華中科技大學

# 本科生毕业设计（论文）参考文献译文

译文出处： *Li S, Maddah-Ali M A, Avestimehr A S. A unified coding framework for distributed computing with straggling servers[C]//2016 IEEE Globecom Workshops (GC Wkshps). IEEE, 2016: 1-6.*


| | |
|---|---|
| 院　　系 | 计算机科学与技术 |
| 专业班级 | 本硕博 2001 班 |
| 姓　　名 | 李茗畦 |
| 学　　号 | U202015630 |
| 指导教师 | 胡燏翀 |


2024 年 3 月 6 日

# 译文要求

一、 译文内容须与课题（或专业内容）联系，并需在封面注明详细出处。

二、 出处格式为

图书：作者.书名.版本（第×版）.译者.出版地：出版者，出版年.起页～止页

期刊：作者.文章名称.期刊名称，年号，卷号（期号）：起页～止页

三、 译文不少于 5000 汉字（或 2 万印刷符）。

四、 翻译内容用五号宋体字编辑，采用 A4 号纸双面打印，封面与封底采用浅蓝色封面纸（卡纸）打印。要求内容明确，语句通顺。

五、 译文及其参考文献原文一起装订，顺序依次为封面、译文、文献。

六、 翻译应在第七学期完成。

# 译文评阅

**导师评语**

应根据学校“译文要求”，对学生译文翻译的准确性、翻译数量以及译文的文字表述情况等做具体的评价后，再评分。

评分：_____（百分制）　　指导教师（签名）：_____

2023 年　2 月 21 日

**题目：** 面向具有滞后节点的分布式计算编码框架

**摘要：**我们提出了一个统一的编码框架，用于具有滞后服务节点的分布式计算，做法是在某些线性计算任务中引入"计算延迟"和"通信负载"之间的权衡。我们展示了在 MapReduce 中进行重复计算以创建编码多播机会来减少通信负载的编码方案，以及进行冗余的中间计算来避免滞后服务节点的编码方案。这两个情况可以被看作是所提出的框架的特殊实例，分别代表这个权衡的两个极端：要么最小化通信负载，要么最小化计算延迟。此外，所提出的编码框架实现的计算延迟-通信负载的权衡允许在系统中的任何点上地执行分布式计算任务。我们还证明了计算延迟-通信负载权衡的信息论下界。

## 1 概述

最近，有两个新提出的方案利用编码来加速分布式计算应用。具体而言，[1]–[3]的作者提出了在分布式计算系统中进行重复的计算任务，从而创造了编码多播机会，显著减少了对中间结果进行洗牌需要的时间。另一方面，[4]的作者提出了在线性计算任务中应用最大距离可分码（MDS），以减轻滞后服务节点的影响并缩短分布式计算中计算阶段花费的时间。

在本文中，我们提出了一个用于具有滞后服务节点的分布式计算的统一编码框架，通过在线性计算任务中引入"计算延迟"和"通信负载"之间的权衡。[1]和[4]的编码方案可以被看作是所提出的编码框架的特殊实例，分别考虑了这个权衡的两个极端：要么最小化通信负载，要么最小化计算延迟。此外，所提出的编码框架在分布式计算中提供了计算延迟和通信负载之间的权衡，并允许在该权衡的任何节点上进行操作。

具体来讲，我们关注一个分布式矩阵乘法问题，对于矩阵 A 和 N 个输入向量$x_1, \dots, x_N$，需要计算 N 个输出向量$y_1 = Ax_1, \dots, y_N = Ax_N$。由于单个服务节点的本地内存太小，无法执行完整的计算，因此我们使用 K 个分布式计算服务节点进行计算。每个节点的本地内存大小足够存储矩阵 A 的比例为 μ 的部分，并且它只能基于其本地内存中存储的内容进行计算。矩阵乘法是解决数据分析和机器学习问题（例如回归和分类）的基本组件之一。许多大数据分析的应用需要在大规模数据集上进行大量的计算和存储，这些计算和存储通常由计算服务集群协作

提供，通常使用高效的分布式计算框架，如 Hadoop MapReduce [5]和 Spark [6]。因此，优化分布式矩阵乘法的性能对于提高分布式计算应用的性能至关重要。

矩阵乘法的分布式实现包括三个阶段：Map 阶段、Shuffle 阶段和 Reduce 阶段。在 Map 阶段，每个节点将输入向量与本地存储的部分 A 矩阵进行乘法运算。当一组节点完成其本地计算，使得它们的中间结果足以恢复输出向量时，我们停止 Map 阶段，并开始在节点之间对中间结果进行 Shuffle，通过特定的 Reduce 函数计算最终的输出向量。

在上述的三阶段实现中，[1]的编码方法旨在最小化中间结果的 Shuffle 阶段的通信负载。它通过利用冗余的 Map 计算来在 Shuffle 阶段利用编码多播以最小化通信负载。我们将这种编码方法称为“最小带宽编码”。[4]的另一种编码方法旨在通过使用 MDS 码对 Map 计算任务进行编码，以最小化 Map 阶段的计算延迟，从而使 Map 阶段的运行时间不受到一定数量的滞后节点的影响。这种编码方案被称为“最小延迟编码”，可以显著降低 Map 阶段的计算延迟。

在本文中，我们对分布式矩阵乘法的 Map 阶段的计算延迟（用$D$表示）和 Shuffle 阶段的通信负载（用$L$表示）之间的权衡进行了形式化阐述，称为计算延迟-负载权衡。如图 1 所示，上述两种编码方案分别对应于最小化$L$和最小化$D$的两个极端点。此外，我们提出了一个统一的编码方案，有机地整合了这两种编码技术，并允许在引入的权衡中进行系统级的操作。

对于给定的计算延迟，我们还证明了完成分布式矩阵乘法所需的最小通信负载的信息论下界。在权衡的两个端点上，所提出的方案在一个常数因子内实现了最小的通信负载。

最后，我们注意到在分布式计算中，Map 阶段的计算负载和 Shuffle 阶段的通信负载之间存在另一个权衡，这在[1]中被引入和描述。在本文中，我们固定了每个节点的计算负载（由存储空间的大小确定），并重点描述了计算延迟（由完成 Map 计算的节点数量确定）和通信负载之间的权衡。

## 2 问题描述

我们考虑下面的矩阵乘法问题，给定矩阵$A$，大小为$m \times n$,$N$个输入向量$x_1, \dots, x_N$，长度均为$n$，需要计算$N$个输出向量$y_1 = Ax_1, \dots, y_N = Ax_N$。

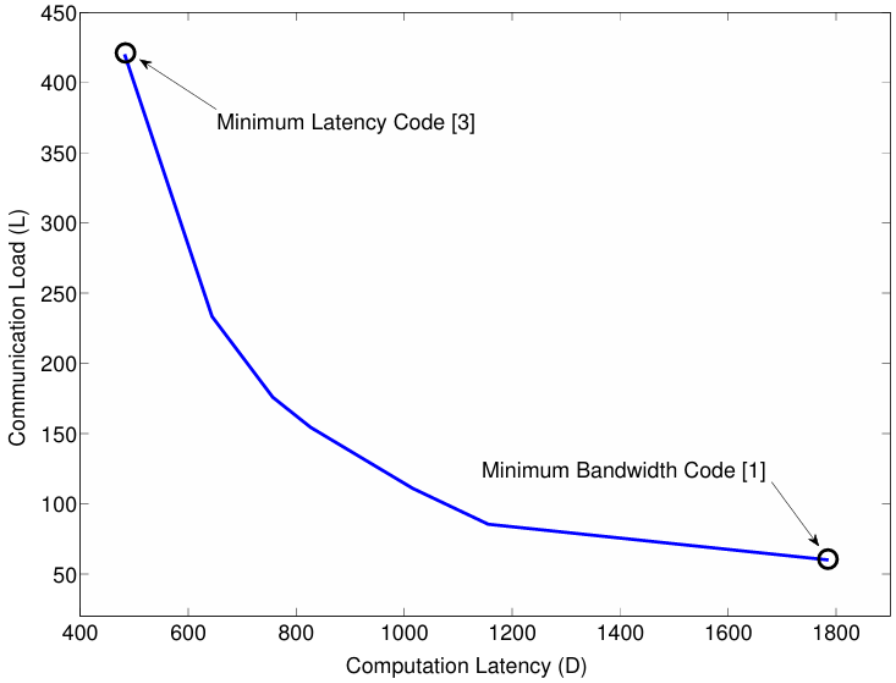



图 1 对于一个输出向量为 N=840，服务节点数 K=14 的矩阵乘法任务的延迟和负载的权衡

我们在 K 个分布式的节点上执行计算。每个节点具有大小为$\mu mnT$比特的本地空间，$T$为矩阵和向量中一个元素的比特位数，也就是一个节点可以保存矩阵$A$的比例为$\mu$的部分。

我们允许在每个节点中存储 A 的行的线性组合。对于节点$k \in 1, \dots, K$和编码矩阵$E_k$(大小为$\mu m \times m$)，该节点可以保存矩阵$U_k = E_k A$。

我们假设输入向量$x_1, \dots, x_N$为所有服务节点共知。整个计算过程分为三个阶段：Map 阶段，Shuffle 阶段和 Reduce 阶段。

A.Map 阶段：

Map 阶段用于计算本地保存的部分矩阵的中间结果，中间结果将用于构造完整的输出向量。准确的来说，对于$j = 1, \dots, N$,以及节点$k(k = 1, \dots, K)$，需要计算中间向量$z_{j,k} = U_k x_j = E_k A x_j = E_k y_j$。我们将节点$k$计算向量$z_{1,k}, \dots, z_{N,k}$的延迟分别为$S_1, \dots, S_k$。这些延迟值为随机变量，第 q 个小的变量表示为$S_{(q)}$。我们主要关注$S_k$的这样一种分布：$E\{S_{(q)}\} = \mu N g(K, q)$，其中$g(K, q)$为一些函数。

当一部分服务节点完成它们的 Map 计算时 Map 阶段将终止，这部分节点表

示为$Q \subseteq \{1, \dots, K\}$。集合$Q$需要保证可以通过利用$Q$中的节点计算的中间向量（即$\{zj,k : j = 1,\dots,N, k \in Q\}$）来重构输出向量$y_1, \dots, y_N$。我们允许在$Q$中进行计算，如果设计得当，这可以用于减少中间结果的通信负载，以便$Q$中的节点哎后续阶段中恢复输出向量。

我们将计算延迟表示为$D$，定义为 Map 阶段花费的平均时间。在 Map 阶段之后，计算输出向量$y_1, \dots, y_N$的任务仅在$Q$中的服务器上进行。输出向量的最终计算在$Q$中的服务器上均匀分布。我们用$W_k$表示分配给节点$k$的输出向量的索引集合，$\{W_k: k \in Q\}$，$W_k$保证没有重复的索引且所有索引平均分布。

B.Shuffle 阶段

Shuffle 阶段的目标是交换在 Map 阶段计算的中间结果，以帮助每个节点恢复其负责的输出向量。为此，$Q$中的每个节点$k$通过编码函数$\phi_k$从本地计算的中间结果$z_{1,k}, \dots, z_{N,k}$生成消息$X_k$，即$X_k = \phi_k(z_{1,k}, \dots, z_{N,k})$。收到所有的消息$\{X_k: k \in Q\}$之后，每个节点 k 可以恢复$W_k$中的所有输出向量。我们假设节点通过共享总线连接。在生成$X_k$之后，服务器$k$向$Q$中的所有其他节点进行多播，将$X_k$发送给它们。我们将通信负载表示为$L$，定义为所有消息$\{Xk : k \in Q\}$中的平均比特数。

C.Reduce 阶段

在 Reduce 阶段，输出向量将被分布式地构建。具体来讲，用户$k$，$k \in Q$，使用本地计算的向量$z_{1,k}, \dots, z_{N,k}$和接收到的多播消息$\{Xk : k \in Q\}$，通过解码函数 $\psi k$ 来得到 $W_k$ 中的索引表示的输出向量，即 $\{y_j: j \in W_k\} = \psi_k(z_{1,k}, \dots, z_{N,k}, \{X_k: k \in Q\})$。

对于这样一个分布式计算系统，如果存在一组编码矩阵$\{E_k\}_{k=1}^{K}$，那么我们称称延迟-负载对$(D, L)$是可实现的。也就是说，具有计算延迟$D$的 Map 计算和具有通信负载$L$的 Shuffle 方案，可以成功得到所有输出向量。

为了阐明这个表述，我们使用以下简单的例子来说明在第 1 节讨论的两种编码方法实现的延迟-负载对。考虑一个由$a_1, \dots, a_{12}$12 个行向量组成的矩阵 A。我们有$N = 4$个输入向量$x_1, \dots x_4$，在$K = 4$个服务节点上进行计算，每个服务器的存储大小为$\mu = 1/2$。我们假设 Map 阶段的延迟$S_k$，$k = 1,\dots, 4$，具有一个移位指

数分布函数$F_{S_k}(t) = 1 - e^{-\left(\frac{t}{\mu N}-1\right)}$。

对于最小带宽方案，最小带宽编码重复地将 A 的每一行存储在$\mu K$个服务器中，以便在 Shuffle 阶段，$\mu K$个中间值可以通过编码的多播消息传递，这获得了一个编码增益为$\mu K$。如图 2 所示，最小带宽编码在 4 个服务节点上重复地将 A 的每一行与所有输入向量$x_1, \ldots, x_4$进行乘法运算，重复次数为$\mu K = 2$ 次，例如$a_1$在节点 1 和节点 2 上进行乘法运算。Map 阶段将进行直致所有节点完成 Map 计算，实现的计算延迟为$D(4) = \frac{37}{6}$。对于$k = 1, 2, 3, 4$，服务器$k$将进行 reduce 操作得到输出向量$y_k$。在 Shuffle 阶段，如图 2 所示，每个服务器进行 3 次按位异或多播，每次多播同时作用于两个服务器。因此，最小带宽编码实现了通信负载$L = 3 \times 4/12 = 1$。最小带宽编码可以被视为特定类型的网络编码，更确切地说是索引编码，其中的关键思想是在服务器上设计“辅助信息”，以便在 Shuffle 阶段实现多播机会从而最小化通信负载。

最小延迟编码使用 MDS 码生成一些冗余的 Map 计算。这种类型的编码利用了多余的服务节点，因此只要足够的编码计算在网络中完成，就可以终止 Map 阶段，而无需等待剩余的慢节点。我们在图 3 中说明了这种编码技术。对于这个例子，最小延迟编码首先让每个服务节点$k$，$k = 1, \ldots, 4$，独立地随机生成矩阵 A 的行的 6 个随机线性组合，表示为$c_{6(k-1)+1}, \ldots, c_{6(k-1)+6}$，实现了一个(24, 12)的 MDS 码。因此，对于任意大小为$|\mathcal{D}| = 12$的子集$\mathcal{D} \subseteq \{1, \ldots, 24\}$，使用中间结果$\{c_i x_j : i \in D\}$可以恢复输出向量$y_j$。Map 阶段在最快的 2 个服务节点完成计算后终止（例如，服务节点 1 和 3），实现的计算延迟为$D(2) = 2 \times (1 + 1/3 + 1/4) = \frac{19}{6}$。然后，服务节点 1 继续对$y_1, y_2$进行 reduce 操作，服务节点 3 继续对$y_3, y_4$进行 reduce 操作。如图 3 所示，服务节点 1 和 3 分别单播由其计算的，其他服务器所需的中间值，实现的通信负载为$L = 6 \times 4/12 = 2$。与最小延迟编码相比，最小带宽编码在映射阶段使用的时间大约是两倍，并且实现了一半的通信负载。它们两种编码方式代表了在下一节中描述的延迟和负载之间的权衡的两个极端点。

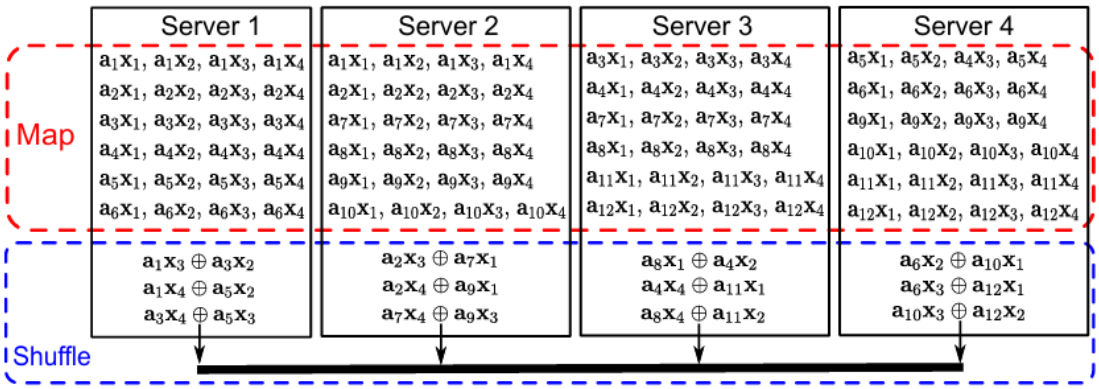



图 2 最小带宽编码方案

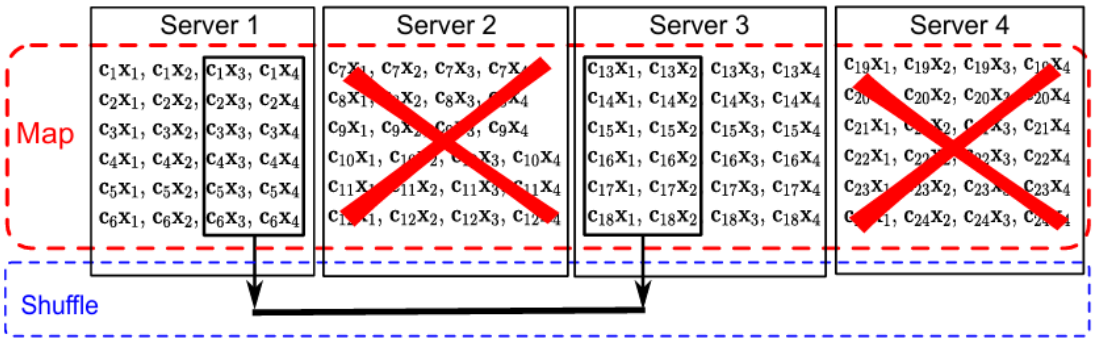



图 3 最小延迟编码方案

## 3 主要结果

该论文的主要结果是：（1）通过开发一个统一的编码框架，对一组可实现的延迟-负载对进行了描述。（2）提出了延迟-负载区域的外部界限，并在以下两个定理中进行了说明。

对于一个分布式矩阵乘法问题，使用$K$个服务器计算$N$个输出向量，每个服务器的存储大小为$\mu \geq \frac{1}{K}$,延迟-负载区域包含以下点的下凸包：$\{(D(q), L(q)): q = \left\lceil \frac{1}{\mu} \right\rceil, \ldots, K\}$。我们在图 4 中通过数值评估了所提出的编码框架实现的延迟-负载对。在计算$N = 180$ 个输出向量时，使用$K = 18$ 个服务器，每个服务器的存储大小为$\mu = 1/3$。实现的权衡大致呈现出延迟和负载之间的反比例关系。例如，将延迟从 120 增加到 240，通信负载从 43 降低到 23，降低了 1.87 倍。

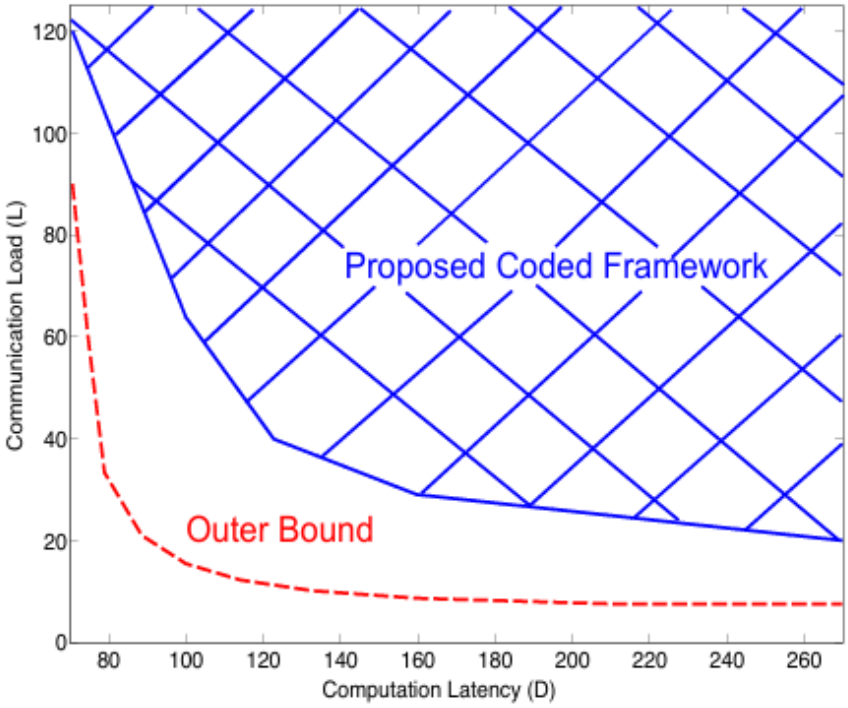



图 4 编码方案和外部约束的延迟，负载的比较

实现$D(q)$和$L(q)$的关键思想是设计 MDS 码和重复 Map 计算的串联，以充分利用最小延迟编码和最小带宽编码的优势。具体来说，我们首先生成矩阵 A 的$\frac{K}{q}m$个 MDS 编码行，然后按将这些 MDS 编码行照特定模式在$K$个服务器上对每行存储每行$\lfloor \mu q \rfloor$次。因此，任意$q$个服务节点的子集都会有足够多的中间结果来得到输出向量，当最快的$q$个服务节点完成 Map 计算时，我们结束 Map 阶段。

我们还利用编码的多播技术来减少 Shuffle 阶段的通信负载。$B_j, j \leq \lfloor \mu q \rfloor$表示在$j$个服务器上重复存储以及计算的矩阵 A 的编码行数。通过同时由$Bj$个中间值，向 j 个节点进行编码多播，可以以通信负载$\frac{B_j}{j}$，实现编码增益为$j$。

## 4 编码框架

在本节中，我们提出和分析一个通用的编码框架，该框架实现了第 3 节中的延迟-负载对。我们首先通过以下示例演示所提出方案的关键思想，然后给出方案的一般描述。

A.示例：$m = 20$，$N = 12$，$K = 6$，$\mu = 1/2$

需要将一个由 20 个行向量组成的矩阵 A 与 12 个输入向量$x_1, \ldots, x_{12}$相乘，计算 12 个输出向量$y_1 = Ax_1, \ldots, y_{12} = Ax_{12}$，使用$K = 6$个服务节点，每个节点的

存储大小为$\mu = 1/2$。

我们假设我们可以等待$q = 4$个节点在 Map 阶段完成计算，接下来描述所提出的编码设计和 shuffle 方案。如图 5 所示，我们首先独立地生成 30 个随机线性组合$c_1, \ldots, c_{30}$，这些组合由 A 的 20 行构成。然后，我们将这些组合$c_1, \ldots, c_{30}$划分为大小为 2 的 15 个批次，并将每个批次的编码行存储在一对唯一的节点上。我们可以假设节点 1、2、3 和 4 是首先完成 Map 计算的前 4 个节点。然后，我们分配 Reduce 任务，使得服务器$k$对输出向量$y_{3(k-1)+1}, y_{3(k-1)+2}, y_{3(k-1)+3}$进行 Reduce 操作，其中$k \in \{1,\ldots,4\}$。由于节点 1 已计算了$\{c_1x_j, \ldots, c_{10}x_j : j = 1, \ldots, 12\}$，为了对 $y_1 = Ax_1$进行 reduce 操作，它需要在 Shuffle 阶段从节点 2、3 和 4 中获取任意 10 个中间值$c_ix_1$，其中$i \in \{11,\ldots, 30\}$。类似的数据需求适用于所有 4 个节点以及其负责的输出向量。

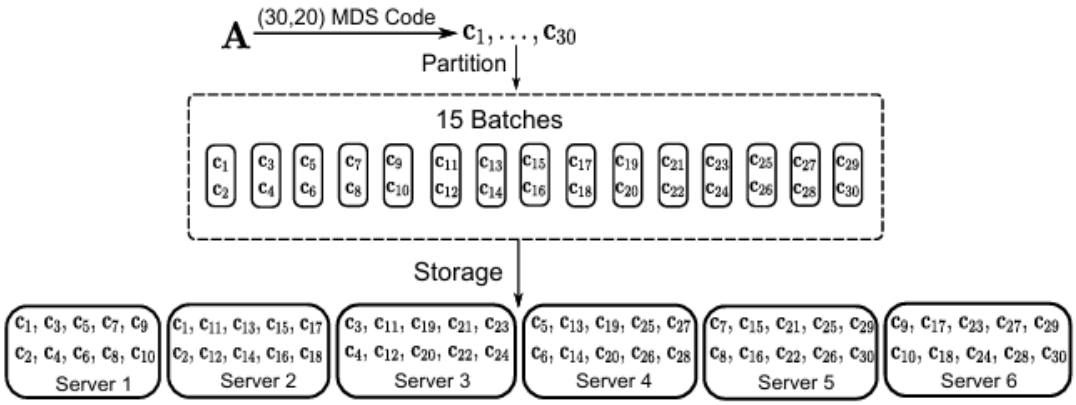



图 5 Map 阶段的存储设计

接下来说明 shuffle 的编码方案。我们首先将这 4 个节点分成 4 个大小为 3 的子集，并在每个子集内进行编码 shuffle。我们在图 6 中展示了节点 1、2 和 3 的编码 shuffle 方案。每个节点将本地计算的中间值进行 3 个位异或的多播，发送给其他两个节点。在接收到 2 个多播消息后，每个节点可以恢复所需的 6 个中间值。例如，节点 1 通过消除$c_2x_7, c_2x_8, c_2x_9$分别恢复$c_{11}x_1, c_{11}x_2, c_{11}x_3$。类似地，我们对另外 3 个大小为 3 的子集执行上述编码 shuffle。每个节点恢复所需的 18 个中间结果。每个节点需要总共 $3 \times (20 - 10) = 30$ 个中间值来计算它分配的 3 个输出向量，在解码所有多播消息后，它需要另外 $30 - 18 = 12$ 个中间值。我们通过简单地使节点单播足够的（即 $12 \times 4 = 48$）中间值来满足剩余的数据需求。总

体而言，通信了 $9\times4+48=84$（可能经过编码）中间值，实现了通信负载$L=4.2$。

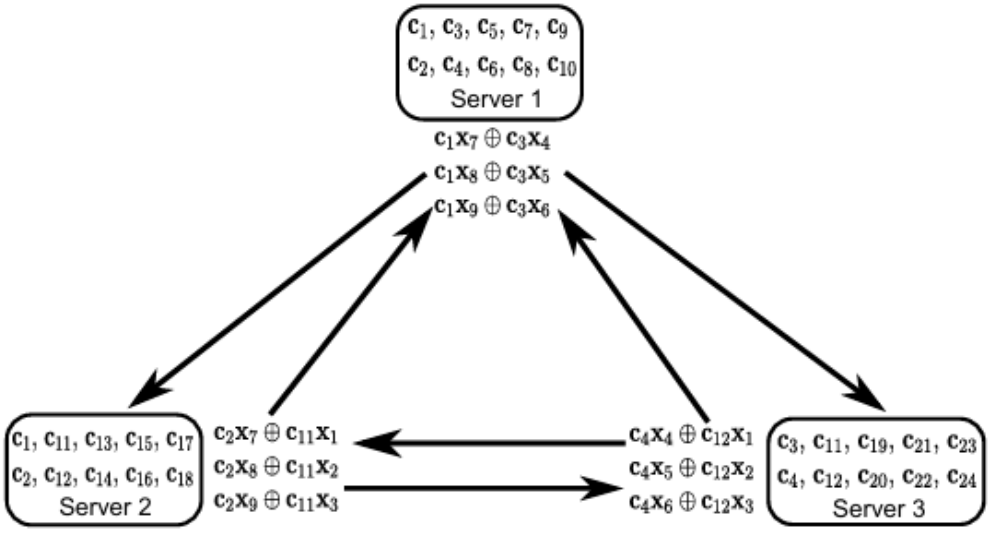



图 6 编码 shuffle 方案

B.通用编码方案

接下来我们描述存储设计、Map 阶段计算和 shuffle 方案，以第 3 节中的延迟-负载对（$D(q), L(q)$）。我们使用一个$(\frac{K}{q}m, m)$的 MDS 码将矩阵 A 的$m$行编码为$\frac{K}{q}m$个编码行$c_1, \ldots, c_{\frac{K}{q}m}$。然后，如图 7 所示，我们将$\frac{K}{q}m$个编码行均匀地分成不相交的批次，每个批次包含$mqK(K\mu q)$个编码行的子集。每个批次由$B_T$表示，并用唯一的子集 T⊂{1,...,$K$}（大小为｜T｜＝$\mu q$）标记。节点$k$，$k\in\{1,\ldots,K\}$，如果$k\in\mathcal{T}$，则将$B_{\mathrm{T}}$中的编码行存储为 U$k$的行。在上面的示例中，$q=4$，$\frac{K}{q}m=6\times20=120$ 个编码行被分成了 15 个批次，每个批次包含 30/15 = 2 个编码行。每个节点属于 5 个大小为 2 的子集，因此存储了 $5\times2=10$ 个编码行的 A。

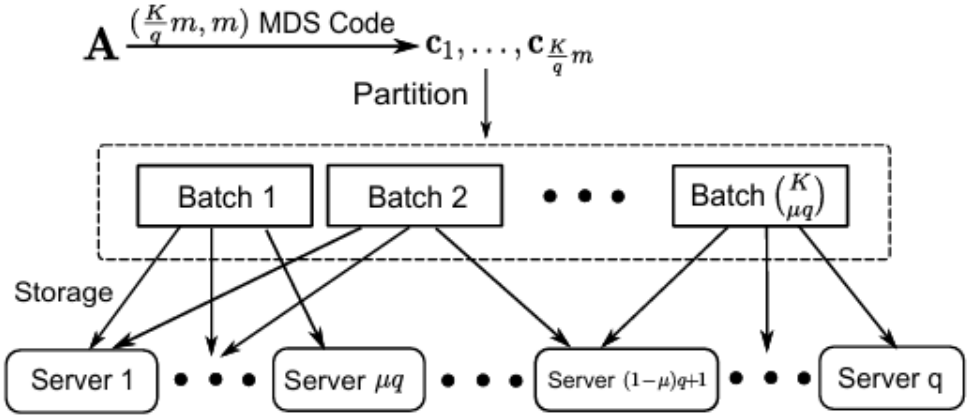



图 7 通用的 Map 阶段的存储设计

下面描述 Map 阶段的执行。每个节点计算本地存储的每个编码行与每个输入向量之间的内积，即节点$k$计算$c_i x_j$（对于所有$j=1,\ldots,N$和所有$i \in \{B_T: k \in T\}$）。我们在最快的$q$个节点完成 Map 计算之前等待，之后中止 Map 阶段，从而实现计算延迟$D(q)$。我们将这些节点的索引集合表示为$\mathcal{Q}$。之后的计算在$\mathcal{Q}$中的$q$个节点进行，每个服务器负责对$N/q$个输出向量$y_1 = Ax_1, \ldots, y_N = Ax_N$执行 reduce 操作。

为了使得 Reduce 阶段能够成功进行，需要存在一种可行的 shuffle 方案，使得每个$q$个服务器的子集（因为我们无法预测哪$q$个服务器会先完成）共同存储至少$m$个不同的编码行$c_i$，其中$i \in \{1,\ldots,\frac{K}{q}m\}$。接下来，我们解释一下我们提出的存储设计如何满足这个要求。首先，$\mathcal{Q}$中的$q$个节点提供了与$\mu qm$行等效的存储大小。然后，由于每个编码行由所有$K$个节点中的$\mu q$个节点存储，它最多可以由$\mu q$个$\mathcal{Q}$中的节点存储，因此$\mathcal{Q}$中的节点共同存储至少$m$个不同的编码行，即$\mu qm/\mu q = m$个不同的编码行。

我们将节点 k 需要的，且仅由集合 S 中的节点独占的中间值表示为$V_S^k$。在上面的例子中，$V_{2,3}^1 = \{c_{11}x_j, c_{12}x_j : j = 1,2,3\}$。在 Shuffle 阶段，$\mathcal{Q}$中的节点创建并多播编码数据包，这些数据包同时对多个其他节点有用，直到$\mathcal{Q}$中的每个节点至少恢复了每个输出向量所需的$m$个中间值。所提出的重排方案是贪婪的，因为$\mathcal{Q}$中的每个节点始终尝试同时为最多的节点广播编码数据包。

## 5 主要参考文献

[1] Li, Songze, et al. "A fundamental tradeoff between computation and communication in distributed computing." IEEE Transactions on Information Theory 64.1 (2017): 109-128.

[2] Li, Songze, et al. "A fundamental tradeoff between computation and communication in distributed computing." IEEE Transactions on Information Theory 64.1 (2017): 109-128.

[3] Li S, Maddah-Ali M A, Avestimehr A S. Coded mapreduce[C]//2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton). IEEE, 2015: 964-971.Lee, Kangwook, et al. "Speeding up distributed machine learning

using codes." IEEE Transactions on Information Theory 64.3 (2017): 1514-1529.

[4] Lee K, Lam M, Pedarsani R, et al. Speeding up distributed machine learning using codes[J]. IEEE Transactions on Information Theory, 2017, 64(3): 1514-1529.Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: Simplified data processing on large clusters." (2004).

[5] Zaharia, Matei, et al. "Spark: Cluster computing with working sets." 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10). 2010.

[6] Li S, Yu Q, Maddah-Ali M A, et al. A scalable framework for wireless distributed computing[J]. IEEE/ACM Transactions on Networking, 2017, 25(5): 2643-2654.

[7] Li S, Yu Q, Maddah-Ali M A, et al. Edge-facilitated wireless distributed computing[C]//2016 IEEE Global Communications Conference (GLOBECOM). IEEE, 2016: 1-7.

[8] Arnold B C, Balakrishnan N, Nagaraja H N. A first course in order statistics[M]. Society for Industrial and Applied Mathematics, 2008.

[9] Ahlswede R, Cai N, Li S Y R, et al. Network information flow[J]. IEEE Transactions on information theory, 2000, 46(4): 1204-1216.

[10] Birk Y, Kol T. Coding on demand by an informed source (ISCOD) for efficient broadcast of different supplemental data to caching clients[J]. IEEE Transactions on Information Theory, 2006, 52(6): 2825-2830.

[11] Bar-Yossef Z, Birk Y, Jayram T S, et al. Index coding with side information[J]. IEEE Transactions on Information Theory, 2011, 57(3): 1479-1494.

[12] Maddah-Ali M A, Niesen U. Fundamental limits of caching[J]. IEEE Transactions on information theory, 2014, 60(5): 2856-2867.

[13] Maddah-Ali M A, Niesen U. Decentralized coded caching attains order-optimal memory-rate tradeoff[J]. IEEE/ACM Transactions On Networking, 2014, 23(4): 1029-1040.

[14] Ji M, Caire G, Molisch A F. Fundamental limits of caching in wireless D2D networks[J]. IEEE Transactions on Information Theory, 2015, 62(2): 849-869.

# A Unified Coding Framework for Distributed Computing with Straggling Servers

Songze Li*, Mohammad Ali Maddah-Ali †, and A. Salman Avestimehr*
* Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA
† Nokia Bell Labs, Holmdel, NJ, USA

arXiv:1609.01690v1 [cs.IT] 6 Sep 2016

***Abstract*—We propose a *unified* coded framework for distributed computing with straggling servers, by introducing a tradeoff between "latency of computation" and "load of communication" for some linear computation tasks. We show that the coded scheme of [1]–[3] that repeats the intermediate computations to create coded multicasting opportunities to reduce communication load, and the coded scheme of [4], [5] that generates redundant intermediate computations to combat against straggling servers can be viewed as special instances of the proposed framework, by considering two extremes of this tradeoff: minimizing either the load of communication or the latency of computation individually. Furthermore, the latency-load tradeoff achieved by the proposed coded framework allows to systematically operate at any point on that tradeoff to perform distributed computing tasks. We also prove an information-theoretic lower bound on the latency-load tradeoff, which is shown to be within a constant multiplicative gap from the achieved tradeoff at the two end points.**

## I. Introduction

Recently, there have been two novel ideas proposed to exploit coding in order to speed up distributed computing applications. Specifically, a repetitive structure of computation tasks across distributed computing servers was proposed in [1]–[3], enabling coded multicast opportunities that significantly reduce the time to shuffle intermediate results. On the other hand, applying Maximum Distance Separable (MDS) codes to some linear computation tasks (e.g., matrix multiplication) was proposed in [4], [5], in order to alleviate the effects of straggling servers and shorten the computation phase of distributed computing.

In this paper, we propose a *unified* coded framework for distributed computing with straggling servers, by introducing a tradeoff between "latency of computation" and "load of communication" for linear computation tasks. We show that the coding schemes of [1] and [4] can then be viewed as special instances of the proposed coding framework by considering two extremes of this tradeoff: minimizing either the load of communication or the latency of computation individually. Furthermore, the proposed coding framework provides a natural tradeoff between computation latency and communication load in distributed computing, and allows to systematically operate at any point on that tradeoff.

More specifically, we focus on a distributed matrix multiplication problem in which for a matrix $\mathbf{A}$ and $N$ input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$, we want to compute $N$ output vectors $\mathbf{y}_1 = \mathbf{A}\mathbf{x}_1, \ldots, \mathbf{y}_N = \mathbf{A}\mathbf{x}_N$. The computation cannot be performed on a single server node since its local memory is too small to hold the entire matrix $\mathbf{A}$. Instead, we carry out this computation using $K$ distributed computing servers collaboratively. Each server has a local memory, with the size enough to store up to equivalent of $\mu$ fraction of the entries of the matrix $\mathbf{A}$, and it can only perform computations based on the contents stored in its local memory. Matrix multiplication is one of the building blocks to solve data analytics and machine learning problems (e.g., regression and classification). Many such applications of big data analytics require massive computation and storage power over large-scale datasets, which are nowadays provided collaboratively by clusters of computing servers, using efficient distributed computing frameworks such as Hadoop MapReduce [6] and Spark [7]. Therefore, optimizing the performance of distributed matrix multiplication is of vital importance to improve the performance of the distributed computing applications.

A distributed implementation of matrix multiplication proceeds in three phases: Map, Shuffle and Reduce. In the Map phase, every server multiplies the input vectors with the locally stored matrix that partially represents the target matrix $\mathbf{A}$. When a subset of servers finish their local computations such that their Map results are sufficient to recover the output vectors, we halt the Map computation and start to Shuffle the Map results across the servers in which the final output vectors are calculated by specific Reduce functions.

Within the above three-phase implementation, the coding approach of [1] targets at minimizing the shuffling load of intermediate Map results. It introduces a particular repetitive structure of Map computations across the servers, and utilizes this redundancy to enable a specific type of network coding in the Shuffle phase (named coded multicasting) to minimize the communication load. We term this coding approach as "Minimum Bandwidth Code". In [8], [9], the Minimum Bandwidth Code was employed in a fully decentralized wireless distributed computing framework, achieving a scalable architecture with a constant load of communication. The other coding approach of [4], however, aims at minimizing the latency of Map computations by encoding the Map tasks using MDS codes, so that the run-time of the Map phase is not affected by up to a certain number of straggling servers. This coding scheme, which we term as "Minimum Latency Code", results in a significant reduction of Map computation latency.

In this paper, we formalize a *tradeoff* between the computation latency in the Map phase (denoted by $D$) and the communication (shuffling) load in the Shuffle phase (denoted

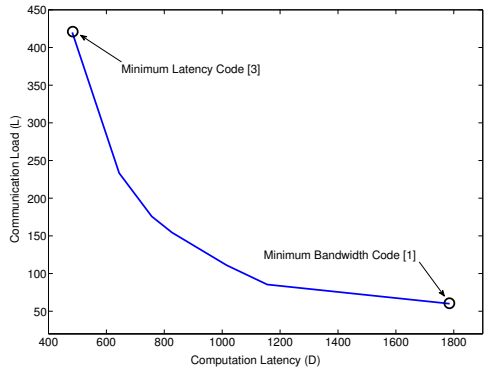



Fig. 1: The Latency-Load tradeoff, for a distributed matrix multiplication job of computing $N = 840$ output vectors using $K = 14$ servers each with a storage size $\mu = 1/2$.

by $L$) for distributed matrix multiplication (in short, the *Latency-Load Tradeoff*), in which as illustrated in Fig. 1, the above two coded schemes correspond to the two extreme points that minimize $L$ and $D$ respectively. Furthermore, we propose a unified coded scheme that organically integrates both of the coding techniques, and allows to systematically operate at any point on the introduced tradeoff.

For a given computation latency, we also prove an information-theoretic lower bound on the minimum required communication load to accomplish the distributed matrix multiplication. This lower bound is proved by first concatenating multiple instances of the problem with different reduction assignments of the output vectors, and then applying the cut-set bound on subsets of servers. At the two end points of the tradeoff, the proposed scheme achieves the minimum communication load to within a constant factor.

We finally note that there has been another tradeoff between the computation load in the Map phase and the communication load in the Shuffle phase for distributed computing, which is introduced and characterized in [1]. In this paper, we are fixing the amount of computation load (determined by the storage size) at each server, and focus on characterizing the tradeoff between the computation latency (determined by the number of servers that finish the Map computations) and the communication load. Hence, the considered tradeoff can be viewed as an extension of the tradeoff in [1] by introducing a third axis, namely the computation latency of the Map phase.

## II. Problem Formulation

### A. System Model

We consider a matrix multiplication problem in which given a matrix $\mathbf{A} \in \mathbb{F}_{2^T}^{m\times n}$ for some integers $T$, $m$ and $n$, and $N$ input vectors $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{F}_{2^T}^n$, we want to compute $N$ output vectors $\mathbf{y}_1 = \mathbf{A}\mathbf{x}_1, \dots, \mathbf{y}_N = \mathbf{A}\mathbf{x}_N$.

We perform the computations using $K$ distributed servers. Each server has a local memory of size $\mu mnT$ bits (i.e., it can store equivalent of $\mu$ fraction of the entries of the matrix $\mathbf{A}$), for some $\frac{1}{K} \le \mu \le 1$.[1]

[1]Thus enough information to recover the entire matrix $\mathbf{A}$ can be stored collectively on the $K$ servers.

We allow applying linear codes for storing the rows of $\mathbf{A}$ at each server. Specifically, Server $k$, $k \in \{1, \dots, K\}$, designs an encoding matrix $\mathbf{E}_k \in \mathbb{F}_{2^T}^{\mu m\times m}$, and stores

$$\mathbf{U}_k = \mathbf{E}_k\mathbf{A}. \tag{1}$$

The encoding matrices $\mathbf{E}_1, \dots, \mathbf{E}_K$ are design parameters and is denoted as *storage design*. The storage design is performed in prior to the computation.

*Remark* 1. For the Minimum Bandwidth Code in [1], each server stores $\mu m$ rows of the matrix $\mathbf{A}$. Thus, the rows of the encoding matrix $\mathbf{E}_k$ was chosen as a size-$\mu m$ subset of the rows of the identity matrix $\mathbf{I}_m$, according to a specific repetition pattern. While for the Minimum Latency Code in [4], $\mathbf{E}_k$ was generated randomly such that every server stores $\mu m$ random linear combinations of the rows of $\mathbf{A}$, achieving a $(\mu mK, m)$ MDS code. □

### B. Distributed Computing Model

We assume that the input vectors $\mathbf{x}_1, \dots, \mathbf{x}_N$ are known to all the servers. The overall computation proceeds in three phases: *Map*, *Shuffle*, and *Reduce*.

**Map Phase:** The role of the Map phase is to compute some coded intermediate values according to the locally stored matrices in (1), which can be used later to re-construct the output vectors. More specifically, for all $j = 1, \dots, N$, Server $k$, $k = 1, \dots, K$, computes the intermediate vectors

$$\mathbf{z}_{j,k} = \mathbf{U}_k\mathbf{x}_j = \mathbf{E}_k\mathbf{A}\mathbf{x}_j = \mathbf{E}_k\mathbf{y}_j. \tag{2}$$

We denote the latency for Server $k$ to compute $\mathbf{z}_{1,k}, \dots, \mathbf{z}_{N,k}$ as $S_k$. We assume that $S_1, \dots, S_K$ are i.i.d. random variables, and denote the $q$th order statistic, i.e., the $q$th smallest variable of $S_1, \dots, S_K$ as $S_{(q)}$, for all $q \in \{1, \dots, K\}$. We focus on a class of distributions of $S_k$ such that

$$\mathbb{E}\{S_{(q)}\} = \mu N g(K, q), \tag{3}$$

for some function $g(K, q)$.

The Map phase terminates when a subset of servers, denoted by $\mathcal{Q} \subseteq \{1, \dots, K\}$, have finished their Map computations in (2). A necessary condition for selecting $\mathcal{Q}$ is that the output vectors $\mathbf{y}_1 \dots, \mathbf{y}_N$ can be re-constructed by jointly utilizing the intermediate vectors calculated by the servers in $\mathcal{Q}$, i.e., $\{\mathbf{z}_{j,k} : j = 1, \dots, N, k \in \mathcal{Q}\}$. However, one can allow redundant computations in $\mathcal{Q}$, since if designed properly, they can be used to reduce the load of communicating intermediate results, for servers in $\mathcal{Q}$ to recover the output vectors in the following stages of the computation.

*Remark* 2. The Minimum Bandwidth Code in [1] waits for all servers to finish their computations, i.e., $\mathcal{Q} = \{1, \dots, K\}$. For the Minimum Latency Code in [4], $\mathcal{Q}$ is the subset of the fastest $\lceil \frac{1}{\mu} \rceil$ servers in performing the Map computations. □

**Definition 1** (Computation Latency)**.** We define the *computation latency*, denoted by $D$, as the average amount of time spent in the Map phase. ◊

After the Map phase, the job of computing the output vectors $\mathbf{y}_1 \dots, \mathbf{y}_N$ is continued *exclusively* over the servers in $\mathcal{Q}$. The final computations of the output vectors are distributed uniformly across the servers in $\mathcal{Q}$. We denote the set of

indices of the output vectors assigned to Server $k$ as $\mathcal{W}_k$, and $\{\mathcal{W}_k : k \in \mathcal{Q}\}$ satisfy 1) $\mathcal{W}_k \cap \mathcal{W}_{k'} = \varnothing$, $\forall k \neq k'$, 2) $|\mathcal{W}_k| = N/|\mathcal{Q}| \in \mathbb{N}$, $\forall k \in \mathcal{Q}$.[2]

**Shuffle Phase:** The goal of the Shuffle phase is to exchange the intermediate values calculated in the Map phase, to help each server recover the output vectors it is responsible for. To do this, every server $k$ in $\mathcal{Q}$ generates a message $X_k$ from the locally computed intermediate vectors $\mathbf{z}_{1,k}, \ldots, \mathbf{z}_{N,k}$ through an encoding function $\phi_k$, i.e., $X_k = \phi_k(\mathbf{z}_{1,k}, \ldots, \mathbf{z}_{N,k})$, such that upon receiving all messages $\{X_k : k \in \mathcal{Q}\}$, every server $k \in \mathcal{Q}$ can recover the output vectors in $\mathcal{W}_k$. We assume that the servers are connected by a shared bus link. After generating $X_k$, Server $k$ multicasts $X_k$ to all the other servers in $\mathcal{Q}$.

**Definition 2** (Communication Load)**.** We define the *communication load*, denoted by $L$, as the average total number of bits in all messages $\{X_k : k \in \mathcal{Q}\}$, normalized by $mT$ (i.e., the total number of bits in an output vector). $\Diamond$

**Reduce Phase:** The output vectors are re-constructed distributedly in the Reduce phase. Specifically, User $k$, $k \in \mathcal{Q}$, uses the locally computed vectors $\mathbf{z}_{1,k}, \ldots, \mathbf{z}_{N,k}$ and the received multicast messages $\{X_k : k \in \mathcal{Q}\}$ to recover the output vectors with indices in $\mathcal{W}_k$ via a decoding function $\psi_k$, i.e.,

$$\{\mathbf{y}_j : j \in \mathcal{W}_k\} = \psi_k(\mathbf{z}_{1,k}, \ldots, \mathbf{z}_{N,k}, \{X_k : k \in \mathcal{Q}\}). \quad (4)$$

For such a distributed computing system, we say a latency-load pair $(D, L) \in \mathbb{R}^2$ is *achievable* if there exist a storage design $\{\mathbf{E}_k\}_{k=1}^K$, a Map phase computation with latency $D$, and a shuffling scheme with communication load $L$, such that all output vectors can be successfully reduced.

**Definition 3.** We define the latency-load region, as the closure of the set of all achievable $(D, L)$ pairs. $\Diamond$

### C. Illustrating Example

In order to clarify the formulation, we use the following simple example to illustrate the latency-load pairs achieved by the two coded approaches discussed in Section I.

We consider a matrix $\mathbf{A}$ consisting of $m = 12$ rows $\mathbf{a}_1, \ldots, \mathbf{a}_{12}$. We have $N = 4$ input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_4$, and the computation is performed on $K = 4$ servers each has a storage size $\mu = \frac{1}{2}$. We assume that the Map latency $S_k$, $k = 1, \ldots, 4$, has a shifted-exponential distribution function

$$F_{S_k}(t) = 1 - e^{-(\frac{t}{\mu N} - 1)}, \ \forall t \geq \mu N, \quad (5)$$

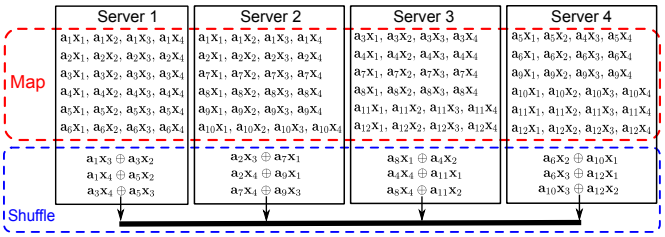
and by e.g., [10], the average latency for the fastest $q$, $1 \leq q \leq 4$, servers to finish the Map computations is

$$D(q) = \mathbb{E}\{S_{(q)}\} = \mu N \Big(1 + \sum_{j=K-q+1}^{K} \tfrac{1}{j}\Big). \quad (6)$$

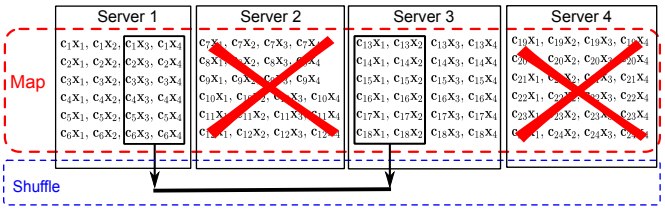
**Minimum Bandwidth Code [1].** The Minimum Bandwidth Code in [1] repeatedly stores each row of $\mathbf{A}$ at $\mu K$ servers with a particular pattern, such that in the Shuffle phase, $\mu K$ required intermediate values can be delivered with a single coded multicast message, which results in a coding gain of $\mu K$. We illustrate such coding technique in Fig. 2(a).

As shown in Fig. 2(a), a Minimum Bandwidth Code repeats the multiplication of each row of $\mathbf{A}$ with all input vectors

[2]We assume that $N \gg K$, and $|\mathcal{Q}|$ divides $N$ for all $\mathcal{Q} \subseteq \{1, \ldots, K\}$.




(a) Minimum Bandwidth Code. Every row of $\mathbf{A}$ is multiplied with the input vectors twice. For $k = 1, 2, 3, 4$, Server $k$ reduces the output vector $\mathbf{y}_k$. In the Shuffle phase, each server multicasts 3 bit-wise XORs, denoted by $\oplus$, of the calculated intermediate values, each of which is simultaneously useful for two other servers.




(b) Minimum Latency Code. $\mathbf{A}$ is encoded into 24 coded rows $\mathbf{c}_1 \ldots, \mathbf{c}_{24}$. Server 1 and 3 finish their Map computations first. They then exchange enough number (6 for each output vector) of intermediate values to reduce $\mathbf{y}_1, \mathbf{y}_2$ at Server 1 and $\mathbf{y}_3, \mathbf{y}_4$ at Server 3.

Fig. 2: Illustration of the Minimum Bandwidth Code in [1] and the Minimum Latency Code in [4].

$\mathbf{x}_1, \ldots, \mathbf{x}_4$, $\mu K = 2$ times across the 4 servers, e.g., $\mathbf{a}_1$ is multiplied at Server 1 and 2. The Map phase continues until all servers have finished their Map computations, achieving a computation latency $D(4) = 2 \times (1 + \sum_{j=1}^{4} \frac{1}{j}) = \frac{37}{6}$. For $k = 1, 2, 3, 4$, Server $k$ will be reducing output vector $\mathbf{y}_k$. In the Shuffle phase, as shown in Fig. 2(a), due to the specific repetition of Map computations, every server multicasts 3 bit-wise XORs, each of which is simultaneously useful for two other servers. For example, upon receiving $\mathbf{a}_1\mathbf{x}_3 \oplus \mathbf{a}_3\mathbf{x}_2$ from Server 1, Server 2 can recover $\mathbf{a}_3\mathbf{x}_2$ by canceling $\mathbf{a}_1\mathbf{x}_3$ and Server 3 can recover $\mathbf{a}_1\mathbf{x}_3$ by canceling $\mathbf{a}_3\mathbf{x}_2$. Similarly, every server decodes the needed values by canceling the interfering values using its local Map results. The Minimum Bandwidth Code achieves a communication load $L = 3 \times 4/12 = 1$.

The Minimum Bandwidth Code can be viewed as a specific type of network coding [11], or more precisely index coding [12], [13], in which the key idea is to design "side information" at the servers (provided by the Map results), enabling multicasting opportunities in the Shuffle phase to minimize the communication load.

**Minimum Latency Code [4].** The Minimum Latency Code in [4] uses MDS codes to generate some redundant Map computations, and assigns the coded computations across many servers. Such type of coding takes advantage of the abundance of servers so that one can terminate the Map phase as soon as enough coded computations are performed across the network, without needing to wait for the remaining straggling servers. We illustrate such coding technique in Fig. 2(b).

For this example, a Minimum Latency Code first has each server $k$, $k = 1, \ldots, 4$, independently and randomly generate 6 random linear combinations of the rows of $\mathbf{A}$, denoted by $\mathbf{c}_{6(k-1)+1}, \ldots, \mathbf{c}_{6(k-1)+6}$ (see Fig. 2(b)). We note that

$\{\mathbf{c}_1, \ldots, \mathbf{c}_{24}\}$ is a $(24, 12)$ MDS code of the rows of $\mathbf{A}$. Therefore, for any subset $\mathcal{D} \subseteq \{1, \ldots, 24\}$ of size $|\mathcal{D}| = 12$, using the intermediate values $\{\mathbf{c}_i \mathbf{x}_j : i \in \mathcal{D}\}$ can recover the output vector $\mathbf{y}_j$. The Map phase terminates once the fastest 2 servers have finished their computations (e.g., Server 1 and 3), achieving a computation latency $D(2) = 2 \times (1 + \frac{1}{3} + \frac{1}{4}) = \frac{19}{6}$. Then Server 1 continues to reduce $\mathbf{y}_1$ and $\mathbf{y}_2$, and Server 3 continues to reduce $\mathbf{y}_3$ and $\mathbf{y}_4$. As illustrated in Fig. 2(b), Server 1 and 3 respectively unicasts the intermediate values it has calculated and needed by the other server to complete the computation, achieving a communication load $L = 6 \times 4/12 = 2$.

From the above descriptions, we note that the Minimum Bandwidth Code uses about twice of the time in the Map phase compared with the Minimum Latency Code, and achieves half of the communication load in the Shuffle phase. They represent the two end points of a general latency-load tradeoff characterized in the next section.

## III. Main Results

The main results of the paper are, 1) a characterization of a set of achievable latency-load pairs by developing a unified coded framework, 2) an outer bound of the latency-load region, which are stated in the following two theorems.

**Theorem 1.** *For a distributed matrix multiplication problem of computing $N$ output vectors using $K$ servers, each with a storage size $\mu \geq \frac{1}{K}$, the latency-load region contains the lower convex envelop of the points*

$$\{(D(q), L(q)) : q = \lceil \tfrac{1}{\mu} \rceil, \ldots, K\}, \tag{7}$$

*in which*

$$D(q) = \mathbb{E}\{S_{(q)}\} = \mu N g(K, q), \tag{8}$$

$$L(q) = N \sum_{j=s_q}^{\lfloor \mu q \rfloor} \frac{B_j}{j} + N \min\left\{1 - \bar{\mu} - \sum_{j=s_q}^{\lfloor \mu q \rfloor} B_j, \frac{B_{s_q - 1}}{s_q - 1}\right\}, \tag{9}$$

*where $S_{(q)}$ is the $q$th smallest latency of the $K$ i.i.d. latencies $S_1, \ldots, S_K$ with some distribution $F$ to compute the Map functions in (2), $g(K, q)$ is a function of $K$ and $q$ computed from $F$, $\bar{\mu} \triangleq \frac{\lfloor \mu q \rfloor}{q}$, $B_j \triangleq \frac{\binom{q-1}{j}\binom{K-q}{\lfloor \mu q \rfloor - j}}{\frac{q}{K}\binom{K}{\lfloor \mu q \rfloor}}$, and $s_q \triangleq \inf\{s : \sum_{j=s}^{\lfloor \mu q \rfloor} B_j \leq 1 - \bar{\mu}\}$.*

We prove Theorem 1 In Section IV, in which we present a unified coded scheme that jointly designs the storage and the data shuffling, which achieves the latency in (8) and the communication load in (9).

*Remark* 3. The Minimum Latency Code and the Minimum Bandwidth Code correspond to $q = \lceil \frac{1}{\mu} \rceil$ and $q = K$, and achieve the two end points $(\mathbb{E}\{S_{(\lceil \frac{1}{\mu} \rceil)}\}, N - N/\lceil \frac{1}{\mu} \rceil)$ and $(\mathbb{E}\{S_{(K)}\}, N \frac{1 - \lfloor \mu K \rfloor / K}{\lfloor \mu K \rfloor})$ respectively. □

*Remark* 4. We numerically evaluate in Fig. 3 the latency-load pairs achieved by the proposed coded framework, for computing $N = 180$ output vectors using $K = 18$ servers each with a storage size $\mu = 1/3$. The achieved tradeoff approximately exhibits an inverse-linearly proportional relationship between the latency and the load. For instance, doubling the latency from 120 to 240 results in a drop of the communication load from 43 to 23 by a factor of 1.87. □

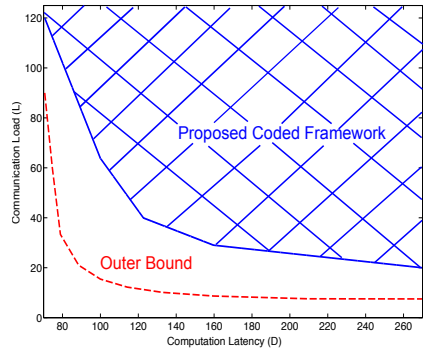



Fig. 3: Comparison of the latency-load pairs achieved by the proposed scheme with the outer bound, for computing $N = 180$ output vectors using $K = 18$ servers each with a storage size $\mu = 1/3$, assuming the the distribution function of the Map time in (5).

*Remark* 5. The key idea to achieve $D(q)$ and $L(q)$ in Theorem 1 is to design the concatenation of the MDS code and the repetitive executions of the Map computations, in order to take advantage of both the Minimum Latency Code and the Minimum Bandwidth Code. More specifically, we first generate $\frac{K}{q} m$ MDS-coded rows of $\mathbf{A}$, and then store each of them $\lfloor \mu q \rfloor$ times across the $K$ servers in a specific pattern. As a result, any subset of $q$ servers would have sufficient amount of intermediate results to reduce the output vectors, and we end the Map phase as soon as the fastest $q$ servers finish their Map computations, achieving the latency in (8).

We also exploit coded multicasting in the Shuffle phase to reduce the communication load. In the load expression (9), $B_j$, $j \leq \lfloor \mu q \rfloor$, represents the (normalized) number of coded rows of $\mathbf{A}$ repeatedly stored/computed at $j$ servers. By multicasting coded packets simultaneously useful for $j$ servers, $B_j$ intermediate values can be delivered to a server with a communication load of $\frac{B_j}{j}$, achieving a coding gain of $j$. We greedily utilize the coding opportunities with a larger coding gain until we get close to satisfying the demand of each server, which accounts for the first term in (9). Then the second term results from two follow-up strategies 1) communicate the rest of the demands uncodedly 2) continue coded multicasting with a smaller coding gain (i.e., $j = s_q - 1$), which may however deliver more than what is needed for reduction. □

**Theorem 2.** *The latency-load region is contained in the lower convex envelop of the points*

$$\{(D(q), \bar{L}(q)) : q = \lceil \tfrac{1}{\mu} \rceil, \ldots, K\}, \tag{10}$$

*in which $D(q)$ is given by (8) and*

$$\bar{L}(q) = N \max_{t=1,\ldots,q-1} \frac{1 - \min\{t\mu, 1\}}{\lceil \frac{q}{t} \rceil (q - t)} q. \tag{11}$$

We prove Theorem 2 in Section V, by deriving an information-theoretic lower bound on the minimum required communication load for a given computation latency, using any storage design and data shuffling scheme.

*Remark* 6. We numerically compare the outer bound in Theorem 2 and the achieved inner bound in Theorem 1 in Fig. 3, from which we make the following observations.

- At the minimum latency point, i.e., $q = 1/\mu = 3$ servers finish the Map computations, the proposed coded scheme achieves $1.33\times$ of the minimum communication load. In general, when $q = 1/\mu \in \mathbb{N}$, the lower bound in Theorem 2 $\bar{L}(\frac{1}{\mu}) = N/\lceil \frac{q}{t} \rceil|_{t=q-1} = N/\lceil \frac{1}{1-\mu} \rceil = \frac{N}{2}$. The proposed coded scheme, or Minimum Latency Code in this case, achieves the load $L(\frac{1}{\mu}) = N(1-\mu)$. Thus the proposed scheme always achieves the lower bound to within a factor of 2 at the minimum latency point.
- At the point with the maximum latency, i.e., all $K = 18$ servers finish the Map computations, the proposed coded scheme achieves $2.67\times$ of the lower bound on the minimum communication load. In general for $q = K$ and $\mu K \in \mathbb{N}$, we demonstrate in Appendix that the proposed coded scheme, or Minimum Bandwidth Code in this case, achieves a communication load $L(K) = N(1-\mu)/(\mu K)$ to within a factor of $3 + \sqrt{5}$ of the lower bound $\bar{L}(K)$.
- For the intermediate latency from 70 to 270, the communication load achieved by the proposed scheme is within a multiplicative gap of at most $4.2\times$ from the lower bound. In general, a complete characterization of the latency-load region (or an approximation to within a constant gap for all system parameters) remains open. □

## IV. Proposed Coded Framework

In this section, we prove Theorem 1 by proposing and analyzing a general coded framework that achieves the latency-load pairs in (7). We first demonstrate the key ideas of the proposed scheme through the following example, and then give the general description of the scheme.

### A. Example: $m = 20$, $N = 12$, $K = 6$ and $\mu = \frac{1}{2}$.

We have a problem of multiplying a matrix $\mathbf{A} \in \mathbb{F}_{2^T}^{m \times n}$ of $m = 20$ rows with $N = 12$ input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_{12}$ to compute 12 output vectors $\mathbf{y}_1 = \mathbf{A}\mathbf{x}_1 \ldots, \mathbf{y}_{12} = \mathbf{A}\mathbf{x}_{12}$, using $K = 6$ servers each with a storage size $\mu = \frac{1}{2}$.

We assume that we can afford to wait for $q = 4$ servers to finish their computations in the Map phase, and we describe the proposed storage design and shuffling scheme.

**Storage Design.** As illustrated in Fig 4, we first independently generate 30 random linear combinations $\mathbf{c}_1, \ldots, \mathbf{c}_{30} \in \mathbb{F}_{2^T}^{n}$ of the 20 rows of $\mathbf{A}$, achieving a $(30, 20)$ MDS code of the rows of $\mathbf{A}$. Then we partition these coded rows $\mathbf{c}_1, \ldots, \mathbf{c}_{30}$ into 15 batches each of size 2, and store every batch of coded rows at a unique pair of servers.

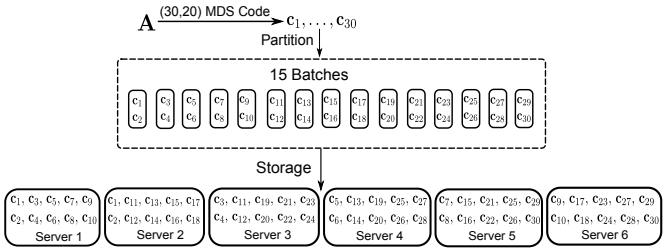



Fig. 4: Storage Design when the Map phase is terminated when 4 servers have finished the computations.

WLOG, due to the symmetry of the storage design, we assume that Servers 1, 2, 3 and 4 are the first 4 servers that finish their Map computations. Then we assign the Reduce tasks such that Server $k$ reduces the output vectors $\mathbf{y}_{3(k-1)+1}$, $\mathbf{y}_{3(k-1)+2}$ and $\mathbf{y}_{3(k-1)+3}$, for all $k \in \{1, \ldots, 4\}$.

After the Map phase, Server 1 has computed the intermediate values $\{\mathbf{c}_1\mathbf{x}_j, \ldots, \mathbf{c}_{10}\mathbf{x}_j : j = 1, \ldots, 12\}$. For Server 1 to recover $\mathbf{y}_1 = \mathbf{A}\mathbf{x}_1$, it needs any subset of 10 intermediate values $\mathbf{c}_i\mathbf{x}_1$ with $i \in \{11, \ldots, 30\}$ from Server 2, 3 and 4 in the Shuffle phase. Similar data demands hold for all 4 servers and the output vectors they are reducing. Therefore, the goal of the Shuffle phase is to exchange these needed intermediate values to accomplish successful reductions.

**Coded Shuffle.** We first group the 4 servers into 4 subsets of size 3 and perform coded shuffling within each subset. We illustrate the coded shuffling scheme for Servers 1, 2 and 3 in Fig. 5. Each server multicasts 3 bit-wise XORs, denoted by $\oplus$, of the locally computed intermediate values to the other two. The intermediate values used to create the multicast messages are the ones known exclusively at two servers and needed by another one. After receiving 2 multicast messages, each server recovers 6 needed intermediate values. For instance, Server 1 recovers $\mathbf{c}_{11}\mathbf{x}_1$, $\mathbf{c}_{11}\mathbf{x}_2$ and $\mathbf{c}_{11}\mathbf{x}_3$ by canceling $\mathbf{c}_2\mathbf{x}_7$, $\mathbf{c}_2\mathbf{x}_8$ and $\mathbf{c}_2\mathbf{x}_9$ respectively, and then recovers $\mathbf{c}_{12}\mathbf{x}_1$, $\mathbf{c}_{12}\mathbf{x}_2$ and $\mathbf{c}_{12}\mathbf{x}_3$ by canceling $\mathbf{c}_4\mathbf{x}_4$, $\mathbf{c}_4\mathbf{x}_5$ and $\mathbf{c}_4\mathbf{x}_6$ respectively.

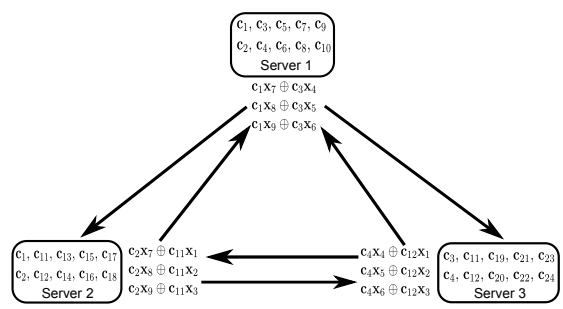



Fig. 5: Multicasting 9 coded intermediate values across Servers 1, 2 and 3. Similar coded multicast communications are performed for another 3 subsets of 3 servers.

Similarly, we perform the above coded shuffling in Fig. 5 for another 3 subsets of 3 servers. After coded multicasting within the 4 subsets of 3 servers, each server recovers 18 needed intermediate values (6 for each of the output vector it is reducing). As mentioned before, since each server needs a total of $3 \times (20 - 10) = 30$ intermediate values to reduce the 3 assigned output vectors, it needs another $30 - 18 = 12$ after decoding all multicast messages. We satisfy the residual data demands by simply having the servers unicast enough (i.e., $12 \times 4 = 48$) intermediate values for reduction. Overall, $9 \times 4 + 48 = 84$ (possibly coded) intermediate values are communicated, achieving a communication load of $L = 4.2$.

### B. General Scheme

We first describe the storage design, Map phase computation and the data shuffling scheme that achieves the latency-load pairs $(D(q), L(q))$ in (7), for all $q \in \{\lceil \frac{1}{\mu} \rceil, \ldots, K\}$. Given

these achieved pairs, we can "memory share" across them to achieve their lower convex envelop as stated in Theorem 1.

For ease of exposition, we assume that $\mu q \in \mathbb{N}$. Otherwise we can replace $\mu$ with $\bar{\mu} = \frac{\lfloor \mu q \rfloor}{q}$, and apply the proposed scheme for a storage size of $\bar{\mu}$.

**Storage Design.** We first use a $(\frac{K}{q}m, m)$ MDS code to encode the $m$ rows of matrix $\mathbf{A}$ into $\frac{K}{q}m$ coded rows $\mathbf{c}_1 \ldots, \mathbf{c}_{\frac{K}{q}m}$ (e.g., $\frac{K}{q}m$ random linear combinations of the rows of $\mathbf{A}$). Then as shown in Fig. 6, we evenly partitioned the $\frac{K}{q}m$ coded rows into $\binom{K}{\mu q}$ disjoint batches, each containing a subset of $\frac{m}{\frac{q}{K}\binom{K}{\mu q}}$ coded rows. [3] Each batch, denoted by $\mathcal{B}_{\mathcal{T}}$, is labelled by a unique subset $\mathcal{T} \subset \{1, \ldots, K\}$ of size $|\mathcal{T}| = \mu q$. That is

$$\{1, \ldots, \frac{K}{q}m\} = \{\mathcal{B}_{\mathcal{T}} : \mathcal{T} \subset \{1, \ldots, K\}, |\mathcal{T}| = \mu q\}. \quad (12)$$

Server $k$, $k \in \{1, \ldots, K\}$ stores the coded rows in $\mathcal{B}_{\mathcal{T}}$ as the rows of $\mathbf{U}_k$ if $k \in \mathcal{T}$.

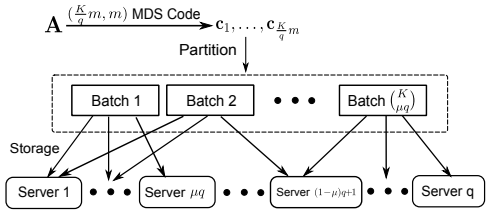



Fig. 6: General MDS coding and storage design.

In the above example, $q = 4$, and $\frac{K}{q}m = \frac{6}{4} \times 20 = 30$ coded rows of $\mathbf{A}$ are partitioned into $\binom{K}{\mu q} = \binom{6}{2} = 15$ batches each containing $\frac{30}{15} = 2$ coded rows. Every node is in 5 subsets of size two, thus storing $5 \times 2 = 10$ coded rows of $\mathbf{A}$.

**Map Phase Execution.** Each server computes the inner products between each of the locally stored coded rows of $\mathbf{A}$ and each of the input vectors, i.e., Server $k$ computes $\mathbf{c}_i \mathbf{x}_j$ for all $j = 1, \ldots, N$, and all $i \in \{\mathcal{B}_{\mathcal{T}} : k \in \mathcal{T}\}$. We wait for the fastest $q$ servers to finish their Map computations before halting the Map phase, achieving a computation latency $D(q)$ in (8). We denote the set of indices of these servers as $\mathcal{Q}$.

The computation then moves on exclusively over the $q$ servers in $\mathcal{Q}$, each of which is assigned to reduce $\frac{N}{q}$ out of the $N$ output vectors $\mathbf{y}_1 = \mathbf{A}\mathbf{x}_1, \ldots, \mathbf{y}_N = \mathbf{A}\mathbf{x}_N$.

For a feasible shuffling scheme to exist such that the Reduce phase can be successfully carried out, every subset of $q$ servers (since we cannot predict which $q$ servers will finish first) should have collectively stored at least $m$ distinct coded rows $\mathbf{c}_i$ for $i \in \{1, \ldots, \frac{K}{q}m\}$. Next, we explain how our proposed storage design meets this requirement. First, the $q$ servers in $\mathcal{Q}$ collectively provide a storage size equivalent to $\mu q m$ rows. Then since each coded row is stored by $\mu q$ out of all $K$ servers, it can be stored by at most $\mu q$ servers in $\mathcal{Q}$, and thus servers in $\mathcal{Q}$ collectively store at least $\frac{\mu q m}{\mu q} = m$ distinct coded rows.

**Coded Shuffle.** For $\mathcal{S} \subset \mathcal{Q}$ and $k \in \mathcal{Q} \backslash \mathcal{S}$, we denote the set of intermediate values needed by Server $k$ and known *exclusively* by the servers in $\mathcal{S}$ as $\mathcal{V}_{\mathcal{S}}^k$. More formally:

$$\mathcal{V}_{\mathcal{S}}^k \triangleq \{\mathbf{c}_i \mathbf{x}_j : j \in \mathcal{W}_k, i \in \{\mathcal{B}_{\mathcal{T}} : \mathcal{T} \cap \mathcal{Q} = \mathcal{S}\}\}. \quad (13)$$

[3] We focus on matrix multiplication problems for large matrices, and assume that $m \gg \frac{q}{K}\binom{K}{\mu q}$, for all $q \in \{\frac{1}{\mu}, \ldots, K\}$.

Due to the proposed storage design, for a particular $\mathcal{S}$ of size $j$, $\mathcal{V}_{\mathcal{S}}^k$ contains $\frac{N}{q} \cdot \frac{\binom{K-q}{\mu q - j} m}{\frac{q}{K}\binom{K}{\mu q}}$ intermediate values.

In the above example, we have $\mathcal{V}_{\{2,3\}}^1 = \{\mathbf{c}_{11}\mathbf{x}_j, \mathbf{c}_{12}\mathbf{x}_j : j = 1, 2, 3\}$, $\mathcal{V}_{\{1,3\}}^2 = \{\mathbf{c}_3\mathbf{x}_j, \mathbf{c}_4\mathbf{x}_j : j = 4, 5, 6\}$, and $\mathcal{V}_{\{1,2\}}^3 = \{\mathbf{c}_1\mathbf{x}_j, \mathbf{c}_2\mathbf{x}_j : j = 7, 8, 9\}$.

In the Shuffle phase, servers in $\mathcal{Q}$ create and multicast coded packets that are simultaneously useful for multiple other servers, until every server in $\mathcal{Q}$ recovers at least $m$ intermediate values for each of the output vectors it is reducing. The proposed shuffling scheme is *greedy* in the sense that every server in $\mathcal{Q}$ will always try to multicast coded packets simultaneously useful for the largest number of servers.

The proposed shuffle scheme proceeds as follows. For each $j = \mu q, \mu q - 1, \ldots, s_q$, where $s_q \triangleq \inf\{s : \sum_{j=s}^{\mu q} \frac{\binom{q-1}{j}\binom{K-q}{\mu q - j}}{\frac{q}{K}\binom{K}{\mu q}} \leq 1 - \mu\}$, and every subset $\mathcal{S} \subseteq \mathcal{Q}$ of size $j+1$:

1) For each $k \in \mathcal{S}$, we evenly and arbitrarily split $\mathcal{V}_{\mathcal{S}\backslash\{k\}}^k$ into $j$ disjoint segments $\mathcal{V}_{\mathcal{S}\backslash\{k\}}^k = \{\mathcal{V}_{\mathcal{S}\backslash\{k\},i}^k : i \in \mathcal{S}\backslash\{k\}\}$, and associate the segment $\mathcal{V}_{\mathcal{S}\backslash\{k\},i}^k$ with the server $i \in \mathcal{S}\backslash\{k\}$.
2) Server $i$, $i \in \mathcal{S}$, multicasts the bit-wise XOR, denoted by $\oplus$, of all the segments associated with it in $\mathcal{S}$, i.e., Server $i$ multicasts $\bigoplus_{k \in \mathcal{S}\backslash\{i\}} \mathcal{V}_{\mathcal{S}\backslash\{k\},i}^k$ to the other servers in $\mathcal{S}\backslash\{i\}$.

For every pair of servers $k$ and $i$ in $\mathcal{S}$, since Server $k$ has computed locally the segments $\mathcal{V}_{\mathcal{S}\backslash\{k'\},i}^{k'}$ for all $k' \in \mathcal{S}\backslash\{i, k\}$, it can cancel them from the message $\bigoplus_{k \in \mathcal{S}\backslash\{i\}} \mathcal{V}_{\mathcal{S}\backslash\{k\},i}^k$ sent by Server $i$, and recover the intended segment $\mathcal{V}_{\mathcal{S}\backslash\{k\},i}^k$.

For each $j$ in the above coded shuffling scheme, each server in $\mathcal{Q}$ recovers $\binom{q-1}{j}\frac{\binom{K-q}{\mu q - j} m}{\frac{q}{K}\binom{K}{\mu q}}$ intermediate values for each of the output vectors it is reducing. Therefore, $j = s_q + 1$ is the smallest size of the subsets in which the above coded multicasting needs to be performed, before enough number of intermediate values for reduction are delivered.

In each subset $\mathcal{S}$ of size $j$, since each server $i \in \mathcal{S}$ multicasts a coded segment of size $\frac{|\mathcal{V}_{\mathcal{S}\backslash\{k\}}^k|}{j}$ for some $k \neq i$, the total communication load so far, for $B_j = \frac{\binom{q-1}{j}\binom{K-q}{\mu q - j}}{\frac{q}{K}\binom{K}{\mu q}}$, is

$$\sum_{j=s_q}^{\mu q} \binom{q}{j+1} \frac{j+1}{j} \cdot \frac{N}{q} \cdot \frac{\binom{K-q}{\mu q - j}}{\frac{q}{K}\binom{K}{\mu q}} = \sum_{j=s_q}^{\mu q} N \frac{B_j}{j}, \quad (14)$$

Next, we can continue to finish the data shuffling in two different ways. The first approach is to have the servers in $\mathcal{Q}$ communicate with each other uncoded intermediate values, until every server has exactly $m$ intermediate values for each of the output vector it is responsible for. Using this approach, we will have a total communication load of

$$L_1 = \sum_{j=s_q}^{\mu q} N \frac{B_j}{j} + N(1 - \mu - \sum_{j=s_q}^{\mu q} B_j). \quad (15)$$

The second approach is to continue the above 2 steps for $j = s_q - 1$. Using this approach, we will have a total communication load of $L_2 = \sum_{j=s_q-1}^{\mu q} N \frac{B_j}{j}$.

Then we take the approach with less communication load, and achieve $L(q) = \min\{L_1, L_2\}$.

*Remark* 7. The ideas of efficiently creating and exploiting coded multicasting opportunities have been introduced in caching problems [14]–[16]. In this section, we illustrated how to create and utilize such coding opportunities in distributed computing to slash the communication load, when facing with straggling servers. □

## V. Converse

In this section, we prove the outer bound on the latency-load region in Theorem 2.

We start by considering a distributed matrix multiplication scheme that stops the Map phase when $q$ servers have finished their computations. For such scheme, as given by (8), the computation latency $D(q)$ is the expected value of the $q$th order statistic of the Map computation times at the $K$ servers. WLOG, we can assume that Servers $1,\dots,q$ first finish their Map computations, and they will be responsible for reducing the $N$ output vectors $\mathbf{y}_1,\dots,\mathbf{y}_N$.

To proceed, we first partition the $\mathbf{y}_1,\dots,\mathbf{y}_N$ into $q$ groups $\mathcal{G}_1,\dots,\mathcal{G}_q$ each of size $N/q$, and define the *output assignment*

$$\mathcal{A}=\left(\mathcal{W}_1^{\mathcal{A}},\mathcal{W}_2^{\mathcal{A}}\dots,\mathcal{W}_q^{\mathcal{A}}\right), \tag{16}$$

where $\mathcal{W}_k^{\mathcal{A}}$ denotes the group of output vectors reduced by Server $k$ in the output assignment $\mathcal{A}$.

Next we choose an integer $t\in\{1,\dots,q-1\}$, and consider the following $\lceil\frac{q}{t}\rceil$ output assignments which are circular shifts of $(\mathcal{G}_1,\dots,\mathcal{G}_q)$ with step size $t$,

$$\begin{aligned}\mathcal{A}_1&=(\mathcal{G}_1,\mathcal{G}_2,\dots,\mathcal{G}_q),\\ \mathcal{A}_2&=(\mathcal{G}_{t+1},\dots,\mathcal{G}_q,\mathcal{G}_1,\dots,\mathcal{G}_t),\\ &\vdots\\ \mathcal{A}_{\lceil\frac{q}{t}\rceil}&=\left(\mathcal{G}_{(\lceil\frac{q}{t}\rceil-1)t+1},\dots,\mathcal{G}_q,\mathcal{G}_1,\dots,\mathcal{G}_{(\lceil\frac{q}{t}\rceil-1)t}\right).\end{aligned} \tag{17}$$

*Remark* 8. We note that by the Map computation in (2), at each server all the input vectors $\mathbf{x}_1,\dots,\mathbf{x}_N$ are multiplied by the same matrix (i.e., $\mathbf{U}_k$ at Server $k$). Therefore, for the same set of $q$ servers and their storage contents, a feasible data shuffling scheme for one of the above output assignments is also feasible for all other $\lceil\frac{q}{t}\rceil-1$ assignments by relabelling the output vectors. As a result, the minimum communication loads for all of the above output assignments are identical. □

For a shuffling scheme admitting an output assignment $\mathcal{A}$, we denote the message sent by Server $k\in\{1,\dots,q\}$ as $X_k^{\mathcal{A}}$, with a size of $R_k^{\mathcal{A}}mT$ bits.

Now we focus on the Servers $1,\dots,t$ and consider the compound setting that includes all $\lceil\frac{q}{t}\rceil$ output assignments in (17). We observe that as shown in Fig. 7, in this compound setting, the first $t$ servers should be able to recover all output vectors $(\mathbf{y}_1\dots,\mathbf{y}_N)=(\mathcal{G}_1,\dots,\mathcal{G}_q)$ using their local computation results $\{\mathbf{U}_k\mathbf{x}_1,\dots,\mathbf{U}_k\mathbf{x}_N:k=1,\dots,t\}$ and the received messages in all the output assignments $\{X_k^{\mathcal{A}_1},\dots,X_k^{\mathcal{A}_{\lceil\frac{q}{t}\rceil}}:k=t+1,\dots,q\}$. Thus we have the following cut-set bound for the first $t$ servers.

$$rank\left(\begin{bmatrix}\mathbf{U}_1\\ \mathbf{U}_2\\ \vdots\\ \mathbf{U}_t\end{bmatrix}\right)NT+\sum_{j=1}^{\lceil\frac{q}{t}\rceil}\sum_{k=t+1}^{K}R_k^{\mathcal{A}_j}mT\geq NmT. \tag{18}$$

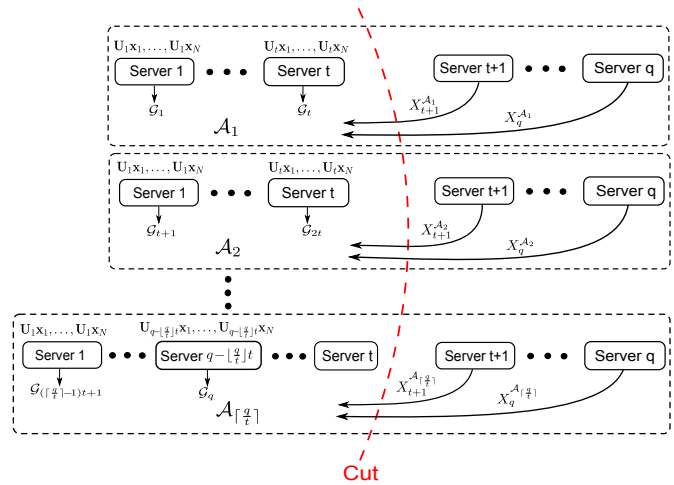



Fig. 7: Cut-set of Servers $1,\dots,t$ for the compound setting consisting of the $\lceil\frac{q}{t}\rceil$ output assignments in (17).

Next we consider $q$ subsets of servers each with size $t$: $\mathcal{N}_i\triangleq\{i,(i+1),\dots,(i+t-1)\}$, $i=1,\dots,q$, where the addition is modular $q$. Similarly, we have the following cut-set bound for $\mathcal{N}_i$:

$$rank\left(\begin{bmatrix}\mathbf{U}_i\\ \mathbf{U}_{i+1}\\ \vdots\\ \mathbf{U}_{i+t-1}\end{bmatrix}\right)NT+\sum_{j=1}^{\lceil\frac{q}{t}\rceil}\sum_{k\notin\mathcal{N}_i}R_k^{\mathcal{A}_j}mT\geq NmT. \tag{19}$$

Summing up these $q$ cut-set bounds, we have

$$NT\sum_{i=1}^{q}rank\left(\begin{bmatrix}\mathbf{U}_i\\ \mathbf{U}_{i+1}\\ \vdots\\ \mathbf{U}_{i+t-1}\end{bmatrix}\right)+\sum_{i=1}^{q}\sum_{j=1}^{\lceil\frac{q}{t}\rceil}\sum_{k\notin\mathcal{N}_i}R_k^{\mathcal{A}_j}mT\geq qNmT, \tag{20}$$

$$\Rightarrow\sum_{j=1}^{\lceil\frac{q}{t}\rceil}\sum_{i=1}^{q}\sum_{k\notin\mathcal{N}_i}R_k^{\mathcal{A}_j}\geq qN-qN\min\{\mu t,1\}. \tag{21}$$

$$\Rightarrow\lceil\tfrac{q}{t}\rceil(q-t)L\overset{(a)}{\geq}(1-\min\{t\mu,1\})qN, \tag{22}$$

where (a) results from the fact mentioned in Remark 8 that the communication load is independent of the output assignment.

Since (22) holds for all $t=1,\dots,q-1$, we have

$$L\geq\bar{L}(q)=N\max_{t=1,\dots,q-1}\frac{1-\min\{t\mu,1\}}{\lceil\frac{q}{t}\rceil(q-t)}q. \tag{23}$$

We assume that the Map phase terminates when $q$ servers finish the computations with probability $P(q)$, for all $q\in\{\lceil\frac{1}{\mu}\rceil,\dots,K\}$, then the communication load for a latency $\mathbb{E}_q(D(q))$ that is a convex combination of $\{\mathbb{E}\{S_{(q)}\}:q=\lceil\frac{1}{\mu}\rceil,\dots,K\}$, is lower bounded by $\mathbb{E}_q(\bar{L}(q))$ that is the same convex combination of $\{\bar{L}(q):q=\lceil\frac{1}{\mu}\rceil,\dots,K)\}$. Considering all distributions of $q$, we achieve all points on the

lower convex envelop of the points $\{(\mathbb{E}\{S_{(q)}\}, \bar{L}(q)) : q = \lceil \frac{1}{\mu} \rceil, \ldots, K\}$, as an outer bound on the latency-load region.

## APPENDIX

In this appendix, we prove that when all $K$ servers finish their Map computations, i.e., $\mathcal{Q} = \{1, \ldots, K\}$ and we operate at the point with the maximum latency, the communication load achieved by the proposed coded scheme (or the Minimum Bandwidth Code) is within a constant multiplicative factor of the lower bound on the communication load in Theorem 2. More specifically,

$$\frac{L(K)}{\bar{L}(K)} < 3 + \sqrt{5}, \tag{24}$$

when $\mu K$ is an integer,[4] where $L(K)$ and $\bar{L}(K)$ are respectively given by (9) and (11).

*Proof.* For $\mu K \in \mathbb{N}$, we have $L(K) = N\frac{1-\mu}{\mu K}$, and

$$\frac{L(K)}{\bar{L}(K)} = \frac{\frac{1-\mu}{\mu K}}{\max\limits_{t=1,\ldots,K-1} \frac{1-\min\{t\mu,1\}}{\lceil \frac{K}{t} \rceil (K-t)} K}. \tag{25}$$

We proceed to bound the RHS of (25) in the following two cases:

1) $1 \leq \frac{1}{\mu} \leq 3 + \sqrt{5}$.

We set $t = 1$ in (25) to have

$$\frac{L(K)}{\bar{L}(K)} \leq \frac{\frac{1-\mu}{\mu K}}{\frac{1-\mu}{K-1}} < \frac{1}{\mu} \leq 3 + \sqrt{5}. \tag{26}$$

2) $\frac{1}{\mu} > 3 + \sqrt{5}$.

Since $\mu K \geq 1$, we have $K - 1 \geq \lceil \frac{K}{2} \rceil \geq \lceil \frac{1}{2\mu} \rceil$.

In this case, we set $t = \lceil \frac{1}{2\mu} \rceil$ in (25) to have

$$\frac{L(K)}{\bar{L}(K)} \leq \frac{(1-\mu)\lceil \frac{K}{\lceil \frac{1}{2\mu} \rceil} \rceil (K - \lceil \frac{1}{2\mu} \rceil)}{\mu K^2 (1 - \mu \lceil \frac{1}{2\mu} \rceil)} \tag{27}$$

$$\leq \frac{2(1-\mu)(K - \lceil \frac{1}{2\mu} \rceil)}{K(1 - \mu \lceil \frac{1}{2\mu} \rceil)} < \frac{2(1-\mu)}{1 - \mu \lceil \frac{1}{2\mu} \rceil} \tag{28}$$

$$\leq \frac{2(1-\mu)}{1 - \mu(\frac{1}{2\mu} + 1)} \tag{29}$$

$$= 4 + \frac{4}{\frac{1}{\mu} - 2} < 3 + \sqrt{5}, \tag{30}$$

Comparing (26) and (30) completes the proof. ■

## REFERENCES


[1] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *e-print arXiv:1604.07086*, Apr. 2016, submitted to IEEE Trans. Inf. Theory.

[2] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Fundamental tradeoff between computation and communication in distributed computing," *IEEE ISIT*, July 2016.

[3] ——, "Coded MapReduce," *53rd Allerton Conference*, Sept. 2015.

[4] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *e-print arXiv:1512.02673*, Dec. 2015.

[5] ——, "Speeding up distributed machine learning using codes," *IEEE ISIT*, July 2016.

[6] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Sixth USENIX OSDI*, Dec. 2004.

[7] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," *2nd USENIX HotCloud*, vol. 10, p. 10, June 2010.

[8] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "A scalable framework for wireless distributed computing," *e-print arXiv:1608.05743*, Aug. 2016, submitted to IEEE/ACM Trans. Netw.

[9] ——, "Edge-facilitated wireless distributed computing," *IEEE GLOBECOM*, Dec. 2016.

[10] B. C. Arnold, N. Balakrishnan, and H. N. Nagaraja, *A first course in order statistics*. Siam, 1992, vol. 54.

[11] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, July 2000.

[12] Y. Birk and T. Kol, "Coding on demand by an informed source (ISCOD) for efficient broadcast of different supplemental data to caching clients," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2825–2830, June 2006.

[13] Z. Bar-Yossef, Y. Birk, T. Jayram, and T. Kol, "Index coding with side information," *IEEE Trans. Inf. Theory*, vol. 57, no. 3, Mar. 2011.

[14] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Inf. Theory*, vol. 60, no. 5, Mar. 2014.

[15] ——, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *IEEE/ACM Trans. Netw.*, Apr. 2014.

[16] M. Ji, G. Caire, and A. F. Molisch, "Fundamental limits of caching in wireless D2D networks," *IEEE Trans. Inf. Theory*, Feb. 2016.


[4] This always holds true for large $K$.