计 算 机 研 究 与 发 展　　DOI:10. 7544/issn1000-1239. 2019. ********
Journal of Computer Research and Development　　卷(期):起止页，年

所属类别：网络信息安全；分布式计算　　稿件编号：2021-10496

编码计算研究综述

郑腾飞，　周桐庆，　蔡志平，　吴虹佳

(国防科技大学计算机学院 长沙　410073)

(zhengtengfei@nudt.edu.cn)

Review of Research on Coded Computing

Zheng Tengfei,　Zhou Tongqing,　Cai Zhiping,　Wu Hongjia

(*College of computer, National University of Defense Technology, Changsha 410073, China*)

Abstract By integrating the coding theory with distributed computing and exploiting flexible coding methods, coded computing manages to relieve the transmission burden and the negative effects of stragglers. In this way, it improves the overall performance of distributed computing systems. Meanwhile, coded computing schemes are also designed and used to provide security and privacy guarantees for distributed computing systems, where mechanisms, such as error-correcting and data masking, are generally adopted. Due to the advantages of coded computing in communication, storage and computational complexity, it has attracted extensive attention and has become a popular direction in the field of distributed computing. In this survey, the background of coded computing is reviewed with its definition and core ideology clarified. Afterward, the existing coding schemes for communication bottleneck, computation delay and security privacy are introduced and comparatively analyzed in detail. Finally, future research directions and technical challenges of coded computing are analyzed and introduced to provide valuable references for related researchers.

Key words coded computing; distributed computing; distributed machine learning; network coding; performance optimizing; system security; data privacy

摘　要　编码计算将编码理论融于分布式计算中,利用灵活多样的编码方式降低数据洗牌造成的高通信负载，缓解掉队节点导致的计算延迟，有效提升分布式计算系统的整体性能，并通过纠错机制和数据掩藏等技术为分布式计算系统提供安全保障. 鉴于其在通信、存储和计算复杂度等方面的优势,受到学术界的广泛关注,成为分布式计算领域的热门方向. 对此，文中首先介绍编码计算的研究背景，明确编码计算的内涵与定义；随后对现有编码计算方案进行评述，从核心挑战入手，分别对面向通信瓶颈，计算延迟和安全隐私的编码计算方案展开介绍、总结和对比分析；最后指出未来可能的研究方向和技术挑战，为相关领域的研究提供有价值的参考.

关键词　编码计算；分布式计算；分布式机器学习；网络编码；性能优化；系统安全；数据隐私

中图法分类号:　TP399

随着机器学习和大数据分析应用的涌现，相关数据集规模不断增长，分布式地执行应用的计算任务可有效整合资源，缓解计算压力. 然而，分布式计算所需的频繁数据洗牌往往导致较高的通信负载，等待计算速度缓慢或发生故障节点的响应还会引入计算延迟，而节点的可信问题进一步制约着分布式计算范式的应用和发展.

近年来，研究人员将编码理论应用到分布式计算领域，提出了一种新的计算框架——编码计算(code computing, CC)，旨在借助灵活多样的编码技术，降低通信负载，缓解计算延迟，并抵抗系统中的拜占庭攻击，保护数据隐私. 例如，文献[1]通过

收稿日期：修回日期：

基金项目：国家重点研发计划(2020YFC2003400， 2018YFB0204301)；国家自然科学基金(62072465)；国防科大研究基金(Zk19-38)

This work is supported by the National Natural Science Foundation of China (62072465)， the National Key Research and Development Program of China (2020YFC2003400， 2018YFB0204301)， and the NUDT Research Grants (Zk19-38).

对分布式计算任务的中间结果进行异或编码，减少工作节点之间需要传输的信息的数量和大小，以此显著减少通信负载. 在另一个例子中，文献[2]利用纠删码对工作节点的任务进行编码，使得主节点能够从部分完成任务的节点中恢复最终结果，从而减少计算延迟.

鉴于其在通信，存储和计算复杂度等方面的优势，编码计算一经提出便引起研究人员的广泛关注，逐渐成为支撑有效分布式计算的热门研究方向. 当前编码计算方案种类繁多，编码形式多样，有必要对当前方案进行总结和分析，以此为分布式计算，高性能计算，安全计算，以及分布式机器学习，联邦学习的研究人员提供启发和思考. 我们将本文和当前相关综述覆盖的编码方案进行了对比，如表 1 所示.

Table 1 Comparison of existing reviews on coded computing

表 1 已有编码计算相关综述对比[1]

综述	文献重点	年份	所覆盖的编码方案		
			最小化通信编码	最小化时延编码	安全隐私编码
Wang 等人[3]	回顾了编码计算在改进大规模机器学习集群性能的研究进展	2019	⊙	⊕	○
Li 等人[4]	对相关领域内部分方案的原理进行了详细的阐述和证明	2020	⊕	⊕	⊕
本文	给出了编码计算的定义，重点对当前编码计算方案进行了梳理和分类，综合总结分析了当前编码计算在上述三个方向的进展	2021	●	●	●

[1] 其中●表示全面，⊕表示较全面，⊙表示不够全面，○表示未涉及.

其中，文献[3]总结了利用编码技术改进分布式机器学习性能的研究进展，缺少对基于编码计算解决通信和安全隐私问题的总结和评述. 同时，除分布式机器学习之外，在其他分布式计算场景中(例如，无线分布式网络，异构分布式网络，边缘计算网络)编码计算技术面临的挑战也不尽相同，本文结合技术分析对编码计算在多场景下的使用进行挖掘分析. 另一方面，文献[4]的综述内容局限于对相关领域内一部分工作的细化阐述，缺乏对编码计算相关技术进展的细粒度对比和解析，参考价值有限. 为满足分布式计算研究者灵活运用编码计算技术，构建更为实用的分布式计算应用(例如联邦迁移学习)，尚且需要对涉及多场景，多类问题的编码计算架构与技术予以全面综述.

本文贡献如下：据我们所知，本文首次相对全面的总结了编码计算的当前研究进展. 首先，对编码计算的基本原理进行介绍，并对现有编码计算方案进行分类. 根据不同的应用目标，本文将编码计算方案分为面向通信瓶颈，面向计算延迟，面向安全隐私三类. 进一步，分别从上述三个方向对现有编码计算方案进行了综述，重点包括：1)介绍分析了 Master-Worker 架构下的面向通信瓶颈的编码计算方案；2)根据不同的计算任务(矩阵乘法，梯度下降等)，分别对面向计算延迟的编码计算方案进行了讨论和总结；3)从对抗恶意节点和防止数据泄露两方面分析总结了编码计算在分布式计算安全和隐私方向的研究进展.

本文第 1 节对分布式计算系统中面临的问题，挑战进行分析，给出了编码计算的定义和顶层分类；在第 2-4 节根据第 2 节中的分类，分别从三个方向具体总结分析编码计算的研究现状；第 5 节展望编码计算的研究方向；第 6 节总结了全文.

1 编码计算概述

1.1 问题分析

分布式计算相关技术和理念已经在各种应用和场景中得以运用，然而，当在大量节点上分布式执行计算任务时，分布式计算系统将面临以下挑战：

1)数据洗牌带来的通信瓶颈. 数据洗牌是分布式计算系统的核心步骤，其目的是在分布式计算节点之间交换中间值或原始数据. 例如，在

MapReduce 架构中，数据从 Mapper 被传输到 Reducer. 通过对 Facebook 的 Hadoop 架构进行追踪分析，平均有 33%的作业执行时间都花费在数据洗牌上[5].在 TeraSort，WordCount，RankedInvertedIndex 和 SelfJoin 等应用程序中，50%-70%的执行时间用于数据洗牌[6]. 然而，在每一次数据洗牌过程中，整个数据集都通过网络进行通信. 频繁数据交互带来的通信开销造成了分布式计算系统的性能瓶颈.

2)掉队节点带来的计算延迟. 分布式计算系统由大量计算节点执行计算任务，其中一部分工作节点的计算速度可能比平均速度慢 5-8 倍，甚至会出现未知故障[7](这类节点被称为“掉队节点(straggler)”). 等待掉队节点的反馈会给整个计算任务造成不可预测的延迟[8]，降低系统性能.

3)安全和隐私. 计算分布引入的另一个重要问题是计算/工作节点存在不可控，不可靠等问题. 与传统集中式计算不同，分布式计算中的工作节点很可能是多个所有人的资产，这就使得数据输入到系统后访问面被动增加，导致数据的隐私受到威胁. 例如，将涉及到用户个人健康情况的医疗数据分发到多个节点可能会造成隐私泄露. 与此同时，拜占庭攻击是分布式系统面临的一个传统的安全威胁，节点提交错误信息将误导最终的计算结果，极大的影响系统可用性.

1.2 编码计算方案分类

由 2. 1 分析可知分布式计算系统主要面临三个方面的挑战：1)数据洗牌带来的通信瓶颈；2)掉队节点造成的不可预测的延迟；3)系统安全和数据隐私. 这三者严重制约着系统的扩展性和服务性能. 为应对上述挑战，研究人员提出了一系列编码计算方案. 根据各方案的主要功能和目标，可以相应地将编码计算方案分为以下三类：

1)优化通信负载编码：以降低分布式计算系统的通信开销. 优化通信负载编码方案通过增加额外的计算操作，创建编码机会，从而减少数据洗牌所需的通信负载. 文献[1]是该方向的第一篇研究，其在分布式计算负载和通信负载之间实现了逆线性平衡——将计算负载增加r倍(即，在r个节点上计算每个任务)，则可以将通信负载降低r倍. 随后，文献[9-12]将方案[1]分别扩展到无线分布式网络，多阶段数据流应用程序，计算任务密集型分布式系统和异构分布式网络中，有效降低了不同分布式计算场景下的通信负载. Attia 等人[13]提出了一种用于分布式机器学习的近乎最佳的编码数据洗牌方案，得到工作节点不同存储方式下通信开销的最优下界. Li 等人[14]提出了一种压缩编码分布式计算方案，相比于 CDC 方案进一步降低了分布式计算系统中的通信负载. Li 等人[15]对索引编码方案[11]进行修改，并在此基础上提出了一种用于分布式计算系统的半随机柔韧性索引编码数据洗牌方案，该方案平均能节省 87%的传输开销.

2)最小化计算延迟编码：以减轻分布式计算系统的掉队节点导致的延迟弊端. 最小化计算延迟编码计算方案能够在计算负载和计算延迟(即，整个作业响应时间)之间进行逆线性平衡. 换言之，该方向的编码计算方案利用编码来有效地注入冗余计算，以此来减轻掉队节点的影响，并通过注入与冗余量成比例的乘法因子来加速计算. Lee 等人[2]利用最大距离可分码(maximum distance separable code, MDS 码)首次解决分布式矩阵-向量乘法中的延迟问题，并且降低了分布式机器学习算法中数据洗牌的通信成本. 和文献[2]目标一致，文献[16-18]分别提出了不同的编码计算方案，以降低分布式矩阵-向量乘法中的计算延迟. 除此之外，针对其他分布式计算任务(如，矩阵-矩阵乘法[19-23]，梯度下降[24-28,29]，卷积计算，傅里叶转换[30,31]和非线性计算[32]等)中的计算延迟，研究人员也提出了相应的编码计算方案.

为了处理异构分布式计算系统中的掉队节点，必须考虑到为异构节点设计负载平衡策略[33-38]，以最大程度地减少总体作业执行时间. 考虑到分布式计算系统中的掉队节点的动态性，如何有效的利用掉队节点所做的计算结果优化计算延迟也引起了研究人员的广泛关注[39-43].

3)安全和隐私编码：为分布式计算系统提供安全的计算过程. 为抵抗梯度下降中的拜占庭攻击，Chen 等人[44]利用编码理论的思想提出了 DRACO 编码计算方案. Gupta 等人[45]利用“响应冗余”的，可在梯度聚合时检测出计算错误的节点. Data 等人[46]通过对数据进行编码，基于错误校正[47]设计了一种抗拜占庭攻击的分布式优化算法. Yu 等人[48]提出一种拉格朗日编码计算(Lagrange coded computing, LCC)方案，该方案对输入数据进行编码，不仅可以减少计算延迟，而且可以抵抗恶意节点的攻击，保护数据隐私. 作为 LCC 的扩展，So 等人[49]提出了一种快速且具有隐私保护功能的分布式机器学习框架——CodedPrivateML. Nodehi 等人[50]将多项式码与 BGW 方案[51]结合在一起，提出了一种多项式共享方案. 随后，针对矩阵乘法，研究人员分别提出了单

边隐私[52-54]和双边隐私[55-58]的编码计算方案. 文献[59]针对边缘计算架构中源节点，工作节点，主节点不同的隐私需求设计了相应的编码计算方案.

表 2 按照以上分类思路列出了代表性工作，后文将在第 3，4，5 节详细介绍，分析各分类的典型工作.

Table 2 Classification of coded computing schemes

表 2 编码计算方案分类

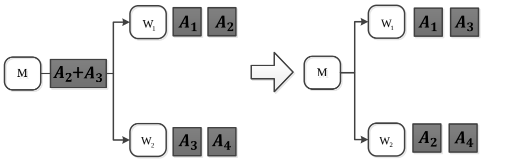
拟解决问题	示例
通信瓶颈	CDC[1],CWDC[9],S-CDC[11], SIP/SIU[13],CCDC[14],Pliable Index coding[15]
计算延迟	ShortDot[16],S-对角线[17],多项式码[20], MatDot[21],FRC[24],BCC[27], PCR[29],HCMM[33], LCC[48]
安全隐私	DRACO[44], LCC[48],Coded PrivateML[49],阶梯码[53]PRAC[54], 调和编码[60]

1.3 编码计算定义

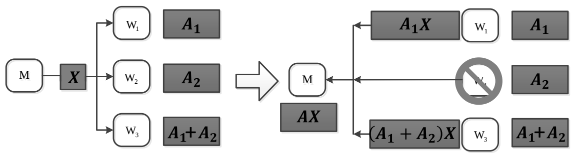
编码计算目前还未有一个严格的统一的定义，为简要说明编码计算的思想，列举以下两个简单示例：

示例 1. 降低通信负载编码[2]：假设一个分布式计算系统具有 2 个工作节点和 1 个主节点，现有一个大数据矩阵被分为 4 个子矩阵，即，$\boldsymbol{A_1}, \boldsymbol{A_2}, \boldsymbol{A_3}, \boldsymbol{A_4}$，分别存储在节点$W_1$和$W_2$中，如图 1(a)所示. 其目标是主节点将$\boldsymbol{A_3}$传输至$W_1$，并将$\boldsymbol{A_2}$传输至$W_2$. 可以设计这样一种编码，使主节点发送多播编码信息$\boldsymbol{A_2}+\boldsymbol{A_3}$到 2 个工作节点，后者使用本地已存储的数据便可解码获得所需的数据. 显然，与未编码的数据洗牌方案相比，编码方案可降低 50%的通信开销.

示例 2. 减少计算延迟编码[2]：接下来考虑另一个简单例子，图 1(b)展示一个具有 3 个工作节点和 1 个主节点的分布式计算系统，其目标是计算矩阵乘法$\boldsymbol{AX}$，其中$\boldsymbol{A} \in \mathbb{R}^{q\times r}$，$\boldsymbol{X} \in \mathbb{R}^{q\times r}$. 数据矩阵$\boldsymbol{A}$被划分为$\boldsymbol{A}_1$和$\boldsymbol{A}_2$两个子矩阵. 可以这样设计编码，在进行计算任务分配前，由主节点对子矩阵进行编码生成数据$\boldsymbol{A}_3=\boldsymbol{A}_1+\boldsymbol{A}_2$. 而后，主节点将$\boldsymbol{A}_1$，$\boldsymbol{A}_2$，$\boldsymbol{A}_3$分别分配给$W_1$，$W_2$，$W_3$. 当工作节点接收到矩阵$\boldsymbol{X}$时，每个节点将$\boldsymbol{X}$与存储的数据相乘，并将计算结果返回给主节点. 通过观察可知，主节点在接收到任意两个工作节点的结果时都可以恢复最终结果$\boldsymbol{AX}$，而不用等待最慢的节点(掉队节点)响应.



（a）降低通信负载编码示例



（b）减少计算延迟编码示例

Fig. 1 Simple example of coded computing

图 1 编码计算简单示例

值得注意的是，示例 1 为了在传输数据时可以进行编码，引入了冗余，即节点W_1和W_2分别额外存储了数据$\boldsymbol{A_2}$和$\boldsymbol{A_4}$. 示例 2 同样引入了冗余——给W_3分配了额外的计算任务$(\boldsymbol{A}_1+\boldsymbol{A}_2)\boldsymbol{X}$，以此使得分布式计算系统可以容忍 1 个掉队节点的存在. 由此可见，编码计算的核心思想是注入并充分利用分布式计算系统中的数据或计算冗余. 本文将编码计算定义如下：编码计算是利用编码理论注入冗余，通过对存储-通信-计算的权衡,从而解决或缓解分布式计算系统中通信瓶颈，计算延迟，安全和隐私等问题的一系列技术手段.

2 面向通信瓶颈的编码计算

近年来，研究人员对 Master-Worker 分布式计算架构下的数据洗牌问题进行了大量研究. 其中根据主节点是否参与运算及存储数据，Master-Worker 架构可分为 Map-Reduce 和典型 Master-Worker 两种略有区别的分布式计算架构. 而在上述两种不同分布式计算架构下的数据洗牌编码方案也不尽相同. 因此，在随后两节，将分别对基于 Map-Reduce 和基于典型 Master-Worker 的数据洗牌编码计算方案进行介绍和分析.

2.1 基于Map-Reduce的数据洗牌编码计算

MapReduce 是一种编程范式，可以并行处理海

量数据集. 在 MapReduce 架构中，主节点不参与运算，且不存储数据，只为各工作节点协调分配不同的输入数据. 更具体地说，在 MapReduce 中整个计算被分解为“Map”，“Shuffle”和“Reduce”三个阶段. 在 Map 阶段，工作节点根据设计的 Map 函数使用输入数据来计算中间值的一部分；在 Shuffle 阶段，工作节点相互交换一组中间值；在 Reduce 阶段，工作节点计算并输出最终结果. 针对 Map-Reduce 架构，研究人员通过将子数据集映射到多个工作节点，并仔细设计放置策略，以创建编码机会. 编码计算利用该机会将各工作节点计算的中间值进行异或编码，然后将编码信息广播给其他工作节点，以此降低通信负载.

2.1.1　通用数据洗牌编码方案

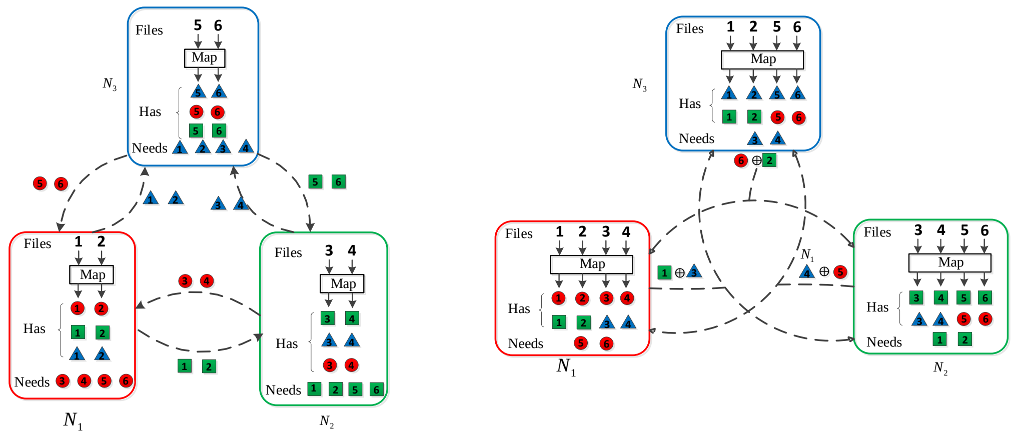
针对 Map-Reduce 架构，Li 等人[1]提出了编码分布式计算方案(coded distributed computing, CDC)，并在后续工作中将 CDC 方案扩展到无线分布式计算系统[9]，多级数据流[10]和 TeraSort 排序算法[61]等. 这里结合图 2 所示计算用例概述 CDC 方案的基本思路.

假设客户端需从 6 个输入文件中计算 3 个输出函数(分别用红色/圆形，绿色/正方形和蓝色/三角形表示)，计算任务由节点N_1，N_2和N_3协同完成. 每个节点计算唯一的输出函数，例如N_1计算红色/圆形函数，N_2计算绿色/方形函数，N_3计算蓝色/三角形函数.

在计算上不加冗余时，如图 2(a)所示，如果每个节点在本地存储 2 个输入文件，这样便可在本地生成 6 个所需的中间值中的 2 个. 因此，每个节点需要从其他节点接收另外 4 个中间值，产生的通信负载为$4 \times 3 = 12$.

如图 2(b)所示，CDC 使每个输入文件映射到两个节点. 显然，由于执行了更多的本地计算任务，因此每个节点现在仅需要 2 个其他中间值，此时的通信负载为$2 \times 3 = 6$. 由于每个节点计算出了更多的中间值，因此在数据洗牌时便有了编码的机会. CDC 将每个节点处生成的两个中间值进行异或编码，并多播到另外两个节点，此时的通信负载为 3. 因此，CDC 产生的通信负载比没有计算冗余的情况下的通信负载降低了 4 倍，比未编码的数据洗牌方案低 2 倍. 可见，这样以计算换通信的方式可有效降低洗牌时的传输开销. 随后，文献[1]从数据映射，计算，数据洗牌和归约 4 个阶段对 CDC 的一般化过程进行了形式化定义.

CDC 关注的是由 MapReduce 驱动的通用框架中的通信流，并且适用于任意数量的输出结果，输入数据文件和计算节点，不要求计算函数具备任何特殊性质(如线性).



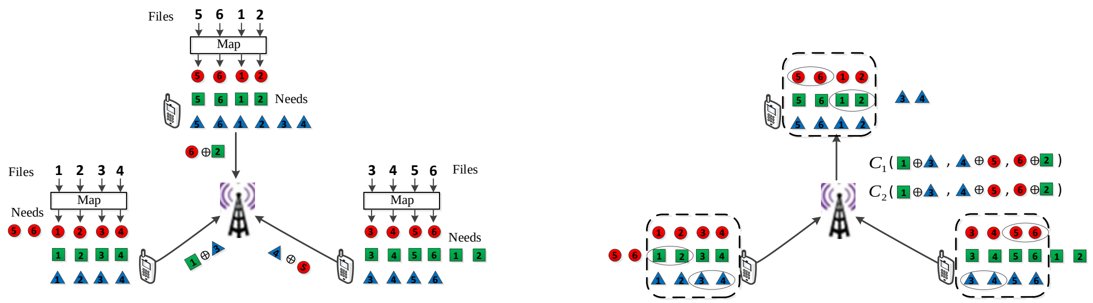
(a) 一般方案图解　　(b) CDC方案图解

Fig. 2　Data shuffling between general scheme and CDC

图 2　一般方案和 CDC 方案数据洗牌对比

2.1.2　有领域知识的数据洗牌编码方案

Li 等人[9]将 CDC 方案推广到无线分布式计算系统，设计了一种无线分布式编码计算的框架(coded wireless distributed computing, CWDC). CWDC 由上行链路和下行链路两部分组成，每个用户在上行链路发送 2 个中间值的异或值至接入点，如图 3(a)所示. 然后接入点无需解码任何单独的值，只需生成接收到的消息的 2 个随机线性组合$C_1(\cdot,\cdot,\cdot)$和$C_2(\cdot,\cdot,\cdot)$，并将它们广播给用户，即可同时满足所有的数据请求，如图 3(b)所示. 图示编码方法中上行链路通信负载为 3，下行链路通信负载为 2.



(a) 无线分布式编码上行链路

(b) 无线分布式编码下行链路

Fig. 3 The diagram of data shuffling in CWDC

图 3 CWDC 编码方案数据洗牌图解

对于具有K个用户的无线分布式计算应用程序，假设每个工作节点可以存储整个数据集的$\mu(0 < \mu \le 1)$倍，所提出的 CWDC 方案的通信负载为：

$$L_{uplink} \approx L_{downlink} \approx \frac{1}{\mu} - 1 \qquad (1)$$

与未编码方案相比，CWDC 可将通信负载降低μK倍，并且其通信负载是固定的和工作节点数量无关.

许多分布式计算应用程序包含多个 MapReduce 阶段. 例如机器学习算法[62]，数据库 SQL 查询[63-64]和数据分析[65]. 基于此，Li 等人[10]为多阶段数据流应用程序形式化定义了分布式计算模型. 其将多级数据流表示为一个分层的有向无环图(directed acyclic graph, DAG)，在这个 DAG 中，每个顶点代表一个 MapReduce 类型计算，方案通过对每个顶点实施 CDC 编码策略，有效降低通信负载.

CDC 中的一个隐含假设是每个服务器对存储在其内存中的所有文件执行所有可能的计算. 然而，当工作节点需要执行计算密集型任务时，可能没有足够的时间来执行所有计算. 针对这种情况，Ezzeldin 等人[11]在 CDC 方案的基础上提出了一种分割编码计算(spilt coded distributed computing, S-CDC)方案. 作者通过给定节点的计算能力阈值，从而得出相关通信负载的下限，并基于 CDC 提出一种启发式方案，以达到该通信下限.

在异构分布式计算系统中设计编码计算方案时，主要面临以下两个挑战：如何在异构节点上分配合适的数据，以及在合适的数据分配时，如何创建尽可能多的编码机会. Kiamari 等人[12]通过给定工作节点的存储能力$M = \{M_k\}_{k=1}^{K}$(K为工作节点总数)，进一步考虑子集M_1，M_2，…，M_k之间的关系，从而基于 CDC 思想来寻找异构系统下的编码机会.

在文献[61]中，作者将 CDC 的思路应用于 TeraSort 排序算法，并设计了一种名为 CodedTeraSort 的新分布式排序算法，该算法在数据中施加结构化冗余，以在数据洗牌阶段创造有效的编码机会. 作者通过实验评估了 CodedTeraSort 算法在 Amazon EC2 集群上的性能，与传统的 TeraSort 相比，CodedTeraSort 速率高 1. 97 倍-3. 39 倍.

2. 1. 3 线性假设下的数据洗牌编码方案

在没有进一步假设的情况下，CDC[1]实现了最优的计算和通信负载之间的平衡. 然而，工作节点计算的函数通常有一些结构，利用这些结构可以进一步降低通信负载.

在 Reduce 阶段是线性聚合的假设下，Li 等人[14]将压缩技术和 CDC 相结合提出了一种压缩编码分布式计算方案(compressed coded distributed computing, CCDC). 考虑如图 4 所示的分布式计算场景，假设最终计算结果是计算各中间值之和，则可以在发送节点上预先组合相同函数的中间值以减少通信.

例如，节点 1 将两对中间值相加以生成两个分组，然后将每个分组(绿色/正方形和蓝色/三角形)分割为两段，并取各自的一段逐位进行异或，以生成大小为中间值一半的编码数据. 最后，节点 1 将该编码数据多播到节点 2 和 3. 节点 2 和 3 处执行类似的操作.

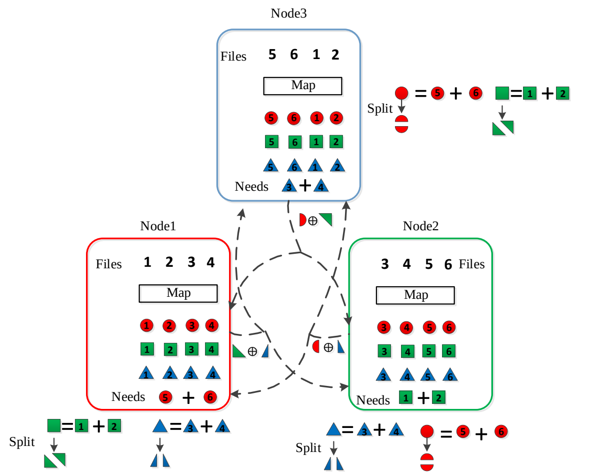


Fig. 4 The diagram of data shuffling in CCDC

图 4 CCDC 编码方案数据洗牌图解

最后，每个节点可以利用本地已有的中间值来解码获得所需的数据. 该编码方案可用于需在最后阶段对中间结果线性聚合的分布式机器学习中.

Horii 等人[66]指出在 Map 阶段计算的中间值可以看作是$\mathbb{F}_2$中的向量. 假设工作节点发送的向量数为r，由这些向量构造的线性子空间的维数可能小于r. 例如，在计算大量文档中的单词数时，工作节点计算的许多中间值是相同的，并且中间值的一些线性组合也是相同的. Horii 等人基于上述观点和假设，在编码时让工作节点发送中间值的线性子空间的基和线性组合系数，从而进一步降低通信负载.

2.2 基于典型Master-Worker的数据洗牌编码计算

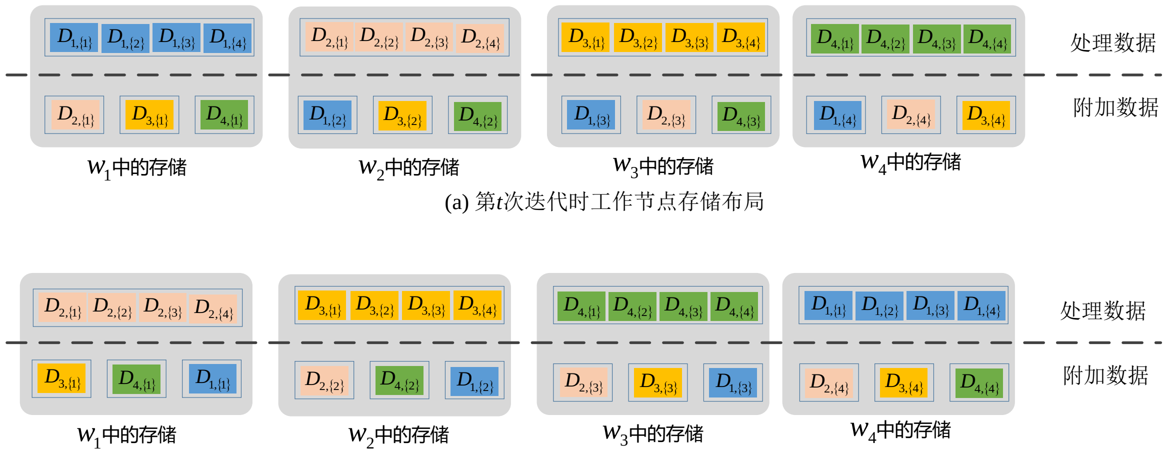
与 Map-Reduce 架构不同，在典型 Master-Worker 计算框架中，主节点可以访问整个数据库，并且只有主节点可以发送数据，而各工作节点之间无法进行通信. 主节点在每次迭代中将整个数据集排列划分为多个子数据集，并将每个子数据集传输到相应的工作节点，以便工作节点执行本地计算任务. 工作节点在完成计算后将结果返回给主节点. 典型 Master-Worker 架构下的编码计算方案可分为数据洗牌和存储更新两个阶段. 在数据洗牌阶段，主节点将整体数据分为大小相同的子数据集，并将子数据集的编码信息广播发送给工作节点，每个工作节点从主节点广播的编码信息和本地存储的数据中恢复出下次迭代所需的数据. 在存储更新阶段，每个工作节点存储新分配的数据单元并更新存储结构以实现下次迭代的数据洗牌. 值得注意的是，在编码计算方案中工作节点需额外存储一些关于其他数据单元的信息. 编码计算将此类附加数据进行仔细设计，以帮助在下次迭代时从编码信息中解码所需数据.

Lee 等人[2]首次提出了典型 Master-Worker 分布式计算架构下的编码数据洗牌方案，旨在提高分布式机器学习算法的训练速率. 假设数据集一共有q个数据行，工作节点数为n，每次迭代时将数据集随机均匀的分配给各个工作节点，则每个工作节点需要处理q/n个数据行. 假设每个工作节点的内存大小为$s(s > q/n)$个数据行，由此可知工作节点在每次算法迭代时除了存储计算任务所需的q/n个数据行外，还有额外$s - q/n$个数据行的存储空间. 文献[2]充分利用这部分存储空间，让每个工作节点从剩余的数据行中随机均匀的选择$s - q/n$个数据行进行存储，形成“侧信息”. 在数据洗牌时，主节点利用异或编码的方式将数据集编码，并将编码信息广播至各工作节点. 各工作节点利用“侧信息”完成解码，获取下次迭代所需数据. Lee 等人通过实验表明在$n = 50$，$q = 1000$，$s/q = 0.1$时，用于数据洗牌的通信开销减少了 81%. 因此，在缓存的存储开销非常低的情况下，与未编码方案相比可以显著地降低分布式系统的通信开销.

在给定一组(K, N, S)(K为节点数，N为文件总数，S为每个节点的存储大小)的情况下，Attia 等人[67-68]刻画了每个工作节点存储容量与最大通信负载之间的关系. 具体来说，在文献[67]中，针对工作节点数$K = \{2, 3\}$的情况，Attia 等人将最大通信负载描述为一个关于可用存储的函数. 在文献[68]中，Attia 等考虑了无多余存储的情况(即，$S = N/K$)，表明即使对于最小存储值，编码机会仍然存在. 然而，上述方案的参数不能取任意值.

随后，Attia 等人[13]在编码缓存方案[69-70]的启发下，提出了一种新颖的编码数据洗牌方案，该方案基于一种保持存储结构不变的存储/更新过程，称为“结构不变的放置和更新(structural invariant placement and update, SIP/SIU)”.

SIP/SIU 和文献[2]类似，工作节点除了存储必要的处理数据外，需额外存储一些附加数据，如图 5 所示. 和文献[2]中随机存储分配不同，SIP/SIU 以一种确定性和系统化的存储更新策略创造了更多的编码机会.



(a) 第t次迭代时工作节点存储布局

(b) 第$t+1$次迭代时工作节点存储布局

Fig. 5 Storage of work nodes in SIP/SIU

图 5 SIP/SIU 方案中工作节点的存储布局

下面以有$K=4(w_1,w_2,w_3,w_4)$个工作节点的分布式计算系统为例，简要说明 SIP/SIU 方案的流程. 设整个数据集$\mathcal{A}$被随机分为不相交的4个子数据集，$\mathcal{A}=\{D_1,D_2,D_3,D_4\}$，其中每个子数据被分为 4 个更小的部分，例如$D_1=\{D_{1,\{1\}},D_{1,\{2\}},D_{1,\{3\}},D_{1,\{4\}}\}$. 在为工作节点分配数据时，每个工作节点除了存储必要的处理数据外，额外存储附加数据，如w_1中存储$D_1=\{D_{1,\{1\}},D_{1,\{2\}},D_{1,\{3\}},D_{1,\{4\}}\}$和$\{D_{2,\{1\}},D_{3,\{1\}},D_{4,\{1\}}\}$，如图 5(a)所示. SIP 存储结构为数据洗牌提供了编码机会，在数据洗牌时主节点不必向工作节点传输整个数据集，只需广播式(2)所示编码信息. 工作节点收到编码信息后，可利用本地存储的数据解码获得所需的数据，并更新存储内容，完成一次数据洗牌，如图 5(b)所示. 完成一次数据洗牌后，各工作节点的存储结构不变，因此可为下次迭代时的数据洗牌提供编码机会.

$$X_{t,\ t+1}=\begin{cases}D_{2,\ \{2\}}\oplus D_{3,\ \{1\}},\ for\ w_1,\ w_2\\ D_{2,\ \{3\}}\oplus D_{4,\ \{1\}},\ for\ w_1,\ w_3\\ D_{2,\ \{4\}}\oplus D_{1,\ \{1\}},\ for\ w_1,\ w_4\\ D_{3,\ \{3\}}\oplus D_{4,\ \{2\}},\ for\ w_2,\ w_3\\ D_{3,\ \{4\}}\oplus D_{1,\ \{2\}},\ for\ w_2,\ w_4\\ D_{4,\ \{4\}}\oplus D_{1,\ \{3\}},\ for\ w_3,\ w_4\end{cases}\tag{2}$$

Attia 等人通过数值模拟实验表明，对于具有较大存储容量的工作节点的分布式计算系统，文献[13]要优于文献[2]中的编码方案，并且具有较低的计算复杂度.

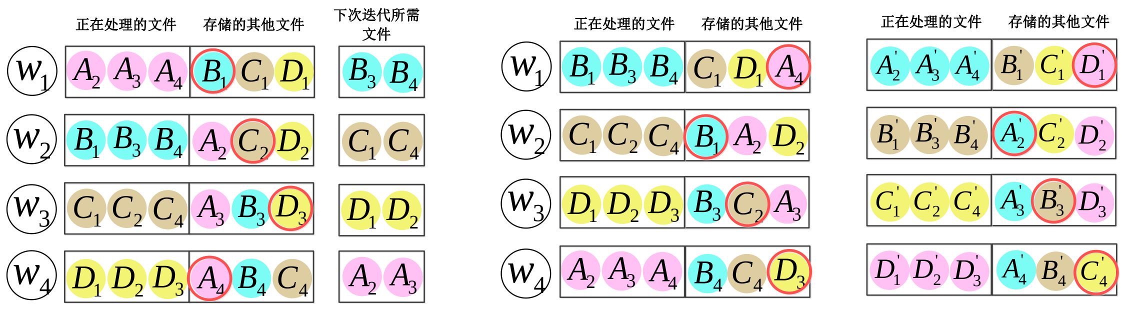
Elmahdy 等人[71]基于 SIP/SIU 方案提出了一种不同的编码洗牌方案，并证明了当文件数等于工作节点数时，其编码数据洗牌方案是最优的. 下面仍以有$K=4(w_1,w_2,w_3,w_4)$个工作节点的分布式计算系统为例，简要说明该方案. 设有 4 个输入文件$N=4$，分别命名$\{A,B,C,D\}$，每个工作节点的内存大小为$S=2$个文件. 不失一般性，假设w_1,w_2,w_3,w_4在第t次迭代时正在处理的文件分别为$\{A,B,C,D\}$.

Elmahdy 等人提出的放置策略将每个文件分为$\binom{K-1}{S-1}=3$个相同大小的子文件. 每个子文件被集合$\Gamma\subseteq[4]$标注，其中$|\Gamma|=S/(N/K)-1=1$. 例如，正在被工作节点w_1处理的文件A，被分为A_2，A_3和A_4三个子文件. 和文献[13]类似，该方案中工作节点w_i的内存也被分为两部分：一部分存储正在处理的文件，另一部分专门存储其他文件. 图 6(a)展示了工作节点的内存组织方式以及在下次迭代时需要计算的文件. 在此存储结构下，主节点只需广播$\mathcal{X}=\{X_{12},X_{13},X_{23}\}$至各工作节点便可完成一次数据洗牌. $\mathcal{X}$如式(3)所示.

$$\begin{cases}X_{12}=A_2\oplus B_3\oplus B_4\oplus C_1\\ X_{13}=A_3\oplus B_3\oplus C_1\oplus D_1\\ X_{23}=B_3\oplus C_1\oplus C_4\oplus D_2\end{cases}\tag{3}$$

每个工作节点w_i都可以从$\mathcal{X}$解码获得自己在第$t+1$次迭代时需要计算的文件. 例如，w_1可以从X_{13}解码获得B_3，从$X_{12}\oplus X_{13}$中解码获得B_4. 要说明的是，各工作节点更新存储后维持相同的内存结构，以完成从第$t+1$到第$t+2$次迭代的数据洗牌. Elmahdy 等人将该过程分两个阶段完成：缓存更新和子文件重新标记. 假设在$t+1$次迭代时w_1需要处理的文件为$B=\{B_1,B_3,B_4\}$，子文件C_1和D_1会和子文件A_4一起被保存下来，用于第$t+1$至第$t+2$次迭代时的数据洗牌，如图 6(b)左侧所示. 最后，为了维持当前内

存结构和原始内存结构保持一致，工作节将存储的文件重新命名，如图 6(b)右侧所示.



(a) 第t次迭代时节点存储结构及t+1次迭代所需文件　　(b) 第t+1次迭代前缓存更新和子文件重新标记过程

Fig. 6　Storage of work nodes and the processes of storage update in [71]

图 6　文献[71]中工作节点存储布局及更新过程

由上述分析可知，每次迭代时主节点需发送 3 个子消息，假设每个子消息的大小为 1/3，则文献[71]所提出的方案的通信负载为 1. 在相同的内存布局策略下，非编码洗牌方案需发送 8 个子消息，最终的通信负载为 8/3. 因此，在上述场景下，与非编码洗牌方案相比，Elmahdy 等人所提出的编码洗牌方案可节省约 62%的通信负载.

“好”的数据洗牌方案需要缓存的数据在各工作节点之间和算法迭代过程中具有足够的差异[72-74]，受这一想法的激励，Li 等人[15]提出了一种受约束的柔韧性索引编码数据洗牌方案，该方案在通信成本和计算性能之间取得了平衡.

为了达到上述目的，Li 等人在索引编码方案的基础上做了两项修改：1)添加一条约束，即，一条消息最多可以发送给 c 个工作节点，以确保只有一小部分工作节点可以缓存同一条消息. 该约束旨在降低工作节点间消息的相关性，确保工作节点间缓存内容的差异足够大. 2)设计了一个分层结构，即，将消息分成组，在迭代过程中，每个工作节点只缓存特定的消息. 该修改旨在降低迭代过程中消息的相关性，以确保迭代过程中各工作节点缓存内容的差异足够大. 通过在一个真实数据集上进行实验，与基于索引编码的替换式随机洗牌方案[2]相比，文献[15]提出的方案平均能减少 87%的传输消耗.

2.3　小结

在本节中，我们总结了 Master-Worker 架构下两种不同分布式计算框架中优化通信负载的编码计算方案. 首先我们对上述方案进行总结和回顾，如表 3 所示.

在 Map-Reduce 架构下的编码计算方案通过将输入数据映射到多个工作节点，并精心设计分配策略，从而创造编码机会. 而在不注入计算冗余的情况下，工作节点交换中间值时，由于没有解码所需的信息，所以不能对中间值进行编码. 通过注入冗余，CDC[1]，CodedTeraSort[61]可以将中间值进行异或编码；CWDC[9]可以在无线分布式网络上下行链路分别将中间值异或编码和线性组合编码；CCDC[14]在 Reduce 函数是线性相关的假设下，可以对中间值进行压缩，然后异或编码；文献[66]则在中间值具有线性相关性假设下，可以只交换编码信息的线性子空间的基和线性组合系数，但其在编码阶段需要更多的计算开销；文献[11-12]则分别是 CDC 方案在多级数据流应用和异构分布式系统下的推广，其考虑了更多的约束条件，但编码思想和方式与 CDC 方案相同.

与 Map-Reduce 架构在数据洗牌阶段传输计算结果信息不同，在典型的 Master-Worker 架构中数据洗牌阶段需传输原始数据. 由上述方案分析可知，典型 Master-Worker 架构中工作节点存储计算任务所需的数据外，还需存储一些“冗余数据”. 正是因为引入了存储冗余，所以数据洗牌阶段才有编码机会. 因此典型 Master-Worker 架构中编码洗牌方案的通信负载和各节点多余存储空间大小相关. 一个极端的情况是，当所有的工作节点都有足够的存储空间来存储整个数据集时，数据洗牌阶段不需要通信. 而另一个极端情况，当所有工作节点的存储空间刚好足以存储任务所需的数据时(称为无多余存储空间的情况)，则通信量将达到最大. 文献[2]，[13]和[71]考虑了无替换的数据洗牌，即附加数据用于提高通信效率，而不用于计算. 然而，当额外的存储数据用于计算时，可以获得潜在的计算增益. 例如，为了训练分类器模型，额外存储的数据样本也有助

于模型训练.

通过对上述方案的分析可知，编码计算可以有效降低数据洗牌阶段的通信负载. 然而，通信负载的降低往往以更高的计算负载和更高的存储为代价. 因此，需要权衡计算-存储-通信三者之间的关系，才能更好的评估方案的性能.

Table 3　Summary and review of coded computing schemes for communication bottleneck

表 3　面向通信瓶颈的编码计算方案总结与回顾

计算框架	方案	适用场景	核心思想	优点/缺点
Map-Reduce	CDC[1]	同构	注入计算冗余，传输中间值的异或编码	优点：分布式编码的基础方案 缺点：未根据不同计算任务定制编码策略
	CWDC[9]	同构无线分布式网络	在无线网络中的上下行链路传输过程对中间值异或编码	优点：发掘上下行链路的两次编码机会 缺点：解码会带来额外的计算开销
	Allerton2016[10]	多级数据流应用程序	提出了一种更广义的 CDC 方案来处理由分层 DAG 表示的多级数据流计算任务	优点：支撑多阶段数据流任务 缺点：各节点任务量分配不灵活
	S-CDC[11]	同构计算密集型	为每个工作节点分配较少的计算任务	优点:有效权衡计算负载和通信负载 缺点:划分的文件数会随工作节点数量呈指数级增长
	GLOBECOM2017[12]	异构	考虑异构节点的存储能力之间的关系，从而寻找更多的编码机会	优点:挖掘处理节点的异构存储能力进行输入文件分配 缺点：Reduce 函数在处理节点之间是均匀分布的
	CCDC[14]	同构	预合并同一函数的计算中间值，然后对预组合的数据包进行异或编码，以便在不同的工作节点之间进行通信	优点:融合压缩和异或编码控制通信开销 缺点：需同步处理大量任务
	arXiv2020[66]	同构	假设中间值具有线性依赖结构，工作节点只发送线性子空间的基和线性组合系数	优点:针对满足线性假设的任务优化通信开销 缺点：应用范围较窄
典型 Master-Worker	TIT2018[2]	同构	存储节点不仅存储计算所需数据，并且从剩下数据中随机均匀选择数据进行存储以提供编码机会	优点：基础编码计算框架 缺点:传输量理论值对数据量有假设
	SIP/SIU[13]	同构	设计一个存储结构不变的存储/更新过程	优点:面向大存储值提供通信和计算优化
	Pliable Index Coding[15]	同构	基于改进柔韧索引编码减少通信轮数的半随机数据洗牌方案	优点：优化洗牌的通信回合，冗余数据可用于计算 缺点：与 TIT2020[71]相比施加的约束条件非常宽松
	TIT2020[71]	同构	将文件转换过程定义为一个有向图，利用存储结构不变的存储/更新过程提供更多编码机会	优点:提供输入文件数和工作节点数相当时的最优通信 缺点：对其他场景需研究权衡策略

3 面向计算延迟的编码计算

面向计算延迟的编码计算方案的核心思想是通过使用编码技术，创建任务冗余，即在更多的工作节点上分配计算任务，使得主节点在不接收掉队节点结果的情况下依然可以恢复最终结果. 在面向计算延迟的编码计算方案中，一个重要的性能指标是恢复阈值，它是指在最坏情况下，主节点解码最终结果需要等待的工作节点的数量[2]. 一般来说恢复阈值越小，完成最终任务需等待完成计算的工作节点的个数越少，计算延迟越短. 面向计算延迟的编码计算方案的目标是降低恢复阈值，以便通过等待较少的工作节点来恢复最终结果，从而减少计算延迟.

为了降低不同分布式计算任务的计算延迟，可以利用具体运算的代数结构来设计编码方案. 在随后章节，本文将介绍分析当前编码计算方案在不同计算任务中的应用. 除此之外，本文还将简单讨论研究人员在异构场景和利用掉队节点结果等方面所提出的编码计算方案.

3.1 面向矩阵乘法运算的编码计算

3.1.1 矩阵-向量乘法

分布式矩阵-向量乘法是线性变换的组成部分，是机器学习和信号处理应用中的一个重要步骤. 为方便方案描述，首先定义以下分布式计算矩阵-向量乘法系统，该系统具有 1 个主节点和P个工作节点，其目标是分布式计算大小为$m \times n$的矩阵$\boldsymbol{A}$和$n \times 1$的向量$\boldsymbol{x}$的积：$\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}$. 如无特殊说明外，本节编码计算方案的计算场景和目标遵循上述定义. 下面我们将对不同的编码方案进行简要介绍和分析.

MDS 码：与简单的将同一个计算任务分配给多个工作节点(重复编码)不同，Lee 等人[2]利用 MDS 码来克服矩阵-向量乘法中的计算延迟问题. 基于 MDS 码的方案的编码策略是首先将矩阵$\boldsymbol{A}$按列划分为k个子矩阵，每个子矩阵的大小为$m \times n/k$. 随后使用(P,k)MDS 码对k个子矩阵进行编码，从而生成P个编码子矩阵. 随后将编码子矩阵发送至工作节点，工作节点计算编码子矩阵与向量$\boldsymbol{x}$的积，并返回主节点. 则主节点可以根据最快的k个计算节点恢复出最终结果$\boldsymbol{b}$. 如 2.2 节图 1(b)所示，为一个使用(3,2)MDS 码的编码计算方案实例.

Short-Dot：在基于重复编码或者 MDS 码的编码计算方案中，每个工作节点需计算长度为n的点积，而 Dutta 等人[16]认为通过缩短点积的长度，可以减少工作节点的计算时间. 因此 Dutta 等人提出了一种“短点积”(short-dot)的编码计算方案. 该方案通过对子矩阵施加稀疏性以使工作节点计算较短的点积. 然而，工作节点计算的点积的长度越短，该方案的恢复阈值越高.

s-对角线编码：Wang 等人[17]提出了一种名为“s-对角线”(s为系统中掉队节点的个数)的编码计算方案，该方案利用编码矩阵的对角结构来降低计算负载，同时实现和 MDS 码方案一样的恢复阈值. 下面以掉队节点数$s = 1$，工作节点数$P = 5$，来简要说明该方案的编解码过程. 首先将矩阵$\boldsymbol{A}$沿行划分为$k = 4$个子矩阵，即$\boldsymbol{A}^T = [\boldsymbol{A}_1^T\ \boldsymbol{A}_2^T\ \boldsymbol{A}_3^T\ \boldsymbol{A}_4^T]$. 则根据 1-对角线编码，可产生如下编码数据：$\widetilde{\boldsymbol{A}}_1 = \boldsymbol{A}_1$，$\widetilde{\boldsymbol{A}}_2 = \boldsymbol{A}_1 + \boldsymbol{A}_2$，$\widetilde{\boldsymbol{A}}_3 = \boldsymbol{A}_2 + \boldsymbol{A}_3$，$\widetilde{\boldsymbol{A}}_4 = \boldsymbol{A}_3 + \boldsymbol{A}_4$，$\widetilde{\boldsymbol{A}}_5 = \boldsymbol{A}_4$. 假设工作节点 1 为掉队节点，则其余工作节点返回的结果如式(4)所示：

$$\begin{bmatrix} \widetilde{\boldsymbol{b}}_2 \\ \widetilde{\boldsymbol{b}}_3 \\ \widetilde{\boldsymbol{b}}_4 \\ \widetilde{\boldsymbol{b}}_5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{A}_1\boldsymbol{x} \\ \boldsymbol{A}_2\boldsymbol{x} \\ \boldsymbol{A}_3\boldsymbol{x} \\ \boldsymbol{A}_4\boldsymbol{x} \end{bmatrix} \tag{4}$$

由式(4)可知，系数组成的矩阵为上三角矩阵，由于主对角线中的元素是非零的，所以它是可逆的. 因此可以通过求系数矩阵的逆来恢复最终结果. 利用对角线结构，s-对角线编码方案不仅降低了计算负载，并且可以实现和文献[2]相同的恢复阈值.

无码率编码：LT 码[75]是一类用于从有限的源符号集生成无限多个编码符号的纠删码. Mallick 等人[18]通过将矩阵$\boldsymbol{A}$的m行视为源符号，将 LT 码用于矩阵-向量乘法. 该方案首先根据 Robust Soliton 度分布选择参数d，随后从$\boldsymbol{A}$中随机均匀的选择d个数据行并将它们随机相加生成编码行. 即，d决定了每个编码行中的原始数据行的数量. 原数据行与编码数据行之间的映射对于成功解码至关重要，因此，此映射需存储在主节点上.

假设对m个原始数据行进行编码生成了αm个编码行，则主节点将αm个编码行平均分配给P个工作节点. 工作节点计算编码行和向量$\boldsymbol{x}$的乘积，并将结果返回给主节点. 如果一个工作节点在主节点能够解码$\boldsymbol{b}$之前完成了分配给它的所有向量积，则它将保持空闲状态，主节点继续从其他工作节点收集更多的行向量积. 一旦主节点获得了足够的结果可以解码$\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}$，则它将向所有工作节点发送完成信号以停止计算. 和其他方案相比，无码率编码方案可以实现理想的负载平衡并且具有较低的解码复杂

度.

3.1.2　矩阵-矩阵乘法

矩阵乘法是许多数据分析应用程序中关键操作之一. 此类应用程序需要处理TB级甚至PB级的数据，这需要大量计算和存储资源，而单台计算机往往无法满足. 因此，在大型分布式系统上部署矩阵乘法计算任务已经引起了广泛的研究[76-77].

本文首先定义以下分布式矩阵乘法计算场景. 该分布式计算系统有一个主节点和N个工作节点$(w_1, w_2, \dots w_N)$组成，其目标是计算大矩阵乘法$\boldsymbol{C} = \boldsymbol{A}^T\boldsymbol{B}$. 其中$\boldsymbol{A} \in \mathbb{R}^{r\times q}$，$\boldsymbol{B} \in \mathbb{R}^{r\times s}$. 为了分布式计算该矩阵乘法，首先将两个输入矩阵分别(任意)划分为$p\times m$和$p\times n$个子矩阵块，其中同一输入矩阵内的所有子矩阵大小相等. 然后将$\boldsymbol{A},\boldsymbol{B}$中每个子矩阵分配给各工作节点，工作节点完成计算任务后，将计算结果返回给主节点. 主节点恢复最终结果输出$\boldsymbol{C} = \boldsymbol{A}^T\boldsymbol{B}$. 如无特殊说明外，本节编码计算方案的计算场景和目标遵循上述定义. 下面我们将对不同的编码方案进行简要介绍和分析.

1D MDS码：Lee等人[19]将大矩阵乘法$\boldsymbol{A}^T\boldsymbol{B}$视为$n$个小矩阵乘法. 其将矩阵$\boldsymbol{B}$按列划分为$n$个子矩阵，即$\boldsymbol{C} = \boldsymbol{A}^T\boldsymbol{B} = [\boldsymbol{A}^T\boldsymbol{b}_1\ \ \boldsymbol{A}^T\boldsymbol{b}_2 \dots \boldsymbol{A}^T\boldsymbol{b}_n]$. 随后将工作节点分为$n$组，假设$N = dn$，则每组包含$d$个工作节点. 每组节点只负责计算一个小矩阵乘法$\boldsymbol{A}^T\boldsymbol{b}_i$，例如第一组节点计算$\boldsymbol{A}^T\boldsymbol{b}_1$. 随后将$\boldsymbol{A}$按列划分为$m$个子矩阵，并使用$(d,m)$MDS码对$\boldsymbol{A}$的$m$个子矩阵进行编码获得$d$个编码列$(d \geq m)$，如$\boldsymbol{a}_1$到$\boldsymbol{a}_d$. 然后将计算任务$\boldsymbol{a}_i^T\boldsymbol{b}_1$分配给第一组中第$i$个工作节点. 以此类推，直到将$n$个小矩阵乘法完全分配给各工作节点. 由于该方法只对矩阵$\boldsymbol{A}$进行编码，故Lee等人将其称为一维(one-dimensional, 1D) MDS编码方案. 该编码方案的整体计算时间由n个组中最慢的组的计算时间决定，而每个组的计算时间由该组中第m个完成计算任务的工作节点决定. 一般而言，1D MDS编码计算方案的恢复阈值为$K_{\text{1D MDS}} = N - \frac{N}{n} + m$.

乘积码：随后Lee等人基于乘积码[78]，在文献[19]中提出了另一种新颖的编码矩阵乘法方案，该方案较1D-MDS编码方案具有更低的恢复阈值. 乘积码是用小编码块作为构建块构建较大编码块的一种方法. 下面以$N = 9$，$m = n = 2$，$p = 1$为例，简要说明乘积码的编码解码过程. 由于$m = n = 2$，因此有：

$$\boldsymbol{C} = \boldsymbol{A}^T\boldsymbol{B} = \begin{bmatrix}\boldsymbol{a}_1^T\\ \boldsymbol{a}_2^T\end{bmatrix}[\boldsymbol{b}_1\quad \boldsymbol{b}_2] = \begin{bmatrix}\boldsymbol{a}_1^T\boldsymbol{b}_1 & \boldsymbol{a}_1^T\boldsymbol{b}_2\\ \boldsymbol{a}_2^T\boldsymbol{b}_1 & \boldsymbol{a}_2^T\boldsymbol{b}_2\end{bmatrix} \tag{5}$$

Lee等人首先利用(3,2)MDS码分别对矩阵$\boldsymbol{A}$和$\boldsymbol{B}$进行编码，从而得到$\boldsymbol{a}_1$到$\boldsymbol{a}_3$, $\boldsymbol{b}_1$到$\boldsymbol{b}_3$，其中$\boldsymbol{a}_3 = \boldsymbol{a}_1 + \boldsymbol{a}_2, \boldsymbol{b}_3 = \boldsymbol{b}_1 + \boldsymbol{b}_2$. 随后将计算任务$\boldsymbol{a}_i^T\boldsymbol{b}_j$分配给各工作节点，如图7所示. 假设其中有4个工作节点为掉队节点，则主节点在接收到其他5个计算结果：$\boldsymbol{a}_1^T(\boldsymbol{b}_1+\boldsymbol{b}_2), \boldsymbol{a}_2^T\boldsymbol{b}_1, \boldsymbol{a}_2^T\boldsymbol{b}_2, \boldsymbol{a}_2^T(\boldsymbol{b}_1+\boldsymbol{b}_2), (\boldsymbol{a}_1+\boldsymbol{a}_2)^T\boldsymbol{b}_2$时，可以通过计算$\boldsymbol{a}_1^T\boldsymbol{b}_2 = (\boldsymbol{a}_1+\boldsymbol{a}_2)^T\boldsymbol{b}_2 - \boldsymbol{a}_2^T\boldsymbol{b}_2$，然后计算$\boldsymbol{a}_1^T\boldsymbol{b}_1 = \boldsymbol{a}_1^T(\boldsymbol{b}_1+\boldsymbol{b}_2) - \boldsymbol{a}_1^T\boldsymbol{b}_2$，从而计算出最终结果$\boldsymbol{C}$. 更一般地，乘积码的恢复阈值为：

$$K_{\text{PRODUCT}} = 2(m-1)\sqrt{N} - (m-1)^2 + 1 \tag{6}$$

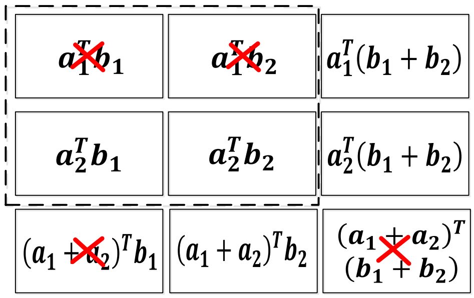


Fig. 7　An example of computing tasks assignment

图7　计算任务分配示例

下面介绍一类以多项式码为基础的编码计算方案[20-22]，这类方案的共同特征是为每个工作节点分配不同的随机数，并使用该随机数对输入矩阵进行编码，从而使得最后解码过程可视为多项式插值问题，不同之处在于各方案对输入矩阵的切割方式不同. 基于多项式码的编码计算方案的主要优点是可实现最佳恢复阈值，且恢复阈值不随工作节点的数量的变化而变化.

多项式码：为了寻找最佳恢复阈值，Yu等人[20]提出了一种基于多项式码的编码计算方案，该方案的恢复阈值与工作节点数量无关，且远小于MDS码和乘积码方案中的恢复阈值. 为构造多项式编码计算方案，Yu等人首先将输入矩阵$\boldsymbol{A}$和$\boldsymbol{B}$沿垂直方向分别划分为m和n个子矩阵，即：$\boldsymbol{A} = [\boldsymbol{A}_0\ \ \boldsymbol{A}_1 \dots \boldsymbol{A}_{m-1}]$，$\boldsymbol{B} = [\boldsymbol{B}_0\ \ \boldsymbol{B}_1 \dots \boldsymbol{B}_{n-1}]$. 随后随机分配给每个工作节点$w_i$一个互不相同的数，记为$x_i \in \mathbb{F}_q$($\mathbb{F}_q$为一个足够大的有限域). 接下来，对于给定的参数$\alpha$，$\beta \in \mathbb{N}$，Yu等人定义如下$(\alpha,\beta)$-多项式码：对$\forall i \in \{0,1,\dots,N-1\}$，计算式(7)和(8).

$$\widetilde{\boldsymbol{A}}_i = \sum_{j=0}^{m-1} \boldsymbol{A}_j\, x_i^{j\alpha} \tag{7}$$

$$\widetilde{\boldsymbol{B}}_i = \sum_{j=0}^{n-1} \boldsymbol{B}_j x_i^{j\beta} \tag{8}$$

随后将编码数据$\widetilde{\boldsymbol{A}}_i$和$\widetilde{\boldsymbol{B}}_i$分配给工作节点$w_i$. 工作节点$w_i$在接收到编码数据后根据式(9)计算$\widetilde{\boldsymbol{C}}_i$，并将结果返回主节点.

$$\widetilde{\boldsymbol{C}}_i = \widetilde{\boldsymbol{A}}_i^T \widetilde{\boldsymbol{B}}_i = \sum_{j=0}^{m-1}\sum_{k=0}^{n-1} \boldsymbol{A}_j^T \boldsymbol{B}_k x_i^{j\alpha+k\beta} \tag{9}$$

当$(\alpha,\beta)=(1,m)$时，可将$\widetilde{\boldsymbol{C}}_i$看作一个$mn-1$次多项式，如式(10)所示. 则每个工作节点的任务可看成是计算该多项式在$x=x_i$处的值. 观察(10)可知，由$h(x)$的系数可以恢复最后结果$\boldsymbol{C}$. 由于x_i各不相同，且$h(x)$为$mn-1$次多项式，所以只需mn个值便可以确定该多项式. 因而，从工作节点的计算结果中解码$\boldsymbol{C}$可以看作是给定mn个点的值的情况下插值多项式$h(x)$. 通过选择仔细选择参数(α,β)，该方案的恢复阈值为$K_{\text{Polynomial Codes}}=mn$，其与工作节点的数量无关.

$$h(x) \triangleq \sum_{j=0}^{m-1}\sum_{k=0}^{n-1} \boldsymbol{A}_j^T \boldsymbol{B}_k x^{j+km} \tag{10}$$

MatDot：由文献[20]可知，当$m=n$时，多项式码方案的恢复阈值为m^2. 针对$m=n$的情况，Fahim等人[21]提出了一种恢复阈值更低的编码计算方案，称为“MatDot”. MatDot 假设矩阵$\boldsymbol{A}$和$\boldsymbol{B}$都为$P\times P$的方阵，并将两矩阵分别在垂直和水平方向划分为m个大小相等子矩阵. 其编码思想和多项式编码[20]相似，在$m=n$时，MatDot 方案的恢复阈值为$K_{\text{MatDot}}=2m-1$，其远远小于多项式编码的恢复阈值m^2.

PolyDot：虽然 MatDot 的恢复阈值比多项式码方案[20]低，但在通信成本方面，MatDot 中每个工作节点的通信成本为$O(N^2)$，要比多项式码方案的通信成本$O(N^2/m^2)$高. 因此 Fahim 等人[21]针对矩阵$\boldsymbol{A}$和$\boldsymbol{B}$都是方阵的情况，在 MatDot 的基础上提出了一种名为“PolyDot”的编码方案. 该方案权衡了恢复阈值和通信成本之间的关系，是 MatDot 和多项式码[20]方案的折中. 一般而言，PolyDot 的恢复阈值为$K_{\text{PolyDot}}=t^2(2s-1)(st=m)$，通信成本为$O(N^2/t^2)$.

纠缠多项式码：与只允许将矩阵按列划分的多项式码方案不同，Yu 等人[22]在多项式码方案的基础上提出了一种名为纠缠多项式码的编码计算方案. 该方案允许对输入矩阵进行任意划分. 在纠缠多项式编码方案中，矩阵$\boldsymbol{A}$和$\boldsymbol{B}$分别被分割为$p\times m$和$p\times n$个子矩阵块，其中同一矩阵内的所有子矩阵大小相等. 换句话说，多项式码方案[20]是$p=1$的特殊情况.

通过选择参数p，m和n的不同的值，纠缠多项式码可以实现系统资源的不同利用，从而平衡存储和通信成本. 更一般地，该方案的恢复阈值为$K_{\text{Entangled polynomial}}=pmn+p-1$. 此外，纠缠多项式码启发了通用多项式计算[48]，安全/私有计算[79]和区块链系统[80]中编码计算方案的发展.

稀疏编码：大规模机器学习中许多问题都表现出极大规模的稀疏性，上述编码方案可能会破坏这种稀疏结构，并且导致更高的计算开销. Wang 等人[23]通过实验证明，在稀疏矩阵乘法中，多项式码方案的最终完成时间与未编码方案相比显著增加. 针对稀疏矩阵乘法问题，Wang 等人[23]提出了一种“稀疏编码”方案，其在编码过程中充分利用了稀疏矩阵的特性，其恢复阈值可以以较高的概率达到mn，并且可以降低工作节点计算量. 实验结果表明，与未编码方案，MDS 码[2]，乘积码[19]，多项式编码[20]和 LT 码[81]相比，在不同稀疏矩阵乘法中，稀疏编码方案都具有较快的计算速度，且对真实数据集的影响更为明显.

我们在表 4 中对上述编码计算方案的恢复阈值进行了总结和对比. 其中工作节点个数为N，输入矩阵$\boldsymbol{A},\boldsymbol{B}$分别被划分为$p\times m$和$p\times n$个子矩阵块.

Table 4 Summary of recovery threshold in coded computing schemes for matrix-matrix multiplication

表 4 矩阵-矩阵乘法编码计算方案恢复阈值总结

方案	恢复阈值	限制条件
1D MDS 码[19]	$N-\frac{N}{n}+m$	-
乘积码[19]	$2(m-1)\sqrt{N}-(m-1)^2+1$	$m=n$
多项式码[20]	mn	-
MatDot[21]	$2m-1$	$m=n$
PolyDot[21]	$t^2(2s-1)$	$m=n$且 $st=m$
纠缠多项式[22]	$pmn+p-1$	-
稀疏编码[23]	mn	稀疏矩阵

3.2 面向机器学习典型运算的编码计算

3.2.1 梯度下降算法编码方案

Tandon 等人[24]首次提出了梯度编码(gradient coding, GC)这一概念，通过有效利用工作节点额外的计算和存储，使得主节点能够容忍部分随机掉队节点. 基于 GC 思想有一系列研究方案[24-28]，这些方案的共同特征是对梯度进行编码. 具体来说，对于一个由n个工作节点组成的分布式系统，GC 的核心思想是首先将训练数据集划分n个不同的批次，随后将$r(1 \le r \le n)$个数据批分配给每个工作节点，接下来工作节点根据分配的数据集计算出r个部分梯度，并返回r个梯度的线性组合. 这些线性组合使得主节点可以从任意$n-r+1$个工作节点的结果中恢复出全梯度(即，所有部分梯度的和). 换句话说，基于 GC 的编码方案的恢复阈值为$K=n-r+1$.

基于这种思想 Tandon 等人在文献[24]中构造了两种梯度编码方案：1)部分重复编码(fractional repetition coding, FRC)和 2)循环重复编码(cyclic repetition coding, CRC). 下面分别对两种方案作简要介绍.

FRC：假设系统中有s个掉队节点，该方案首先将n个工作节点平均分为$(s+1)$个组，则每组有$(n/s+1)$个工作节点. 随后将训练数据均等划分为n个数据批，并分配给每个工作节点$r=(s+1)$个不相交的数据批，所有小组彼此互为副本. 完成计算后，每个工作节点将其部分梯度的总和传输给工作节点. 然而，这种构造只在n为$s+1$的倍数时适用.

CRC：与 FRC 构造不同，CRC 编码方案不需要n被$(s+1)$整除. 在 CRC 中，不是分配不相交的数据批，而是考虑将$(s+1)$个数据批循环分配给工作节点. 如图 8 所示，为$n=3$，$s=1$的数据分配示例. 假设每个数据批对应的梯度向量分别为$\boldsymbol{g}_1,\boldsymbol{g}_2,\boldsymbol{g}_3$，各个工作节点分别发送梯度的线性组合，例如$\boldsymbol{g}_1/2+\boldsymbol{g}_2,\boldsymbol{g}_2-\boldsymbol{g}_3,\boldsymbol{g}_1/2+\boldsymbol{g}_3$. 主节点可以从任意两个向量中恢复$\boldsymbol{g}_1+\boldsymbol{g}_2+\boldsymbol{g}_3$，如式(11)所示.

$$\boldsymbol{g}_1+\boldsymbol{g}_2+\boldsymbol{g}_3=2\left(\frac{1}{2}\boldsymbol{g}_1+\boldsymbol{g}_2\right)-(\boldsymbol{g}_2-\boldsymbol{g}_3) \quad (11)$$

特别地，文献[24]通过构造一个随机编码矩阵，从而指定数据分配以及局部梯度的线性组合的系数. 以图 8 为例，构造的编码矩阵$\boldsymbol{B}$应为：

$$\boldsymbol{B}=\begin{bmatrix}1/2 & 1 & 0\\ 0 & 1 & -1\\ 1/2 & 0 & 1\end{bmatrix}.$$

$\boldsymbol{B}$中每一行的非零项的索引决定了分配给每个工作节点的数据批次，而每一行的值是每个工作节点对梯度进行线性组合编码时的系数. 在此基础上文献[26]和[27]分别使用循环$(n,n-s)$MDS 码[82]和 Reed-Solomon 码[83]构造了相同功能的编码矩阵，并达到了相同的性能，两者的恢复阈值都为$K=n-r+1$.

BCC 编码方案：BCC 方案[27]的核心思想是在主节点处获得部分梯度的“覆盖率”. 简单来说，将训练样本分成若干批，每个工作节点独立地随机选择

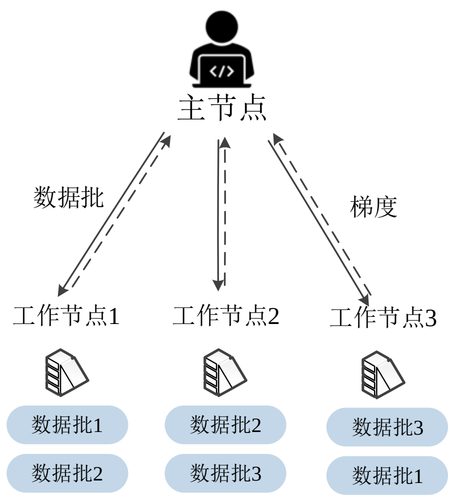


Fig. 8 Example of CRC when n=3, s=1

图 8 n=3, s=1 CRC 编码示例

其中一个数据批进行梯度计算，随后工作节点将计算的部分梯度的和返回给主节点. 如果主节点之前已经接收到同一数据批的梯度，那么主节点将丢弃该消息，否则保留该消息. 在接收到所有数据批的处理结果之前，主节点一直收集消息. 最后，主节点通过简单地计算保留的消息的总和恢复出最终结果.

BCC 方案的优势在于其是完全分布和无需协调的. 即每个工作节点独立选择其数据批，并以完全异步的方式执行本地计算和通信. 不需要主节点向工作节点提供任何反馈. 所有这些特性使得该方案易于在实际场景中进行应用. Li 等人将主节点恢复梯度所需的平均工作节点数定义为平均恢复阈值，BCC 方案的平均恢复阈值为$K=\left\lceil\frac{n}{r}\right\rceil\log\left\lceil\frac{n}{r}\right\rceil$. 在$n=50$，$m=50$时，与未编码方案和 CRC[79]相比，BCC 方案将任务执行速度分别提高了 85.4%和 69.9%.

通信-恢复阈值平衡方案：除了恢复阈值外，通信成本也是影响分布式梯度下降算法的重要因素之一. 然而上述文献的重点在于实现最佳恢复阈值，

并没有考虑通信成本的问题. Ye 等人[28]通过将梯度记为一个l维矢量，并对其中的元素进行线性编码，从而用更大的恢复阈值来换取每个工作节点更少的通信量.

虽然文献[28]同时权衡了恢复阈值和通信开销，但该方案存在解码复杂度高和数值稳定性差的问题. 为了恢复梯度和，主节点需要计算一个大小为$n-s$（n为总节点数，s为掉队节点数）的矩阵的逆矩阵，这导致文献[28]的解码复杂度为$O(n^3)$. Kadhe 等人[84]针对上述问题，设计一个允许使用任意线性代码来实现编解码功能的系统框架. 该框架使用 FRC[24]方案在工作节点之间分配训练数据. 当在这个框架中使用特定的码时，它的块长度决定了计算负载，维度决定了通信开销，最小距离决定了恢复阈值. 作者使用MSD码在Amazon EC2上评估了该框架的性能，与文献[28]相比，其平均迭代时间减少了 16%.

PCR：与 GC 编码方案[24-28]通过对梯度编码不同，Li 等人[29]提出了一种多项式编码回归(polynomial coded regression, PCR)方案. PCR 利用潜在的代数结构来生成编码子矩阵，使得编码矩阵是未编码子矩阵的线性组合. 和多项式码[20]方案类似，最后解码阶段可以看作多项式插值问题，主节点可以通过确定多项式系数从而恢复最终结果. 仿真结果表明，与梯度编码方案相比，该方案的恢复阈值更低，计算和通信时间更短. 在工作节点数为n，训练数据被分为n个数据批，为每个工作节点分配r个数据批的分布式计算场景下，PCR 的恢复阈值为$K=2\left\lceil\frac{n}{r}\right\rceil-1$. 与 PCR 方案[29]中对数据进行编码的想法类似，其他梯度编码方法，如随机梯度编码(stochastic gradient coding, SGC)[85]和 LDPC 码[86]，对数据变量进行编码，以降低更一般的大规模优化问题中的计算延迟.

AGC：与上述针对固定数目的掉队节点的方案不同，Cao 等人[87]提出了一种具有灵活容忍度的自适应梯度编码(adaptive gradient coding, AGC)方案. 通过让工作节点按轮次向主节点发送信号，该方案在计算负载，通信开销和恢复阈值之间实现了最佳折衷. 在 AGC 中，假如系统中没有掉队节点，则主节点在接收到所有工作节点在第一轮返回的编码梯度后即可解码获得总梯度；假如系统中有掉队节点，则正常工作节点需要继续返回编码梯度，直至主节点可以解码总梯度. 因此，该方案适用于掉队节点数量未知并且随着算法迭代而变化的实际应用. 表5 给出了上述 GC 方案和多项式编码方案恢复阈值的对比.

Table 5 Classification and summary of coded computing schemes for gradient

表 5 编码梯度下降方案分类和总结

方法分类	方案名称	恢复阈值
梯度编码	FRC [24]	$n-r+1$
	CRC [24]	
	Cyclic MDS Code[25]	
	Reed-Solomon Codes[26]	
	BCC [27]	$\left\lceil\frac{n}{r}\right\rceil\log\left\lceil\frac{n}{r}\right\rceil$
	ISIT[84]	和码的最小距离有关
	AGC[87]	自适应
数据编码	PCR[29]	$2\left\lceil\frac{n}{r}\right\rceil-1$

3.2.2 卷积计算和傅里叶变换

卷积计算：卷积运算在数学，物理，统计和信号处理中具有广泛的应用. 特别是对于卷积神经网络来说，卷积常被用于过滤或提取特征. Dutta 等人[30]针对受掉队节点影响的分布式卷积计算系统，提出了一种新颖的编码卷积策略. 该编码策略首先将输入向量分割为长度一定的短向量，并使用 MDS 码对预先指定的向量进行编码，从而可以在目标时间内快速可靠的完成计算.

离散傅里叶变换：离散傅里叶变换是包括信号处理，数据分析和机器学习算法等在内的许多应用程序的基础操作之一. Yu 等人[31]为了减少分布式离散傅里叶变换算法的计算延迟，提出了一种编码傅里叶变换方案. 该方案利用离散傅里叶变换的递归结构，首先将其分解为多个短向量上的离散傅里叶变换操作，其次利用傅里叶变换的线性特性，对输入数据进行线性编码，使工作节点的输出具有一定的 MDS 码特性，从而减少分布式计算的计算延迟.

3.2.3 非线性计算

由于神经网络的某些层，如激活层是非线性的，所以算法的整体计算是非线性的. 然而，上述讨论的编码计算方案并不适用于具有非线性计算的神经网络. 但这些网络的性能也受到掉队节点的限制. Dutra 等人[88]提出的一种用于矩阵-向量乘法的 Generalized PolyDot 编码方案，是可以扩展到深度神

经网络(deep neural networks, DNN)的方法之一. Generalized PolyDot 通过对 DNN 中每一层的线性运算进行编码，从而允许每一层的训练中出现错误. 换句话说，在错误量不超过最大错误容忍度的情况下，解码仍然可以正确执行. Hadidi 等人[89]指出编码技术可以有效降低 IoT 系统中不同神经网络架构(如 AlexNet[90])中的计算延迟. 但是，这种统一编码 DNN 的策略可能不适用于其它具有大量非线性函数的神经网络. 现有的编码计算方法侧重于手工设计新的编码方法，大多都不适合预测服务系统.

因此，Kosaian 等人[32]在 CNN 和多层感知器的基础上，提出了一种用来学习编码和解码功能的神经网络结构和训练方法. 在没有掉队节点的情况下，解码函数返回的预测结果与其他预测服务系统的结果相同. 当出现掉队节点时，解码函数的输出是对缓慢或失败预测的近似重建. 通过使用 ResNet-18 分类器在数据集 MNIST，Fashion-MNIST 和 CIFAR-10 上进行实验，结果显示该方案重建不可用输出结果的正确率分别为 95.71％，82.77％和 60.74％.

3.3 其他面向计算延迟的编码计算方案

就适用场景而言，前三节所介绍的编码计算方案中工作节点皆是同构的，而分布式计算中另一个常见特性是工作节点的异构性. 因此，研究如何减少异构分布式计算场景下的计算延迟也是非常必要的.

异构场景下编码计算方案：为了更好地处理异构分布式计算系统中的掉队节点，必须考虑设计有效的负载分配策略，以最大程度地减少总体作业执行时间. 在给定计算时间参数，即每个工作节点计算时间服从一个移位的指数分布时，异构编码矩阵乘法(heterogeneous coded matrix multiplication, HCMM)算法[33]解决了最优负载分配问题. HCMM 方案利用编码技术和计算负载分配策略，最大程度地减少了平均计算时间. 仿真结果表明，相比于未编码方案，未编码负载平衡方案和统一的负载分配编码方案，HCMM 分别将平均计算时间分别减少了 71％，53％和 39％.

虽然 HCMM 显著降低了计算延迟，但其解码复杂度很高. 在实际的分布式计算系统中，某些处理节点具有相同的计算能力分布，因此可以将它们组合在一起，形成一个群. 通过利用这种群结构和不同群之间的异构性[34-35]，并结合最优负载分配策略，不仅可以实现接近基于 MDS 码的编码方案的最佳计算时间，而且可以降低解码复杂度.

除了工作节点的异构能力之外，工作节点的可用资源也可能随时间而变化. 为了最大化工作节点的资源利用率，研究人员提出了适应工作节点时变特性的动态负载分配算法[36-38]. Keshtkarjahromi 等人[36]提出了一种编码协同计算协议(coded cooperative computation protocol, C3P). 在 C3P 中，主节点基于工作节点的响应来确定编码数据包的传输间隔. 对于不能在给定的传输间隔内完成任务的工作节点，它们等待下一个编码数据包的时间更长. 与不考虑工作节点动态资源异构性的 HCMM[33]相比，C3P 协议的计算延迟降低了 30％.

为了避免网络中掉队节点造成的延迟，大多数编码计算方案都将掉队节点视为“纠删节点”，这意味着它们的计算结果将被完全忽略. 然而很少有工作节点是完全不活动的，因此，掉队节点，特别是非持久性掉队节点所完成的计算结果是不可忽略的，需要更好地利用[91].

利用掉队节点的编码计算方案：为了利用这些掉队节点的计算能力，Ozfatura 等人[39]使用了多信息通信(multi-message communication, MMC)，其允许工作节点在完成分配任务的一部分时传输其计算结果. Ozfatura 等人将 MMC 和拉格朗日编码计算方案[48](Lagrange coded computing, LCC)相结合，以最大程度地缩短作业执行时间，但由于工作节点传输到主节点的消息数量增加而导致通信负载增加. Kiani 等人[40]将分配给工作节点的任务进一步分为较小的部分，并且允许工作节点之间交流其各自计算结果，这使得掉队节点所做的工作可以被充分利用.

Ferdinand 等人[41]提出一种分层编码计算方案，以利用所有工作节点的计算结果. 在该方案中，每个工作节点将分配的计算任务按层划分为多个子计算，然后按顺序进行处理，即在下一层计算开始之前，需要将已完成层的子计算的结果传输到主节点. Ferdinand 等人使用 MDS 码对每层任务进行编码，以使工作节点完成每层任务的计算时间大致相同. 随后该分层编码计算方案被 Kiani 等人[42]扩展到分布式矩阵-矩阵乘法和矩阵-向量乘法中. 对于这两种类型的乘法，Kiani 等人通过实验表明，虽然解码时间有所减少，但是分层编码的计算时间要比文献[39]中的方案长.

虽然编码计算方案可以降低通信负载并减少作业执行时间，但未编码的计算方案具有其自身的优

势，即，不需解码并允许部分梯度更新. 为了结合两种方案的优点，Ozfatura 等人[43]提出了编码部分梯度计算(coded partial gradient computation, CPGC)方案. 因为每个工作节点都有很高的概率完成第一次分配的任务，所以 CPGC 首先分配给工作节点未编码的子矩阵，在工作节点完成计算后，再分配给其编码子矩阵. 主节点能够使用一部分工作节点的全部计算结果和掉队节点的部分计算结果来更新梯度参数，从而减少任务执行的平均时间.

3.4 小结

在本节中，我们探讨了编码技术在不同分布式计算任务中的应用. 首先对上述方案进行总结回顾，如表 6 所示.

在矩阵-向量乘法任务中，研究人员不仅考虑如何使得恢复阈值最小，而且考虑了如何降低工作节点处的计算负载，以从整体上降低计算时间. 特别地，无码率编码[18]可以有效利用掉队节点所做的工作，但该方案工作节点和主节点通信轮次增高，带来较大的通信开销. 在矩阵-矩阵乘法中，现有研究方案都先将输入矩阵划分为较小的子矩阵，然后对子矩阵进行编码，生成编码矩阵. 乘积码[19]使用 MDS 码在两个维度上对矩阵进行编码，多项式码[20]，MatDot[21]，PolyDot[21]，纠缠多项式码[22]则使用随机数对子矩阵进行编码，只不过在矩阵分割形式上有所不同. 文献[20-22]解码最终结果的方法都可看成多项式插值问题. 但上述方案都未考虑矩阵的稀疏性，因此研究人员提出了稀疏码方案[23]，在计算速度上较其他方案有较大优势. 梯度下降编码方案则大致可分为两类，一类是基于 GC 的编码方案[24-27]，这些方案中工作节点返回局部梯度的线性组合；另一类方案是对数据进行编码的方案，虽然该方案的恢复阈值比 GC 编码方案低，但其对数据进行编码带来了额外的计算开销.

虽然大多数的研究都集中在编码的设计上，但是译码复杂度和工作节点的计算负载也会对计算时间产生很大的影响. 因此面向计算延迟的编码计算方案不应仅以降低恢复阈值为目标，而应权衡恢复阈值，计算负载和译码复杂度三者之间的关系，从而使整体计算时间最短. 除了 Reed-Solomon 码[81]和 LDPC 码[85]，更有效的低复杂度的解码方式需要进一步探索.

Table 6 Summary and review of coded computing schemes for computing delay

表 6 面向计算延迟的编码计算方案总结与回顾

计算任务类型	方案名称	核心思想/主要贡献	采用的编码技术	优/缺点
矩阵-向量乘法	MDS code[2]	使用 MDS 码对输入矩阵编码，使得主节点不必等待最慢节点便可恢复最终结果	MDS 码	优点：应对计算延迟的基础方法 缺点：引入额外计算量
	ShortDot[16]	通过在编码矩阵中引入稀疏性，减少在工作节点计算的点积长度	MDS 码	优点：相较与 MDS code 减少了工作节点的计算负载 缺点：恢复阈值比 MDS code 高
	S-对角线[17]	利用矩阵的对角结构来获得最佳恢复阈值和最佳计算负载	线性组合	优点：在相同恢复阈值下，较 MDS code 降低了工作节点计算负载 缺点：平均计算负载随掉队节点数目增加而增加
	无码率编码[18]	利用 LT 码的属性从有限的源符号集生成无限数量的编码符号	LT 码	优点：在相同恢复阈值下，具有更低的冗余计算和解码复杂度 缺点：需记录工作节点和存储数据的映射关系，增加存储开销
矩阵-	1D MDS code[19]	将输入矩阵沿垂直方向划分，对其中一个输入矩阵使用 MDS 码进行	MDS 码	优点：同 MDS code 缺点：引入额外计算量

<table>
<tr><td rowspan="7">矩阵乘法</td><td></td><td>编码</td><td></td><td></td></tr>
<tr><td>乘积码[19]</td><td>用 MDS 码对两个矩阵进行编码，随后将计算任务以$\sqrt{N}\times\sqrt{N}$的形式分配给工作节点</td><td>乘积码</td><td>优点：相比于 1D MDS code[19]进一步降低了恢复阈值
缺点：解码需要迭代进行，复杂度高</td></tr>
<tr><td>多项式码[20]</td><td>使用随机数对子矩阵进行编码，形成编码子矩阵，通过重构多项式来恢复最终结果</td><td>多项式码</td><td rowspan="4">优点：相较于 MDS 码和乘积码进一步降低恢复阈值，且恢复阈值不再随工作节点数量的变化而变化
缺点：随着工作节点数量的增加，编码和解码的计算成本远远高于乘积码[19]. 此外，其可以处理的工作节点的数量是有限制的，这对于涉及多达数千个节点的系统可能是不实用的</td></tr>
<tr><td>MatDot[21]</td><td>在$m=n$的情况下，通过只计算相关的交叉积，牺牲一定通信开销，实现比多项式码更低的恢复阈值</td><td>多项式码</td></tr>
<tr><td>PolyDot[21]</td><td>形式化恢复阈值和通信开销之间的折衷(多项式码和 MatDot 码是这个折衷曲线上的两个极端)</td><td>多项式码</td></tr>
<tr><td>纠缠多项式码[22]</td><td>该方案允许将输入矩阵在水平和垂直两个维度进行划分，划分后利用多项式码进行编码</td><td>多项式码</td></tr>
<tr><td>稀疏码[23]</td><td>利用输入和输出矩阵的稀疏性减少计算量，逼近最优阈值</td><td>加线性组合</td><td>优点：获得最优化的恢复阈值和计算负载
缺点：仅限于稀疏矩阵乘法，扩展性差</td></tr>
<tr><td rowspan="7">梯度下降算法</td><td>FRC[24]</td><td>工作节点被分为多个组，将数据平均分发给组中每个成员，每个数据批由多个工作节点进行计算</td><td>线性组合</td><td rowspan="4">优点：发掘梯度编码优势提高计算过程对掉队节点的容忍度
缺点：需要事先知道掉队节点的数据量. 实际上，一般情况下很难预测集群中掉队节点的个数，且数目存在动态变化可能</td></tr>
<tr><td>CRC[24]</td><td>基于一种循环分配策略对数据进行分配</td><td>线性组合</td></tr>
<tr><td>Cyclic MDS Codes[25]</td><td>编码矩阵中后续每列都是第一列的循环移位</td><td>CyclicMDS 码</td></tr>
<tr><td>Reed-Solomon Codes[26]</td><td>利用掩模矩阵，从 RS 码中选择合适的码字来构造编码矩阵</td><td>Reed-Solomon</td></tr>
<tr><td>BCC[27]</td><td>将数据分成多个数据批，由工作节点随机选择数据批进行计算</td><td>梯度编码</td><td>优点：无需关于掉队节点的先验知识；每个工作节点独立选择其数据批，方案扩展性好
缺点：本地任务完成快的计算节点在计算结束后一直处于空闲状态，造成计算资源的浪费</td></tr>
<tr><td>ICML2018[28]</td><td>将部分梯度按维度进行划分，降低通信成本</td><td>梯度编码</td><td>优点：提供了一种考量通信开销的梯度编码方案
缺点：恢复阈值较其他梯度编码方案高</td></tr>
<tr><td>PCR[29]</td><td>利用代数结构，直接对数据批进行编码，无需工作节点对计算的梯度进行编码</td><td>多项式码</td><td>优点：相比于基础梯度编码提供更低恢复阈值，计算延迟更小
缺点：相比于基础梯度编码，本方案的编码过程会带来额外计算开销</td></tr>
<tr><td>卷积运算</td><td>Coded Convolution[30]</td><td>将两个向量拆分为指定长度的多个部分，并使用 MDS 码对其中一个向量进行编码</td><td>MDS 码</td><td rowspan="2">优点：支撑以卷积和傅里叶变换为基础计算元素的任务场景
缺点：适用场景单一</td></tr>
<tr><td>傅里叶</td><td>编码傅里叶变换[31]</td><td>利用递归结构和离散傅里叶变换运算的线性度</td><td>MDS 码</td></tr>
</table>

变换				
非线性计算	Generalized PolyDot[88]	对神经网络中每一层的线性运算进行编码，允许每一层的训练中出现错误	多项式码	优点：有效降低神经网络训练中的计算延迟 缺点：不适用于具有大量非线性运算的神经网络
	SOSP2019[32]	设计一种学习编码和解码功能的神经网络结构和训练方法	端到端模型	优点：不需要手工设计编码方法，能更好的应用于非线性运算任务 缺点：神经网络训练的解码函数存在误差
异构计算	HCMM[33]	有机整合编码和负载分配策略	LT 码	优点：通过最佳负载分配策略降低异构计算延迟 缺点：解码复杂度高
	C3P[36]	设计一种动态的负载分配策略，主节点基于工作节点的响应性来确定编码数据包的传输间隔	LT 码	优点：根据工作节点响应来确定传输间隔，实现了动态的负载分配，和 HCMM 相比速度提高30%. 缺点：通信负载较高
其他	分层编码[41]	每个工作节点将分配的计算任务按层划分为多个子计算	MDS 码	优点：充分利用掉队节点的计算结果，进一步减少了计算延迟 缺点：主节点和工作节点需要耗费更多的通信成本
	CPGC[43]	通过在在迭代第一轮发送未编码数据，第二轮使用 MDS 码编码数据，融合了编码方案和未编码方案的优势，	MDS 码	

4 面向安全隐私的编码计算

在分布式计算系统中，好奇的工作节点可能串通起来以获取原始数据的信息，而受到拜占庭攻击的恶意工作节点可能故意提供错误的结果[92]，从而误导最终结果. 抗恶意节点的编码计算方案往往通过设计具有纠错能力的解码方式以定位恶意节点，从而获取正确结果. 防隐私泄露的编码计算方案则通过引入一个随机均匀矩阵对输入数据进行编码，从而达到掩藏真实数据的目的.

面向安全隐私的编码计算方案的目标是通过利用编码技术，在系统中存在M个恶意节点和T个共谋节点时，依然可以获得正确结果，并且不泄露原始数据的任何信息. 下面我们将分别从抗恶意节点和防隐私泄露两方面对当前编码计算方案进行分析.

4.1 抗恶意节点的编码计算

在分布式计算的应用场景中，如战场物联网[93]，联邦学习[94]等，工作节点的计算可能是不可信的. 因此，一个重要的问题是是否能够在拜占庭敌手的存在下可靠地执行分布式计算，并且该问题由来已久[95]. 最近，编码技术被应用到分布式计算系统中，以抵抗分布式计算系统中的拜占庭攻击问题.

DRACO：在分布式梯度下降算法中，Chen 等人分别利用 FRC 和 CRC[24]对数据进行编码(具体编码方法见 4. 3 节)，并针对两种不同的编码方法，设计了两种不同的解码方案.

具体来说，利用 FRC 对数据进行编码时，DRACO[44]让每一组中的节点来计算相同的梯度的和. 为了解码同一组中的计算节点返回的输出，主节点使用多数投票算法来选择其中一个值. 这保证了只要每组中少于一半的节点是恶意的，主节点将选择到正确的结果. 由此，主节点可以获得所有组的正确的结果. 利用 CRC 对数据进行编码时，DRACO 利用离散傅里叶逆变换矩阵构造一个函数$\varphi(\cdot)$. 函数$\varphi(\cdot)$可以利用文件分配矩阵来计算恶意工作节点的索引. 因此，DRACO 可以利用非恶意节点返回的值来解码最终结果. 仿真结果表明，DRACO 算法在 MNIST 数据集上实现 90%的测试精度时，比中值聚合方案[96]快 3 倍以上. 随后，Rajput 等人[97]使用梯度滤波器进一步提高了 DRACO 的计算效率.

响应冗余：Gupta 等人[45]提出“响应冗余”的概念，并利用随机检测的方法来提高系统的工作效率. 其核心思想是让工作节点响应两次，且在每次响应中，为同一个工作节点分配不同的子数据集. 通过对比两次响应的结果，主节点可以确定系统中恶意节点的索引.

然而，引入响应冗余使得工作节点的计算成本比 DRACO[44]高两倍. 为了降低计算成本，Gupta 等人提出了随机方案，即主节点随机选择中间迭代的结果进行检测，或者，在每一次迭代中，主节点以概率$p(0<p<1)$进行错误检查. 通过这种方式，可以减少计算冗余，同时可以确定恶意工作节点.

数据编码方案：然而，上述方案忽略了梯度计算中的代数结构，这使得工作节点都需额外存储多组数据，并且存储量会随着恶意节点的个数的增加而增加. 与上述方案不同，Data 等人[46]通过对数据进行编码，基于错误校正[47]提出了一种抗拜占庭攻击的分布式优化算法. 文献[46]的要点是通过设计一个具有错误校正功能的矩阵对原始数据进行编码，从而可以在解码阶段定位恶意节点. 和上述方案相比，该方案的存储冗余较少，并且减少了工作节点的计算时间. 其可容忍的最高恶意节点数为$\left\lfloor\frac{m-1}{2}\right\rfloor$，达到了信息论中的最优值.

LCC：针对任意多元多项式的计算任务，Yu 等人[48]提出一种拉格朗日编码计算(Lagrange coded computing, LCC)方案，该方案对输入数据进行编码，不仅可以降低计算延迟，而且可以抵抗恶意节点的攻击，同时保护数据的隐私.

考虑一个具有N个工作节点的分布式计算系统，其目标是对大型数据集$X=(X_1,X_2,\dots,X_K)$中每一个X_i，计算$f(X_i)$(f是给定的阶为$\deg(f)$的多元多项式). 如果通过利用合适的编码方法可以在系统中存在S个掉队节点，A个恶意节点，T个合谋节点时仍然获得正确结果且不暴露数据隐私，则就说该方案可以实现三元组(S,A,T).

如果$N\geq(K+T-1)\deg(f)+S+2A+1$，则 LCC 可以实现三元组$(S,A,T)$. 该结果的意义在于，增加一个工作节点(即$N$增加 1)，LCC 可以使掉队节点的弹性增加 1，或者使恶意节点的鲁棒性提高$1/2$，同时保持数据的隐私.

下面以$f(\boldsymbol{X}_i)=\boldsymbol{X}_i^2, K=2, N=8, (S,A,T)=(1,1,1)$为例简要介绍 LCC 的编码过程. 其中$\boldsymbol{X}_i$为$\sqrt{M}\times\sqrt{M}$的方阵. 因为$K=2$，所以输入数据$\boldsymbol{X}$被分为两个子矩阵$\boldsymbol{X}_1$和$\boldsymbol{X}_2$，LCC 的要点是选取一个均匀随机矩阵$\boldsymbol{Z}$，并使用拉格朗日插值多项式编码$(\boldsymbol{X}_1,\boldsymbol{X}_2,\boldsymbol{Z})$，编码过程如式(12)所示.

$$u(z)\triangleq\boldsymbol{X}_1\cdot\frac{(z-2)(z-3)}{(1-2)(1-3)}+\boldsymbol{X}_2\cdot\frac{(z-1)(z-3)}{(2-1)(2-3)}+\boldsymbol{Z}\cdot\frac{(z-1)(z-2)}{(3-1)(3-2)} \quad (12)$$

然后，在有限域$\mathbb{F}$中确定 8 个不同的数$\{\alpha_i\}_{i=1}^8$，并且$\{\alpha_i\}_{i=1}^8\cap[2]=\emptyset$. 接下来，让工作节点 1-8 分别存储$u(\alpha_1),\dots,u(\alpha_8)$. 值得注意的是，每个工作节点获取的数据为通过使用$\lambda\boldsymbol{Z}$掩藏的$\boldsymbol{X}_1$和$\boldsymbol{X}_2$的线性组合，其中λ是一非零值. 因为$\boldsymbol{Z}$是均匀随机的，所以可以保证$T=1$时数据的隐私性. 其次，对于工作节点j，其计算$f(\widetilde{\boldsymbol{X}}_j)=f\left(u(\alpha_j)\right)$，本质上是计算多项式$f\left(u(z)\right)$在点$\alpha_j$的值，该多项式的阶数最高为 4. 通常来说，一个 4 次多项式可以利用 5 个不同点的值插值得到. 然而，如果存在$A=1$个恶意节点，$S=1$个掉队节点，则需要主节点使用 Reed-Solomon 解码器，并且需要额外的 3 个点处的值(每多一个掉队节点需要额外的 1 个值，每多 1 个恶意节点需要额外的 2 个值). 最后，主节点解码多项式$f\left(u(z)\right)$后，可以计算$f\left(u(1)\right)$和$f\left(u(2)\right)$来获得$f(\boldsymbol{X}_1)$和$f(\boldsymbol{X}_2)$.

LCC 可以应用于计算任务是多元多项式的任何计算场景，因此涵盖了机器学习中许多计算任务，例如，线性计算，双线性计算，一般张量代数和梯度下降.

4.2 防隐私泄露的编码计算

在分布式计算模型中，要计算的输入数据可能包含大量敏感信息，如个人位置信息和医疗信息. 但是主节点有时需要利用可疑但有用的工作节点. 因此保护输入数据的敏感信息不被泄露变得至关重要.

4.2.1 面向多项式运算的隐私编码方案

作为 LCC 的扩展，So 等人[49]提出了一种快速且具有隐私保护功能的分布式机器学习框架——CodedPrivateML. CodedPrivateML 在每一轮训练中分两步秘密共享数据集和模型参数. 首先，采用随机量化方法将数据集和每轮的权重向量转换为有限域. 然后，CodedPrivateML 使用 LCC 编码技术将量化值进行编码，并将编码数据分发给各工作节点，以保证数据隐私. CodedPrivateML 面临的挑战是，LCC 只能用于多项式求值形式的计算. 因此，CodedPrivateML 利用多项式逼近来处理涉及到 sigmoid 函数时的非线性计算，从而可以对 LCC 编码的数据进行 Logistic 回归. 在 Amazon EC2 集群上的实验表明，CodedPrivateML 比基于 BGW[51]的安全多方计算方法快 34 倍，并且可以保护数据的隐私.

针对梯度类型的计算，Yu 等人[60]提出了一种新颖的编码计算方案，称为“调和编码”，该方案适用于任何类型的梯度计算，同时可以保护输入数据的隐私. 和 LCC 不同的是，调和编码使用一个随机变

量Z来构造具有递归结构的中间值P，然后使用P对数据进行编码. 由于Z是随机的，因此原始数据被掩藏，数据隐私得到保护. 又由于中间值P的特殊结构，所以使得该方案相比于 Shamir 秘密共享方案[98]和LCC[48]方案，在计算梯度型函数时需要更少的工作节点. 表 7 总结了三种方案在保护数据隐私时所需的最低工作节点数.

Table 7 Comparison of the minimum number of working nodes

表 7 最低工作节点数对比

	Shamir[98]	LCC[48]	调和编码[60]
最少工作节点数	$K(\deg g+1)$	$K\deg g+1$	$K(\deg g-1)+2$

利用多项式码[20]可以有效降低分布式计算系统中的计算延迟，且恢复阈值与工作节点的数量无关. 在此基础上，Nodehi 等人[50]将多项式码与 BGW 方案[51]结合在一起，提出了一种多项式共享方案. 在该方案中，数据源自外部资源，因此必须对工作节点和主节点同时保持私有. 与 BGW 方案不同，Nodehi 等人使用多项式码对数据集进行编码. 文献[50]可用于执行加法,乘法和多项式函数等多种计算过程，同时可以减少完成任务所需的工作节点数.

在文献[50]中,执行计算的数据集被分成多个子数据集，每个子数据集被编码并分配给工作节点. 在这种情况下，较快完成任务的工作节点将在等待掉队节点时处于空闲状态. 为了进一步减少计算延迟，Kim 等人[99]利用计算冗余提出了一种私有异步多项式编码方案，该方案将一个计算任务分成几个相对较小的子任务，并将子任务分配给每个工作节点. 除了保留多项式共享方案的隐私保护特性外，该方案有以下两个优点：1)计算能力有限的掉队节点可以成功地完成较小的子任务；2)计算速度较快的工作节点被分配更多的任务，在整个任务计算期间持续工作，从而减少了整个任务的执行时间.

然而，文献[50，99]主要利用多项式码来保护数据隐私，这在某些方面是有限制的，例如，它只允许将矩阵按列划分. 因此，Nodehi 等人[100]利用纠缠多项式码[22]提出了一种纠缠多项式码共享方案，该方案作为多项式共享的扩展，进一步减少了数据共享阶段的限制，从而在满足隐私约束的同时减少执行相同计算任务所需的工作节点的数量.

4. 2. 2 面向矩阵乘法运算的隐私编码方案

针对矩阵乘法的特点研究人员考虑了以下两种隐私情况：

1)单边隐私：输入数据中只有一个是私有的，另一个输入对工作节点是公开的.

2)双边隐私：两个输入数据都是私有的，工作节点无法获得所有输入的有关信息.

下面将分别对两种隐私情况下的编码计算方案进行介绍和分析.

单边隐私：Bitar 等人[52-53]提出使用阶梯码替换线性秘密共享方案[101]来对数据进行编码. 作为说明，考虑一个有 3 个工作节点的分布式计算场景，其目标是分布式计算矩阵-向量乘法$\boldsymbol{Ax}$，并保护输入数据$\boldsymbol{A}$的隐私(单边隐私). 为保护数据隐私，主节点生成一个与$\boldsymbol{A}$具有相同维数的随机矩阵$\boldsymbol{R}$. 当使用线性秘密共享方案时，数据和随机矩阵不被分割，作为一个整体进行编码和传输，如表 8 所示.

Table 8 Transmission contents of two different schemes

表 8 两种不同共享方案的传输内容

	S_1	S_2	S_3
线性秘密共享编码	$\boldsymbol{R}$	$\boldsymbol{R}+\boldsymbol{A}$	$\boldsymbol{R}+2\boldsymbol{A}$
阶梯码	$\boldsymbol{A}_1+\boldsymbol{A}_2+4\boldsymbol{R}_1$，$\boldsymbol{R}_1+\boldsymbol{R}_2$	$\boldsymbol{A}_1+2\boldsymbol{A}_2+4\boldsymbol{R}_1$，$\boldsymbol{R}_1+2\boldsymbol{R}_2$	$\boldsymbol{A}_1+3\boldsymbol{A}_2+4\boldsymbol{R}_1$，$\boldsymbol{R}_1+3\boldsymbol{R}_2$

因此，主节点必须等待其中任意两个工作节点的完整结果,才能解码最终结果,如$\boldsymbol{Ax}=S_2\boldsymbol{x}-S_1\boldsymbol{x}$. 当使用阶梯码时，数据和随机矩阵在传输给工作节点之前被分割成子矩阵. 随后主节点发送给工作节点两组编码数据，如表 8 所示. 因此，每个工作节点有两个子任务. 每个工作节点按顺序执行计算任务，并将计算结果返回给主节点. 当工作节点完成了足够多的子任务后，主节点可以通知工作节点停止计算. 例如，当主节点接收到所有工作节点的第一个子任务的计算结果，或者主节点接收到任意两个工作节点的所有任务的结果时，可以从中解码获得最终的结果. 在有 4 个工作节点的分布式计算场景下，Bitar 等人在 Amazon EC2 集群上进行实验，结果表明使用阶梯码的平均等待时间比线性秘密共享方案减少了 59%.

Bitar 等人[54]考虑到战场物联网(internet of battlefield things, IoBT)应用和设备的隐私要求，以及战场边缘设备资源的异构和时变性，提出了一种私密无速率自适应(private and rateless adaptive coded computation, PRAC)编码计算方案. 和阶梯码相同，PRAC 也引入随机矩阵来掩藏原始数据，但

不同的是 PRAC 考虑了工作节点的异构性，使用喷泉码[81]对数据进行编码，并根据工作节点的响应情况，动态的生成和分配随机矩阵和编码数据.

双边隐私：Chang 等人[55]首次考虑了矩阵乘法中双边隐私的情况，并设计了一个可行性方案. 具体来说，输入矩阵被分割成子矩阵并用随机矩阵编码. 下面以工作节点数为 8，共谋节点数为 1，来说明文献[55]的编码过程.

主节点首先将输入矩阵$\boldsymbol{A}$，$\boldsymbol{B}$按如式(13)所示方式进行划分.

$$\boldsymbol{A}=\begin{bmatrix}\boldsymbol{A}_1\\ \boldsymbol{A}_2\end{bmatrix}\qquad \boldsymbol{B}=[\boldsymbol{B}_1\ \ \boldsymbol{B}_2] \tag{13}$$

其中$\boldsymbol{A}_1,\boldsymbol{A}_2\in\mathbb{F}^{m/2\times n}$，$\boldsymbol{B}_1,\boldsymbol{B}_2\in\mathbb{F}^{n\times p/2}$. 然后，分别为$\boldsymbol{A},\boldsymbol{B}$各生成一个随机矩阵$\boldsymbol{K}^A\in\mathbb{F}^{m/2\times n},\boldsymbol{K}^B\in\mathbb{F}^{n\times p/2}$. 接下来，主节点为每个工作节点$i$选择不同的参数$x_i$，生成编码数据，如式(14)，(15)所示.

$$\widetilde{\boldsymbol{A}}_i=\boldsymbol{A}_1+\boldsymbol{A}_2x_i+\boldsymbol{K}^Ax_i^2 \tag{14}$$

$$\widetilde{\boldsymbol{B}}_i=\boldsymbol{B}_1+\boldsymbol{B}_2x_i^3+\boldsymbol{K}^Bx_i^5 \tag{15}$$

工作节点在接收到编码数据后，每个工作节点i计算$\widetilde{\boldsymbol{A}}_i\widetilde{\boldsymbol{B}}_i$，并将结果返回给主节点. 则每个工作节点的计算任务相当于计算多项式$h(x)$在点$x=x_i$的值，$h(x)$如式(16)所示.

$$h(x)=\boldsymbol{A}_1\boldsymbol{B}_1+\boldsymbol{A}_2\boldsymbol{B}_1x+\boldsymbol{K}^A\boldsymbol{B}_1x^2+\boldsymbol{A}_1\boldsymbol{B}_2x^3+\boldsymbol{A}_2\boldsymbol{B}_2x^4+(\boldsymbol{K}^A\boldsymbol{B}_2+\boldsymbol{A}_1\boldsymbol{K}^B)x^5+\boldsymbol{A}_2\boldsymbol{K}^Bx^6+\boldsymbol{K}^A\boldsymbol{K}^Bx^7 \tag{16}$$

观察式(16)可知，多项式$h(x)$中 4 项$(\boldsymbol{A}_1\boldsymbol{B}_1,\boldsymbol{A}_2\boldsymbol{B}_1\boldsymbol{x},\boldsymbol{A}_1\boldsymbol{B}_2\boldsymbol{x},\boldsymbol{A}_2\boldsymbol{B}_2\boldsymbol{x})$的系数可组成最终结果. 因此，主节点可以采取多项式插值法确定该多项式，从而获得所需的系数.

在上述编码思想的基础上，Kakar 等人[56]针对双边隐私提出了一种新的任意矩阵划分下的对齐秘密共享方案，以优化下载比率和恢复阈值. 与文献[55]将两个输入矩阵划分为相同数量的子矩阵不同，Kakar 等人[56]将输入矩阵划分为不同数量的子矩阵. 与文献[55]相比，文献[56]在下载比率，可容忍共谋服务器数量和计算复杂度等方面都有所改进.

受多项式码[20]的启发，Yang 等人[57]将多项式码扩展到保护双边隐私的矩阵乘法中. 和文献[55-56]不同，该方案仅需为每个输入矩阵生成 1 个随机矩阵便可完成编码. 主节点的解码过程与多项式码方案相似，都可视为多项式插值问题. 和文献[56]相比，该方案的编码复杂度要低，并且在下载比率相似的情况下，可实现更低的恢复阈值.

4.3 小结

在本节中，我们讨论了编码计算在分布式计算系统中对抗恶意节点以及保护数据隐私方面的应用和研究. 首先我们对上述方案进行总结和回顾，如表 9 所示.

Table 9 Summary and review of coded computing schemes for security and privacy

表 9 面向安全和隐私的编码计算方案总结和回顾

问题	方案名称	核心思想/主要贡献	编码技术	优/缺点
拜占庭攻击	DRACO[44]	采用了 FRC 和 CRC 两种编码方式，针对这两种方式，分别采用多数投票算法和检测函数，对恶意节进行检测和定位	梯度编码	优点：相同精度下，比未编码方案速度更快 缺点：需要计算冗余，工作节点计算效率较低
	ArXiv2019[45]	引入“响应冗余”，主节点随机选择中间迭代的结果进行检测	线性组合	优点：采用随机检测的方式提供良好计算速度 缺点：需要分配给工作节点更多的数据，存储开销较大
	ISIT2019[46]	对数据进行编码，利用错误校正方法定位恶意节点索引	其他	优点：对原始数据进行编码，计算冗余低，存储开销低 缺点：编解码复杂度高
	LCC[48]	利用拉格朗日多项式对数据进行编码，并使用 Reed-Solomon 解码器进行纠错	拉格朗日编码	优点：可以同时减少计算延迟，抵抗恶意节点的攻击和保护数据的隐私，编解码复杂度低 缺点：只适用于多项式计算
数据隐	LCC[48]	引入随机矩阵掩藏原始数据，并利用拉格朗日多项式对数据进行编码	拉格朗日编码	优点：工作节点的存储和计算开销较少 缺点：需引入更多工作节点

私	CodedPrivateML[49]		使用 LCC 方案对数据和模型参数进行编码，将 sigmoid 函数用 sigmoid 函数的多项式逼近来替代	拉格朗日编码	LCC 在非线性梯度计算中的扩展
	调和编码[60]		引入随机变量 Z 掩藏输入数据，并设计“调和”参数降低所需工作节点数量	其他	优点：和 LCC 相比在相同共谋节点数的情况下进一步降低了所需的工作节点数 缺点：编码复杂度高
	单边隐私	阶梯码[53]	使用阶梯码替换线性秘密共享编码方案来对数据进行编码	阶梯码	优点：与线性秘密共享方案相比可以显著降低计算和通信成本 缺点：只考虑了单边隐私
		PRAC[54]	使用喷泉码对数据进行编码，并根据工作节点的响应情况，动态的生成和分配随机矩阵以及编码数据	喷泉码	优点：充分利用工作节点的计算资源 缺点：主节点和工作节点之间通信成本增加
	双边隐私	对齐编码[55]	采用随机矩阵对数据进行编码，通过多项式插值法解码获得所需结果	对齐编码	优点：首次考虑了双边隐私的情况，为两个输入矩阵都提供隐私保障 缺点：输入矩阵分割时只能分割为相同的数目，且需要生成多个编码矩阵
		ArXiv2018[56]	将输入矩阵划分为不同数量的子矩阵，降低下载比率	对齐编码	优点：和对齐编码相比，下载比率，可容忍共谋服务器数量和计算复杂度等方面都有所改进 缺点：需要生成多个编码矩阵才能提供隐私保障
		TIFS2019[57]	将多项式码在保持双边隐私的矩阵乘法中扩展应用	多项式码	优点：编码计算开销低，且下载比率较对齐编码高 缺点： 随着工作节点数量的增加，编解码的计算成本升高

抗恶意节点的编码计算方案往往通过设计具有纠错能力的解码方式以获取正确结果. DRACO[44]采用文献[24]中 FRC 和 CRC 两种编码方式，但在解码阶段分别引入了多数投票算法和检测函数. Gupta 等人[45]采用基于 GC 的编码方式对梯度进行编码，但解码阶段则引入响应冗余，用以检测恶意节点. Data 等人[46]在编码阶段设计具有错误校正能力的编码矩阵，利用该矩阵对输入数据进行编码，以在解码阶段可以对恶意节点进行定位. LCC[48]通过构造多项式的方式对数据进行编码，并在解码阶段利用具有纠错能力的 Reed-Solomon 解码器进行解码. 防隐私泄露的编码计算方案在对数据进行编码时，额外引入一个随机矩阵，以此对原数据进行掩藏. 特别是，针对矩阵乘法任务，本节讨论了单，双边隐私两种情况. 一般来说，利用特定运算的代数结构，例如卷积任务或梯度计算，实现编码和解码的方案通常表现更高效.

5 研究展望

目前，关于编码计算在学术界的研究才刚刚起步，存在许多问题需要解决，提供一系列研究机遇，本文从以下 3 个方面对未来可能的研究方向进行阐述.

1)通信与计算的松耦合. 在将编码应用于数据洗牌的相关工作中，通信和计算是松耦合的. 即，其只关心通信目标：主节点希望以最小的通信负载使得每个工作节点都能获得执行计算任务所需要的数据. 为了实现最小化通信负载，往往需要计算节点额外存储大量数据(存储冗余)，而这些冗余数据只帮助传输并不用于计算[15].

未来一个可能的工作方向是提高通信与计算的耦合性. 如果多余的存储数据用于计算，则可以获得潜在的计算增益. 例如，在训练一个分类器模型时，多余的数据样本也有助于模型训练. 设计一个计算和通信耦合度较高的方案，并研究其中通信与计算两者之间的折衷是必要的.

2)掉队节点的利用和通信负载的平衡. 虽然掉队节点的运行速度比平均速度慢，但其完成的计算结果对仍然有助于实现最终计算任务. 例如，在文献[58]中，为了使线性逆求解器在截止期约束下的均方误差最小，掉队节点的结果被视为软误差. 然而，当前利用掉队节点计算结果的方案中，往往需要执行更多的通信轮次. 而高通信成本是分布式计算系统的另一个瓶颈. 未来一个可能的研究方向是在利用掉队节点的计算结果时，探索使主节点和工作节点之间通信成本最小化的优化方法.

3)安全隐私中的异构. 由于掉队节点是分布式计算中的一个基本问题，因此面向计算延迟的编码计算方案考虑了异构场景. 然而，在编码计算相关工作中，其他方面的异构网络问题还没有得到充分的研究[57]. 例如，工作节点可能有不同程度的声誉[102]. 保护数据隐私的手段可以只针对新工作节点或声誉低的工作节点进行，而对于受信任的工作节点则可能不需要. 因此，未来一个可能的研究方向是考虑工作节点在声誉上的异构性，从而采取不同的隐私保护策略，以此降低可信节点的计算复杂度.

6 结束语

几十年来，编码理论在抗噪声方面的作用得以深入研究，并被广泛应用于多种场景中，成为日常基础设施(如，智能手机，笔记本电脑，WiFi 和蜂窝系统等)的一部分. 编码计算将编码理论和分布式计算系统相结合，旨在通过注入计算冗余创造编码机会，实现降低通信成本，提高计算速度和保护数据安全等目标. 本文通过区分研究目标将已有编码计算方案分为 3 类，对相应范畴下的研究现状进行了详细阐述，对比分析了典型方案，总结归纳了编码计算面临的挑战及研究方向，预期为分布式计算的研究人员带来启发和参考.

本文作者中蔡志平教授是本项目的构思者及负责人，指导论文写作; 周桐庆博士对本文关键性理论和知识性内容进行批评性审阅; 郑腾飞是综述的主要撰写人; 吴虹佳完成相关资料的收集和分析. 周桐庆博士和蔡志平教授贡献等同，同为通信作者. 全体作者都阅读并同意最终的文本.

参 考 文 献

[1] Li Songze, Mohammad Ali Maddah-Ali, A. Salman Avestimehr. A fundamental tradeoff between computation and communication in distributed computing[J]. IEEE Transactions on Information Theory, 2018,64(1): 109-128

[2] 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

[3] Wang Yan, Li Nianshuang, Wang Xiling, et al. Coding-based performance improvement of distributed machine learning in large-scale clusters[J]. Journal of Computer Research and Development, 2020,57(03):542-561(in Chinese)
(王艳,李念爽,王希龄,钟凤艳.编码技术改进大规模分布式机器学习性能综述[J].计算机研究与发展,2020,57(03):542-561.)

[4] Li Songze, Avestimehr S. Coded Computing: Mitigating Fundamental Bottlenecks in Large-scale Distributed Computing and Machine Learning[M]. Now Foundations and Trends, 2020

[5] Chowdhury M, Zaharia M, Ma J, et al. Managing data transfers in computer clusters with orchestra [J]. Acm Sigcomm Computer Communication Review, 2011, 41(4):98-109

[6] Zhang Zhuoyao, Cherkasova L, Loo B T. Performance Modeling of MapReduce Jobs in Heterogeneous Cloud Environments[C]//Proc of the Sixth International Conference on Cloud Computing, Piscataway, NJ: IEEE, 2013:839–846

[7] Neeraja J Yadwadkar, Bharath Hariharan, Joseph E Gonzalez, et al. Multi-task learning for straggler avoiding predictive job scheduling [J]. Journal of Machine Learning Research, 2016, 17(1): 3692–3728.

[8] Jeffrey D, Luiz A B. The tail at scale[J]. Communications of the Acm, 2013, 56(2):74-80

[9] Li Songze, Yu Qian, Mohammad Ali Maddah-Ali, et al. A Scalable framework for wireless distributed computing [J]. IEEE/ACM Transactions on Networking, 2017, 25(5):2643-2654

[10] Li Songze, Maddah-Ali M A, Avestimehr A S. Coded distributed computing: straggling servers and multistage dataflows[C]//Proc of the 54th Annual Allerton Conference on Communication, Control, and Computing, Piscataway, NJ: IEEE, 2016:164-171

[11] Ezzeldin Y H, Karmoose M, Fragouli C, et al. Communication vs distributed computation: an alternative trade-off curve[C]// Proc of the IEEE Information Theory Workshop, Kaohsiung, Piscataway, NJ: IEEE, 2017: 279-283

[12] Kiamari M, Wang Chenwei, Avestimehr A S, et al. On heterogeneous coded distributed computing[C]// Proc of the IEEE Global Communications Conference, Piscataway, NJ: IEEE, 2017: 1-7

[13] Attia M A, Tandon R. Near optimal coded data shuffling for distributed learning[J]. IEEE Transactions on Information Theory, 2019, 65(11): 7325-7349

[14] Li Songze, Maddahali M A, Avestimehr A S, et al. Compressed coded distributed computing[C]// Proc of the IEEE International Symposium on Information Theory (ISIT), Piscataway, NJ: IEEE, 2018: 2032-2036

[15] Song Linqi, Fragouli C, Zhao Tianchu. A pliable index coding approach to data shuffling[J]. IEEE Transactions on Information Theory, 2020, 66(3): 1333–1353

[16] Dutta S, Cadambe V R, Grover P, et al. “Short-Dot”: computing large linear transforms distributedly using coded short dot products[J]. IEEE Transactions on Information Theory, 2019, 65(10): 6171-6193

[17] Wang Sinong, Liu J, Shroff N B, et al. Fundamental limits of coded linear transform[J]. arXiv: Information Theory, 2018

[18] Mallick A, Chaudhari M, Joshi G, et al. Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication [J]. Measurement and Modeling of Computer Systems, 2020, 3(1):95–96

[19] Lee K, Suh C, Ramchandran K. High-dimensional coded matrix multiplication. [C]// Proc of the IEEE International Symposium on Information Theory (ISIT), Piscataway, NJ: IEEE, 2017:2418–2422

[20] Yu Qian, Maddah-Ali M A, Avestimehr A S. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication[J]. arXiv preprint arXiv:1705.10464, 2017.

[21] Fahim M, Jeong H, Haddadpour F, et al. On the optimal recovery threshold of coded matrix multiplication[J].2018, arXiv:1859.10761

[22] Yu Qian, Ali M, Avestimehr A S, et al. Straggler mitigation in distributed matrix multiplication: fundamental limits and optimal coding[J]. IEEE Transactions on Information Theory, 2020,66(3):1920–1933

[23] Wang Sinong, Liu J, Shroff N. Coded sparse matrix multiplication[J]. arXiv preprint arXiv:1802.03430, 2018

[24] Tandon R, Lei Q, Dimakis A G, et al. Gradient coding: avoiding stragglers in distributed learning[C]//Proc of the International Conference on Machine Learning (ICML), New York, NY: ACM, 2017: 3368-3376

[25] Raviv N, Tandon R, Dimakis A G, et al. Gradient Coding from cyclic mds codes and expander graphs[C]//Proc of the International Conference on Machine Learning (ICML), New York, NY: ACM, 2018: 4302-4310

[26] Halbawi W, Azizan N, Salehi F, et al. Improving distributed gradient descent using reed-solomon codes[C]//Proc of the International Symposium on Information Theory (ISIT), Piscataway, NJ: IEEE, 2018: 2027-2031

[27] Li Songze, Kalan S M M, Avestimehr A S, et al. Near-optimal straggler mitigation for distributed gradient methods [C]// Proc of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Piscataway, NJ: IEEE, 2018:857–866

[28] Ye Min, Abbe E. Communication-computation efficient gradient coding [C]//Proc of the International Conference on Machine Learning (ICML), New York, NY: ACM, 2018:5606–5615

[29] Li Songze, Kalan S M, Yu Qian, et al. Polynomially coded regression: optimal straggler mitigation via data encoding[J]. 2018, arXiv:1878.16781

[30] Dutta S, Cadambe V R, Grover P, et al. Coded convolution for parallel and distributed computing within a deadline[C]//Proc of the IEEE International Symposium on Information Theory (ISIT), Piscataway, NJ: IEEE, 2017: 2403–2407

[31] Yu Qian, Maddahali M A, Avestimehr A S, et al. Coded fourier transform[J] arXiv:Information Theory, 2018

[32] Kosaian J, Rashmi K V, Venkataraman S, et al. Parity models: erasure-coded resilience for prediction serving systems. //Proc of the 27th ACM Symposium on Operating Systems Principles, New York, NY: ACM, 2019:30–46

[33] Reisizadeh A, Prakash S, Pedarsani R, et al. Coded computation over heterogeneous clusters[J]. IEEE Transactions on Information Theory, 2019, 65(7): 4227-4242

[34] Kim M, Sohn J, Moon J, et al. Coded matrix multiplication on a group-based model[C]//Proc of the International Symposium on Information Theory (ISIT), Piscataway, NJ: IEEE, 2019: 722-726

[35] Kim D, Park H, Choi J, et al. Optimal load allocation for coded distributed computation in heterogeneous clusters[J]. arXiv: Distributed, Parallel, and Cluster Computing, 2019

[36] Keshtkarjahromi Y, Xing Yuxuan, Seferoglu H, et al. Dynamic heterogeneity-aware coded cooperative computation at the edge[C]//Proc of the IEEE International Conference on Network Protocols, Piscataway, NJ: IEEE, 2018: 23-33

[37] Narra K G, Lin Z, Kiamari M, et al. Slack squeeze coded computing for adaptive straggler mitigation[C]//Proc of the International Conference for High Performance Computing, New York, NY: ACM, 2019: 1-16

[38] Kim, Kwang Taik, Ma Chiang et al. Coded edge computing[C]// Proc of the IEEE International Conference on Computer Communications (INFOCOM), Piscataway, NJ: IEEE, 2020:237-246

[39] Ozfatura E, Gunduz D, Ulukus S, et al. Speeding up distributed gradient descent by utilizing non-persistent stragglers[C] // Proc of the International Symposium on Information Theory (ISIT), Piscataway, NJ: IEEE, 2019: 2729-2733

[40] Kiani S, Ferdinand N S, Draper S C, et al. Exploitation of stragglers in coded computation[C] //Proc of the International Symposium on Information Theory (ISIT), Piscataway, NJ: IEEE, 2018: 1988-1992

[41] Ferdinand N S, Draper S C. Hierarchical coded computation[C]// Proc of the International Symposium on Information Theory (ISIT), Piscataway, NJ: IEEE, 2018: 1620-1624

[42] Kiani S, Ferdinand N S, Draper S C, et al. Hierarchical coded matrix multiplication[J]. arXiv: Information Theory, 2019

[43] Ozfatura E, Ulukus S, Gündüz D. Distributed gradient descent with coded partial gradient computations[C]//Proc of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Piscataway, NJ: IEEE, 2019: 3492-3496

[44] Chen Lingjiao, Wang Hongyi, Charles Z, et al. DRACO: byzantine-resilient distributed training via redundant gradients[C]// Proc of the International Conference on Machine Learning (ICML), New York, NY: ACM, 2018: 903-912

[45] Gupta N, Vaidya N H. Randomized reactive redundancy for byzantine fault-tolerance in parallelized learning[J]. arXiv: Distributed, Parallel, and Cluster Computing, 2019

[46] Data D, Li Songze, Diggavi S. Data encoding methods for byzantine-resilient distributed optimization[C]//Proc of the IEEE International Symposium on Information Theory (ISIT), Piscataway, NJ: IEEE, 2019: 2719-2723

[47] Candes E J, Tao T. Decoding by linear programming [J]. IEEE Transactions on Information Theory, 2005, 51(12): 4203-4215

[48] Yu Qian, Raviv N, So J, et al. Lagrange coded computing: optimal design for resiliency, security and privacy[C]//Proc of the International Conference on Artificial Intelligence and Statistics, New York, NY: JMLR, 2019:1215–1225

[49] So J, Guler B, Avestimehr A S, et al. Codedprivateml: A fast and privacy-preserving framework for distributed machine learning[J]. arXiv: Learning, 2019

[50] Nodehi H A, Maddahali M A. Limited-sharing multi-party computation for massive matrix operations. [C]//Proc of the IEEE International Symposium on Information Theory (ISIT), Piscataway, NJ: IEEE, 2018: 1231-1235

[51] Ben-Or M, Goldwasser S, Wigderson A. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract) [C]//Proc of the 20th Annual ACM Symposium on Theory of Computing, York, NY: ACM, 1988:1-10

[52] Bitar R, Parag P, Rouayheb S E, et al. Minimizing latency for secure distributed computing[C]//Proc of the International Symposium on Information Theory (ISIT), Piscataway, NJ: IEEE, 2017: 2900-2904

[53] Bitar R, Parag P, Rouayheb S E, et al. Minimizing latency for secure coded computing using secret sharing via staircase codes[J]. IEEE Transactions on Communications, 2020, 68(8): 4609–4619

[54] Bitar R, Xing Yuxuan, Keshtkarjahromi Y, et al. PRAC: private and rateless adaptive coded computation at the edge[C]//Proc of SPIE Defense Commercial Sensing, Baltimore, MD, May 2019.

[55] Chang Weiting, Tandon R. On the capacity of secure distributed matrix multiplication[C]//Proc of the IEEE Global Communications Conference (GLOBECOM), Piscataway, NJ: IEEE, 2018: 1-6.

[56] Kakar J, Seyedhamed E, Aydin S. Rate-efficiency and straggler-robustness through partition in distributed two-sided secure matrix computation. arXiv: Information Theory, 2018

[57] Yang H, Lee J. Secure distributed computing with straggling servers using polynomial codes[J]. IEEE Transactions on Information Forensics and Security, 2019, 14(1): 141-150

[58] Yang Yaoqing, Grover P, Kar S, et al. Coded distributed computing for inverse problems[C]// Proc of the Annual Conference on Neural Information Processing, Cambridge, MA: MIT Press 2017: 709-719

[59] Seferoglu E V Y K H. Coded Privacy-Preserving Computation at Edge Networks[J]. arXiv preprint arXiv:2106.08290, 2021

[60] Yu Qian, Avestimehr A S. Harmonic coding: An optimal linear code for privacy-preserving gradient-type computation[C]//2019 IEEE International Symposium on Information Theory (ISIT). Piscataway, NJ: IEEE, 2019: 1102-1106.

[61] Li Songze, Supittayapornpong S, Maddah-Ali M A, et al. Coded terasort[C]//Proc of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Piscataway, NJ: IEEE, 2017: 389-398

[62] Schölkopf, B, Platt, J, Hofmann, T. Map-reduce for machine learning on multicore[C]//Proc of the IEEE International Conference on Neural Information Processing System, Piscataway, NJ: IEEE, 2007:281-288

[63] Isard M, Budiu M, Yu Y, et al. Dryad: distributed data-parallel programs from sequential building blocks[C]//Proc of the ACM SIGOPS/EuroSys European Conference on Computer Systems. New York: ACM, 2007: 59-72

[64] Abouzeid A, Bajda-Pawlikowski K, Abadi D J, et al. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads [J].VLDB Endowment, 2009, 2(1):922-933

[65] Ekanayake, Jaliya. DryadLINQ for scientific analyses[C]// Proc of the Fifth IEEE International Conference on e-Science, Piscataway, NJ: IEEE, 2009:329–336

[66] Horii S. Improved computation-communication trade-off for coded distributed computing using linear dependence of intermediate values. ArXiv Preprint ArXiv:2005.06118, 2020

[67] Attia M A, Tandon R. Information theoretic limits of data shuffling for distributed learning[C]// Proc of the IEEE Global Communications Conference (GLOBECOM), Piscataway, NJ: IEEE, 2016: 1-6

[68] Attia M A, Tandon R. On the worst-case communication overhead for distributed data shuffling[C]//Proc of the Allerton Conference on Communication, Control, and Computing (Allerton), Piscataway, NJ: IEEE, 2016: 961-968

[69] Sengupta A, Tandon R, Clancy T C. Fundamental limits of caching with secure delivery[J]. IEEE Transactions on Information Forensics and Security, 2014, 10(2): 355-370

[70] Pedarsani R, Maddah-Ali M A, Niesen U. Online coded caching[J]. IEEE/ACM Transactions on Networking, 2016, 24(2):836-845.

[71] Elmahdy A, Mohajer S. On the fundamental limits of coded data shuffling for distributed machine learning[J]. IEEE Transactions on Information Theory, 2020, 66(5): 3098-3131

[72] Gürbüzbalaban M, Ozdaglar A, Parrilo P A. Why random reshuffling beats stochastic gradient descent[J]. Mathematical Programming, 2019: 1-36

[73] Bottou L. Large-scale machine learning with stochastic gradient descent[C]//Proc of the COMPSTA, Berlin: Springer, 2010: 177-186

[74] Parker, Charles. Unexpected challenges in large scale machine learning. [C]//Proc of the International Workshop on Big Data, Streams and Heterogeneous Source Mining, New York: ACM, 2012:1–6

[75] Luby M. LT codes[C]// Proc of the IEEE Symposium on Foundations of Computer Science, Piscataway, NJ: IEEE, 2002: 271-280

[76] De Geijn R A, Watts J. SUMMA: Scalable universal matrix multiplication algorithm[J]. Concurrency and Computation: Practice and Experience, 1995, 9(4): 255-274

[77] Solomonik E, Demmel J. Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms[C]//Proc of the European Conference on Parallel Processing, Berlin: Springer, 2011: 90-109

[78] Costello D J, Hagenauer J, Imai H, et al. Applications of error-control coding[J]. IEEE Transactions on Information Theory, 1998, 44(6): 2531-2560

[79] So J, Guler B, Avestimehr A S, et al. Coded privateml: a fast and privacy-preserving framework for distributed machine learning[J]. arXiv preprint arXiv:1902.00641, 2019

[80] Li Songze, Yu Mingchao, Yang C S, et al. Polyshard: coded sharding achieves linearly scaling efficiency and security simultaneously[J]. 2018, arXiv:1809.10361

[81] MacKay D J C. Fountain codes[J]. IEE Proceedings-Communications, 2005, 152(6): 1062-1068

[82] Marshall J T. Coding of real-number sequences for error correction: a digital signal processing problem[J]. IEEE Journal on Selected Areas in Communications, 1984, 2(2): 381-392

[83] Halbawi W, Liu Zihan, Hassibi B, et al. Balanced Reed-solomon codes[C]//Proc of the International Symposium on Information Theory (ISIT), Piscataway, NJ: IEEE, 2016: 935-939
[84] Kadhe S, Koyluoglu O O, Ramchandran K. Communication-efficient gradient coding for straggler mitigation in distributed learning[C]// roc of 0 IEEE International Symposium on Information Theory (ISIT). Piscataway, NJ: IEEE, 2020: 2634-2639
[85] Bitar R, Wootters M, Rouayheb S E, et al. Stochastic gradient coding for straggler mitigation in distributed learning[J]. IEEE Journal on Selected Areas in Information Theory, 2020,1(1):277–291
[86] Maity R K, Rawa A S, Mazumdar A, et al. Robust gradient descent via moment encoding and LDPC codes[C]//Proc of the International Symposium on Information Theory (ISIT), Piscataway, NJ: IEEE, 2019: 2734-2738
[87] Cao Hankun, Yan Qifa, Tang Xiaohu, et al. Adaptive gradient coding[J]. arXiv preprint arXiv:2006.04845, 2020.
[88] Dutra S, Bai Z, Jeong H, et al. A unified coded deep neural network training strategy based on generalized polydot codes[J]. arXiv: Information Theory, 2017
[89] Hadidi R, Cao J, Ryoo M S, et al. Robustly executing dnns in IoT systems using coded distributed computing[C]//Proc of the IEEE/ Design Automation Conference, Piscataway, NJ: IEEE, 2019:1-2
[90] Krizhevsky A, Sutskever I, Hinton G E, et al. Imagenet classification with deep convolutional neural networks[C]//Proc of the Annual Conference on Neural Information Processing Systems, Cambridge, MA: MIT Press, 2012: 1097-1105
[91] Hasircioglu B, Gomezvilardebo J, Gunduz D, et al. Bivariate polynomial coding for exploiting stragglers in heterogeneous coded computing systems[J]. arXiv: Information Theory, 2020
[92] Lim W Y, Luong N C, Hoang D T, et al. Federated learning in mobile edge networks: a comprehensive survey[J]. arXiv:1805.10381,2019
[93] Abdelzaher T, Ayanian N, Basar T, et al. Will distributed computing revolutionize peace? the emergence of battlefield IoT [C]// Proc of the IEEE International Conference on Distributed Computing Systems (ICDCS), Piscataway, NJ: IEEE,2018: 1129-1138
[94] Konecný J. Stochastic, distributed and federated optimization for machine learning[J]. arXiv: Learning, 2017
[95] Lamport L, Shostak R E, Pease M, et al. The byzantine generals problem[J]. ACM Transactions on Programming Languages and Systems, 1982, 4(3): 382-401
[96] Chen Yudong, Su Lili, Xu Jiaming. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent[J]. Measurement and Analysis of Computing Systems, 2017, 1(2): 1-25
[97] Rajput S, Wang Hongyi, Charles Z, et al. DETOX: a redundancy-based framework for faster and more robust gradient aggregation[C]//Proc of the Annual Conference on Neural Information Processing Systems, Cambridge, MA: MIT Press, 2019: 10320-10330
[98] Shamir A. How to share a secret[J]. Communications of the ACM, 1979, 22(11): 612-613
[99] Kim M, Yang H, Lee J, et al. Private coded computation for machine learning[J]. arXiv: Information Theory, 2018
[100] Nodehi H A, Najarkolaei S R, Maddahali M A, et al. Entangled Polynomial Coding in Limited-Sharing Multi-Party Computation[C]// Proc of the Information Theory Workshop, Piscataway, NJ: IEEE, 2018: 1-5
[101] Atallah M J, Frikken K B. Securely outsourcing linear algebra computations[C]//Proc of the ACM Symposium on Information, Computer and Communications Security, New York: ACM, 2010: 48-59
[102] Jøsang A, Ismail R, Boyd C. A survey of trust and reputation systems for online service provision [J]. Decision Support Systems, 2007, 43(2): 618-644

Zheng Tengfei, born in 1993. PhD candidate. His main research interests include information security and distributed computing.

郑腾飞，1993 年生．博士生．主要研究方向为信息安全和分布式计算.

Zhou Tongqing, born in 1991. Ph. D. His main research interests include distributed machine learning and privacy protection.

周桐庆，1991 年生．博士后．主要研究方向为分布式机器学习和隐私保护.

Cai Zhiping, born in 1975, Ph. D. Professor. Senior Member of China Computer Federation. His main research interests include network security and distributed computing.

蔡志平，1975 年生．博士，教授，CCF 会员．主要研究方向包括网络安全和分布式计算等.

Wu Hongjia, born in 1994, Ph. D. candidate. Her research interests include edge computing and federal learning.

吴虹佳， 1994 年生．博士生．主要研究方向为边缘计算和联邦学习．

校对负责人：郑腾飞

手机：15930718896 Email: zhengtengfei@nudt.edu.cn