entropy

Article

Minimizing Computation and Communication Costs of Two-Sided Secure Distributed Matrix Multiplication under Arbitrary Collusion Pattern

Jin Li [1], Nan Liu [1] and Wei Kang [2,*]

[1] National Mobile Communications Research Laboratory, Southeast University, Nanjing 211189, China; lijin@seu.edu.cn (J.L.); nanliu@seu.edu.cn (N.L.)
[2] School of Information Science and Engineering, Southeast University, Nanjing 211189, China
* Correspondence: wkang@seu.edu.cn

Abstract: This paper studies the problem of minimizing the total cost, including computation cost and communication cost, in the system of two-sided secure distributed matrix multiplication (SDMM) under an arbitrary collusion pattern. In order to perform SDMM, the two input matrices are split into some blocks, blocks of random matrices are appended to protect the security of the two input matrices, and encoded copies of the blocks are distributed to all computing nodes for matrix multiplication calculation. Our aim is to minimize the total cost, overall matrix splitting factors, number of appended random matrices, and distribution vector, while satisfying the security constraint of the two input matrices, the decodability constraint of the desired result of the multiplication, the storage capacity of the computing nodes, and the delay constraint. First, a strategy of appending zeros to the input matrices is proposed to overcome the divisibility problem of matrix splitting. Next, the optimization problem is divided into two subproblems with the aid of alternating optimization (AO), where a feasible solution can be obtained. In addition, some necessary conditions for the problem to be feasible are provided. Simulation results demonstrate the superiority of our proposed scheme compared to the scheme without appending zeros and the scheme with no alternating optimization.

Keywords: secure distributed matrix multiplication; arbitrary collusion pattern; integer linear programming; integer geometric programming

Citation: Li, J.; Liu, N.; Kang, W. Minimizing Computation and Communication Costs of Two-Sided Secure Distributed Matrix Multiplication under Arbitrary Collusion Pattern. *Entropy* **2024**, *26*, 407. https://doi.org/10.3390/e26050407

Academic Editor: Boris Ryabko

Received: 14 March 2024
Revised: 2 May 2024
Accepted: 3 May 2024
Published: 8 May 2024

1. Introduction

With the development of the Internet of Things (IoT), the ubiquitous wireless devices can generate massive data via environment monitoring or target tracking [1]. However, due to the limited power or hardware architecture, these wireless devices cannot satisfy the data processing and computation requirements by themselves. This inspires wireless devices to seek help from online computing nodes who can assist in computation and data processing. Furthermore, distributed computing nodes can be employed to further accelerate the computation and data processing tasks, which means wireless devices can assign computation tasks to many different computing nodes, e.g., Apache Spark [2] and MapReduce [3]. On the other hand, if the online computing nodes are untrustworthy, we should also guarantee data security. Hence, how to perform computation of data with the aid of distributed computing nodes in a secure fashion is an important problem.

In this paper, we focus on the secure distributed matrix multiplication (SDMM) problem [4–8]. In [7,8], the trace-mapping framework has been employed to achieve communication-efficient schemes in the SDMM. The authors of [9] proposed a model of SDMM from an information-theoretic perspective. The user wishes to compute the product of two input matrices **A** and **B** with the aid of distributed computing nodes while guaranteeing the security of the information about the two input matrices. Two cases are considered: one-sided security and two-sided security. In the first case, the user only wants to protect the information security of matrix **A**, and **B** is a public matrix known to all computing nodes [10]. In the second case, we need to consider the information security of both matrices **A** and **B** [9,11]. The information theft by the distributed computing nodes can be modeled by the collusion pattern, which has also been studied in problems of secret sharing [12] and private information retrieval [13,14]. Some of the existing literature has studied the SDMM problem under homogeneous collusion patterns, where up to l computing nodes may collude to obtain the information of the two input matrices [9,15–18]. To balance the tradeoff between the uplink and downlink cost, the works proposed two schemes based on the secure cross subspace alignment [15]. In [9], the authors characterized the fundamental limits of minimum communication overhead for the SDMM problem under homogeneous collusion pattern. The work in [16] proposed a scheme based on the polynomial codes on sub-tasks assigned to computing nodes, which can mitigate the straggling effects efficiently. In [18], the authors have adopted some random matrices to encode two input matrices for the purpose of meeting the requirement of security. Then, many encoded copies are sent to different computing nodes for computation. Finally, the user receives these computation results from computing nodes and recovers the product of the two input matrices. It has considered two cases: (1) encoding the input matrices without extra random matrices, i.e., generalized polydot code, and (2) encoding the input matrices with some random matrices to satisfy the security constraint, i.e., secure the generalized polydot code. They also show the superiority of the proposed scheme on the recovery threshold, i.e., the number of computation results that is needed for users to decode the desired result without error, and the communication load between the user and computing nodes, i.e., the amount of downloaded information from computing nodes. Recently, rather than focusing on the homogeneous collusion pattern, ref. [19] studied the SDMM problem under the arbitrary collusion pattern. Considering the two proposed performance metrics, i.e., the normalized download cost and normalized upload cost, they provide the optimal scheme for the one-sided SDMM problem and an achievable scheme for the two-sided SDMM problem.

Both the private information retrieval and SDMM problem considered in [14,19] deal with the non-homogeneous collusion pattern scenario. The common approach of these two problems is assigning different number of copies to different servers. Intuitively speaking, the servers that collude more will be assigned a lower number of copies. More specifically, in [14], the authors considered the ratio between the message size and the amount of downloaded information from the servers. Then, the work of [19] studied the SDMM problem under the arbitrary collusion pattern for a fixed matrix splitting factor, and different numbers of copies were distributed to different computing nodes based on the collusion pattern to minimize the performance of normalized download and upload costs. However, the heterogeneity of the computing nodes in terms of storage capacity, communication capability, and computing capability was not taken into consideration. When full heterogeneity is taken into consideration, the numbers of copies assigned to different servers will not only depend on its colluding behavior but also on its storage capacity, communication capability, and computing capability. Furthermore, the fixed matrix splitting factor may affect the performance of SDMM. Hence, in this work, we study the problem of two-sided SDMM under an arbitrary collusion pattern with the flexible matrix splitting factor. Furthermore, in order to measure the communication and computation performance of the system, a new performance metric called the total cost, which is composed of the computation cost and communication cost, has been proposed in our paper. Additionally, the storage capability of the computing nodes and the delay requirement of the user are also considered. Then, an optimization problem is formulated by minimizing the total cost, subject to the security constraint of the two input matrices, the decodability constraint of the desired result of the multiplication, the storage capacity of the computing nodes, and the delay constraint. In order to overcome the divisibility problem of matrix splitting, we also propose a strategy of appending zeros to the input matrices and discuss the feasible set of some matrix splitting factors for the optimality of the problem. Finally, an alternating optimization (AO) algorithm based on some solvers is adopted to obtain a feasible solution, and some necessary conditions for the feasibility of problem have been provided.

The contributions of our paper are summarized as follows:

- We propose a new performance metric, the total cost, which includes communication cost and computation cost, to measure the performance of the SDMM problem under arbitrary collusion pattern. Our aim is to minimize the total cost, overall matrix splitting factors, number of appended random matrices, and distribution vector, while satisfying the security constraint of the two input matrices, the decodability constraint of the desired result of the multiplication, the storage capacity of the computing nodes, and the delay constraint.
- To overcome the divisibility problem of matrix splitting, we propose a strategy of padding zeros to the input matrices, which can split the input matrices into an arbitrary number of blocks compared to the scheme without appending zeros. Moreover, the value ranges of some matrix splitting factors are discussed for the optimality of the problem.
- The formulated optimization problem is solved by an AO algorithm based on some solvers. More specifically, for the optimization subproblem corresponding to number of appended random matrices and distribution vector, the relationship between number of appended random matrices and distribution vector can be found so that the subproblem is transformed into an integer linear programming over the distribution vector, which can be solved by the MATLAB function "intlinprog". Furthermore, we also provide some necessary conditions to verify the feasibility of this subproblem. Then, for the optimization subproblem corresponding to all matrix splitting factors, by relaxing the ceiling function and integer constraints, the subproblem can be transformed into an integer geometric programming problem solved by using "YALMIP". Simulation results show that our proposed scheme with padding zeros is superior to the scheme without appending zeros and the scheme with no alternating optimization.

The rest of this paper is organized as follows: Section 2 introduces the system model of the two-sided SDMM under arbitrary collusion pattern. Section 3 proposes a zero-padding strategy, discusses the feasible set of some matrix splitting factors, and formulates an optimization problem. Section 4 provides the algorithm to solve the problem. Simulation results and conclusions are shown in Sections 5 and 6, respectively.

Notation 1. *In this paper, the following notations are used.* $[1:N]$ *denotes the set* $\{1, 2, \cdots, N\}$. $\mathbf{h}_n$ *represents the n-th column vector of the matrix* $\mathbf{h}$. $\mathbf{1_N}$ *denotes the* $N \times 1$ *column vector. Positive integer is represented by* $\mathbb{Z}^+$*, natural number is denoted by* $\mathbb{N}$*, and the ceiling function is denoted by* $\lceil \cdot \rceil$.

2. System Model

As shown in Figure 1, we consider a user who wants to calculate the multiplication of two input matrices $\mathbf{A} \in \mathbb{F}^{T \times S}$ and $\mathbf{B} \in \mathbb{F}^{S \times D}$. We suppose that T, S and D are all integers and the finite field $\mathbb{F}$ is sufficiently large. Due to its own limited computational ability, the user wishes to split the two matrices $\mathbf{A}$ and $\mathbf{B}$ into many blocks and upload them to N computing nodes for computation. At the same time, both matrices $\mathbf{A}$ and $\mathbf{B}$ contain sensitive information, and the user does not want to leak any information to the N computing nodes.

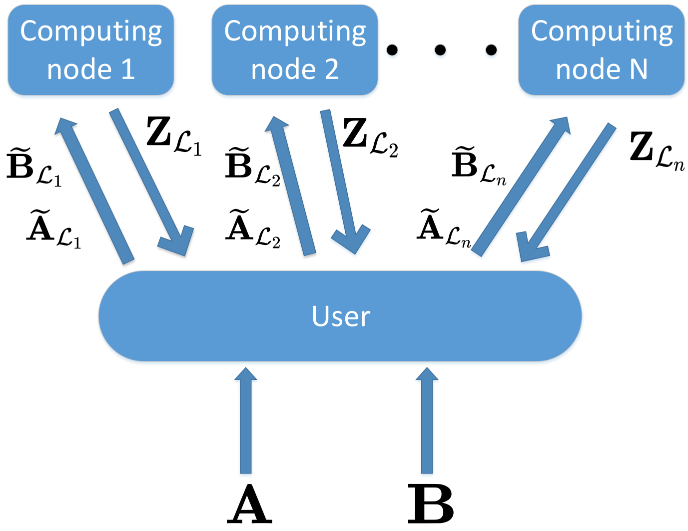

Figure 1. Two-sided secure distributed matrix multiplication.

We study the case where the computing nodes may collude with others to obtain information about the two matrices $\mathbf{A}$ and $\mathbf{B}$. We represent the colluding behaviors by a collusion pattern $\mathcal{P}$, which contains M colluding sets, i.e., $\mathcal{P} = \{\mathcal{T}_1, \mathcal{T}_2, \cdots \mathcal{T}_M\}$. Here, $\mathcal{T}_m \subseteq [1:N]$ is the m-th colluding set, which means that computing nodes in $\mathcal{T}_m$ may collude to obtain the information of the two matrices. We make the following two assumptions about the collusion pattern $\mathcal{P}$:

(1) For ease of presentation, we only include the maximal colluding set in $\mathcal{P}$. For instance, a colluding set $\{3,4,5,6\}$ means that computing nodes 3, 4, 5, and 6 collude. This implies that computing nodes belonging to any subset of $\{3,4,5,6\}$ also collude. However, for ease of presentation, we do not include the subsets of $\{3,4,5,6\}$ in $\mathcal{P}$.
(2) Every computing node must appear in at least one colluding set. This is because we assume that all computing nodes are curious, and no computing node can be trusted with the sensitive information of $\mathbf{A}$ and $\mathbf{B}$.

A collusion pattern $\mathcal{P}$ can be represented by its incidence matrix $\mathbf{B}_{\mathcal{P}}$, of size $N \times M$, i.e., if computing node i in the j-th colluding set of $\mathcal{P}$, the value of the (i,j)-th element in $\mathbf{B}_{\mathcal{P}}$ is 1. For example, when $\mathcal{P} = \{\{1,2,3\},\{1,4\},\{2,4\},\{3,4\},\{5\}\}$, its incidence matrix is

$$\mathbf{B}_{\mathcal{P}} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{1}$$

Due to the need to keep the two matrices secure, the user must encode $\mathbf{A}, \mathbf{B}$ before uploading them to the computing nodes for computation. Assume that there are N_1 encoded copies with $N_1 \geq N$, then these encoding functions are denoted as: $\boldsymbol{f} = (f_1, f_2, \cdots, f_{N_1}), \boldsymbol{g} = (g_1, g_2, \cdots, g_{N_1})$. We use $\widetilde{\mathbf{A}}_i$ and $\widetilde{\mathbf{B}}_i$ to represent the i-th encoded copy of matrices $\mathbf{A}$ and $\mathbf{B}$, respectively, $i \in [1:N_1]$, i.e., $\widetilde{\mathbf{A}}_i = f_i(\mathbf{A}), \widetilde{\mathbf{B}}_i = g_i(\mathbf{B})$. The user distributes a subset of the encoded matrices to computing node n, where the indices of this subset are written as $\mathcal{L}_n$, $\mathcal{L}_n \subseteq [1:N_1]$. This is termed *the upload phase.*

The computing node n computes the product, i.e., $\mathbf{Z}_i = \widetilde{\mathbf{A}}_i \widetilde{\mathbf{B}}_i, i \in \mathcal{L}_n$. Then, computing node n would send the computed results $\mathbf{Z}_i$, $i \in \mathcal{L}_n$ back to the user. This is termed *the download phase*.

In order to ensure the security of matrices $\mathbf{A}$ and $\mathbf{B}$, the following security constraint must be satisfied,

$$I\left(\mathbf{A},\mathbf{B};\left(\widetilde{\mathbf{A}}_{\mathcal{L}_n},\widetilde{\mathbf{B}}_{\mathcal{L}_n}\right)_{n\in\mathcal{T}_m}\right)=0,\quad \forall m\in[1:M]. \tag{2}$$

which indicates that computing nodes in each colluding set, when putting their received copies together, can not obtain any information about the two matrices.

In addition, the user must be able to decode the desired product $\mathbf{C}=\mathbf{AB}$ from the answers received from all the computing nodes, i.e., the decodability constraint

$$H(\mathbf{AB}|\mathbf{Z}_1,\mathbf{Z}_2,\cdots,\mathbf{Z}_{N_1})=0 \tag{3}$$

must be satisfied.

2.1. Matrix Encoding Scheme

We use the secure generalized polydot code (SGPD) in [18] to encode the two input matrices. First, we split $\mathbf{A}$ into $t\times s$ blocks, while $\mathbf{B}$ can be split into $s\times d$ blocks, i.e.,

$$\mathbf{A}=\begin{bmatrix}\mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,s}\\ \vdots & \ddots & \vdots\\ \mathbf{A}_{t,1} & \cdots & \mathbf{A}_{t,s}\end{bmatrix},\quad \mathbf{B}=\begin{bmatrix}\mathbf{B}_{1,1} & \cdots & \mathbf{B}_{1,d}\\ \vdots & \ddots & \vdots\\ \mathbf{B}_{s,1} & \cdots & \mathbf{B}_{s,d}\end{bmatrix}, \tag{4}$$

where T is divisible by t, S is divisible by s, and D is divisible by d. Then, $\mathbf{A}_{i,j}$ is of size $t_0\times s_0$, and $\mathbf{B}_{i,j}$ is of size $s_0\times d_0$, where we have defined

$$t_0=\frac{T}{t},\quad s_0=\frac{S}{s},\quad d_0=\frac{D}{d}. \tag{5}$$

In view of the security constraint (2), we append some random matrices $\mathbf{K}_{i,j}\in\mathbb{F}^{(T/t)\times(S/s)}$ and $\mathbf{K}'_{i,j}\in\mathbb{F}^{(S/s)\times(D/d)}$ as

$$\mathbf{A}^*=\begin{bmatrix}\mathbf{A}_{l,1} & \cdots & \mathbf{A}_{1,s}\\ \vdots & \ddots & \vdots\\ \mathbf{A}_{t,1} & \cdots & \mathbf{A}_{t,s}\\ \mathbf{K}_{1,1} & \cdots & \mathbf{K}_{1,s}\\ \vdots & \ddots & \vdots\\ \mathbf{K}_{l_\Delta,1} & \cdots & \mathbf{K}_{l_\Delta,s}\end{bmatrix}, \tag{6}$$

$$\mathbf{B}^*=\begin{bmatrix}\mathbf{B}_{1,1} & \cdots & \mathbf{B}_{1,d} & \mathbf{K}'_{1,1} & \cdots & \mathbf{K}'_{1,l_\Delta}\\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots\\ \mathbf{B}_{s,1} & \cdots & \mathbf{B}_{s,d} & \mathbf{K}'_{s,1} & \cdots & \mathbf{K}'_{s,l_\Delta}\end{bmatrix}, \tag{7}$$

where l_Δ rows of random matrices are appended to matrix $\mathbf{A}$, and l_Δ columns of random matrices are appended to matrix $\mathbf{B}$, where l_Δ is a positive integer. Each element of the random matrices $\mathbf{K}_{i,j}$ and $\mathbf{K}'_{i,j}$ are generated in an i.i.d. fashion according to the uniform distribution on $\mathbb{F}$. Note that (6) and (7) are just one way of appending random matrices. The other case is given by Method 2 in [19]. For simplicity, we only study the case of (6) and (7), and the other case of appending random matrices can be treated in a similar fashion.

In this case, the encoded matrices are generated according to

$$\begin{aligned}\widetilde{\mathbf{A}}_i =& \sum_{j=1}^{t}\sum_{k=1}^{s}\mathbf{A}^*_{j,k}x_i^{s(j-1)+k-1}\\ &+\sum_{j=t+1}^{t^*}\sum_{k=1}^{s}\mathbf{A}^*_{j,k}x_i^{s(j-1)+k-1} \quad i=1,\cdots,N_1, \end{aligned} \tag{8}$$

$$\begin{aligned}\widetilde{\mathbf{B}}_i =& \sum_{k=1}^{s}\sum_{p=1}^{d}\mathbf{B}^*_{k,p}x_i^{s-k+t^*s(p-1)}\\ &+\sum_{k=1}^{s}\sum_{p=d+1}^{d^*}\mathbf{B}^*_{k,p}x_i^{t^*sd+s(p-d)-k} \quad i=1,\cdots,N_1, \end{aligned} \tag{9}$$

where x_i, $i = 1, 2, \cdots, N_1$ are N_1 distinct non-zero elements in $\mathbb{F}$, and we have defined $t^* = t + l_\Delta, d^* = d + l_\Delta$.

The N_1 generated encoded copies of (8) and (9), i.e., $(\widetilde{\mathbf{A}}_i, \widetilde{\mathbf{B}}_i)$, $i \in [1 : N_1]$, will be distributed to the computing nodes, where computing node n will receive $\widetilde{\mathbf{A}}_{\mathcal{L}_n}$ and $\widetilde{\mathbf{B}}_{\mathcal{L}_n}$, where $\mathcal{L}_n \subseteq [1 : N_1]$ is the index set of the encoded matrices distributed to computing node n. We assume that $\mathcal{L}_n, n \in [1 : N]$, form a partition of the set $[1 : N_1]$, which means that each encoded copy will be distributed to one and only one computing node. Upon receiving $\widetilde{\mathbf{A}}_{\mathcal{L}_n}$ and $\widetilde{\mathbf{B}}_{\mathcal{L}_n}$, computing node n will calculate $\mathbf{Z}_i = \widetilde{\mathbf{A}}_i\widetilde{\mathbf{B}}_i$, $i \in \mathcal{L}_n$, and return $\mathbf{Z}_{\mathcal{L}_n}$ to the user. We distribute the encoded matrices to the computing nodes in the following way. Let $\mathbf{J} = \begin{bmatrix} J_1 & J_2 & \cdots J_N \end{bmatrix}^T$ be the distribution vector where $J_n \in [0 : N_1]$ is the number of distributed encoded matrices given to the n-th computing node. Then, we have $|\mathcal{L}_n| = J_n, \mathbf{1}^T\mathbf{J} = N_1$. It has been proved in [19] that when

$$\mathbf{B}_{\mathcal{P}}^T\mathbf{J} \leq (l_\Delta s)\mathbf{1}_M, \tag{10}$$

the security constraint (2) is satisfied. The physical meaning of (10) is that the number of encoded matrices for computing nodes in every colluding set must be smaller than the minimal number of random matrices appended in $\mathbf{A}^*$ or $\mathbf{B}^*$, which is $l_\Delta s$. Furthermore, the decodability constraint (3) is guaranteed by the following inequality [19]:

$$\mathbf{1^T J} = N_1 \geq l_\Delta(sd + 2s) + ts(d + 1) - 1. \tag{11}$$

It means that the encoded copies N_1 must be no smaller than $l_\Delta(sd + 2s) + ts(d + 1) - 1$ for decoding the desired results $\mathbf{C} = \mathbf{AB}$ without error.

2.2. Storage, Communication and Computing Requirements of Each Computing Node

The amount of storage each encoded copy $(\widetilde{\mathbf{A}}_i, \widetilde{\mathbf{B}}_i)$ occupies is $t_0s_0 + s_0d_0$. Suppose computing node n's storage capacity is M_n, then, if $t_0s_0 + s_0d_0 + t_0d_0 > M_n$, computing node n can not even store one encoded copy of $(\widetilde{\mathbf{A}}_i, \widetilde{\mathbf{B}}_i)$ and its corresponding answer, i.e., $\mathbf{Z}_i = \widetilde{\mathbf{A}}_i\widetilde{\mathbf{B}}_i$. If

$$M_n \geq t_0s_0 + s_0d_0 + t_0d_0, \tag{12}$$

then the computing node could store one encoded copy $(\widetilde{\mathbf{A}}_i, \widetilde{\mathbf{B}}_i)$, $i \in \mathcal{L}_n$, compute the multiplication $\mathbf{Z}_i = \widetilde{\mathbf{A}}_i\widetilde{\mathbf{B}}_i$, return the corresponding result and then retrieve another encoded copies from the user for further computation. Hence, (12) must be satisfied for all $n \in [1 : N]$. Written in vector form, we have

$$(t_0s_0 + s_0d_0 + t_0d_0)\mathbf{1_N} \leq \mathbf{M}. \tag{13}$$

Suppose computing node n's computation speed is V_n multiplications per second, then the time it takes for the user to complete the computation assigned to it, is

$$Q_n^{\mathrm{C}} = \frac{J_n}{V_n} t_0 s_0 d_0.$$

Further suppose that the uplink and downlink capacity between the user and computing node n are C_n^{U} and C_n^{D} symbols per second, respectively. Then, the amount of upload delay incurred at computing node n is

$$Q_n^{\mathrm{U}} = \frac{J_n}{C_n^{\mathrm{U}}} (t_0 s_0 + s_0 d_0),$$

and the amount of download delay incurred at computing node n is

$$Q_n^{\mathrm{D}} = \frac{J_n}{C_n^{\mathrm{D}}} t_0 d_0.$$

Then the total amount of delay incurred at computing node n when assigned with J_n number of encoded copies is

$$Q_n = Q_n^{\mathrm{U}} + Q_n^{\mathrm{D}} + Q_n^{\mathrm{C}}, \tag{14}$$

where we have assumed that the computing nodes can only do one of the three actions at any time instant: compute or receive upload or send download. This is also in line with the assumption that the computing nodes may not have enough memory to store all J_n copies all at once. Rather, it receives one copy, computes, and then sends it back to the user and then retrieves the next copy and repeats.

Thus, the total delay incurred for this computation is

$$Q = \max_{n \in [1:N]} Q_n = \max_{n \in [1:N]} \left\{ \frac{J_n}{V_n} t_0 s_0 d_0 + \frac{J_n}{C_n^{\mathrm{U}}} (t_0 s_0 + s_0 d_0) + \frac{J_n}{C_n^{\mathrm{D}}} t_0 d_0 \right\}, \tag{15}$$

and we require that the total delay is no larger than a given threshold Q_{th}, i.e.,

$$\max_{n \in [1:N]} \left\{ \frac{J_n}{V_n} t_0 s_0 d_0 + \frac{J_n}{C_n^{\mathrm{U}}} (t_0 s_0 + s_0 d_0) + \frac{J_n}{C_n^{\mathrm{D}}} t_0 d_0 \right\} \leq Q_{\mathrm{th}}. \tag{16}$$

Besides the delay constraint, cost should also be considered for efficient SDMM. More specifically, the cost we consider is comprised of the computation cost of computing nodes and the data transmission cost, where the data transmission cost can be can be written twice divided into the upload and download transmission cost. More specifically, we assume that the upload and download transmission cost for computing node n is c_n^{U}, and c_n^{D} per symbol, and the computation cost of each multiplication at computing node n is c_n^{C}, then the total required cost for the user doing the secure matrix multiplication of matrices **A** and **B** is

$$U = U_{\mathrm{U}} + U_{\mathrm{D}} + U_{\mathrm{C}}, \tag{17}$$

where U_{U} is the upload cost, which is given by

$$U_{\mathrm{U}} = (t_0 s_0 + s_0 d_0) \sum_{n=1}^{N} J_n c_n^{\mathrm{U}},$$

U_D is the download cost, which is given by

$$U_D = t_0 d_0 \sum_{n=1}^{N} J_n c_n^D,$$

and U_C is the computation cost, which is given by

$$U_C = t_0 s_0 d_0 \sum_{n=1}^{N} J_n c_n^C.$$

2.3. Problem Formulation

In this work, we would like to jointly optimize the distribution vector $\mathbf{J}$, and the matrix split parameter (t, s, d, l_Δ) such that the cost of the user, defined in (17), is minimized. At the same time, the security constraint (10), the decodability constraint (11), the storage constraint (13), and the delay constraint (16) must be satisfied.

3. The Feasible Set of (T, S, D)

Since we are splitting the two matrices $\mathbf{A}$ and $\mathbf{B}$ as shown in (4), it is natural to assume that t, s, and d have to take values such that T, S, and D be divisible by t, s, and d, respectively. For example, if $T = 5$, t can only take values in the set $\{1, 5\}$, because $T = 5$ is not divisible by $2, 3, 4$. However, this significantly limit the values that (t, s, d) can take and may provide a high cost for the user.

In this section, we propose a better and more general way as follows: we allow any t, s, d values, and to make the matrix splittable, we append zeros to the original matrix, i.e., append $\bar{s}$ columns and $\bar{t}$ rows to the matrix $\mathbf{A}$ and append $\bar{s}$ rows and $\bar{d}$ columns to the matrix $\mathbf{B}$, such that $(T + \bar{t})/t$, $(S + \bar{s})/s$, and $(D + \bar{d})/d$ are integers. This increases the dimension of the two matrices but enables us to split them into blocks in a more flexible way. For example, $\mathbf{A} \in \mathbb{F}^{5\times 4}$, i.e., $T = 5, S = 4$, and we would like to take $t = 2, s = 2$. However, $T = 5$ is not divisible by $t = 2$. Then, we can append one row of zeros to $\mathbf{A}$ so that the appended matrix has dimension 6×4 and thus can be divisible by $t = 2, s = 2$.

More generally, we propose that for any (t, s, d) with $t \in [1 : T], s \in [1 : S], d \in [1 : D]$, we may append $(T \mod t)$ many rows to the bottom of matrix $\mathbf{A}$ and $(S \mod s)$ many columns to the right side of matrix $\mathbf{A}$. Similarly, we append $(S \mod s)$ many rows to the bottom of matrix $\mathbf{B}$ and $(D \mod d)$ many columns to the right side of matrix $\mathbf{B}$. As a result, instead of (5), we have

$$t_0 = \left\lceil \frac{T}{t} \right\rceil, \quad s_0 = \left\lceil \frac{S}{s} \right\rceil, \quad d_0 = \left\lceil \frac{D}{d} \right\rceil. \tag{18}$$

As can be seen, not padding zeros and only using (t, s, d) that is a divisor of (T, S, D) is a special case.

Since we are considering padding zeros, it is also possible to have $t \in \{T + 1, T + 2, \cdots\}$ or $s \in \{S + 1, S + 2, \cdots\}$ or $d \in \{D + 1, D + 2, \cdots\}$. We show in the next lemma that this will only increase the cost at the user, defined in (17), for $t \in \{T + 1, T + 2, \cdots\}$ and $d \in \{D + 1, D + 2, \cdots\}$.

Lemma 1. *To minimize the cost at the user, i.e.,* (17)*, it is sufficient to consider* $t \in [1 : T]$ *and* $d \in [1 : D]$.

Proof. For $t \in \{T + 1, T + 2, \cdots\}$ and $d \in \{D + 1, D + 2, \cdots\}$, the only decodability constraint (11) becomes relaxed, and other constraints are unchanged. In this case, we can prove that the optimal cost will increase compared to $t \in [1 : T]$ and $d \in [1 : D]$. Please refer to Appendix A for detailed proof. □

Remark 1. *The case of s is different from the cases of t and d. When (l_Δ, t, d) is fixed, from security constraint (10) and decodability constraint (11), we see that on one hand, increasing s increases the number of blocks, but on the other hand, it also relaxes the security constraint. When computing nodes are heterogeneous, i.e., computing nodes have different computation cost, upload transmission cost and download transmission cost, the increase in s does not necessarily increase the total cost, because due to the more relaxed security constraint, we can distribute more blocks to computing nodes with lower costs. As a result, when we apply the strategy of appending zeros, the optimal s may not take values in $[1:S]$.*

After the above discussions, the problem described in Section 2.3 can be formally formulated as

$$\begin{aligned}
\min_{\mathbf{J},t,s,d,l_\Delta} \quad & \mathbf{J}^T\mathbf{c}^{\mathrm{U}}(t_0s_0+s_0d_0)+\mathbf{J}^T\mathbf{c}^{\mathrm{D}}t_0d_0+\mathbf{J}^T\mathbf{c}^{\mathrm{C}}t_0s_0d_0 && \text{(19a)}\\
\text{s.t.} \quad & (10), (11), && \text{(19b)}\\
& (t_0s_0+s_0d_0+t_0d_0)\mathbf{1_N} \le \mathbf{M}, && \text{(19c)}\\
& \max_{n\in[1:N]} \left\{ \frac{J_n}{V_n}t_0s_0d_0 + \frac{J_n}{C_n^{\mathrm{U}}}(t_0s_0+s_0d_0) + \frac{J_n}{C_n^{\mathrm{D}}}t_0d_0 \right\} \le Q_{\mathrm{th}}, && \text{(19d)}\\
& t_0 = \left\lceil \frac{T}{t} \right\rceil, \quad s_0 = \left\lceil \frac{S}{s} \right\rceil, \quad d_0 = \left\lceil \frac{D}{d} \right\rceil && \text{(19e)}\\
& 1 \le t \le T, 1 \le s, 1 \le d \le D, l_\Delta \ge 1, && \text{(19f)}\\
& t,s,d,l_\Delta \in \mathbb{Z}^+, \mathbf{J} \in \mathbb{N}^{N\times 1}. && \text{(19g)}
\end{aligned}$$

where (19b) provides the security constraint and decodability constraint, (19c) is the storage constraint with N computing nodes' storage capacity vector defined as $\mathbf{M} = \begin{bmatrix} M_1 & \cdots & M_N \end{bmatrix}^T$, and (19d) is the delay constraint. In the cost function (19a), we have defined the upload transmission cost vector, the download transmission cost vector and the computation cost vector of N computing nodes as $\mathbf{c}^{\mathrm{U}} = [c_1^{\mathrm{U}}, c_2^{\mathrm{U}}, \cdots, c_N^{\mathrm{U}}]^T \in \mathbb{R}^{N\times 1}$, $\mathbf{c}^{\mathrm{D}} = [c_1^{\mathrm{D}}, c_2^{\mathrm{D}}, \cdots, c_N^{\mathrm{D}}]^T \in \mathbb{R}^{N\times 1}$ and $\mathbf{c}^{\mathrm{C}} = [c_1^{\mathrm{C}}, c_2^{\mathrm{C}}, \cdots, c_N^{\mathrm{C}}]^T \in \mathbb{R}^{N\times 1}$, respectively. Note that the scheme of appending zeros makes the dimension of every block in $\mathbf{A}$ and $\mathbf{B}$ to be $t_0 \times s_0 = \left\lceil \frac{T}{t} \right\rceil \times \left\lceil \frac{S}{s} \right\rceil$ and $s_0 \times d_0 = \left\lceil \frac{S}{s} \right\rceil \times \left\lceil \frac{D}{d} \right\rceil$, respectively, as indicated by (19e). Furthermore, note that in (19f), while the values of t and d are limited to $[1:T]$ and $[1:D]$, respectively, the value of s does not have an upper bound due to Remark 1.

4. Algorithm Design

Due to coupling variables, integer constraints and nonlinear constraints and objective function of the problem in (19) , it is hard to find a global optimal or suboptimal solution. In the following, we propose an algorithm to obtain a feasible solution.

Coupling variables in Problem (19) inspires us to utilize the alternating optimization (AO) technique. Then, a feasible solution to Problem (19) can be obtained by solving the next two subproblems: one is fixing (t, s, d) to optimize $(\mathbf{J}, l_\Delta)$, and the other is optimizing (t, s, d) given $(\mathbf{J}, l_\Delta)$.

4.1. Optimization Subproblem of $(\mathbf{J}, l_\Delta)$ for a Fixed (T, S, D)

In this subsection, for a fixed (t, s, d), the optimization subproblem of Problem (19) corresponding to $(\mathbf{J}, l_\Delta)$ is given as

$$\min_{\mathbf{J}, l_\Delta} \quad (19a) \tag{20a}$$
$$\text{s.t.} \quad (10), (11), \tag{20b}$$
$$J_n \leq \frac{Q_{\text{th}}}{\frac{1}{V_n} t_0 s_0 d_0 + \frac{1}{C_n^{\text{U}}}(t_0 s_0 + s_0 d_0) + \frac{1}{C_n^{\text{D}}} t_0 d_0}, \quad \forall n \in [1:N] \tag{20c}$$
$$l_\Delta \geq 1, l_\Delta \in \mathbb{Z}^+, \mathbf{J} \in \mathbb{N}^{N\times 1}. \tag{20d}$$

Note that when (t, s, d) is fixed, the corresponding (t_0, s_0, d_0) is also fixed according to (19e). Further note that when (t, s, d) is fixed, the objective function (20a) is only a function of $\mathbf{J}$, and not l_Δ. Due to the fact that $J_n, c_n^{\text{U}}, c_n^{\text{D}}, c_n^{\text{C}} \geq 0, n = 1, ..., N$, the inequality of (11) must be satisfied with the equality when $\mathbf{J}^*$ is optimal. Hence, (11) can be rewritten as follows

$$\mathbf{1}^{\mathbf{T}}\mathbf{J} = l_\Delta(sd + 2s) + ts(d+1) - 1. \tag{21}$$

With equality (21), l_Δ can be expressed as a function of $\mathbf{J}$, i.e., $l_\Delta = \frac{\mathbf{1}^{\mathbf{T}}\mathbf{J} - ts(d+1)+1}{sd+2s}$. Then, substituting l_Δ in Problem (20) as a the function of $\mathbf{J}$, Problem (20) can be reformulated as

$$\min_{\mathbf{J}} \quad (19a) \tag{22a}$$
$$\text{s.t.} \quad (20c), \tag{22b}$$
$$\mathbf{B}_{\mathcal{P}}^T \mathbf{J} \leq (\frac{\mathbf{1}^{\mathbf{T}}\mathbf{J} - ts(d+1)+1}{d+2})\mathbf{1}_M, \tag{22c}$$
$$\frac{\mathbf{1}^{\mathbf{T}}\mathbf{J} - ts(d+1)+1}{sd+2s} \geq 1, \frac{\mathbf{1}^{\mathbf{T}}\mathbf{J} - ts(d+1)+1}{sd+2s} \in \mathbb{Z}^+, \mathbf{J} \in \mathbb{N}^{N\times 1}. \tag{22d}$$

Problem (22) is an integer linear programming problem with only one optimizing variable $\mathbf{J}$. This problem can be solved using MATLAB function "intlinprog". MATLAB's built-in "intlinprog" function is based on the branch and bound (BnB) algorithm and the interior point method [20,21] and is typically used to solve integer linear programming problems, such as the one in (22).

For certain system parameters and (t, s, d) values, Problem (22) is not feasible. To identify a necessary condition for the feasibility of Problem (22), we have the following lemma.

Before presenting the lemma, we define a variable p as the smallest number of colluding sets that contain all computing nodes. For example, for the collusion pattern represented by incidence matrix (1), p is equal to 3, because three colluding sets, i.e., $\{1,2,3\}$, $\{1,4\}$, $\{5\}$, include all computing nodes, and any 2 colluding sets in the collusion pattern can not include all computing nodes.

Lemma 2. *For fixed parameters (t, s, d), if Problem (20) is feasible, the following inequalities must be satisfied:*

$$\left\lceil \frac{tsd + ts - 1}{s(p-d-2)} \right\rceil \leq \frac{Y - tsd - ts + 1}{sd + 2s} \quad \textit{and} \quad p - d - 2 > 0$$

where Y is defined as

$$Y \triangleq \sum_{n=1}^{N} \frac{Q_{th}}{\frac{1}{V_n} t_0 s_0 d_0 + \frac{1}{C_n^U}(t_0 s_0 + s_0 d_0) + \frac{1}{C_n^D} t_0 d_0}, \tag{23}$$

where (t_0, s_0, d_0) satisfies (19e). Variable p is defined as the smallest number of colluding sets that contain all computing nodes.

Proof. Let us first derive a lower bound of l_Δ. According to the second assumption made about the collusion pattern in Section 2, every computing node must appear in at least one colluding set. So, we have

$$\sum_{n=1}^{N} J_n \leq \sum_{m\in\mathcal{P}'} \sum_{n\in\mathcal{T}_m} J_n \tag{24}$$

for any $\mathcal{P}'$ which is a subset of colluding sets in $\mathcal{P}$ that include all computing nodes, i.e., $\{i\} \in \mathcal{T}_j$ for some $\mathcal{T}_j \in \mathcal{P}'$ for any $i = 1, 2, \cdots, N$. For example, in the collusion pattern represented by the incidence matrix in (1), $\mathcal{P}'$ may be $\{\{1,2,3\},\{1,4\},\{5\}\}$, $\{\{1,2,3\},\{2,4\},\{5\}\}$, $\{\{1,2,3\},\{3,4\},\{5\}\}$, or $\{\{1,4\},\{2,4\},\{3,4\},\{5\}\}$.

The constraint (10) can be rewritten as

$$\sum_{n\in\mathcal{T}_m} J_n \leq l_\Delta s, \tag{25}$$

where $\mathcal{T}_m$ is the m-th colluding set. Inequality (25) shows that the total number of encoded matrices received by computing nodes in every colluding set can not be more than that of random matrices. Hence, from (25), we have

$$\sum_{m\in\mathcal{P}'} \sum_{n\in\mathcal{T}_m} J_n \leq \sum_{m\in\mathcal{P}'} l_\Delta s \leq p l_\Delta s \tag{26}$$

Thus, from (21), (24), and (26), we have

$$l_\Delta(sd+2s) + ts(d+1) - 1 = \sum_{n=1}^{N} J_n \leq p l_\Delta s.$$

When $p - d - 2 > 0$ is satisfied, l_Δ must satisfy

$$l_\Delta \geq \left\lceil \frac{tsd + ts - 1}{s(p-d-2)} \right\rceil. \tag{27}$$

On the other hand, when $p - d - 2 > 0$ is not satisfied, there exists no feasible l_Δ.

Next, we derive an upper bound on l_Δ. We have

$$l_\Delta(sd+2s) + ts(d+1) - 1 = \sum_{n=1}^{N} J_n \tag{28}$$

$$\leq \sum_{n=1}^{N} \frac{Q_{\text{th}}}{\frac{1}{V_n} t_0 s_0 d_0 + \frac{1}{C_n^{\text{U}}}(t_0 s_0 + s_0 d_0) + \frac{1}{C_n^{\text{D}}} t_0 d_0}, \tag{29}$$

where (28) follows from (21), and (29) follows from (20c). Hence, an upper bound on l_Δ is given by

$$l_\Delta \leq \frac{Y - tsd - ts + 1}{sd + 2s}, \tag{30}$$

where Y is as defined in (23).

If Problem (20) is feasible, we must have that $p - d - 2 > 0$, and the upper bound of l_Δ in (30) must be greater than or equal to the lower bound of l_Δ in (27).

Hence, the proof is complete. □

Based on Lemma 2, Algorithm 1 is proposed to solve Problem (20), where we check the necessary conditions of the feasibility of Problem (20) before solving it using the MATLAB "intlinprog" function.

Algorithm 1 Iterative Algorithm for Problem (20)

Input: (t, s, d), and (t_0, s_0, d_0) calculated according to (19e)
Output: $\mathbf{J}, l_\Delta$.
1: **if** $\frac{Y-tsd-ts+1}{sd+2s} \geq \left\lceil \frac{tsd+ts-1}{s(p-d-2)} \right\rceil$ and $p-d-2>0$ **then**
2: Solve Problem (22) with MATLAB function "intlinprog".
3: **else**
4: Problem (20) is infeasible.
5: **end if**

4.2. Optimization Subproblem of (T, S, D) for a Fixed $(\mathbf{J}, l_\Delta)$

In this subsection, given $(\mathbf{J}, l_\Delta)$, the optimization subproblem of Problem (19) corresponding to (t, s, d) is formulated as

$$\begin{aligned} \min_{t,s,d} \quad & (19a) && (31a)\\ \text{s.t.} \quad & (10), (11), (19c), (19d), (19e) && (31b)\\ & 1 \leq t \leq T, 1 \leq s, 1 \leq d \leq D, && (31c)\\ & d \leq p-3, && (31d)\\ & t, s, d \in \mathbb{Z}^+, && (31e) \end{aligned}$$

where constraint (31d) is derived from Lemma 2.

Ceiling functions, i.e., $\left\lceil \frac{T}{t} \right\rceil, \left\lceil \frac{S}{s} \right\rceil, \left\lceil \frac{D}{d} \right\rceil$ in (19e), and integer constraint (31e) make Problem (31) hard to address. We can solve this subproblem by relaxing the ceiling functions, i.e., Problem (31) can be recast as

$$\begin{aligned} \min_{t,s,d} \quad & \mathbf{J}^T\mathbf{c}^{\mathrm{U}}\left(TSt^{-1}s^{-1} + SDs^{-1}d^{-1}\right) + \mathbf{J}^T\mathbf{c}^{\mathrm{D}}TDt^{-1}d^{-1} + \mathbf{J}^T\mathbf{c}^{\mathrm{C}}TSDt^{-1}s^{-1}d^{-1} && (32a)\\ \text{s.t.} \quad & \frac{\mathbf{B}_P^T\mathbf{J}}{l_\Delta}s^{-1} \leq \mathbf{1_M}, && (32b)\\ & \frac{l_\Delta(sd+2s) + ts(d+1) - 1}{\mathbf{1^T J}} \leq 1, && (32c)\\ & \left(TSt^{-1}s^{-1} + SDs^{-1}d^{-1} + TDt^{-1}d^{-1}\right)\mathbf{1_N} \leq \mathbf{M}, && (32d)\\ & \frac{J_n}{V_n}TSDt^{-1}s^{-1}d^{-1} + \frac{J_n}{C_n^{\mathrm{U}}}\left(TSt^{-1}s^{-1} + SDs^{-1}d^{-1}\right) + \frac{J_n}{C_n^{\mathrm{D}}}TDt^{-1}d^{-1} \leq Q_{\mathrm{th}}, \forall n \in [1:N] && (32e)\\ & (31c), (31d), (31e). && (32f) \end{aligned}$$

Problem (32) is an integer geometric programming problem which can be solved using the Matlab toolbox YALMIP directly. MATLAB's built-in "YALMIP" function is based on the interior point method and BnB algorithm [22,23] and is typically used to solve integer geometric programming problems, such as the one in (32).

4.3. The Proposed Alternating Optimization (AO) Algorithm

Based on the above discussions of the two subproblems, we propose an AO algorithm as follows: In every AO iteration, for a fixed $(t, s, d) = (t^{(\tau)}, s^{(\tau)}, d^{(\tau)})$, and (t_0, s_0, d_0), which is calculated as $(t_0, s_0, d_0) = (\left\lceil \frac{T}{t^{(\tau)}} \right\rceil, \left\lceil \frac{S}{s^{(\tau)}} \right\rceil, \left\lceil \frac{D}{d^{(\tau)}} \right\rceil)$, we use Algorithm 1 to solve (22). Then, for the output $(\mathbf{J}, l_\Delta)$ of Algorithm 1, we solve Problem (32) with YALMIP directly and obtain $(t^{(\tau+1)}, s^{(\tau+1)}, d^{(\tau+1)})$.

Since Problem (32) is obtained by the relaxation of the ceiling functions, we may face the problem where even though the (t, s, d) found by YALMIP are integers, which we call $(\bar{t}, \bar{s}, \bar{d})$, the corresponding $\frac{T}{\bar{t}}$, $\frac{S}{\bar{s}}$, and $\frac{D}{\bar{d}}$ in Problem (32) may not be integers. In order to overcome this problem, for the converged solution (t^*, s^*, d^*) of the AO, we check whether constraints (19c) and (19d) in the original problem (19) are satisfied according to the definition of (19e). If they are, (t^*, s^*, d^*) is taken as the solution to Problem (31), and the corresponding block dimensions (t_0, s_0, d_0) are taken to be $\left(\left\lceil \frac{T}{t^*} \right\rceil, \left\lceil \frac{S}{s^*} \right\rceil, \left\lceil \frac{D}{d^*} \right\rceil\right)$ by padding zeros. If they are not, then we can employ an exhaustive search within a neighborhood near the converged solution (t^*, s^*, d^*) for a feasible solution or restart the algorithm with a new random initial point $(t^{(0)}, s^{(0)}, d^{(0)})$. In a time-constrained system, we can also abandon optimizing (t, s, d) and simply use the initial values $(t^{(0)}, s^{(0)}, d^{(0)})$ to obtain a timely solution.

Finally, the proposed AO algorithm to solve Problem (19) is summarized in Algorithm 2. (The source code can be found in the following link: https://github.com/SendBullet/SDMM-opt (accessed on 14 March 2024))

Algorithm 2 Alternating Optimization Algorithm for Problem (19)

1: Initialize $t^{(0)}, s^{(0)}, d^{(0)}, \tau = 0$ and the tolerance ϵ, where $t^{(0)}, s^{(0)}, d^{(0)}$ are chosen from divisors of (T, S, D) randomly.
2: **repeat**
3: Given $(t, s, d) = (t^{(\tau)}, s^{(\tau)}, d^{(\tau)})$, calculate $\mathbf{J}^{(\tau+1)}, l_\Delta^{(\tau+1)}$ by Algorithm 1.
4: Given $\mathbf{J} = \mathbf{J}^{(\tau+1)}, l_\Delta = l_\Delta^{(\tau+1)}$, calculate $(t^{(\tau+1)}, s^{(\tau+1)}, d^{(\tau+1)})$ by solving Problem (32) with YALMIP directly.
5: Set $\tau = \tau + 1$.
6: **until** The fractional increase of the objective function of Problem (19) is less than ϵ.
7: **if** Constraints (19c), (19d) and (19e) are satisfied simultaneously **then**
8: Output $\mathbf{J}^{(\tau)}, l_\Delta^{(\tau)}, (t^{(\tau)}, s^{(\tau)}, d^{(\tau)})$ and block dimension $(t_0, s_0, d_0) = \left(\left\lceil \frac{T}{t^{(\tau)}} \right\rceil, \left\lceil \frac{S}{s^{(\tau)}} \right\rceil, \left\lceil \frac{D}{d^{(\tau)}} \right\rceil\right)$.
9: **else**
10: Return to step 1 and restart with a new random initial point.
11: **end if**

4.4. Complexity Analysis

The complexity of Algorithm 2 per iteration mainly lies in Steps 3 and 4. In Step 3, the complexity of Algorithm 1 is derived from solving Problem (20) by MATLAB function "intlinprog" which uses BnB method. By omitting the lower-order terms, the main complexity of Algorithm 1 per iteration is $\mathcal{O}(2^N)$, where N is the number of computing nodes. In Step 4, similarly, the main complexity of solving Problem (32) by YALMIP with BnB method is $\mathcal{O}(2^3)$, where 3 is the dimension of optimizing variables (t, s, d) [24]. Hence, by neglecting the lower-order terms, the approximate computational complexity of Algorithm 2 per iteration is $\mathcal{O}(2^N)$ when $N \geq 3$, and $\mathcal{O}(2^3)$ when $N \leq 2$. As can be seen, the complexity scales exponentially with the number of computing nodes.

5. Simulation Results

In this section, we provide simulation results to evaluate the performance of the two-sided SDMM under arbitrary collusion pattern. We consider two collusion patterns. The first one has $N = 11$ computing nodes with the collusion pattern being $\mathcal{P}_1 = \{\{1,4\}, \{2,5\}, \{1,2,6\}, \{3,7\}, \{4,5,6,7\}, \{8\}, \{9\}, \{10\}, \{11\}\}$, while the second one consists of $N = 20$ computing nodes with the collusion pattern being $\mathcal{P}_2 = \{\{1,2,3,4,5,6\}, \{6,7,8,9,10,11\}, \{12,13\}, \{14\}, \{13,15\}, \{16,17\}, \{18\}, \{19\}, \{20\}\}$ So, for these two collusion patterns, the smallest number of colluding sets containing all of the computing nodes is $p = 7$ and $p = 9$, respectively. The stopping criterion in Algorithms 2 is set to $\epsilon = 10^{-8}$ [25]. Other system parameters are listed in the Table 1.

For simplicity, our proposed scheme in this paper is denoted by "**Pro.**". Then, the following two benchmarks are considered to compare with our proposed scheme:

(1) "**N/0.**": In this scenario, we do not append zeros to the input matrices. The optimization subproblem corresponding to (t, s, d) for a fixed $(\mathbf{J}, l_\Delta)$ is solved by exhaustive search in feasible pairs $(\tilde{t}, \tilde{s}, \tilde{d})$, which are divisors of (T, S, D). Other details are similar to Algorithm 2. This corresponds to the optimal performance of AO when no zeros are appended.

(2) "**SE.**": First, (t, s, d) is initialized by divisors of (T, S, D) randomly. Then, we solve Problem (22) to obtain $(\mathbf{J}, l_\Delta)$. This is a low complexity algorithm where no zeros are appended, and also, (t, s, d) are randomly chosen without being optimized. Only $(\mathbf{J}, l_\Delta)$ are optimized for the fixed randomly chosen (t, s, d).

First, we consider the collusion pattern $\mathcal{P}_1$ and the number of computing nodes being 11. Figure 2 shows the total cost versus the number of rows of matrix $\mathbf{A}$, i.e., T. Firstly, with the increase in T, the total cost of all schemes increases, which is caused by the growth of input matrix dimension. Secondly, our proposed algorithm outperforms the "**N/0.**" scheme when $T \geq 2500$. This means that when $T \geq 2500$, it is better to append zeros to the matrices to obtain a lower cost. Thirdly, our proposed algorithm always performs better than the "**SE**" scheme, which demonstrates the necessity of both appending zeros and performing AO. Lastly, from the comparison between $(S, D) = (2500, 2500)$ and $(S, D) = (3500, 3500)$, we can observe that the cost of the proposed scheme increases with the increase of the size of the matrices.

Table 1. System parameters.

Parameters	Values for $N = 11$	Values for $N = 20$
Storage capacity $\mathbf{M}$	$M_1 = M_2 = M_3 = M_4 = 1 \times 10^6$, $M_5 = M_6 = M_7 = M_8 = 2 \times 10^6$, $M_9 = M_{10} = M_{11} = 3 \times 10^6$	$M_1 = \cdots = M_7 = 1 \times 10^6$, $M_8 = \cdots = M_{15} = 2 \times 10^6$, $M_{16} = \cdots = M_{20} = 3 \times 10^6$
Computation speed V_n	$V_1 = V_2 = V_3 = V_4 = 2 \times 10^8$, $V_5 = V_6 = V_7 = V_8 = 5 \times 10^8$, $V_9 = V_{10} = V_{11} = 8 \times 10^8$	$V_1 = \cdots = V_7 = 2 \times 10^8$, $V_8 = \cdots = V_{15} = 5 \times 10^8$, $V_{16} = \cdots = V_{20} = 8 \times 10^8$
Uplink Capacity C_n^{U}	$C_1^{\mathrm{U}} = C_2^{\mathrm{U}} = C_3^{\mathrm{U}} = C_4^{\mathrm{U}} = 1024 \times 10^4$, $C_5^{\mathrm{U}} = C_6^{\mathrm{U}} = C_7^{\mathrm{U}} = C_8^{\mathrm{U}} = 2048 \times 10^4$, $C_9^{\mathrm{U}} = C_{10}^{\mathrm{U}} = C_{11}^{\mathrm{U}} = 4096 \times 10^4$	$C_1^{\mathrm{U}} = \cdots = C_7^{\mathrm{U}} = 1024 \times 10^4$, $C_8^{\mathrm{U}} = \cdots = C_{15}^{\mathrm{U}} = 2048 \times 10^4$, $C_{16}^{\mathrm{U}} = \cdots = C_{20}^{\mathrm{U}} = 4096 \times 10^4$
Downlink Capacity C_n^{D}	$C_1^{\mathrm{D}} = C_2^{\mathrm{D}} = C_3^{\mathrm{D}} = C_4^{\mathrm{D}} = 1024 \times 10^4$, $C_5^{\mathrm{D}} = C_6^{\mathrm{D}} = C_7^{\mathrm{D}} = C_8^{\mathrm{D}} = 2048 \times 10^4$, $C_9^{\mathrm{D}} = C_{10}^{\mathrm{D}} = C_{11}^{\mathrm{D}} = 4096 \times 10^4$	$C_1^{\mathrm{D}} = \cdots = C_7^{\mathrm{D}} = 1024 \times 10^4$, $C_8^{\mathrm{D}} = \cdots = C_{15}^{\mathrm{D}} = 2048 \times 10^4$, $C_{16}^{\mathrm{D}} = \cdots = C_{20}^{\mathrm{D}} = 4096 \times 10^4$
Upload cost $\mathbf{c}^{\mathrm{U}}$	$c_1^{\mathrm{U}} = c_2^{\mathrm{U}} = c_3^{\mathrm{U}} = c_4^{\mathrm{U}} = 3 \times 10^{-8}$, $c_5^{\mathrm{U}} = c_6^{\mathrm{U}} = c_7^{\mathrm{U}} = c_8^{\mathrm{U}} = 4 \times 10^{-8}$, $c_9^{\mathrm{U}} = c_{10}^{\mathrm{U}} = c_{11}^{\mathrm{U}} = 5 \times 10^{-8}$	$c_1^{\mathrm{U}} = \cdots = c_7^{\mathrm{U}} = 3 \times 10^{-8}$, $c_8^{\mathrm{U}} = \cdots = c_{15}^{\mathrm{U}} = 4 \times 10^{-8}$, $c_{16}^{\mathrm{U}} = \cdots = c_{20}^{\mathrm{U}} = 5 \times 10^{-8}$
Download cost $\mathbf{c}^{\mathrm{D}}$	$c_1^{\mathrm{D}} = c_2^{\mathrm{D}} = c_3^{\mathrm{D}} = c_4^{\mathrm{D}} = 3 \times 10^{-8}$, $c_5^{\mathrm{D}} = c_6^{\mathrm{D}} = c_7^{\mathrm{D}} = c_8^{\mathrm{D}} = 4 \times 10^{-8}$, $c_9^{\mathrm{D}} = c_{10}^{\mathrm{D}} = c_{11}^{\mathrm{D}} = 5 \times 10^{-8}$	$c_1^{\mathrm{D}} = \cdots = c_7^{\mathrm{D}} = 3 \times 10^{-8}$, $c_8^{\mathrm{D}} = \cdots = c_{15}^{\mathrm{D}} = 4 \times 10^{-8}$, $c_{16}^{\mathrm{D}} = \cdots = c_{20}^{\mathrm{D}} = 5 \times 10^{-8}$
Computation cost $\mathbf{c}^{\mathrm{C}}$	$c_1^{\mathrm{C}} = c_2^{\mathrm{C}} = c_3^{\mathrm{C}} = c_4^{\mathrm{C}} = 2 \times 10^{-8}$, $c_5^{\mathrm{C}} = c_6^{\mathrm{C}} = c_7^{\mathrm{C}} = c_8^{\mathrm{C}} = 6 \times 10^{-8}$, $c_9^{\mathrm{C}} = c_{10}^{\mathrm{C}} = c_{11}^{\mathrm{C}} = 8 \times 10^{-8}$	$c_1^{\mathrm{C}} = \cdots = c_7^{\mathrm{C}} = 2 \times 10^{-8}$, $c_8^{\mathrm{C}} = \cdots = c_{15}^{\mathrm{C}} = 6 \times 10^{-8}$, $c_{16}^{\mathrm{C}} = \cdots = c_{20}^{\mathrm{C}} = 8 \times 10^{-8}$
Delay threshold Q_{th}	1000	1000

Figure 3 plots the total cost versus the number of columns of matrix $\mathbf{B}$, i.e., D. Similar to Figure 2, the difference between our proposed scheme and the "**SE**" scheme becomes larger with the increase in the dimensions of the input matrices. However, the "**N/0.**" scheme achieves the same total cost as our proposed algorithm. This shows that, in this case, there is no need to pad zeros. Though the proposed scheme and the "**N/0.**" scheme have the same performance, the proposed scheme has less complexity because it can avoid the exhaustive search of the "**N/0.**" scheme.

Figure 4 illustrates the total cost versus the number of columns of matrix $\mathbf{A}$, i.e., S, which is also the number of rows of the matrix $\mathbf{B}$. Although the total cost of our proposed scheme is the same as that of the "**N/0.**" scheme for some S values, the gain of our proposed algorithm over the "**N/0.**" scheme increases with the increase in S. In fact, the gain is very significant for large S values, for example, when $S = 4000$, the total cost incurred by the proposed scheme is only 55.77% of the "**N/0.**" scheme when $(T, D) = (2500, 2500)$ and 55.07% when $(T, D) = (3500, 3500)$.

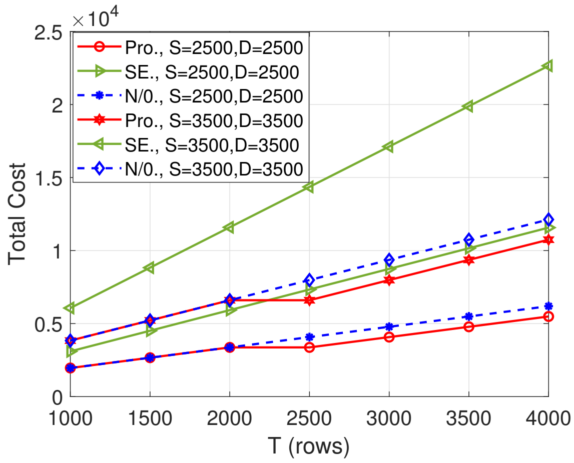

Figure 2. Total cost versus T when $N = 11$.

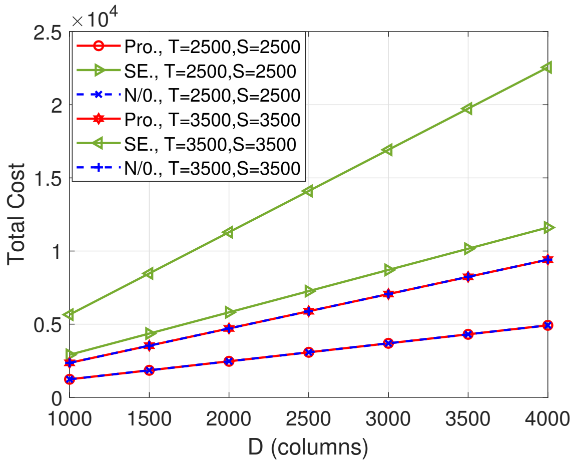

Figure 3. Total cost versus D when $N = 11$.

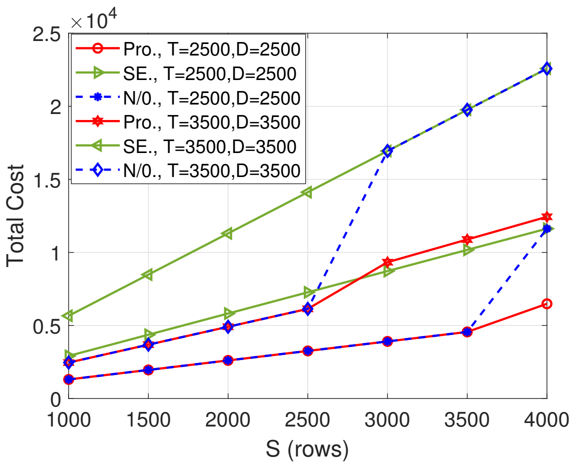

Figure 4. Total cost versus S when $N = 11$.

Figures 5, 6 and 7, respectively, depict the total cost with respect to T, S, and D when the number of computing nodes is 20. Similarly, our proposed scheme strictly outperforms the other two benchmarks in some cases, which further shows the superiority of the proposed scheme. Comparing Figures 2, 3 and 4 with Figures 5, 6 and 7, respectively, we see that the total cost decreases significantly with the increase in the number of computing nodes. Thus, when possible, the user should utilize more computing nodes to reduce the total cost.

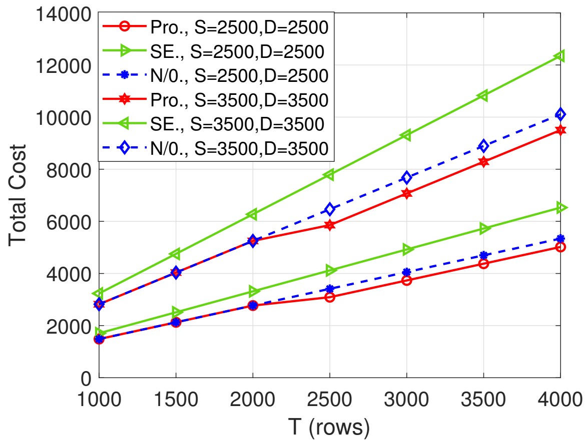

Figure 5. Total cost versus T when $N = 20$.

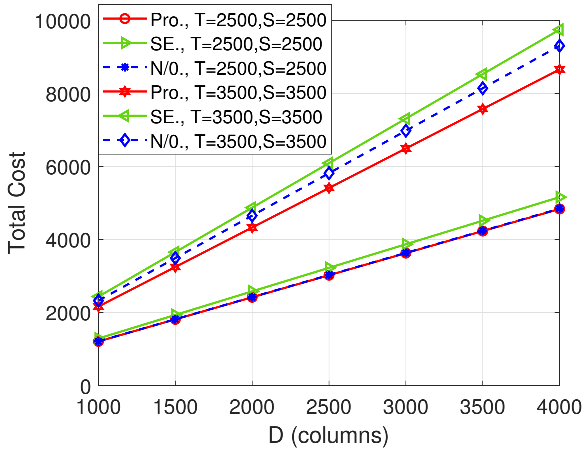

Figure 6. Total cost versus D when $N = 20$.

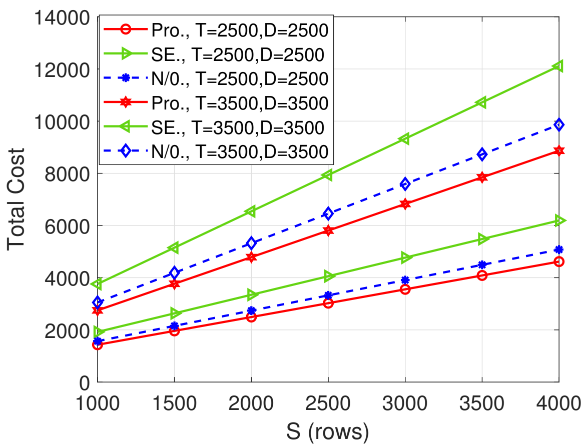

Figure 7. Total cost versus S when $N = 20$.

6. Conclusions

In this paper, we investigated the minimization problem of the total cost, comprised of the computation cost and the communication cost, in the system of two-sided SDMM under an arbitrary collusion pattern. For realizing SDMM, we split the two input matrices into many blocks and appended some extra blocks of random matrices to guarantee the security of the two input matrices. Then, the matrix multiplication is calculated based on the encoded copies in the computing nodes. Our aim is to minimize the total cost, while ensuring the security constraint of the two input matrices, the decodability constraint of the desired result of the multiplication, the storage capacity of the computing nodes, and the delay constraint. The distribution vector, the number of appended random matrices, and all matrix splitting factors were optimized. In order to overcome divisibility problem of matrix splitting, we firstly proposed a strategy of appending zeros to the two input matrices and then discussed the value ranges of some matrix splitting factors for the optimality of the problem. Next, an AO algorithm was provided to obtain a feasible solution. Furthermore, to verify the feasibility of the proposed optimization problem, some necessary conditions were provided. Numerical results demonstrated that our proposed scheme achieves a lower total cost compared to the scheme without appending zeros and the scheme without AO optimization.

Author Contributions: Conceptualization, J.L., N.L. and W.K.; methodology, J.L., N.L. and W.K.; formal analysis, J.L., N.L. and W.K.; writing—original draft preparation, J.L.; writing—review and editing, J.L., N.L. and W.K.; supervision, N.L. and W.K.; Funding acquisition, N.L. and W.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work is partially supported by the National Natural Science Foundation of China under Grants 62071115, 62361146853, 62371129, and the Research Fund of National Mobile Communications Research Laboratory, Southeast University (No. 2024A03).

Institutional Review Board Statement: Not applicable.

Data Availability Statement: Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Appendix A

Proof of Lemma 1. First, we consider the case of t for a fixed (l_Δ, s, d), i.e., appending $\mathbf{A} \in \mathbb{F}^{T\times S}$ with arbitrary rows of zeros.

We prove by contradiction. For a fixed (l_Δ, s, d), suppose the optimal value of t, i.e., t^*, takes values in $\{T+1, T+2, \cdots\}$. This means that in the padded matrix, denoted as $\mathbf{A}_1$, there is at least an entire row of blocks, whose values are all $\mathbf{0}_{t_0\times s_0}$. We may remove this entire row of zero blocks and obtain another matrix, denoted as $\mathbf{A}_2$. We will show that the cost at the user is smaller if we use $\mathbf{A}_2$ in place of $\mathbf{A}_1$. This means that it is suboptimal to have the padded matrix contain an entire row of zero small blocks. In other words, t^* can not take values in $\{T+1, T+2, \cdots\}$.

To see that the optimal cost at the user is smaller, if we use $\mathbf{A}_2$ in place of $\mathbf{A}_1$, we note that both $\mathbf{A}_2$ and $\mathbf{A}_1$ have the same dimension for the blocks, i.e., $t_0 \times s_0$. The difference is that $\mathbf{A}_2$ has a lower number of blocks, more specifically, t for $\mathbf{A}_2$ is less. Any $\mathbf{J}$ feasible for the problem of $\mathbf{A}_1$ is feasible for the problem of $\mathbf{A}_2$, as the security constraint (10) is the same for both problems due to the fixed (l_Δ, s, d), and the storage constraint (13) and the delay constraint (16) are the same due to both problems having the same dimension for the blocks, i.e., $t_0 \times s_0$. The decodability constraint (11) is more stringent for the problem of $\mathbf{A}_1$ as t is larger for $\mathbf{A}_1$. Hence, any $\mathbf{J}$ feasible for the problem of $\mathbf{A}_1$ is feasible for the problem of $\mathbf{A}_2$. Furthermore, the same $\mathbf{J}$ that are feasible for both problems incur the same cost (17) due to both problems having the same dimension for the blocks, i.e., $t_0 \times s_0$. Therefore, since the feasibility set of the problem for $\mathbf{A}_2$ is larger, the optimal cost at the user is smaller.

The same argument can be made for d for a fixed (l_Δ, s, t).

Hence, the proof is complete. □

References

1. El-Sayed, H.; Sankar, S.; Prasad, M.; Puthal, D.; Gupta, A.; Mohanty, M.; Lin, C.T. Edge of Things: The Big Picture on the Integration of Edge, IoT and the Cloud in a Distributed Computing Environment. *IEEE Access* **2018**, *6*, 1706–1717. [CrossRef]
2. Zaharia, M.; Chowdhury, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Spark: Cluster computing with working sets. In Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10), Boston, MA, USA, 22–25 June 2010.
3. Dean, J.; Ghemawat, S. MapReduce: Simplified data processing on large clusters. In Proceedings of the 6th Symposium on Operating Systems Design and Implementation, San Francisco, CA, USA, 6–8 December 2004.
4. D'Oliveira, R.G.; El Rouayheb, S.; Karpuk, D. GASP codes for secure distributed matrix multiplication. *IEEE Trans. Inf. Theory* **2020**, *66*, 4038–4050. [CrossRef]
5. D'Oliveira, R.G.; El Rouayheb, S.; Heinlein, D.; Karpuk, D. Degree tables for secure distributed matrix multiplication. *IEEE J. Sel. Areas Inf. Theory* **2021**, *2*, 907–918. [CrossRef]
6. Jia, Z.; Jafar, S.A. On the capacity of secure distributed batch matrix multiplication. *IEEE Trans. Inf. Theory* **2021**, *67*, 7420–7437. [CrossRef]
7. Kiah, H.M.; Kim, W.; Kruglik, S.; Ling, S.; Wang, H. Explicit Low-Bandwidth Evaluation Schemes for Weighted Sums of Reed-Solomon-Coded Symbols. *IEEE Trans. Inf. Theory* **2024**. [CrossRef]
8. Machado, R.A.; D'Oliveira, R.G.L.; Rouayheb, S.E.; Heinlein, D. Field Trace Polynomial Codes for Secure Distributed Matrix Multiplication. In Proceedings of the 2021 XVII International Symposium "Problems of Redundancy in Information and Control Systems" (REDUNDANCY), Moscow, Russia, 25–29 October 2021; pp. 188–193. [CrossRef]
9. Chang, W.T.; Tandon, R. On the capacity of secure distributed matrix multiplication. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6.
10. Bitar, R.; Parag, P.; El Rouayheb, S. Minimizing latency for secure coded computing using secret sharing via staircase codes. *IEEE Trans. Commun.* **2020**, *68*, 4609–4619. [CrossRef]
11. Ebadifar, S.; Kakar, J.; Sezgin, A. The Need for Alignment in Rate-Efficient Distributed Two-Sided Secure Matrix Computation. In Proceedings of the ICC 2019—2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6. [CrossRef]
12. Stinson, D.R. An explication of secret sharing schemes. *Des. Codes Cryptogr.* **1992**, *2*, 357–390. [CrossRef]
13. Tajeddine, R.; Gnilke, O.W.; Karpuk, D.; Freij-Hollanti, R.; Hollanti, C.; El Rouayheb, S. Private information retrieval schemes for coded data with arbitrary collusion patterns. In Proceedings of the 2017 IEEE International Symposium on Information Theory (ISIT), Aachen, Germany, 25–30 June , 2017; pp. 1908–1912.
14. Yao, X.; Liu, N.; Kang, W. The capacity of private information retrieval under arbitrary collusion patterns for replicated databases. *IEEE Trans. Inf. Theory* **2021**, *67*, 6841–6855. [CrossRef]
15. Kakar, J.; Khristoforov, A.; Ebadifar, S.; Sezgin, A. Uplink-downlink tradeoff in secure distributed matrix multiplication. *arXiv* **2019**, arXiv:1910.13849.
16. Yang, H.; Lee, J. Secure Distributed Computing With Straggling Servers Using Polynomial Codes. *IEEE Trans. Inf. Forensics Secur.* **2019**, *14*, 141–150. [CrossRef]
17. Chen, Z.; Jia, Z.; Wang, Z.; Jafar, S.A. GCSA codes with noise alignment for secure coded multi-party batch matrix multiplication. *IEEE J. Sel. Areas Inf. Theory* **2021**, *2*, 306–316. [CrossRef]
18. Aliasgari, M.; Simeone, O.; Kliewer, J. Private and Secure Distributed Matrix Multiplication With Flexible Communication Load. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 2722–2734. [CrossRef]
19. Yao, Y.; Liu, N.; Kang, W.; Li, C. Secure Distributed Matrix Multiplication Under Arbitrary Collusion Pattern. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 85–100. [CrossRef]
20. Van Roy, T.J.; Wolsey, L.A. *Integer Programming*; Core Discussion Papers Rp; Center for Operations Research and Econometrics: Ottignies-Louvain-la-Neuve, Belgium, 2009.
21. Boyd, S.P.; Vandenberghe, L. *Convex Optimization*; Cambridge University Press: Cambridge, UK, 2004.
22. Boyd, S.; Kim, S.J.; Vandenberghe, L.; Hassibi, A. A tutorial on geometric programming. *Optim. Eng.* **2007**, *8*, 67–127. [CrossRef]
23. Lofberg, J. YALMIP: A toolbox for modeling and optimization in MATLAB. In Proceedings of the 2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508), Taipei, Taiwan, 2–4 September 2004; pp. 284–289. [CrossRef]
24. Wang, Y.; Chen, M.; Li, Z.; Hu, Y. Joint Allocations of Radio and Computational Resource for User Energy Consumption Minimization Under Latency Constraints in Multi-Cell MEC Systems. *IEEE Trans. Veh. Technol.* **2023**, *72*, 3304–3320. [CrossRef]
25. Zhou, G.; Pan, C.; Ren, H.; Wang, K.; Nallanathan, A. Intelligent reflecting surface aided multigroup multicast MISO communication systems. *IEEE Trans. Signal Process.* **2020**, *68*, 3236–3251. [CrossRef]