

**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**

1. 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。
2. 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

1. 译文不少于5000汉字（或2万印刷符）。
2. 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。
3. 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。
4. 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：\_\_\_\_\_\_\_\_\_\_100\_\_\_\_\_\_\_\_\_（百分制） 指导教师（签名）：  2024年6月12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。

# 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​，需要计算N个输出向量​。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。

# 问题描述

我们考虑下面的矩阵乘法问题，给定矩阵，大小为,个输入向量​，长度均为，需要计算个输出向量​。

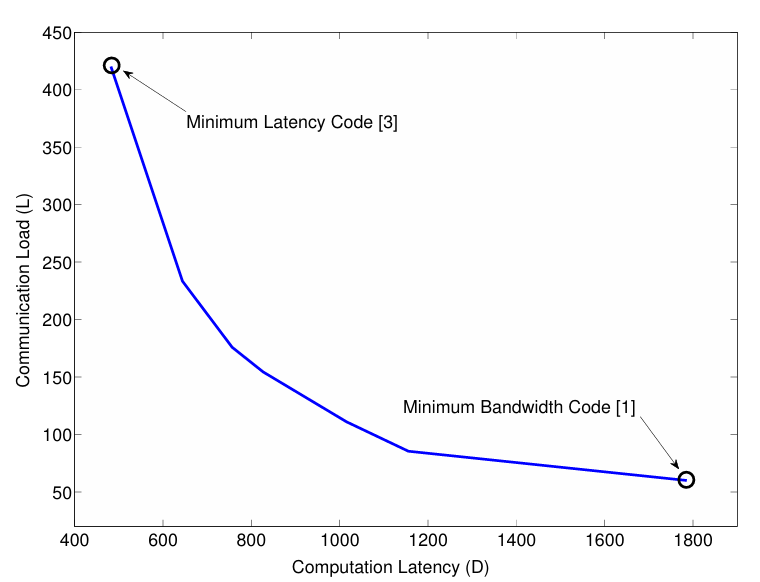


图 1 对于一个输出向量为N=840，服务节点数K=14的矩阵乘法任务的延迟和负载的权衡

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比特的本地空间，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的比例为的部分。

我们允许在每个节点中存储A的行的线性组合。对于节点和编码矩阵(大小为)，该节点可以保存矩阵。

我们假设输入向量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,以及节点()，需要计算中间向量。我们将节点计算向量的延迟分别为。这些延迟值为随机变量，第q个小的变量表示为。我们主要关注的这样一种分布：，其中为一些函数。

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。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,...,𝑁, 𝑘 ∈ 𝒬}）来重构输出向量。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。我们用表示分配给节点𝑘的输出向量的索引集合，{: 𝑘 ∈ 𝒬}，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通过编码函数从本地计算的中间结果生成消息，即。收到所有的消息之后，每个节点k可以恢复中的所有输出向量。我们假设节点通过共享总线连接。在生成之后，服务器𝑘向𝒬中的所有其他节点进行多播，将发送给它们。我们将通信负载表示为𝐿，定义为所有消息{𝑋𝑘 : 𝑘 ∈ 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 ∈ 𝒬，使用本地计算的向量和接收到的多播消息{𝑋𝑘 : 𝑘 ∈ 𝒬}，通过解码函数𝜓𝑘来得到中的索引表示的输出向量，即。

对于这样一个分布式计算系统，如果存在一组编码矩阵，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12个行向量组成的矩阵A。我们有𝑁 = 4个输入向量，在𝐾 = 4个服务节点上进行计算，每个服务器的存储大小为𝜇 = 1/2。我们假设Map阶段的延迟，𝑘 = 1,..., 4，具有一个移位指数分布函数。

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𝜇𝐾个服务器中，以便在Shuffle阶段，𝜇𝐾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𝜇𝐾。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进行乘法运算，重复次数为𝜇𝐾 = 2次，例如在节点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。对于𝑘 = 1, 2, 3, 4，服务器𝑘将进行reduce操作得到输出向量。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 × 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,..., 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，实现了一个(24, 12)的MDS码。因此，对于任意大小为∣𝒟∣ = 12的子集𝒟⊆{1,..., 24}，使用中间结果可以恢复输出向量。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)= 。然后，服务节点1继续对进行reduce操作，服务节点3继续对进行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= 6×4/12= 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。

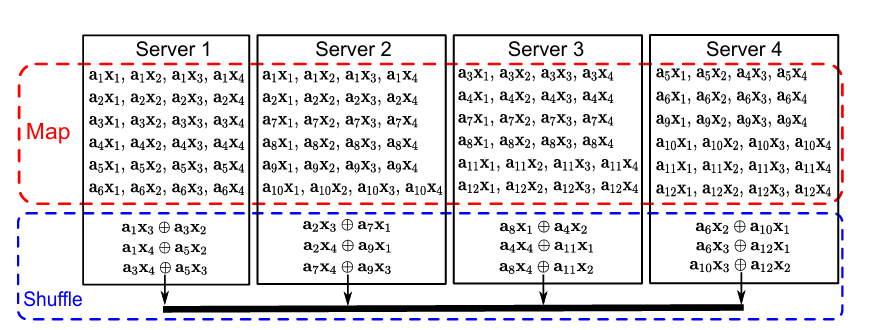


图 2 最小带宽编码方案

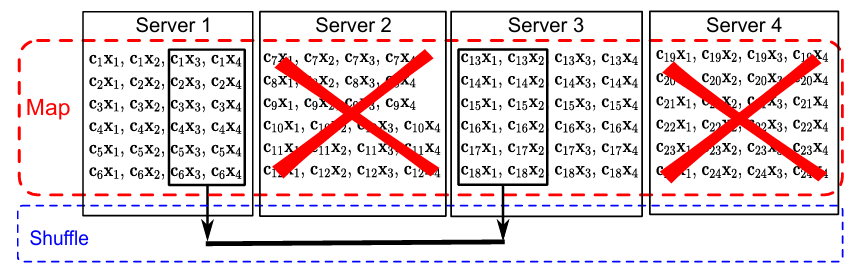


图 3 最小延迟编码方案

# 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𝜇 ≥,延迟-负载区域包含以下点的下凸包：。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𝜇 = 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。

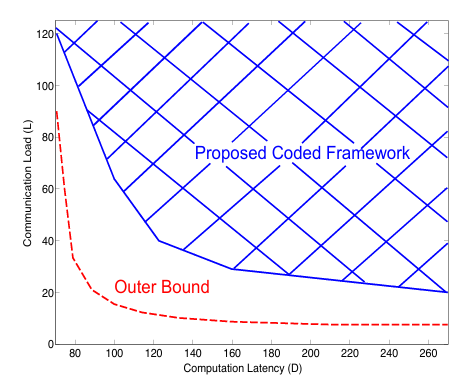


图 4 编码方案和外部约束的延迟，负载的比较

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。

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，实现编码增益为𝑗。

# 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，𝜇 = 1/2

需要将一个由20个行向量组成的矩阵A 与12个输入向量相乘，计算12个输出向量，使用𝐾 = 6个服务节点，每个节点的存储大小为𝜇 = 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，这些组合由A的20行构成。然后，我们将这些组合划分为大小为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对输出向量进行Reduce操作，其中𝑘 ∈ {1,..., 4}。由于节点1已计算了，

为了对 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，其中𝑖 ∈ {11,..., 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。

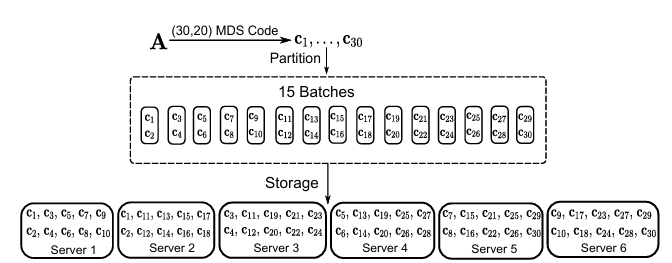


图 5 Map阶段的存储设计

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通过消除分别恢复。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 × (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 × 4 = 48）中间值来满足剩余的数据需求。总体而言，通信了9 × 4 + 48 = 84（可能经过编码）中间值，实现了通信负载𝐿 = 4.2。

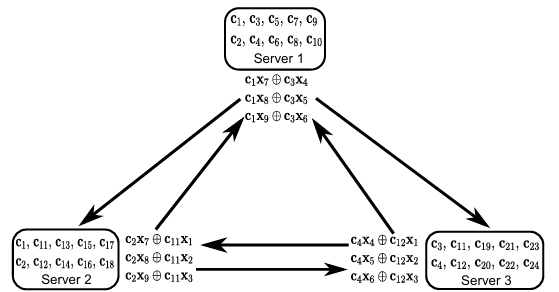


图 6 编码shuffle方案

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()的MDS码将矩阵A的𝑚行编码为个编码行。然后，如图 7所示，我们将个编码行均匀地分成不相交的批次，每个批次包含𝑚𝑞𝐾(𝐾𝜇𝑞)个编码行的子集。每个批次由表示，并用唯一的子集T ⊂{1,...,𝐾}（大小为∣T∣ = 𝜇𝑞）标记。节点𝑘，𝑘 ∈ {1,...,𝐾}，如果𝑘 ∈ 𝒯，则将中的编码行存储为U𝑘的行。在上面的示例中，𝑞 = 4， = 6 × 20 = 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 × 2 = 10个编码行的A。

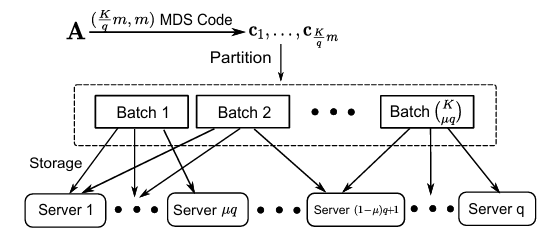


图 7 通用的Map阶段的存储设计

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（对于所有𝑗 = 1,...,𝑁和所有𝑖 ∈ ）。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𝒬。之后的计算在𝒬中的𝑞个节点进行，每个服务器负责对𝑁/𝑞个输出向量执行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，其中𝑖 ∈ {1,...,}。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，𝒬中的𝑞个节点提供了与行等效的存储大小。然后，由于每个编码行由所有𝐾个节点中的𝜇𝑞个节点存储，它最多可以由𝜇𝑞个𝒬中的节点存储，因此𝒬中的节点共同存储至少𝑚个不同的编码行，即𝜇𝑞𝑚/𝜇𝑞 = 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。在上面的例子中，。在Shuffle阶段，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。

# 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.