文章编号:1005-0523(2021)03-0041-11

面向大规模矩阵乘法的编码计算性能研究

王 艳,王希龄,赖宏达,李念爽

(华东交通大学软件学院,江西 南昌 330013)

摘要:为了更好地使用编码计算提高分布式机器学习算法运行效率,需要对大规模矩阵乘法的编码计算方案性能开销进行充分的研究。该文考察了面向大规模矩阵乘法的编码计算方案的任务完成时间,同时也考虑了所有参与分布式计算的节点总的计算开销,给出了各个工作节点完成计算任务的时间,均匀分布场景下总的任务完成时间和集群机器总的计算时间的表达式,对比分析了3种编码方案的性能,并通过实验对比了不同情况对任务完成时间与计算节点总计算开销影响,提出了一个启发式算法,提供了不同编码计算方案的选择依据。

关键词:编码计算;分布式机器学习;矩阵乘法;掉队节点;性能研究

中图分类号:TP391 **文献标志码**:A

本文引用格式:王艳,王希龄,赖宏达,等. 面向大规模矩阵乘法的编码计算性能研究[J]. 华东交通大学学报,2021,38(3):41-51.

DOI:10.16749/j.cnki.jecjtu.20210706.012

Research on the Performance of Coding Calculation for Large-Scale Matrix Multiplication

Wang Yan, Wang Xiling, Lai Hongda, Li Nianshuang

(School of Software, East China Jiaotong University, Nanchang 330013, China)

Abstract: With the growth of machine learning algorithm models and data sets, a single node cannot effectively bear the computing and storage requirements required for large-scale training. A common solution is to run large-scale machine learning algorithms on distributed clusters. However, the performance of distributed clusters is significantly affected by stragglers. In recent studies, researchers have used coding calculations to solve the straggler problem, but the performance of coding calculation schemes for large-scale matrix multiplication has not been fully studied and analyzed. This paper examines the task completion time of the coding calculation scheme for large-scale matrix multiplication, and considers the total calculation overhead of all nodes participating in distributed computing. The expression of the task completion time for each working node to complete the calculation task according with the total time under the uniform distribution scenario and the total computing time of the cluster machines is given. The performance of the three coding schemes is compared and analyzed. The effects of different situations on the task completion time and the total computing cost of the computing node are compared through experiments, and a heuristic algorithm is proposed to provide the basis for the selection of different coding calculation schemes.

Key words: coding computing; distributed machine learning; matrix multiplication; lagging nodes; performance research

收稿日期:2021-01-21

基金项目:国家自然科学基金项目(61402172);江西省自然科学基金项目(20192BAB217006)

作者简介:王艳(1982—),副教授,博士,研究方向为分布式存储、分布式机器学习。E-mail:313624307@qq.com。

Citation format: WANG Y, WANG X L, LAI H D, et al. Research on the performance of coding calculation for large-scale matrix multiplication[J]. Journal of East China Jiaotong University, 2021, 38(3): 41-51.

互联网的飞速发展和大数据处理框架的进步推动了大规模的机器学习部署,并促进了分布式机器学习算法在更多领域中的应用[1-5]。为了更高效地利用大数据训练更准确的大模型,在机器学习中引入大量矩阵乘法,一旦采用了分布式机器学习模式,就意味着大量矩阵乘法的分布式计算。然而,分布式集群中存在某些计算节点由于各种因素的影响(如节点失效、系统故障、通信瓶颈等),计算速度会以某种随机的方式变慢,从而使分布式机器学习算法执行时间增加,成为分布式计算系统的主要瓶颈[6],这种节点被称作掉队节点。目前减缓掉队节点影响的方法主要是增加某种形式的"计算冗余",例如,采用副本的方式在多个节点上执行相同的计算任务[7]。然而,最近的研究结果表明,编码计算可以更加有效地减轻掉队节点的影响[8-12],同时相较于副本方案成本更低。

Tandon 等[13]研究了分布式系统中梯度计算的最优编码设计,提出了一种新的编码计算方案,用于计算函数的和,展示了对梯度进行编码可以提供同步梯度下降法对失效节点和掉队节点的容忍。他们根据工作节点运行速度变慢的程度将掉队节点分成完全掉队节点(Full Stragglers)和部分掉队节点(Partial Stragglers)两种,针对这两种情况,提出了一种编码方案来实现机器学习集群对掉队节点的鲁棒性。Ye 等[14]提出了通信计算高效梯度编码方案,它从计算负载、掉队节点容忍和通信成本 3 个方面描述了梯度计算的基本权衡。Li 等[15]提出了一种 MapReduce 的编码框架,称为"Coded MapReduce",它在 $r(r\in\mathbf{N})$ 个精心选择的节点上分配每个任务的映射计算,以使网络内的节点减少 r 倍的通信负载。Li 等[16]将文献[15]中提出的编码思想应用于 TeraSort,提出了一种新的分布式排序算法"Coded TeraSort",大大提高了 Hadoop MapReduce 中 TeraSort 基准的执行时间。Reisizadeh 等[17]研究由各种不同性能的计算机组成的通用异构分布式计算集群,提出了一个编码框架,通过交换冗余来减少计算延迟,从而加速存在掉队节点的异构集群的分布式计算。Ferdinandn 等[18]提出了一种用于近似矩阵乘法的任意时刻编码方案,通过近似计算的形式来加速分布式计算。Dutta 等[19]考虑了存在掉队节点的情况下,使用并行处理器计算两个长向量的卷积问题,提出了存在截止时间的卷积问题的编码计算方案。Park 等[20]提出了一个由工作节点组成的层次计算结构。

上述研究都是通过提出编码方案来增加对掉队节点的鲁棒性,并未对各自编码方案的性能开销展开研究,本文重点考察了面向分布式机器学习的各编码计算方案的任务完成时间和机器计算总时间两类开销。本文定义了面向分布式机器学习的编码计算的两个性能指标:一是某个使用该编码方案的计算任务完成时间,即完成整个计算任务所需要的时间;二是整个计算任务的机器计算总时间,即整个分布式计算系统中所有工作节点对该计算任务进行计算的时间的总和。通过计算由 n 个工作节点组成的分布式系统中第 i 个完成计算任务的节点的密度函数,给出工作节点计算任务完成时间符合均匀分布场景下计算任务的完成时间和机器计算总时间的表达式;对比分析了这一场景下 3 种应用于矩阵乘法的编码方案的完成时间和机器计算总时间,并通过实验对比了指数分布场景下不同情况对任务完成时间与计算节点总计算开销影响,提供了计算方案选择的依据。

1 分布式计算开销示例

在机器学习算法中,我们经常要执行矩阵乘法的操作。我们考虑这样一个矩阵乘法问题:通过输入矩阵 $\boldsymbol{A}$ 和向量 $\boldsymbol{X}$ 计算出矩阵 $\boldsymbol{B}=\boldsymbol{AX}$。由于工作节点的内存大小有限,矩阵 $\boldsymbol{A}$ 按列被分成 3 个大小相等的子矩阵,即 $\boldsymbol{A}=[\boldsymbol{A}_1\ \boldsymbol{A}_2\ \boldsymbol{A}_3]$,计算工作分别在图 1 和图 2 所示的两个分布式计算系统中进行的。

图 1 所示的分布式计算系统由 1 个主节点和 3 个工作节点组成,每个工作节点分别预先存储子矩阵 $\boldsymbol{A}_1,\boldsymbol{A}_2,\boldsymbol{A}_3$ 中的一个,且各个节点存储的矩阵互不相同。主节点将向量 $\boldsymbol{X}$ 发送给每个工作节点,工作节点 $\boldsymbol{W}_i(i=1,2,3)$ 将本地存储的矩阵与向量 $\boldsymbol{X}$ 进行乘法运算。主节点接收到 3 个工作节点的计算结

果，可计算出 $\boldsymbol{B}$。假设图 1 的计算系统中节点 $\boldsymbol{W}_1$，$\boldsymbol{W}_2$ 和 $\boldsymbol{W}_3$ 的计算任务完成时间（T）分别为 2,6 ms 和 10 ms,那么第一个计算系统 T=10 ms,机器计算总时间 C=2+6+10=18 ms。

图 2 所示的分布式计算系统由 1 个主节点和 5 个工作节点组成,使用(5,3)MDS 编码将矩阵 $\boldsymbol{A}$ 分成的 3 个子矩阵编码为 5 个编码矩阵 $\boldsymbol{A}_1$,$\boldsymbol{A}_2$,$\boldsymbol{A}_3$,$\boldsymbol{A}_1+\boldsymbol{A}_2+\boldsymbol{A}_3$,$\boldsymbol{A}_1+2\boldsymbol{A}_2+3\boldsymbol{A}_3$,并分别预先存储在 5 个工作节点 W_1,W_2,W_3,W_4,W_5 中。主节点将向量 $\boldsymbol{X}$ 发送给每个工作节点,工作节点 $\boldsymbol{W}_i(i$=1,2,3,4,5)将本地存储的矩阵与向量 $\boldsymbol{X}$ 进行乘法运算,并在完成计算任务后,将结果发送回主节点。一旦主节点接收到 5 个计算结果中的任意 3 个,其他工作节点就会停止工作,主节点就可以计算出 $\boldsymbol{AX}$。例如,当主节点接收到 $\boldsymbol{A}_1\boldsymbol{X}$,$\boldsymbol{A}_3\boldsymbol{X}$ 和$(\boldsymbol{A}_1+\boldsymbol{A}_2+\boldsymbol{A}_3)\boldsymbol{X}$ 时,它可以通过计算$(\boldsymbol{A}_1+\boldsymbol{A}_2+\boldsymbol{A}_3)\boldsymbol{X}-\boldsymbol{A}_3\boldsymbol{X}-\boldsymbol{A}_1\boldsymbol{X}$ 得到 $\boldsymbol{A}_2\boldsymbol{X}$,然后计算出 $\boldsymbol{AX}$。这一编码方案,相较于未编码方案必须接收到全部计算结果才能完成最终计算,能容忍 2 个节点计算速度较慢或完全失效。

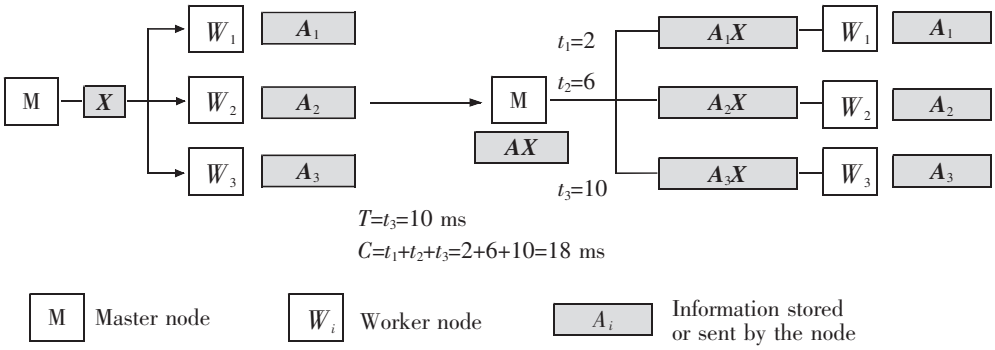

图 1　未编码方案示例

Fig.1　Illustration of uncoded scheme

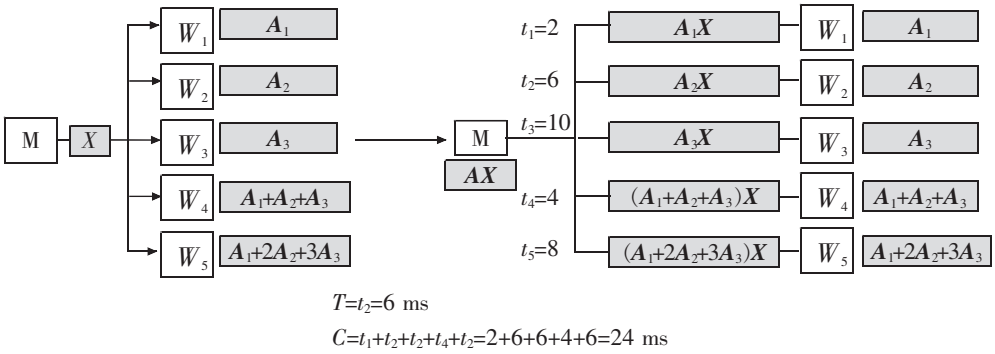

图 2　编码方案示例

Fig.2　Illustration of coding scheme

假设图 2 中 5 个计算节点的完成时间分别为 2,4,6,8,10 ms，主节点在接收到任意 3 个节点的计算结果后就立刻通知剩余的其他 2 个节点停止计算(根据 MDS 编码性质,任意 3 个节点的计算已足够得到矩阵 $\boldsymbol{B}$),所以任务完成时间 T=6 ms,机器计算总时间 C=2+4+6+6+6=24 ms。

从上述结果可以看出，相较于未编码方案,虽然编码方案的任务完成时间减少了,但是机器计算总时间比未编码方案更高。

2　模型建立

在一个由 1 个主节点和 n 个工作节点组成的分布式计算系统中，由于每个工作节点内存有限,先将总的计算任务分成若干个大小相等的子任务,然后使用编码技术将这些子任务编码成 n 个大小相同的编码计算任务,发送给每个工作节点进行计算,主节点在接收到任意 k 个工作节点的计算结果后,即可计算出最终结果。

我们通过以下两个性能指标评估一个编码方案的性能：

任务完成时间(T)：从主节点将子任务分配给每个工作节点开始，直到主节点接收到足够多的工作节点提交的计算结果，能够完成整个计算任务所需要的时间；

机器计算总时间(C)：整个分布式计算系统中所有工作节点对该计算任务进行计算的时间的总和。

假设这 n 个工作节点的计算任务完成时间分别为 $t_1,t_2,\cdots,t_n$，记 X_i 为这些任务完成时间中第 i 个最小的完成时间

$$T\triangleq X_k \tag{1}$$

$$C\triangleq \sum_{i=1}^{k} X_i+(n-k)\ X_k \tag{2}$$

由于接收到 k 个工作节点的计算结果后，主节点即可计算出最终结果，任务完成时间 T 为主节点接收到第 k 个计算结果的时间，即 X_k。此时分布式计算系统的计算任务就已完成，主节点会向所有工作节点发送一个停止工作的信号，未完成计算任务的$(n-k)$个工作节点也在 X_k 时刻停止计算，机器计算总时间 C 即为式(2)所示。

由于工作节点在进行计算任务时可能会由于各种原因导致计算速度相较于正常节点更慢或是更快，使得每个工作节点完成计算任务的时间可能会不同，每个工作节点计算任务的完成时间 $t_i(i\in\{1,2,\cdots,n\})$是一个独立同分布的连续随机变量，具有概率分布 F 和密度函数 f。为了得到 X_i 的分布，因 X_i 小于或等于 x 当且仅当这 n 个节点的计算时间 $X_1,X_2,\cdots,X_n$ 至少有 i 个小于或等于 x。

$$P\{X_i\leqslant x\}=\sum_{k=1}^{n}\binom{n}{k}(F(x))^k(1-F(x))^{n-k} \tag{3}$$

微分可得 X_i 的密度函数如下

$$\begin{aligned} f_{X_i}(x)&=f(x)\sum_{k=1}^{n}\binom{n}{k}k(F(x))^{k-1}(1-F(x))^{n-k}\\ &=\frac{n!}{(n-i)!\ (i-1)!}f(x)F(x)^{i-1}(1-F(x))^{n-i} \end{aligned} \tag{4}$$

上面的密度十分直观，为了使 X_i 等于 x，n 个值 $X_1,X_2,\cdots,X_n$ 中的 $i-1$ 个必须小于 x，$n-i$ 个必须大于 x，而且有一个必须等于 x。现在，对于指定的 $i-1$ 个 X_i 的每个成员都小于 x，指定的 $n-i$ 个 X_i 的每个成员都大于 x，而余下的值等于 x 的密度概率是 $(F(x))^{i-1}(1-F(x))^{n-i}f(x)$。于是，由于 n 个随机变量分成这样 3 组不同划分数为 $\frac{n!}{(n-i)!\ (i-1)!}$，得到了上述的密度函数。

在分布式计算系统中，存在这样一种情形，工作节点的计算速度不仅仅会因为某些原因变慢，还会因为一些原因变快。例如系统中大多数的计算节点要同时执行若干计算任务，而当一些节点只执行更少的甚至只有一个计算任务时，计算速度会随之不断增加。假设工作节点在执行一个计算任务时，节点的计算完成时间最快为 a，最慢为 b，每个工作节点的完成时间在(a,b)上是等可能的，那么这些工作节点的计算完成时间在 (a,b) 上服从均匀分布。另外，由于 b 的取值可以无限大，而 a 的取值最小为一个大于 0 的值，为了便于实际计算，可以取大多数节点的完成时间为 $\frac{a+b}{2}$，再根据具体情形确定 a 和 b 的值。

均匀分布的概率分布为 $F(x)=\frac{x-a}{b-a}$，密度函数为 $f(x)=\frac{1}{b-a}$，$a<x<b$，代入式(4)可得

$$\begin{aligned} f_{X_i}(x)&=\frac{n!}{(n-i)!\ (i-1)!}\cdot\frac{1}{b-a}\cdot\left(\frac{x-a}{b-a}\right)^{i-1}\left(1-\frac{x-a}{b-a}\right)^{n-i}\\ &=\frac{n!}{(n-i)!\ (i-1)!}\cdot\frac{(x-a)^{i-1}(b-x)^{n-i}}{(b-a)^n} \end{aligned} \tag{5}$$

求解该密度函数的期望为

$$\begin{aligned} E[f_{X_i}(x)]&=\int_a^b\frac{n!}{(n-i)!\ (i-1)!}\cdot\frac{(x-a)^{i-1}(b-x)^{n-i}}{(b-a)^n}x\ \mathrm{d}x\\ &=a+\frac{(b-a)i}{n+1} \end{aligned} \tag{6}$$

任务完成时间，即第 i 个计算任务的完成时间为

$$T=a+\frac{(b-a)i}{n+1} \tag{7}$$

机器计算总时间为

$$\begin{aligned} C&=\left(a+\frac{b-a}{n+1}\right)+\left(a+\frac{2(b-a)}{n+1}\right)+\cdots+\left(a+\frac{i(b-a)}{n+1}\right)+\\ &(n-i)\left(a+\frac{i(b-a)}{n+1}\right)=na+\frac{i(b-a)}{n+1}\left(\frac{1-i}{2}+n\right) \end{aligned} \tag{8}$$

3 3种编码方案性能的对比分析

对于给定的任何一种编码策略，本文将完成一

个计算任务所需等待的最小工作节点数定义为该编码策略的恢复阈值。也就是说,如果任何大小不小于恢复阈值的工作节点子集完成了它们的工作,主节点就能够计算出最终结果。在本文中,恢复阈值可以理解为:当主节点接收到这 n 个工作节点中最快的前 i 个工作节点的计算结果后,即可计算出最终结果。

在计算各编码方案的任务完成时间 T 和机器计算总时间 C 之前,必须要说明的是,由于编码方案的不同,每种编码方案中工作节点的计算量是不同的,因而计算时间也不相同。这在模型中可以体现为,当工作节点的计算时间符合均匀分布的场景下,各个编码方案的区间(a,b)会有不同。为了方便,本文将进行一次乘法的计算量记为 1,进行加法的计算量忽略不计,如一个 $1\times n$ 的向量与一个 $n\times 1$ 的向量相乘的计算量为 n,而一个 $n\times 1$ 的向量与一个 $1\times n$ 的向量相乘的计算量为 n^2。

为了更好的对比各个编码方案任务完成时间 T 和机器计算总时间 C,还需要给这些编码方案构造一个相同的计算场景,目标都是通过输入矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 计算出 $\boldsymbol{H}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{B}$,如图 3 所示。矩阵 $\boldsymbol{A}$ 的维度为 $\boldsymbol{Z}\times\boldsymbol{X}$,矩阵 $\boldsymbol{B}$ 的维度为 $\boldsymbol{Z}\times\boldsymbol{Y}$,计算任务是在拥有 1 个主节点和 N 个工作节点的分布式系统中进行。其中,两个输入矩阵分别被(任意)划分为 $p\times m$ 和 $p\times n$ 个子矩阵块,每一个输入矩阵划分的子矩阵大小是相同的,且每个工作节点只能在本地存储 2 个子矩阵。另外,每个编码方案的 p,m,n 的取值是不同的。

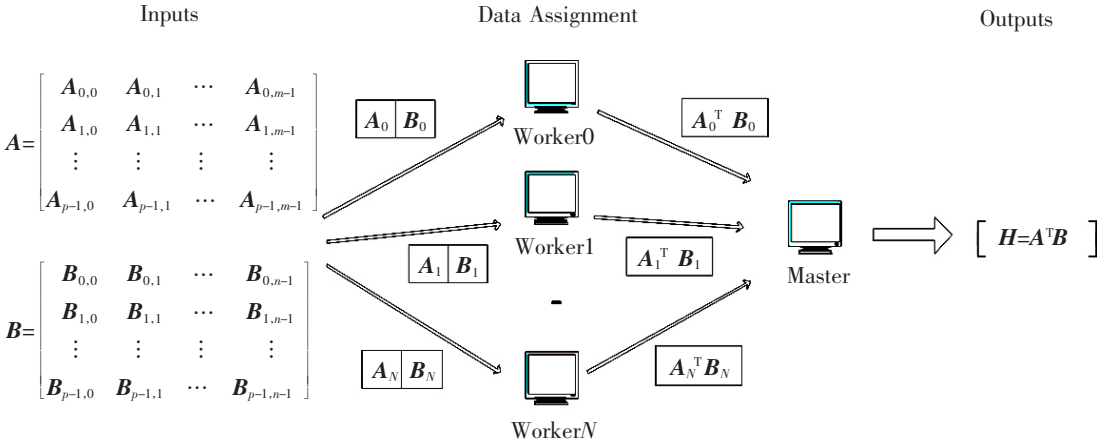

图 3　矩阵乘法问题示例

Fig.3　Example of matrix multiplication problem

目前最常用于矩阵乘法的编码方案为:一维 MDS 编码、乘积编码、多项式编码。本文将对这 3 种编码方案进行介绍以及实验对比其性能。

3.1　一维 MDS 编码方案

一维 MDS 编码方案是 Lee 等[21]对文献[22]的编码思想进行扩展得出的,将 MDS 码在一个输入矩阵中加入冗余数据这种方法称为一维 MDS 码(1D MDS 码)。该编码方案的思想是:将大矩阵乘法的问题看作 n 个小矩阵乘法的问题,即 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{B}=[\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}_1\ \ \boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}_2\ \cdots\ \boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}_n]$,然后对 n 个小矩阵中的每一个分别应用 MDS 编码矩阵乘法。假设 $N=nk$,工作节点被分成大小为 k 的 n 个组,每个组都专门计算一个 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}_j$。例如对于第一个组,它用于计算 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}_1$,该方案首先使用$(k,m)$MDS 编码对矩阵 $\boldsymbol{A}$ 的沿列分成的 m 个大小相同的子矩阵进行编码,以获得 k 个编码矩阵,比如 $\boldsymbol{a}_1$ 到 $\boldsymbol{a}_k$,然后将 $\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{b}_1$ 的计算分配给这组的第 i 个工作节点,以此类推。MDS 编码计算的计算时间由 n 个组的计算时间中的最大值决定,每个组的计算时间由组中 k 个工作节点中第 m 个完成计算任务的工作节点决定。

图 4 所示为 $N=3$,$m=2$,$n=1$,$p=1$ 时的 1D MDS 编码方案示例,主节点接收到 3 个工作节点中的任意 2 个返回的计算结果后,即可计算出矩阵 $\boldsymbol{H}$。

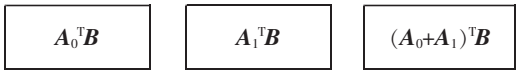

图4 3工作节点的1D MDS码示例
Fig.4 Illustration of 1D MDS code with 3 work nodes

1D MDS编码方案的恢复阈值为$N-\frac{N}{n}+m$，每个工作节点的计算量为$\frac{ZXY}{mn}$，在均匀分布场景下该方案的任务完成时间为$\frac{ZXY}{mn}a+\frac{\frac{ZXY}{mn}(b-a)(N-\frac{N}{n}+m)}{N+1}$，机器计算总时间为$\frac{ZXYN}{mn}a+\frac{\frac{ZXY}{mn}(N-\frac{N}{n}+m)(b-a)}{N+1}(\frac{1-(N-\frac{N}{n}+m)}{2}+N)$。其中$a,b$分别为工作节点在执行一个计算任务时的最快计算完成时间和最慢计算完成时间，每个工作节点的完成时间在(a,b)上是等可能的。

3.2 乘积码编码方案

乘积码矩阵乘法是Lee等[21]中提出的一种基于乘积码的编码方案，它在两个输入矩阵中都加入了冗余，但是该方案只是针对$m=n$的情况设计的。乘积码矩阵乘法编码方案先把工作节点分配成$(\sqrt{N}\times\sqrt{N})$的矩阵，接着将矩阵$\boldsymbol{A}$沿着列分成m个子矩阵，用$(\sqrt{N},m)$MDS码编码成$\sqrt{N}$个编码矩阵，然后分配给工作节点的$\sqrt{N}$列，其中每一列分配的编码矩阵是相同的。同理，将矩阵$\boldsymbol{B}$沿着列分成m个子矩阵，编码后分配给工作节点的$\sqrt{N}$行。根据MDS码的特性，主节点在获得该行/列中的任意m个结果后，即可对整行/列进行解码。因此，主服务器可以在行和列上迭代地解码MDS编码块，直到输出完整的矩阵$\boldsymbol{H}$为止。

图5所示为$N=9,m=2,n=2,p=1$时的乘积码编码方案示例，主节点接收到9个工作节点中的任意6个返回的计算结果后，即可计算出矩阵$\boldsymbol{H}$。

乘积码编码方案的恢复阈值为$2(m-1)\sqrt{N}-(m-1)^2+1$，每个节点的计算量为$\frac{ZXY}{mn}$，在均匀分布场景下该方案的任务完成时间为$\frac{ZXY}{mn}a+\frac{\frac{ZXY}{mn}(b-a)((2m-1)\sqrt{N}-(m-1)^2+1)}{N+1}$，机器计算总时间为$\frac{ZXY}{mn}Na\cdot\frac{\frac{ZXY}{mn}((2m-1)\sqrt{N}-(m-1)^2+1)}{N+1}(\frac{1-((2m-1)\sqrt{N}-(m-1)^2+1)}{2}+N)$。

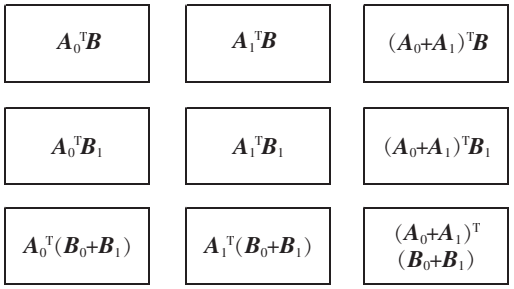

图5 9工作节点的乘积码示例
Fig.5 Illustration of product code with 9 work nodes

3.3 多项式码编码方案

多项式码编码方案是Yu等[23]提出的一种编码计算策略，利用编码理论的思想来设计工作节点上的中间计算，以实现对掉队节点的容忍。该方案首先沿着列将2个输入矩阵分别平均分成m和n个大小相等的子矩阵

$$\boldsymbol{A}=[\boldsymbol{A}_0\ \ \boldsymbol{A}_1\ \cdots\ \boldsymbol{A}_{m-1}]$$
$$\boldsymbol{B}=[\boldsymbol{B}_0\ \ \boldsymbol{B}_1\ \cdots\ \boldsymbol{B}_{n-1}] \tag{9}$$

然后给每个工作节点$i(i\in\{0,1,\cdots,N-1\})$分配一个数字$x_i$，同时使得所有的$x_i$都互不相同，在工作节点$i$本地存储以下2个编码子矩阵

$$\widetilde{\boldsymbol{A}}_i=\sum_{j=0}^{m-1}\boldsymbol{A}_j x_i^j,\widetilde{\boldsymbol{B}}_i=\sum_{j=0}^{n-1}\boldsymbol{B}_j x_i^{jm} \tag{10}$$

每个工作节点i对本地存储的2个编码矩阵继续乘法运算，得到以下结果并将其发送给主节点

$$\widetilde{\boldsymbol{H}}_i=\widetilde{\boldsymbol{A}}_i^T\widetilde{\boldsymbol{B}}_i=\sum_{j=0}^{m-1}\sum_{j=0}^{n-1}\widetilde{\boldsymbol{A}}_i^T\boldsymbol{B}_k x_i^{j+km} \tag{11}$$

也就是以下多项式在$x=x_i$处的值

$$h(x)\triangleq\sum_{j=0}^{m-1}\sum_{k=0}^{n-1}\boldsymbol{A}_j^T\boldsymbol{B}_k x_i^{j+km} \tag{12}$$

由于所设计的计算策略具有特殊的代数结构，解码过程可以看作是一个多项式插值问题（或是一个解码Reed-Solomon码的问题），可有效求解。这种多项式编码的主要创新之处和优点在于，通过精心设计编码子矩阵的代数结构，可以确保工作节点上的任何mn个中间计算都是以在主节点上恢复矩阵乘法的最终乘积。从某种意义上来说，这是在中间

计算中创建了一个 MDS 结构，而不是像以前的工作那样仅仅是对矩阵进行编码。

图 6 所示为 N=5,m=2,n=2,p=1 时的多项式码编码方案示例,主节点接收到 5 个工作节点中的任意 4 个返回的计算结果后,即可计算出矩阵 $\boldsymbol{H}$。

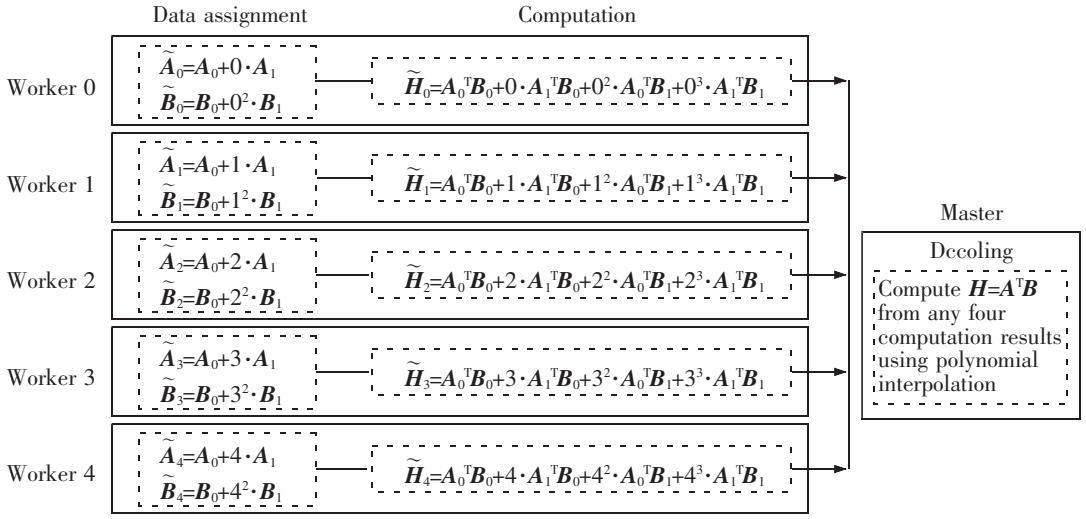

图 6　5 工作节点的多项式码示例

Fig.6　Illustration of polynomial code with 5 work nodes

多项式码编码方案的恢复阈值为 mn，每个节点的计算量为$\frac{ZXY}{mn}$，在均匀分布场景下该方案的任务完成时间为$\frac{ZXY}{mn}a+\frac{ZXY(b-a)}{N+1}$,机器计算总时间为$\frac{ZXYN}{mn}a+\frac{ZXY(b-a)}{N+2}(\frac{1-mn}{N+2}+N)$。

另外,在多项式编码的基础上,Yu 等[24]提出了纠缠多项式编码方案，实现了 $pmn+p-1$ 的恢复阈值;Dutta 等[25]提出了 MatDot 编码方案,实现了 $2m-1$ 的恢复阈值。由于这几种方案都是通过设计编码子矩阵的代数结构实现对掉队节点的容忍,本文仅将多项式编码方案与其他编码方案进行对比分析。

4　实验及结果

为了更加直观的比较各个编码方案的任务完成时间 T 和机器计算总时间 C,我们通过实验对3种编码方案的性能指标进行了测试。

本文使用 Python 语言实现了 3 种编码方案的编码计算过程。当编码参数设置为 a=1,b=9,X=Y=Z=4,N=16 时，本文分别对未编码方案和 3 种编码方案的任务完成时间 T 和机器计算总时间 C 进行了数值上的测量,实验结果如表 1 所示。

表 1　未编码方案和 3 种编码方案的对比

Tab.1　Comparison between uncoded scheme and three coding schemes

方案	其他参数值	T/ms	C/ms
未编码方案	m=2,n=2,p=1	118.4	320
1D MDS 码	m=2,n=2,p=1	91.2	1 121.9
乘积码	m=2,n=2,p=1	76.2	1 008.9
多项式码	m=2,n=2,p=1	46.1	692.7

从表中结果可以看出,编码方案与未编码方案相比:编码方案的任务完成时间普遍要短,这是因为每种编码方案只需等待达到其恢复阈值的工作节点子集完成计算任务即可得到最终结果,而不必等待所有节点完成计算任务;但是编码方案的机器计算总时间普遍更长，这是因为相较于未编码方案,编码方案使用了更多的工作节点来完成计算任务。将 3 种编码方案进行对比:乘积码的任务完成时间和机器计算总时间相较于 1D MDS 编码方案都要短,这是由于乘积码的恢复阈值更低;多项式码在 3 种编码方案中,任务完成时间和机器计算总

时间的表现都是最优,这是因为在给定其他参数情况下3种编码方案 p,m,n 的取值都相同时,多项式码的恢复阈值远小于另外两种编码方案,但是该方案额外增加了主节点的编解码,是以增加主节点的计算时间为代价的。

4.1 编码方案的参数选择

选定一个合适的编码方案之后,比如某用户准备使用多项式编码计算方案对2个大小为100×100的矩阵进行乘法计算,若该用户选择将两个输入矩阵分别划分为4个大小相同的子矩阵,即 m=n=4,假设该用户选择在20个工作节点上执行计算任务,且在该用户选择的工作节点的工作环境下 a=1,b=9,那么此时任务完成时间 T=443 252 ms,机器计算总时间 C=4 824 405 ms。若是该用户选择将两个输入矩阵分别划分为4个大小相同的子矩阵,即 m=n=5,假设该用户选择在29个工作节点上执行计算任务,且在该用户选择的工作节点的工作环境下 a=1,b=9,那么此时任务完成时间 T=306 667 ms,机器计算总时间 C=4 573 333 ms。从上述例子中可以看出,不同的参数选择会导致机器计算总时间 C,即资源消耗的不同。那么我们又面临着这样一个新的问题:如何选择编码方案的参数,使得用于编码计算任务的资源消耗最少。

将这一问题使用数学语言可以描述为:在给定工作节点数量 N 和任务完成时间 T 的限制时,如何选择 m,n,N 和其他参数的值,使得机器计算总时间C最小。

要研究选择 m,n,N 和其他参数的值,使得机器计算总时间 C 最小,我们首先要了解机器计算总时间 C 与这些参数之间有什么关系,即参数变化时 C 的变化趋势。另外,由于对工作节点数量 N 和任务完成时间 T 的限制可以理解为给定了 N 和 T 的上限,并不代表着所给工作节点数量 N 和任务完成时间 T 的值就是最终结果,我们还需要对工作节点数量 N 和任务完成时间 T 进行研究,分析这两个参数之间、两个参数与机器计算总时间 C 之间以及两个参数与其他参数之间的关系。

我们首先对前文求出的3种编码方案的任务完成时间 T 和机器计算总时间 C 的表达式进行研究。从3种编码方案性能指标的表达式中,可以很明显地观察到,当其他参数保持不变时,增加 Z,X,Y 的值,T 和 C 都会随之增加;增加 mn 的值,T 和 C 都会随之减小。但是当 N 增加时,T 和 C 的值不容易判断,本文又对工作节点数 N 变化的情况下3种编码方案的 T 和 C 的变化情况进行了实验分析。在其他参数分别设置为 a=1, b=9,X=Y=Z=100,m=n=5,p=1 时,计算出 N 分别取值为 30,35,40,45,50,55,60,65,70,80,90,100,120,160,200,240,280,320,360 时各编码方案的 T 和 C 的值,如图7所示。

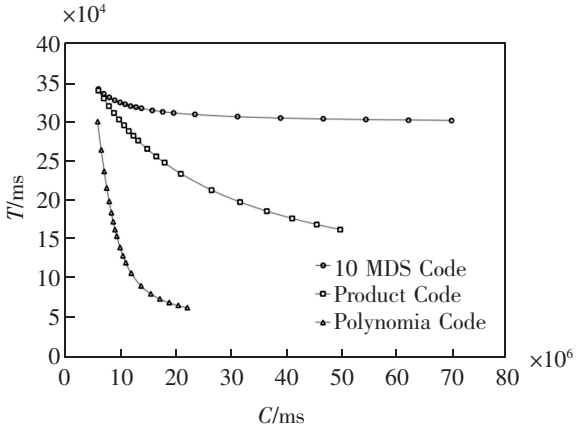

图7 3种编码方案在不同 N 取值下的 T 和 C 变化情况

Fig.7 Changes of T and C in three coding schemes with different values of N

从图7中可以看出,随着工作节点数 N 的增加,3种编码方案的机器计算总时间 C 都在不断增加,而任务完成时间 T 都在不断减小,这说明增加更多的冗余工作节点,能更快的完成计算任务,但是计算任务成本开销也会相应地增加。另外,从图中还可以看出,在 C 增加到一定程度后,T 的减小会变得缓慢,这说明增加的冗余达到一定程度之后,再继续增加冗余,成本开销依然会继续增加,但是任务完成时间却减少的很慢甚至不再减少,此时收益很低。

当输入矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 划分成的子矩阵数量不同,即参数 m,n 不同时,我们通过实验对编码方案的任务完成时间 T 和机器计算总时间 C 进行了研究。以多项式码编码方案为例,当 a=1,b=9,X=Y=Z=4 时,本文分别给出了 m,n 取值不同的情况下 T 和 C 的变化情况,研究结果如图8和图9所示。

从图8可以看出,当给定 N 时,选择 m×n 较大的输入矩阵划分方案,可以得到较小的 T,这是因为当划分成的子矩阵越小时,每个工作节点的计算量就越小,因而工作节点的工作时间也就越短。但是,

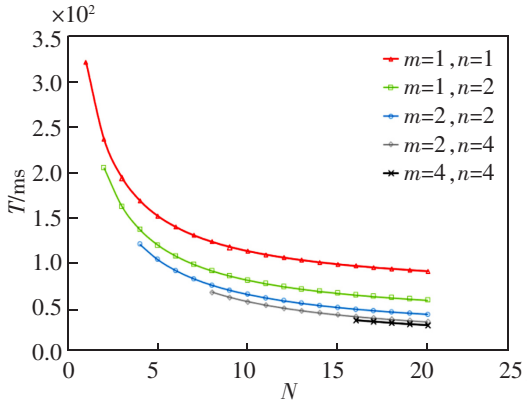

图 8　*m* 和 *n* 值不同时多项式编码框架的 *T* 的变化情况

Fig.8　Changes of *T* for polynomial code scheme with different values of *m* and *n*

并非是选择的 $m\times n$ 越大的输入矩阵划分方案,编码计算方案的整体运行时间就越短。因为选择的 $m\times n$ 越大,根据多项式编码方案的编码原理,进行计算所需要的工作节点数量自然也就越多。另外,一个编码计算方案的运行过程并不是只涉及工作节点的工作时间,还有主节点在对初始子矩阵进行编码生成编码子矩阵以及对编码子矩阵进行解码得到最终计算结果所花费的编解码时间。多项式方案需要给每个工作节点 i 发送以下两个编码子矩阵:

$\widetilde{A}_i=\sum_{j=0}^{m-1}A_j x_i^j$,$\widetilde{B}_i=\sum_{j=0}^{n-1}B_j x_i^{jm}$。

可以看出,当 m 和 n 的取值越大时,编码子矩阵就越复杂,生成编码子矩阵以及对编码子矩阵进行编解码所需要的时间也就越长。m 和 n 的取值越大并不能使进行编码计算任务花费的全部时间越短。

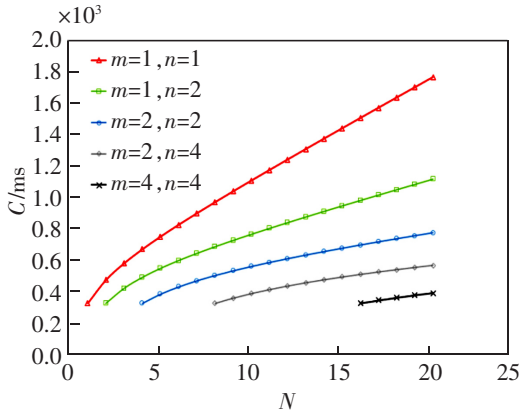

图 9　*m* 和 *n* 值不同时多项式编码框架的 *C* 的变化情况

Fig.9　Changes of *C* for polynomial code scheme with different values of *m* and *n*

从图 9 可以看出,当给定 N 时,选择 $m\times n$ 较大的输入矩阵划分方案,可以得到较小的 C。这是因为当工作节点的总数量保持不变时,每个工作节点的工作时间减少,机器计算总时间自然也就随之减少。此外,当工作节点数量 N 与多项式码的恢复阈值 mn 相等时,也就是说在分布式计算系统中没有冗余存在,此时无论 N 的取值如何变化,多项式编码方案的机器计算总时间 C 总是相等的,这是由 C 的计算表达式决定的,这也是多项式编码方案的机器计算总时间 C 如图 7 变化的原因。

4.2　参数的启发式算法

在对 4.1 节实验结果进行分析的基础上,本文给出了在给定 T 和 N 的上限时,如何选取 m,n,T 的值来最小化机器计算总时间 C 的启发式算法,算法过程如下所示:

输入:矩阵维度 X,Y,Z,计算节点完成最快时间 a,最慢时间 b,给定 T 和 N 的上限;

输出:计算总时间 C 的预测值,及最小化 C 的参数 m,n 的取值,任务完成时间 T 和工作节点数量 N;

1) 根据输入矩阵大小求出所有可用的矩阵划分方案 m 和 n 的可能值;

2) 根据用户所给的任务完成时间 T 的限制选择合适的 m 和 n 的取值;

3) 根据工作节点数量 N 的上限判断在给定 T 和上限内是否能完成计算工作;

4) 若能则给出使得机器计算总时间 C 最小的各项参数的取值。

在算法中输入给定的任务完成时间 T 和工作节点数量 N 的上限以及需要进行乘法计算的矩阵大小,然后算法会输出可以使得机器计算总时间 C 最小化的参数 m,n 的取值,任务完成时间 T 和工作节点数量 N 的预测值以及机器计算总时间 C 的预测值。

当算法输入为 a=1,b=9,X=Y=Z=100,N=46,T=200 000 ms 时,输出为 m=2,n=20 或 m=4,n=10 或 m=10,n=4 或 m=20,n=2,T=195 212.8 ms,C=5 660 638.3 ms。

在 mn 不同取值下 C 的取值如表 2 所示,可以看出算法结果的确可以输出使得机器计算总时间 C 最小化的参数 m,n 的取值。

表2 在 mn 不同取值下 C 的值

Tab.2 C with different values of $m\times n$

m	n=1	n=2	n=4	n=5	n=10	n=20
1	53 829 787.23	30 744 680.85	19 074 468.09	16 689 361.70	11 663 829.79	8 512 765.96
2	30 744 680.85	19 074 468.09	12 984 042.55	11 663 829.79	8 512 765.96	5 660 638.30
4	19 074 468.09	12 984 042.55	9 428 191.49	8 512 765.96	5 660 638.30	$mn>N$
5	16 689 361.70	11 663 829.79	8 512 765.96	7 627 234.04	$mn>N$	$mn>N$
10	11 663 829.79	8 512 765.96	5 660 638.30	$mn>N$	$mn>N$	$mn>N$
20	8 512 765.96	5 660 638.30	$mn>N$	$mn>N$	$mn>N$	$mn>N$

5 结论

1）本文对基于分布式机器学习的编码方案的性能开销进行了研究和分析，计算出了 n 个工作节点中第 i 个完成计算任务的节点的密度函数，并给出了工作节点计算任务完成时间符合均匀分布场景下计算任务的任务完成时间 C 和机器计算总时间 T 的表达式。

2）分析对比了3个应用于矩阵乘法的编码方案与未编码方案的任务完成时间和机器计算总时间，提供了计算方案选择的依据。并且在此基础上，通过 Python 实验了指数分布情况下的编码方案并实验得出了其任务完成时间 C 和机器计算总时间 T，还在此基础上进行实验，对3种编码方案进行比较。

3）在此基础上还对参数选择进行分析，提出一个启发式算法为参数的选择提供参考。

参考文献：

[1] 何清，李宁，罗文娟，等. 大数据下的机器学习算法综述[J]. 模式识别与人工智能，2014，27(4)：327-336.

[2] 杨刚，贺冬葛，戴丽珍. 基于 CNN 和粒子群优化 SVM 的手写数字识别研究[J]. 华东交通大学学报，2020，37(4)：41-47.

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

[4] ABADI M，BARHAM P，CHEN J，et al. TensorFlow：A system for large-scale machine learning[C]//Proceedings of the 12 th USENIX Symposium on Operating Systems Design and Implementation. Berkeley，USA:USENIX Association，2016.

[5] CHEN T，LI M，LI Y，et al. MXNet：A flexible and efficient machine learning library for heterogeneous distributed systems[EB/OL].(2015-12-03)[2021-01-21]. https://arxiv.org/abs/1512.01274v1.

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

[7] ZAHARIA M，KONWINSKI A，JOSEPH A D，et al. Improving map reduce performance in heterogeneous environments[C]//Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation. Berkeley，USA：USENIX Association，2008.

[8] LI S，MADDAH-ALI M A，AVESTIMEHR A S. A unified coding framework for distributed computing with straggling servers[EB/OL].(2016-12-10)[2021-01-21]. https://arxiv.org/abs/1609.01690v1.

[9] KIANI S，FERDINAND N，DRAPER S C. Exploitation of stragglers in coded computation[C]//2018 IEEE International Symposium on Information Theory(ISIT). Piscataway，USA：IEEE，2018.

[10] TANDON R，LEI Q，DIMAKIS A G，et al. Gradient coding[EB/OL]. (2016-09-16)[2021-01-21]. https://arxiv.org/abs/1612.03301v2.

[11] RAVIV N，TAMO I，TANDON R，et al. Gradient coding from cyclic MDS codes and expander graphs[EB/OL].(2017-07-12)[2021-01-21]. https://arxiv.org/abs/1609.01690v1.

[12] MALLICK A，CHAUDHARI M，JOSHI G. Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication[EB/OL]. (2018-04-27)[2021-01-21]. https://arxiv.org/abs/1804.10331.

[13] TANDON R，LEI Q，DIMAKIS A G，et al. Gradient coding：Avoiding stragglers in distributed learning[C]//Proceedings of the 34th International Conference on Machine Learning. Sydney，Australia：PMLR，2017.

[14] YE M，ABBE E. Communication-computation efficient

gradient coding[EB/OL].(2018-02-09)[2021-01-21]. https://arxiv.org/abs/1802.03475.

[15] LI S,MADDAH-ALI M A,AVESTIMEHR A S. Coded mapreduce[C]//53rd Annual Allerton Conference on Communication,Control and Computing (Allerton). Piscataway, USA:IEEE,2015.

[16] LI S,SUPITTAYAPORNPONG S,MADDAH-ALI M A,et al. Coded terasort[C]//2017 IEEE International Parallel and Distributed Processing Symposium Workshops(IPDPSW). Piscataway,USA:IEEE,2017.

[17] REISIZADEH A,PRAKASH S,PEDARSANI R,et al. Coded computation over heterogeneous clusters[EB/OL]. (2017-01-21)[2021-01-21]. https://arxiv.org/abs/1701.05973v1

[18] FERDINANDN S,DRAPER S C. Anytime coding for distributed computation[C]//54th Annual Allerton Conference on Communication,Control and Computing(Allerton). Piscataway,USA:IEEE,2016.

[19] 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). Piscataway,USA:IEEE,2017.

[20] PARK H,LEE K,SOHN J,et al. Hierarchical coding for distributed computing[EB/OL].(2018-01-15)[2021-01-21]. https://arxiv.org/abs/1801.04686.

[21] LEE K,LAM M,PEDARSANI R,et al. Speeding up distributed machine learning using codes[J]. IEEE Transactions on Information Theory,2018,64(3):1514-1529.

[22] LEE K,SUH C,RAMCHANDRAN K. High-dimensional coded matrix multiplication[C]//2017 IEEE International Symposium on Information Theory(ISIT). Piscataway,USA:IEEE,2017.

[23] YU Q,MADDAH-ALI M,AVESTIMEHR A S. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication[EB/OL]. (2017-05-30)[2021-01-21]. https://arxiv.org/abs/1705.10464.

[24] YU Q,MADDAH-ALI M,AVESTIMEHR A S. Straggler mitigation in distributed matrix multiplication:Fundamental limits and optimal coding[EB/OL]. (2018-01-23)[2021-01-21]. https://arxiv.org/abs/1801.07487v5.

[25] DUTTA S,FAHIM M,HADDADPOUR F,et al. On the optimal recovery threshold of coded matrix multiplication[EB/OL]. (2018-01-21)[2021-01-21]. https://arxiv.org/abs/1801.10292v1.