Multiplication

Let A and B, produce matrix C, as the result of the multiplication process. Here they split the matrix B into a set of column blocks and the matrix A into a set of row blocks. In each iteration, all the map tasks process two inputs: (i) a column block of matrix B, and (ii) a row block of matrix A; collectively, they produce a row block of the resultant matrix C. The column block associated with a particular map task is fixed throughout the computation, while the row blocks are changed in each iteration. However, in Hadoop's programming model (a typical MapReduce model), there is no way to specify this behavior. Hence, it loads both the column block and the row block in each iteration of the computation. Twister supports the notion of long running map/reduce tasks where these tasks are allowed to retain static data in the memory across invocations, yielding better performance for "Iterative MapReduce" computations[1][10].

4.3. Page Rank

In Twister implementation of PageRank, we constructed web graphs with vertices where in-link degree of all pages comply with the power law distribution. These input data are partitioned into few parts and stored in the format of adjacency list. Each map function runs on one of the partitioned data, which are constant over the iterations. The variable data are the PageRank values calculated during previous iteration which in turns used as the input value for the next iteration. In each iteration, the MAP task updates the old PageRank values to new one by analyzing the local partial adjacency list file. The output of MAP task is partial of PageRank values. The reduce task receives all the partial output and produces the new PageRank values[10].

4.4. Graph Search

This algorithm tries to use Twister Framework to process breadth-first graph search problem in parallel. This algorithm is based on Cailin's Hadoop version of breadth-first graph search[10]. The basic idea of this algorithm is to exploring the nodes of the same level parallel, and then explore the next levels iteratively[10].

4.5. Word Count

In typical word count applications implemented using other MapReduce runtimes, the map task outputs (word,1) pairs for all the words it encounter. This approach is not optimized for performance rather simple to program. With small amount of more complexity we can simply convert this map task to produce a list of (word,count) pairs corresponding to the local data partion. This is the approach used in Twister word count application[10].

4.6. High Energy Physics(HEP) Data Analysis

The goal of the analysis is to execute a set of analysis functions on a collection of data files produced by high-energy physics experiments. After processing each data file, the analysis produces a histogram of identified features. These histograms are then combined to produce the final result of the overall analysis. This data analysis task is both data and compute intensive and fits very well for MapReduce computation model. First figure shows the program flow of this analysis once it is converted to a MapReduce implementation and the second figure compares the performance of Twister and Hadoop for this data analysis[12][10].

5. CONCLUSION

It has been observed that changes in the programming model and the way the nodes interact with each other(pub-sub messaging)[8], has produced very good results with Twister MapReduce. This has in turn lead it benificial to many applications of which many have been discussed in the present paper. So it can be safe to assume that Twister has a very high scope for commercial applications even thought it is limited to scientific applications as of now.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," pp. 107–113, 2008.

[2] A. Mohebi, S. Aghabozorgi, T. Ying Wah, T. Herawan, and R. Yahyapour, "Iterative big data clustering algorithms: a review," *Software: Practice and Experience*, vol. 46, no. 1, pp. 107–129, 2016.

[3] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with mapreduce: a survey," *AcM sIGMoD Record*, vol. 40, no. 4, pp. 11–20, 2012.

[4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[5] "MapReduce," Feb. 2017, page Version ID: 764859583. [Online]. Available: https://en.wikipedia.org/w/index.php?title=MapReduce&oldid=764859583

[6] E. A. Mohammed, B. H. Far, and C. Naugler, "Applications of the mapreduce programming framework to clinical big data analysis: current landscape and future trends," p. 22, 2014.

[7] A. Elsayed, O. Ismail, and M. E. El-Sharkawi, "Mapreduce: State-of-the-art and research directions," *International Journal of Computer and Electrical Engineering*, vol. 6, no. 1, p. 34, 2014.

[8] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: a runtime for iterative mapreduce," in *Proceedings of the 19th ACM international symposium on high performance distributed computing*. ACM, 2010, pp. 810–818.

[9] K. Grolinger, M. Hayes, W. A. Higashino, A. L'Heureux, D. S. Allison, and M. A. Capretz, "Challenges for mapreduce in big data," in *Services (SERVICES), 2014 IEEE World Congress on*. IEEE, 2014, pp. 182–189.

[10] "Twister: Iterative MapReduce." [Online]. Available: http://www.iterativemapreduce.org/

[11] C. Doulkeridis and K. Nørvåg, "A survey of large-scale analytical query processing in mapreduce," *The VLDB Journal*, vol. 23, no. 3, pp. 355–380, 2014.

[12] J. Ekanayake, S. Pallickara, and G. Fox, "Mapreduce for data intensive scientific analyses," in *eScience, 2008. eScience'08. IEEE Fourth International Conference on*. IEEE, 2008, pp. 277–284.