Proceedings of the 5th International Conference on Computing and Data Science
DOI: 10.54254/2755-2721/14/20230763

Research on distributed coding computation based on matrix multiplication

Lin Zhang

Xi'an University of Technology, Xi'an, China

3200441180@stu.xaut.edu.cn

Abstract. Matrix multiplication is used in machine learning to train larger, more accurate models with massive data more effectively. The distributed machine learning paradigm necessitates the distributed computing of many matrix multiplications once it is adopted. The execution time of distributed machine learning algorithms increases on dropout nodes, which are computational nodes in distributed clusters that randomly slow down computation due to a variety of factors (e.g., node failure, system failure, communication bottlenecks, etc.), becoming a significant bottleneck in distributed computing systems. It has been discovered that coded computation is less expensive than replica methods and can more effectively reduce the effects of dropped nodes. In this study, we present theoretical insights on how encoded solutions might produce significant gains over unencoded solutions.

Keywords: coding computing; distributed machine learning; matrix multiplication.

1. Introduction

With the rapid development of the Internet and advances in big data processing frameworks, distributed computing has the advantages of high fault tolerance, scalability, openness, fast computation, and has become a common approach to large-scale task computing. Distributed computing improves computational efficiency when two or more software share information with each other, and this software can run either on the same computer or on multiple computers connected through a network, it is possible to decompose a computational task into many smaller parts. However, there exist certain computational nodes in a distributed cluster that slowdown in some random way due to various factors (e.g., unexpected network failures, hardware failures, communication bottlenecks, etc.), and such nodes are called straggler nodes.

Recent results show that coding is introduced into distributed computing, using carefully designed redundant computing power on nodes to create coding multicast between nodes, so as to reduce the number of data exchanges between nodes, reduce the communication load, and effectively reduce the impact of straggler nodes. Coding computing is a great innovation in the field of distributed computing. With the growing demand of computing intensive, coding-based distributed computing will become the mainstream of distributed computing in the future.

DOI: 10.54254/2755-2721/14/20230763

2. Related work: theorem, current status

2.1. Status of domestic and international research on distributed computing

Dropped nodes are a problem in distributed computing systems that can cause substantial computational lag. & Ananthanarayanan et al. demonstrated that some of the compute nodes in distributed computing experience unpredictable latency as a result of a variety of factors, including network latency, shared resources, maintenance tasks, and power limits [1]. Several of the processing nodes in distributed computing experience unpredictable latency due to a variety of reasons, including network latency, shared resources, maintenance tasks, and power limitations [2,3]. Because of the greater overall time needed to execute computational operations, distributed computing's performance is limited. The conventional approaches to dealing with the dropped nodes issue are as follows.

Processing Based on Dropped Node Detection: One of the simplest ways to lessen the effects of dropped nodes is the dropped node detection A MapReduce computational model is put into practice by Dean et al. and is capable of running on huge clusters of commercial equipment. The map and reduce functions can be used to create the highly scalable MapReduce processing architecture [4]. Using the map and reduce functions, the MapReduce computing model may define distributed computation. The MapReduce computation model can express distributed computation by employing map and reduce functions, which automatically handle communication and node failures, and handle deleted nodes. The failure of a node triggers a generic The method continuously monitors the worker node's task progress to identify dropped nodes, and after that, it dispatches the dropped node to the following level. The dropped nodes are then scheduled to be executed on other available worker nodes. Konwinski et al. enhanced the current technique for detecting dropped nodes and showed that the suggested remedy works. An approach that can instantly identify dropped nodes and carry out the task on the other available worker nodes is proposed by Ananthanarayanan et al. This strategy can further cut down on the time it takes for MapReduce processes to complete by detecting lost nodes in real time and eliminating them [5].

Using Direct Replication and Policy-based Processing:. Replication policy-based dropped node processing is another method for dealing with the issue of dropped nodes, and it can greatly boost distributed computing performance by replicating computational workloads and scheduling replicas. Although scheduling copies of computational jobs and replicating them reduces the overall response time of computational processes, it often uses more resources (memory, network bandwidth) In addition to introducing a sing-fork scheduling model, Da Wang et al. presented a theoretical framework to analyze the connection between reaction time and resource use [6]. According to this model, the system distributes N computational jobs among N worker nodes, waits for N-S computational tasks to finish, and then initiates R+1 copies of the computational activities for the S dropped nodes in accordance with a replication strategy. The replication strategy can be put into practice in one of two ways: either keeping the computing tasks of the deleted nodes while starting r new replicas, or cancelling those tasks while starting r+1 new replicas. The authors provide the trade-off criteria for this scheduling strategy to achieve optimal performance given the machine runtime distribution. They also suggest an effective technique to discover the optimal or nearly optimal scheduling policy. Nihar B. Shah et al. analyze and examine the latency performance of redundant requests, determine the best redundant request policy, and evaluate the latency performance of various redundant requests [7].The authors suggested that the best average latency can be obtained by prioritizing the scheduling of redundant requests during server time that is memoryless and allows for the immediate deletion of extra copies of finished computational tasks. Even if slow-responding replica nodes are left in place, Lee et al. demonstrated that the average task latency can be decreased by properly scheduling redundant requests [8].

2.2. The state of domestic and international research in coded distributed computing

Each compute node in a distributed computing system has a distinct processing speed, which affects how long it takes to finish the assigned subtasks. The Master Node must wait for all of the

DOI: 10.54254/2755-2721/14/20230763

computation nodes to finish the subtasks and return the calculation results before it can recover the final results because the computation tasks are distributed among the computation nodes. As a result, the straggler effect, named after the slowest computational node, determines how long it takes to complete a calculation. Recently, academics have suggested coded distributed computing as a potential remedy for this issue, as a combination of distributed computing and coding theory methods. The effect of missing nodes can be reduced for linear algebraic operations like matrix multiplication by employing error-correcting codes, a technique known as coded computing. Coded computing makes it possible to reach the desired outcome for any defined subset of bases that finishes the overall computational task by introducing redundant computation. Convolution [9], Fourier transform [10], distributed gradient descent [11], sparse matrix multiplication [12], heterogeneous coded computing [13] and approximate computing [14] are just a few of the many calculations and uses that use coded computing.

3. Matrix multiplication: definition, example, theorem

3.1. Definition

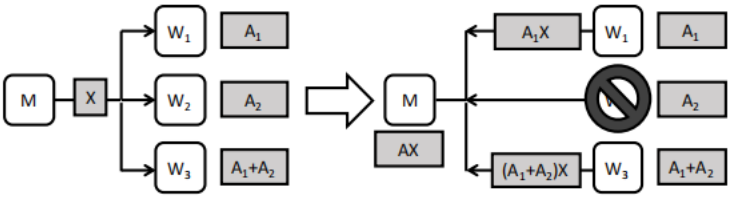

Figure 1. Multiplication of a coded matrix shown.

As shown in Figure 1. Data matrix A is divided into A1 and A2 submatrices. A1 is kept in node W1, A2 is kept in node W2, and A1 and A2 are kept in node W3. Each node receives X, multiplies it with the matrix it has stored, and sends the result to the master node. Keep in mind that the master node can recover AX immediately after receiving any two products, eliminating the need to wait for the slowest answer. Take a case, for instance. where A1X and (A1 + A2)X have been sent to the master node. It can recover A2X and subsequently AX by deducting A1X from (A1 + A2)X.

3.2. Example

For the Fibonacci series(f 1=1, f 2=1, f 3=2, f 4=3,…, fn=fn – 1 + fn 2)。Input n and m, Find the first n sums of fn ,which we defined as Sn , Output its value modulo m.

$$[f_n, f_{n+1}, S_n]\begin{bmatrix}0 & 1 & 0\\ 1 & 1 & 1\\ 0 & 0 & 1\end{bmatrix} = [f_{n+1}, f_{n+2}, S_{n+1}] \quad (F_n \cdot A = F_{n+1})$$

$$===> A = \begin{bmatrix}0 & 1 & 0\\ 1 & 1 & 1\\ 0 & 0 & 1\end{bmatrix} F_n = F_1 \cdot A_{n-1}, F_1 = [1,1,1],\ \mathtt{res}\ = F_n[2] \tag{1}$$

The matrix A and matrix S are first defined according to the derived formula, then the matrix multiplication part of the code is written, divided into two parts, matrix by matrix and matrix by vector. Then write the main program, read in the input numbers, write the fast power part of the code, and then call the program to obtain the result. Besides, binary groups can be used to reduce the time constant for better efficiency.

We can learn that for summation formulas like Fibonacci, you can find transitive matrices, and then you can use matrix multiplication and fast powers. All problems of the Fibonacci sequence in mathematics and signals can be solved in this way With high efficiency and high accuracy.

DOI: 10.54254/2755-2721/14/20230763

3.3. Theorem

Matrix multiplication is a computational process that is carried out in programmed computing in a distributed system with 1 master node and N worker nodes. It (arbitrarily) divides the two input matrices into blocks of p x m and p x n submatrix elements, respectively. Each input matrix's submatrices are separated into equal-sized ones, and each working node can only locally store two submatrices at a time. Moreover, each encoding technique takes various values for p, m, and n.

4. Coded computation: comparison of parallel gradient algorithms, schemes, decoding algorithm

4.1. Encoded computing

Encoding is known to lower storing costs in distributed storage systems and caching networks, as well as to assist in resolving uncertainty in remote communication [15].The idea of efficiently creating and developing coded multicast was originally proposed in the literature [16] and in the caching network of literature [17] and extended in literature [18] and literature [19], where caches prefetch parts of the content in a way that enables encoding during content delivery, thus minimising network traffic. Given that data exchange takes up a significant amount of time in the Map-Reduce computing framework, researchers have applied encoding techniques to the field of distributed computing with the goal of reducing the communication burden using a variety of flexible encoding techniques.

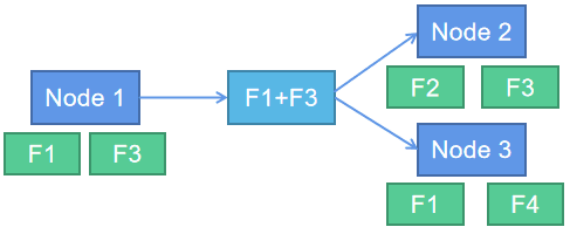

Figure 2. Transmission of signals from node 1 to node 2 and node 3 via heterodyne coding.

As shown in Figure 2, a dispersed system has three nodes: node 1, node 2, and node 3. Files F1 and F3 are kept in node 1, F2 and F3, and F1 and F4 are kept in node 3. In order for node 1 to send multicast encoded signals to nodes 2 and 3, it is now required to send files F1 stored in node 1 to node 2 and files F3 to node 3. If the unencoded technique is employed, files F1 and F3 must be delivered to nodes 2 and 3 respectively, requiring two signals to be sent at this point. This manner, only one signal, F1+F2, is required to satisfy the needs of both nodes. As a result, using the encoded approach results in a 50% reduction in communication overhead as compared to the method of transmitting data without encryption. The straightforward illustration in Figure 2. demonstrates how more data can be stored on each node to lessen the communication strain.

4.2. Examples

Example 1: The one-dimensional MDS coding scheme is an extension of the coding idea of adding redundant data to an input matrix, which is called one-dimensional MDS code (1D MDS code). The idea of this coding scheme is that the problem of multiplying a large matrix is considered as a problem of multiplying n small matrices, i.e., AT B = AT B=[AT b1 AT b2 … AT bn], and then the MDS coded matrix multiplication is applied to each of the n small matrices. Assuming N=nk, the working nodes are divided into n groups of size k, each dedicated to the computation of an AT bj. For example, for the first group, which is used to compute AT b1, the scheme first uses (k, m) MDS encoding to encode m submatrices of the same size divided along the columns of matrix A to obtain k encoding matrices, say a1 to ak, and then assigns the computation of ai T b1 to the i-th working node of this group, and so on. The computation time of the MDS encoding computation is determined by the

DOI: 10.54254/2755-2721/14/20230763

maximum of the computation time of the n groups, and the computation time of each group is determined by the mth worker node that completes the computation task among the k worker nodes in the group. Figure 3 shows an example of the 1D MDS encoding scheme for N = 3, m = 2, n = 1, and p = 1. The matrix H can be computed after the master node receives the computation results returned by any two of the three worker nodes.

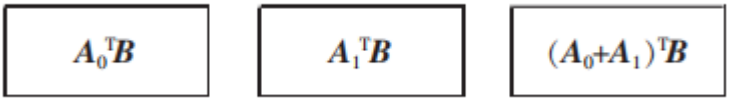

Figure 3. Illustration of 1D MDS code with 3 work nodes.

Example 2: Multiplicative matrix multiplication is a coding scheme based on multiplicative codes, which incorporates redundancy in both input matrices, but it is only designed for the case of m=n. The multiplicative code matrix multiplication coding scheme first assigns the working nodes into ($\sqrt{N} \times \sqrt{N}$) matrices, and then divides matrix A into m submatrices along the columns and encodes them into $\sqrt{N}$ coding matrices using ($\sqrt{N}$, m) MDS codes, and then assigns them to the $\sqrt{N}$ columns of the working nodes, where each column is assigned the same coding matrix. Similarly, the matrix B is divided into m sub-matrices along the columns, encoded and assigned to the $\sqrt{N}$ rows of the working nodes. According to the characteristics of MDS code, the master node can decode the whole row/column after obtaining any m results in that row/column. Therefore, the master server can decode the MDS code block iteratively over the rows and columns until the complete matrix H is output.

Figure 4 shows an example of the product code encoding scheme for N=9, m=2, n=2, p=1. The matrix H is computed by the master node after receiving the computation results from any 6 of the 9 working nodes.

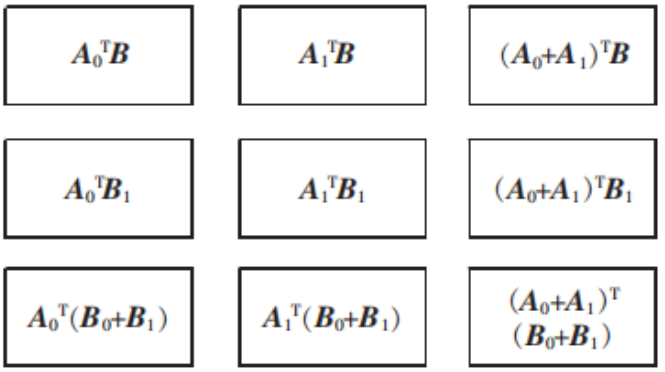

Figure 4. Illustration of product code with 9 work nodes.

Example 3: The polynomial code coding scheme uses ideas from coding theory to design intermediate computations on working nodes to achieve tolerance of dropped nodes. The scheme first divides the 2 input matrices equally along the columns into m and n.

$$A = [A_0 A_1 ... A_{m-1}] \tag{2}$$

$$B = [B_0\, B_1\, ...\, B_{n-1}] \tag{3}$$

Then assign a number xi to each working node i (i ∈ {0, 1, ..., N-1}), while making all xi different from each other, and store the following 2 encoding submatrices locally at working node i. Each

DOI: 10.54254/2755-2721/14/20230763

working node i continues multiplying the 2 locally stored encoding matrices to obtain the following result and sends it to the master node.

Due to the special algebraic structure of the designed computational strategy, the decoding process can be considered as a polynomial interpolation problem (or a problem of decoding Reed-Solomon codes) that can be solved efficiently. The main innovation and advantage of this polynomial encoding is that by carefully designing the algebraic structure of the encoded submatrices, it is possible to ensure that any mn intermediate computations on the working nodes are sufficient to recover the final product of matrix multiplications at the master node. In a sense, this creates an MDS structure in the intermediate computation, rather than just encoding the matrix as in the previous work.

An example of a polynomial code encoding scheme for N=5, m=2, n=2, p=1 is shown in Figure 5, where the matrix H is computed after the master node receives the computation results returned by any 4 of the 5 working nodes.

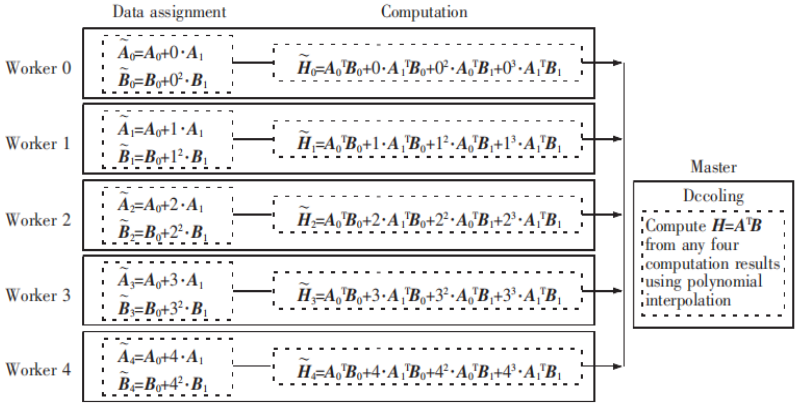

Figure 5. Illustration of polynomial code with 5 work nodes.

5. Conclusion

Because to its high fault tolerance, flexibility, openness, and quick calculation, distributed computing has become a widely utilized technique for large-scale task computing. How to apply coding theory approaches to distributed computing has been a popular research topic in the field of error-correcting codes in recent years as a result of the development of coding theory techniques and their exceptional performance in a variety of fields. Although dropped nodes can cause substantial processing lag, distributed computing can offer extremely dependable and fault-tolerant computing systems. Coded distributed computing is the term for the fusion of distributed computing with coding technology, and it is regarded as the most effective way to address the issue of dropped nodes. Thus, the research area of coded distributed computing is focused on how to address the issue of dropped nodes and decrease computational latency in distributed computing. This work investigates and analyzes the performance overhead of distributed machine learning-based coding methods. Three coding techniques are used to matrix multiplication, and their implementation and guiding principles are contrasted with uncoded schemes. The three coding schemes are contrasted for their benefits and drawbacks, which serves as a foundation for choosing the computational strategy. The paper also clarifies the principles and processes of distributed computing and compares them with traditional methods, analysing their superiority and practicality.

References

[1] Dean J, Barroso L A. The tail at scale [J]. Communications of the ACM, 2013, 56(2): 74-80.

[2] Ananthanarayanan G, Kandula S, Greenberg A G, et al. Reining in the Outliers in Map-Reduce Clusters using Mantri [C]. Osdi. 2010, 10(1): 24.

[3] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107-113

DOI: 10.54254/2755-2721/14/20230763

[4] Wang S, Liu J, Shroff N. Coded sparse matrix multiplication [C]. International Conference on Machine Learning. PMLR, 2018: 5152-5160.
[5] Wang D, Joshi G, Wornell G. Efficient task replication for fast response times in parallel computation [C]. The 2014 ACM international conference on Measurement and modeling of computer systems. 2014: 599-600.
[6] Shah N B, Lee K, Ramchandran K. When do redundant requests reduce latency? [J]. IEEE Transactions on Communications, 2015, 64(2): 715-722.
[7] Lee K, Pedarsani R, Ramchandran K. On scheduling redundant requests with cancellation overheads [J]. IEEE/ACM Transactions on Networking, 2016, 25(2): 1279-1290.
[8] Dutta S, Cadambe V, Grover P. Coded convolution for parallel and distributed computing within a deadline [C]. 2017 IEEE International Symposium on Information Theory (ISIT). IEEE, 2017: 2403-2407.
[9] Yu Q, Maddah-Ali M A, Avestimehr A S. Coded fourier transform [C]. 2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton). IEEE, 2017: 494-501
[10] Tandon R, Lei Q, Dimakis A G, et al. Gradient coding: Avoiding stragglers in distributed learning [C]. International Conference on Machine Learning. PMLR, 2017: 3368-3376
[11] Zaharia M, Konwinski A, Joseph A D, et al. Improving MapReduce performance in heterogeneous environments [C]. Osdi. 2008, 8(4): 7.
[12] Dutta S, Cadambe V, Grover P. " Short-Dot" computing large linear transforms distributedly using coded short dot products [C]. Proceedings of the 30th International Conference on Neural Information Processing Systems. 2016: 2100-2108.
[13] Reisizadeh A, Prakash S, Pedarsani R, et al. Coded computation over heterogeneous clusters [J]. IEEE Transactions on Information Theory, 2019, 65(7): 4227-4242.
[14] Ferdinand N S, Draper S C. Anytime coding for distributed computation [C]. 2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton). IEEE, 2016: 954- 960
[15] Qu, Lingxiao. Construction of a coded distributed computing scheme based on placement distribution tables[D].Guangxi Normal University, 2021.
[16] Maddah-Ali, Ali M, Niesen, et al. Fundamental Limits of Caching[J]. IEEETransactions on Information Theory, 2014, 60(5): 2856- 2867.
[17] Maddah-Ali M A, Niesen U. Decentralized coded caching attains order-optimal memory-rate tradeoff[C]. //Proceedings of the 51st Annual Allerton Conference on Communication, 2013: 421- 427.
[18] Ji M, Caire G, Molisch A F. Fundamental Limits of Caching in Wireless D2D Networks[J]. IEEE Transactions on Information Theory, 2016, 62(2): 849- 869.
[19] Karamchandani N, Niesen U, Maddah-Ali M A, et al. Hierarchical Coded Caching[J]. IEEE Transactions on Information Theory, 2016, 62(6): 3212- 3229.