MapReduce for Big Data Analysis: Benefits, Limitations and Extensions

Yang Song, Hongzhi Wang(✉), Jianzhong Li, and Hong Gao

Harbin Institute of Technology, P.O. Box 750, Harbin 150001, China
wangzh@hit.edu.cn

Abstract. Big data becomes a hot topic. MapReduce is a popular programming paradigm for big data analysis with many benefits. Even though it has widely applications in industry, MapReduce still has limitations in some applications. For these limitations, some extensions have been proposed. In these brief communications, we discuss the benefits and limitations of MapReduce programming paradigm and also its extensions to make MapReduce go beyond the limitations.

Keywords: Big data · MapReduce · Parallel computation · Analysis

1 Introduction

We are at the new frontier of information explosion, in which industry, academia, business and governments are accumulating data in an unimaginable, unpredictable and unprecedentedly high speed. Thus big data processing techniques are in demand, whose goal is to accomplish computation tasks over huge datasets in a reasonable time.

To handle big data, it is a natural way to adopt parallel techniques. In the past few year, many parallel computing platforms have been built, which involve hundreds or even thousands of machines to process big data. Each platform may have a special programming paradigm. One of the most notable one is MapReduce [9], which is proposed by Google. Google also put up a computational diagram Pregel for graph. Spark made some improvements. Others like Hyacks and Graphlab is based on distributed system.

MapReduce provides an easy way to accomplish computational tasks on big data and can scale to large computer clusters. Meanwhile, it also supports the hardware fault tolerance during the process of calculation. To use this paradigm, the user only needs to write two functions, Map and Reduce. The system can manage the task parallel execution and coordination of mappers and reducers, and handle the failures. A MapReduce algorithm instructs these functions to perform a computational task collaboratively.

The MapReduce process a computational task as follows. It involves more than one map job. The input of each map job is one or more DFS files. A Map job converts file blocks to a key-value sequence. The main controller collects a series of key-value pairs from every Map task and sorts them by key sizes. For each time, a reducer process key-value pairs sharing the same key value, and combines the value according to requirement.

DOI: 10.1007/978-981-10-2053-7_40

In MapReduce, grouping and aggregation are according to the same criterion. The Map function converts all input elements into key-value pairs. A map job can generate multiple key-value pairs from the same key. The Reduce function combines the values of a series of key-value table on a demand basis. The output of each reducer is also a key-value sequence, since the reduce task receives the key K of each key pairs as input keys. The outputs of all Reducers are merged into a single file as the final results.

As the earliest big data computing tools, MapReduce unavoidably has some foibles. However, they do not influence its important role in the research and development of big data. It has many advantages beyond other methods. The parallel big data processing improves the performance in big data analysis, which follows by other big data paradigms. This model is easy to use, even for programmers with no experience of distributed development. Also, it hides the details of parallel computing, disaster error, the optimization and load balance. MapReduce can easily handle large-scale parallel computing. For example, Google uses MapReduce to provide web search services, sorting, data mining, machine learning, and other systems. Through MapReduce, applications can run in more than 1000 large cluster nodes which provide optimized disaster error. What's more, the scalability of MapReduce is very awesome. For every server, MapReduce puts computing ability access to the cluster. For most of the past distributed processing frameworks [11, 12], their scalability inferiors to MapReduce a lot. Of course, there are still some limitations about MapReduce. However, with some effective proper algorithm, these problems can be solved completely.

To prove that MapReduce still alive, we will discuss a few representative examples of limitation in MapReduce and give some extensions to solve these problems. MapReduce is unique and can't be replaced. We will discuss in this paper and give evidence to demonstrate this thought.

2 Limitations

MapReduce is becoming a basic framework for processing massive data, due to its excellent scalability, reliability and elasticity. After released, MapReduce went through years of improvement into a mature paradigm. Although this is not a perfect approach, many ways have been proposed to advance it and contribute it to a more efficient, practical and convenient tool for big data analysis. First, we list some limitations to illustrate present problems. Here are some limitations in MapReduce and we take them as examples to share our view about MapReduce.

Limitation 1: incremental iterative MapReduce

Cloud intelligence applications often perform iterative computations on constantly changing datasets [4]. Many data-centric algorithms require efficient iterative computations, such as the well-known PageRank [5] algorithm in web search engines, gradient descent [6] algorithm for optimization, and many other iterative algorithms for applications including k-means algorithm, recommender systems [7] and link prediction [8]. Previous studies are too expensive to perform an entirely new large-scale MapReduce iterative job. It's difficult to timely accommodate new changes to the underlying datasets. To handle sophisticated iteration algorithm, the problem is eager to be solved.

Limitation 2: multiple inputs MapReduce

MapReduce is not designed to directly support operations with multiple inputs such as joins [2]. Many studies on join algorithms including Bloom join [10] in MapReduce have been conducted but they still have non-joining data generated and transmitted over the network. The join operation is considered as a paradigm of such operations. Although there have been many studies on join algorithms in parallel DBMSs, a MapReduce environment is not straightforward to implement joins. To improve MapReduce competence with multiple inputs, the solutions of this problem are in demand.

Limitation 3: minimal algorithms MapReduce

Ideally, a MapReduce system should achieve a high degree of load balancing among the participating machines, and minimize the space usage, CPU and I/O time, and network transfer at each machine. Although these principles have guided the development of MapReduce algorithms, limited emphasis has been placed on enforcing serious constraints on the aforementioned metrics simultaneously [3]. To guarantee the best parallelization and up to small constant factors, many studies about minimal algorithms of MapReduce have been performed.

3 Extensions

With many attentions about MapReduce, researchers have carried on many research work and come up with many extensions to go beyond the limitations. These extensions improve the performance of MapReduce, making it more powerful. Here we introduce some extensions that can effectively make up for the inadequacy of MapReduce.

Extensions 1: incremental iterative MapReduce

I2 MapReduce puts up to deal with incremental iterative MapReduce.

I2 MapReduce introduces a Map-Reduce Bipartite Graph model to represent iterative and incremental computations, which contains a loop between mappers and reducers [4]. A converged iterative computation means that a Map-Reduce Bipartite Graph state is stable. In many cases, only a very small fraction of the underlying dataset has changed, and the newly converged state is quite close to the previously converged state. Based on this observation, the design and implementation of I2 MapReduce to efficiently utilize the converged a Map-Reduce Bipartite Graph state to perform incremental updates is proposed.

An existing MapReduce application needs only slight code modification to take advantage of I2 MapReduce.

Extensions 2: multiple inputs MapReduce

As to multiple inputs MapReduce, a new type of bloom filter [10] called Intersection Bloom filter which represents the intersection of the datasets to be joined is proposed.

Unlike previous solutions that filter only on one dataset, the method filters out disjoint elements or non-joining tuples from both datasets. Each tuple from the input datasets is queried into the intersection filter and will be removed if it is a disjoint element. There are three approaches for building the intersection filter and show their feasibility in two-way joins and join cascades.

It is proved that the join operation using the intersection filter is more efficient than other solutions since it significantly reduces redundant data, and thus produces much less intermediate data [2]. Moreover, the intersection filter provides an extremely important characteristic for a join cascade in which intermediate join results generated from component joins only contain actual joining tuples without filtering. Although the intersection filter has false positives and an extra cost for the pre-processing step, its efficiency in space-saving and filtering often outweighs these drawbacks.

Extensions 3: minimal algorithms MapReduce

In recent years, optimum algorithms of MapReduce are studied. MapReduce has grown into an extremely popular architecture for large-scaled parallel computation. Even though there have been a great variety of algorithms developed for MapReduce, few are able to achieve the ideal goal of parallelization: balanced workload across the participating machines, and a speedup over a sequential algorithm linear to the number of machines [3].

"Minimal MapReduce algorithm" puts together for the first time four strong criteria towards the highest parallel degree. At first glance, the conditions of minimality appear to be fairly stringent. Nonetheless, the existences of simple yet elegant algorithms that minimally settle an array of important database problems.

The proposed algorithms demonstrate that the immediate benefit brought forward by minimality and significantly improve the existing state of the art for all the problems tackled.

4 Discussions

In this section, we discuss important drawbacks and solutions about MapReduce. In recent days, there is a debate whether MapReduce should be abandon. Some new methods have been proposed to take the place of MapReduce and claimed it is not wise to follow MapReduce which has many foibles and weakness. However, as discussed in Sect. 2, MapReduce can be improved through the above algorithm that we found. With these improvements, it is still lively in big data analysis. Through the extensions, MapReduce is becoming more and more mature and one of its implementation, Hadoop, has won more and more users. Besides, many proposed techniques on MapReduce give us more confidence on MapReduce. We have many reasons to believe that MapReduce will not vanish and it will have broad prospects, especially in offline social network. Hadoop has advantages in offline calculation with great accuracy. We are looking forward to breakthrough about MapReduce.

Acknowledgments. This paper was partially supported by National Sci-Tech Support Plan 2015BAH10F01 and NSFC grant U1509216, 61472099, 61133002 and the Scientific Research Foundation for the Returned Overseas Chinese Scholars of Heilongjiang Provience LC2016026.

References

1. Rajaraman, A., Ullman, J.D.: Mining of Massive Datasets. Posts & Telecom Press, Beijing (2012)
2. Phan, T.-C., d'Orazio, L., Rigaux, P.: Toward intersection filter-based optimization for joins in MapReduce. In: Cloud-I, p. 2 (2013)
3. Tao, Y., Lin, W., Xiao, X.: Minimal MapReduce algorithms. In: SIGMOD Conference, pp. 529–540 (2013)
4. Zhang, Y., Chen, S.: i2MapReduce: incremental iterative MapReduce. In: Cloud-I, p. 3 (2013)
5. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Comput. Netw. ISDN Syst. **30**, 107–117 (1998)
6. Avriel, M.: Nonlinear Programming: Analysis and Methods. Courier Dover Publications, Mineola (2003)
7. Baluja, S., Seth, R., Sivakumar, D., Jing, Y., Yagnik, J., Kumar, S., Ravichandran, D., Aly, M.: Video suggestion and discovery for youtube: taking random walks through the view graph. In: Proceedings of the WWW 2008, pp. 895–904 (2008)
8. Liben-Nowell, D., Kleinberg, J.M.: The link-prediction problem for social networks. JASIST **58**(7), 1019–1031 (2007)
9. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: Proceedings of the OSDI 2004 (2004)
10. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
11. Borkar, V.R., Carey, M.J., Grover, R., Onose, N., Vernica, R.: Hyracks: a flexible and extensible foundation for data-intensive computing. In: ICDE, pp. 1151–1162 (2011)
12. Jiang, D., Chen, G., Ooi, B.C., Tan, K.-L., Wu, S.: epiC: an extensible and scalable system for processing big data. PVLDB **7**(7), 541–552 (2014)