of m.

II. Proposed System Model with Varying IV size

The considered system model is motivated by the fact that the relative size of IVs is a design choice for many MapReduce applications. The goal of this work is to utilize the design freedom in defining map and reduce functions to develop new low-complexity CDC designs that support varying IV sizes. This distinguished feature yields novel CDC designs with greater flexibility, i.e., designs that can operate under a wider range of system parameters. This is in contrast to previous CDC designs (e.g., [4], [6], [11]) that can only operate with constant IV sizes. For this reason, we adopt a more general system model of the original CDC work [4].

We consider a network of K nodes, labeled $1,\ldots,K$. The whole dataset is split into N equally sized input files, $\{w_1,\ldots,w_N\}$, based on the specific design. The system aims to compute K output functions, $\phi_k(w_1,\ldots,w_N), k\in[K]$, each of which requires all N files as input. The output function ϕ_k is assigned to node k.[3] In general, as the dataset may be large, node k only has access to a subset of files $\mathcal{M}_k\subseteq\{w_1,\ldots,w_N\}$ and each file is available at r nodes. Since nodes do not have access to all N files such that they cannot directly compute their assigned output function, a MapReduce framework is used which has three phases as follows. While in a typical MapReduce framework, the IV sizes are fixed, in the following, we describe a MapReduce framework that allows varying IV sizes. The proposed FLCD scheme is based on this modified framework.

Map Phase: Nodes compute IVs from locally available files. Each node k uses each locally available file $w_n\in\mathcal{M}_k$ as input to the *map functions*, $\{g_{1,n},\ldots,g_{K,n}\}$, to compute the IVs, $\{v_{1,n},\ldots,v_{K,n}\}$, with possibly different lengths, i.e., $v_{k,n}=g_{k,n}(w_n)$ and $|v_{k,n}|=T_k$ bits. The relative IV sizes, $T_k, k\in[K]$, are based on the choice of the specific map and reduce function designs.

Shuffle Phase: Each node k broadcasts a (coded) message set $\mathcal{X}_k$ on a shared-link, over which $\mathcal{X}_k$ sent by node k can be received by all other nodes without errors. All (coded) messages, $\mathcal{X}_k$, are designed so that each node k can collect every IV $v_{k,n}$ for $n\in[N]$. In general, messages, $\mathcal{X}_k$, may include coded combinations of IVs such that nodes can decode requested IVs using locally computed IVs.

Reduce Phase: Each node k computes the *reduce function*, $h_k(v_{k,1},\ldots,v_{k,N})$, with all IVs, $\{v_{k,1},\ldots,v_{k,N}\}$, as input. The map and reduce functions are designed such that $h_k(v_{k,1},\ldots,v_{k,N})=\phi_k(w_1,\cdots,w_N)$.

Under this framework we define the *computation load*, r, as the mean number of times each file is mapped to the system

$$r\triangleq\frac{1}{N}\sum_{k=1}^{K}|\mathcal{M}_k|, \tag{1}$$

which can be understood as the number of times that each IV is computed in the system. In conventional uncoded MapReduce we find $r=1$ where each file is only mapped once on the computing network. In this paper, for simplicity, we will just consider the case when r is an integer.[4] Next, we will present an example of TeraSort to illustrate the the system model described above and put particular focus on the heterogeneous IV sizes. TeraSort is widely used as a benchmark for MapReduce platform.

Example 1: TeraSort Map and Reduce Function Design: The K computing nodes aim to use TeraSort to sort a large set of integers in the range of $[0,Z)$ in a distributed manner. We design K reduce functions, where node k is assigned reduce function $h_k,\forall k\in[K]$. The reduce functions sort integers of a specific range defined by bounds $z_0,z_1,\ldots,z_K$ in an ascending order with $z_0=0$ and $z_K=Z>0$. Reduce function h_k, sorts integers in the range of $[z_{k-1},z_k)$. Then, map functions are designed to hash the integers of each file into bins defined by the bounds. Map function $g_{k,n}$ returns the IV $v_{k,n}$ which includes the integers of file w_n in the range of $[z_{k-1},z_k)$. After the map phase, the nodes shuffle the corresponding IVs such that each node k collects all integers in range $[z_{k-1},z_k)$ to be sorted with reduce function h_k. After the reduce phase, the integers will be sorted across the computing network. △

Compared to previous works in CDC, we study a more general framework where IVs can have varying sizes that are dictated by the map and reduce function designs. Let the number of bits of IV $v_{k,n}$ be T_k bits, which only depends on the corresponding reduce function, h_k. Then, we define the *communication load*, L, as the number of bits transmitted on the shared-link normalized by the total number of bits from all IVs

$$L\triangleq\frac{\sum_{k=1}^{K}|\mathcal{X}_k|}{N\sum_{k=1}^{K}T_k}, \tag{2}$$

where $|\mathcal{X}_k|$ is the total number of bits from all transmitted messages in $\mathcal{X}_k$.

Definition 1: The optimal communication load or the optimal communication-computation tradeoff is defined as

$$L^*(r)\triangleq\inf\{L:(r,L)\text{ is feasible}\}. \tag{3}$$

◊

Next, we will present an example based on the previous TeraSort example to illustrate the possibility to vary the sizes of IVs in practice.

Example 2: Design Choice of Relative IV Sizes: The design of the map and reduce functions dictates the sizes of the IVs. Continuing Example 1, assuming the N files are the same size and the integers follow a uniform distribution, the size of IV $v_{k,n}$ is $T_k=\frac{z_k-z_{k-1}}{Z}\cdot|w_n|$ with high probability where $|w_n|$ is the size in bits of each file. The bounds, $z_0,z_1,\ldots,z_K$, can be chosen accordingly to have desired varying IV sizes. △

In contrast to the proposed system model that supports varying IV sizes, the state-of-the-art CDC designs typically assume constant IV sizes. We will provide a brief description of these as below.

[3]Each output function can be a collection of many functions. When there are more functions than nodes, we can group the functions into K non-overlapping sets and define these sets as output functions.

[4]The case of non-integer r can be solved by using the similar memory-sharing scheme used in classical coded caching literature [21].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3098593, IEEE Transactions on Cloud Computing

III. State-of-the-Art Achievable Designs with Fixed IV sizes

Currently, there are only two CDC designs whose performance has been demonstrated through empirical evaluations over Amazon EC2 as shown in [4]–[6], [27]. These both assume constant IV sizes. The first is the LMYA design of [4], [27]. Under the system model in [4], the LMYA design achieves the information theoretic optimal communication-computation load tradeoff of

$$L^{\text{LMYA}}(r) = L^*(r) = \frac{1}{r}\left(1 - \frac{r}{K}\right). \tag{4}$$

We will show in this paper that, this tradeoff is only optimal under the specific design framework of [4], which assumes the same IV sizes across the network.

In short, the LMYA design operates by mapping a file at every unique set of r nodes. Then, for the Shuffle phase, every unique node group of size $r+1$ nodes forms a shuffle group. The shuffle group is special in that each node requests IVs from a file that is locally available at every other node in the shuffle group. Each message of $\mathcal{X}_k$ sent from node k is a network coded multicast message that serves r independent requests of the other nodes in the shuffle group. Each of the r nodes in each shuffle group can successfully decode requested IVs from the coded multicast message of $\mathcal{X}_k$. Hence, there is a multiplicative gain of r in terms of communication load using this coded multicasting scheme compared to the conventional unicast approach. Although the promising theoretical performance achieved by the LMYA design, it has a high complexity because it requires $N = \binom{K}{r}$ input files, where each file is mapped to a set of r nodes. Also, the scheme requires $G = \binom{K}{r+1}$ shuffle groups, significantly increasing the overhead in CDC implementations as shown in [4], [27].

The second practically implemented design is the KR design of [5], [6] which has a reduced complexity as it only requires $N = \left(\frac{K}{r}\right)^{r-1}$ files and $G = \left(\frac{K}{r}\right)^{r-1}\left(\frac{K}{r} - 1\right)$ shuffle groups. While files are mapped to node groups of size r nodes, the mapping is not to every unique set of r nodes. This significantly reduces the complexity. Both files and shuffle groups have been reduced exponentially compared to those of the LMYA design. Moreover, the multiplicative gain of coded multicasting is maintained and the communication load is

$$L^{\text{KR}}(r) = \frac{1}{r-1}\left(1 - \frac{r}{K}\right), \tag{5}$$

which is asymptotically optimal as r goes to infinity. However, this scheme only works for homogeneous systems (i.e., the size of all $\mathcal{M}_k$ are the same) and only holds for the limited parameter settings where $m = \frac{K}{r}$ is an integer. This requirement can be very restrictive. For example, if $K = 25$, then the possible choices of r are only $1, 5, 25$, where $r = 1$ (conventional MapReduce system) and $r = 25$ (no needed communication) are not interesting cases. The optimal choice of r might not be achievable with the KR design. Moreover, both the LMYA and KR designs only operate under the assumption of homogeneous IV sizes.

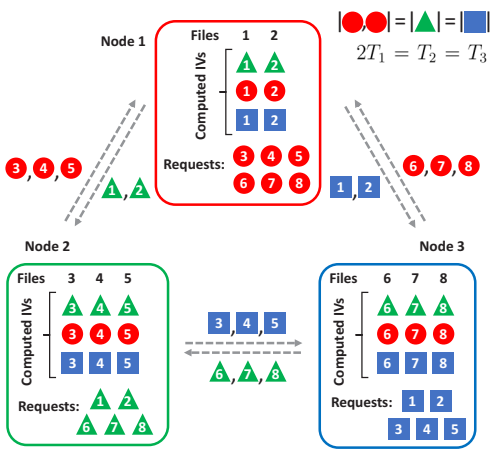

Fig. 1: Uncoded MapReduce with $r = 1$, $N = 8$ files and $K = 3$ nodes from Example 3. Nodes unicast locally computed intermediate values (IVs) to the other nodes over a shared link.

IV. Examples of the Proposed FLCD for $K = 3$

In this section, we present two examples to illustrate the key idea of the proposed FLCD scheme for the special case $K = 3$ nodes. Although the specific designs described here are different from those of the general design for $K > 3$, these examples outline the fundamental concepts of our system model and demonstrate how to design the Shuffle phase when IVs have varying size. Moreover, the examples demonstrate that under our general design framework which allows different IV sizes, the fundamental tradeoff, L^{LMYA} of [4] originally derived for the homogeneous IV sizes no longer holds. Example 3 outlines the conventional uncoded MapReduce approach based on unicast where each input file is mapped at exactly $r = 1$ node. Even for this uncoded case, we show that allowing variable IV sizes results in a lower communication load than that of [4]. In Example 4, FLCD is applied to a network of $K = 3$ nodes and each input file is mapped to $r = 2$ nodes. This example uses coded multicasting, and our design with varying IV sizes again improves on the communication load of [4].

Example 3: Conventional Uncoded MapReduce: As shown in Fig. 1, a network of $K = 3$ nodes aims to compute 3 output functions, one assigned to each node. There are $N = 8$ input files and each is mapped to $r = 1$ computing node. Nodes 1, 2 and 3 map the files of $\mathcal{M}_1 = \{w_1, w_2\}$, $\mathcal{M}_2 = \{w_3, w_4, w_5\}$ and $\mathcal{M}_3 = \{w_6, w_7, w_8\}$, respectively. In the map phase, each node computes 3 IVs, one for each output function, from each of its locally available files. Moreover, the map functions are designed such that $2T_1 = T_2 = T_3$ and IVs for node 1's reduce function (red circles) contain half the number of bits of IVs for node 2 and 3's reduce functions (green triangles and blue squares, respectively).

The shuffle phase is necessary so each computing node can collect the needed (or requested) IVs for its reduce function corresponding to its assigned output function. Nodes 1, 2 and 3 will be required to have the access to all the 8 IVs represented by red circles, green triangles, and blue squares,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3098593, IEEE Transactions on Cloud Computing

respectively, as shown in Fig. 1. In order to accomplish this, each node transmits the required IVs to the other two nodes on the shared-link. For instance, node 1 transmits IVs $v_{2,1}$ and $v_{2,2}$, represented by green triangles numbered 1 and 2, to node 2. Similarly, node 2 transmits IVs $v_{3,3}$, $v_{3,4}$ and $v_{3,5}$, represented by blue squares numbered 3, 4 and 5, to node 3. Finally, after the shuffle phase, each node uses the appropriate IVs as input to reduce functions to compute the desired output. In this example, we see that Node 1 maps 2 files, requests 6 IVs, and the length of each IV is shorter with T_1 bits. Node 2 and 3 each maps 3 files, requests 5 IVs, and the size of each IV is longer with $T_2 = T_3$ bits.

Next, we derive the resulting communication load in this case, denoted by $L^{\text{unicast}}(1)$. Note that, the total number of bits in all IVs is $N(T_1 + T_2 + T_3)$. Also, the number of bits transmitted on the shared-link is $6T_1 + 5T_2 + 5T_3$ as seen in Fig. 1 where 6 IVs of length T_1 (red circles), 5 IVs of length T_2 (green triangles), and 5 IVs of length T_3 (blue squares) are transmitted among the nodes. Hence, given $2T_1 = T_2 = T_3$, we obtain

$$L^{\text{unicast}}(1) = \frac{6T_1 + 5T_2 + 5T_3}{N(T_1 + T_2 + T_3)} = \frac{6T_1 + 10T_1 + 10T_1}{8(T_1 + 2T_1 + 2T_1)} = \frac{13}{20}, \tag{6}$$

which is less than $L^{\text{LMYA}}(1) = \frac{1}{r}\left(1 - \frac{r}{K}\right) = \frac{1}{1}\left(1 - \frac{1}{3}\right) = \frac{2}{3}$ achieved by [4]. △

Example 4: Coded MapReduce with FLCD: We study the same network of Example 3, where $K = 3$ nodes aim to compute 3 output functions from $N = 8$ input files. Here, nodes 1, 2 and 3 will need to collect all IVs represented by the red circles, green triangles and blue squares, respectively, as shown in Fig. 2. Unlike the conventional MapReduce, in FLCD, each file is strategically mapped to $r = 2$ nodes. Specifically, nodes 1, 2 and 3 map the files of $\mathcal{M}_1 = \{w_1, w_2, w_3, w_6\}$, $\mathcal{M}_2 = \{w_1, w_3, w_4, w_5, w_7, w_8\}$ and $\mathcal{M}_3 = \{w_2, w_4, w_5, w_6, w_7, w_8\}$, respectively. Nodes compute IVs from their locally available files. Similar to before, we let IVs have different lengths, i.e., $2T_1 = T_2 = T_3$. In this example, we see that Node 1 maps 4 files, requests 4 IVs, each IV is of a shorter length T_1. Node 2 and 3 each maps 6 files, requests 2 IVs, and each IV is of a longer length T_2.

In the shuffle phase, we look for coded multicasting opportunities where a coded message can serve two independent node requests as shown in Fig. 2. In particular, the IVs $v_{3,1}$ and $v_{3,3}$, represented by blue squares numbered 1 and 3, are available at nodes 1 and 2 and requested by node 3. Similarly, IVs $v_{2,2}$ and $v_{2,6}$, represented by green squares numbered 2 and 6, are available at nodes 1 and 3 and requested by node 2. Hence, node 1 can transmit the coded pair $v_{3,1} \oplus v_{2,2}$ where "$\oplus$" represents the bit-wise XOR operation. Note that $v_{3,1}$ and $v_{2,2}$ have the same size. Nodes 2 and 3 can recover their requested IVs from this coded multicast using their locally computed IVs. The transmitted coded message from node 3 is $(v_{1,7}, v_{1,8}) \oplus v_{2,6}$. Here, "$(,)$" represents the concatenation of two IVs which is necessary since the IVs for output function 1 are half the size of those for output functions 2 and 3. Using

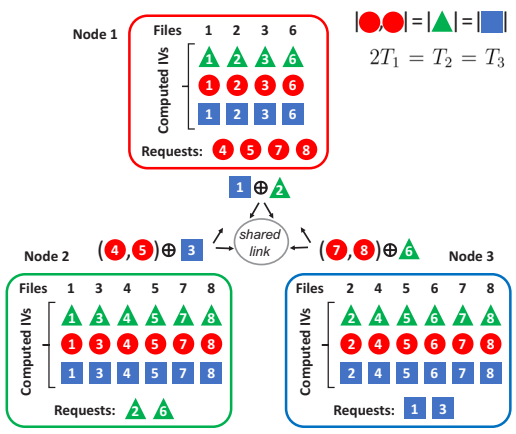

Fig. 2: Coded MapReduce using FLCD with $K = 3$, $r = 2$, and $N = 8$ of Example 4. Coded multicast messages are transmitted from one node to the other two nodes through a shared link.

locally computed IVs node 1 can recover $v_{1,7}$ and $v_{1,8}$ and node 2 can recover $v_{2,6}$. The transmitted coded message from node 2 can be designed similarly to those of node 3.

As shown from Fig. 2, given that $2T_1 = T_2 = T_3$, the coded messages transmitted for all nodes have the same size of $T_2 = T_3$ bits. For instance, the coded message from node 2 to node 1 and 3 is generated by doing the XOR of the concatenation of two IVs of length T_1 (for a total length of $2T_1$ bits) and one IV of length T_3. The resulting coded message has a length of $2T_1 = T_3$ bits. Hence, the communication load is given by

$$L^{\text{FLCD}}(2) = \frac{KT_2}{N(T_1 + T_2 + T_3)} = \frac{3 \cdot 2T_1}{8(T_1 + 2T_1 + 2T_1)} = \frac{3}{20}, \tag{7}$$

which is significantly less than that of that of Example 4 where $L^{\text{unicast}}(1) = \frac{13}{20}$ due to the use of coded multicasting. It also improves upon the fundamental communication-computation trad-eoff $L^{\text{LMYA}}(2) = \frac{1}{6}$, calculated from (4). △

Remark 1: Interestingly, for Examples 3 and 4, the proposed FLCD schemes achieve a lower communication load, L, than those of the LMYA design, even though the latter was proven to be optimal given K and r in [4]. The reason is that the optimality proof of [4] enforces an assumption that the size of each IV is the same. Therefore, the derived lower bound on achievable communication load does not apply to FLCD. We demonstrate here that relaxing this requirement allows for designs with a better communication load. This is because we can reduce the size of IVs that are transmitted more frequently over the network. For instance, in both Examples 3 and 4, there are more transmitted IVs requested by node 1 (red circles), than the IVs requested by node 2 (green triangles) and node 3 (blue squares). Therefore, the size of IVs requested by node 1 are reduced relative to the other IVs. Furthermore, the number of IVs requested by a node is proportional to the difference between the total number of files and the number of locally

mapped files. In other words, by FLCD, the nodes that map less files request IVs of smaller sizes.

V. Achievable communication load and complexity of FLCD

In this section, we will summarize the achievable communication load of the proposed FLCD and provide a theoretical complexity comparison against other state-of-the-art designs. Detailed descriptions of the FLCD schemes and empirical evaluations are deferred until Section VI and Section VII, respectively. Next, we will first discuss results of the FLCD scheme for the special case of $K = 3$, and then discuss results for the general FLCD scheme of $K > 3$.[5]

A. Results of FLCD Scheme for $K = 3$

When $K = 3$, the only non-trivial case is $r = 2$.[6] In this case, for FLCD each multicast from any node serve $r = 2$ independent node requests. This case allows for arbitrary IV sizes and shows the fundamental tradeoff of [4] does not apply under the more general design framework with heterogeneous IV sizes.

Proposition 1: When $K = 3$ and $r = 2$, for general IV sizes $T_1, T_2, T_3 > 0$, the communication load of FLCD is

$$L^{\mathrm{FLCD}}(2) = \frac{3T_1T_2T_3}{2(T_1T_2 + T_1T_3 + T_2T_3)(T_1 + T_2 + T_3)}, \quad (8)$$

where $T_k, k \in [3]$ are the sizes of the IVs for function k. The required number of input files is $\mathrm{LCM}(T_1, T_2, T_3) \times \left(\frac{1}{T_1} + \frac{1}{T_2} + \frac{1}{T_3}\right)$ and the required number of shuffle groups is 1.

Proof: Proposition 1 is proved in Appendix A. ∎

Remark 2: When $K = 3$, $r = 2$ and $T_1 = T_2 = T_3$, FLCD is equivalent to the LMYA design and $L^{\mathrm{FLCD}}(2) = \frac{1}{6} = L^{\mathrm{LMYA}}(2)$. When T_1, T_2 and T_3 are not equal, we have the following corollary.

Corollary 1: When $K = 3$, $r = 2$ and T_1, T_2 and T_3 are not equal,

$$L^{\mathrm{FLCD}}(2) < \frac{1}{6} = L^{\mathrm{LMYA}}(2). \quad (9)$$

Proof: Corollary 1 is proved in Appendix B. ∎

From Corollary 1, it can be seen that in this case, the fundamental limit of [4] is no longer optimal when we allow different IV sizes.

B. Results of General FLCD Scheme of $K > 3$

When $K > 3$, FLCD achieves a multiplicative communication-computation load tradeoff where each multicast serves $r - 1$ nodes. By the design of specific relative IV sizes, FLCD for $K > 3$ is flexible in that it operates for any integer r such that $2 \le r \le \frac{K}{2}$. The performance of FLCD in terms of the communication-computation load tradeoff and the required number of input files and shuffle groups is presented in Theorem 1 in the following.

Theorem 1: When $K > 3$ and $2 \le r \le \frac{K}{2}$, let $m \triangleq \frac{K}{r}$ and $\hat{m} \triangleq \lfloor m \rfloor + 1$, the communication load of FLCD is

$$L^{\mathrm{FLCD}}(r) = \frac{1}{r-1}\left(\frac{\lfloor m \rfloor^2 - \lfloor m \rfloor}{\lfloor m \rfloor \hat{m} - m}\right), \quad (10)$$

and the required number of input files and shuffle groups is $N = G = \lfloor m \rfloor^{(\hat{m}r-K)} \times \hat{m}^{(K-\lfloor m \rfloor r)}$ and IV sizes are either equal to T_1' or T_2' where $\lfloor m \rfloor T_1' = (\lfloor m \rfloor - 1)T_2'$.

Proof: Theorem 1 is proved in Appendx C. ∎

Remark 3: For $K > 3$, when $\frac{K}{r} = m$ is an integer, we find $L^{\mathrm{FLCD}}(r) = L^{\mathrm{KR}}(r) = \frac{1}{r-1}\left(1 - \frac{r}{K}\right)$, i.e., the FLCD and the KR designs have the same communication-computation load tradeoff. When m is not an integer, the FLCD can still operate as shown in Section VI, but the KR scheme is no longer feasible. Note that while the KR scheme can be used in conjunction with a memory sharing approach to operate on a non-integer m, this will result in a communication load that is greater than the original $L^{\mathrm{KR}}(r)$ given in (5). In contrast, the proposed FLCD is directly designed to operate on a non-integer m without the need for memory sharing. Surprisingly, for an non-integer m, we find that when allowing varying IV sizes, the communication load of FLCD is less than that of a system with constant IV sizes. This is described in the following corollary.

Corollary 2: When $K > 3$ and $m = \frac{K}{r}$ is not an integer, then

$$L^{\mathrm{FLCD}}(r) < \frac{1}{r-1}\left(1 - \frac{r}{K}\right), \quad (11)$$

where $r \ge 2$ and $r \in \mathbb{Z}^+$.

Proof: Corollary 2 is proved in Appendix D. ∎

Remark 4: It can be seen from Section VI-B that although the proposed FLCD scheme allows flexible IV lengths, the designed IV lengths and computation loads at each node to achieve (10) will be approximately the same as m becomes large. This result is summarized in the following corollary.

Corollary 3: Assume $K > 3$ and $r \ge 2$. Let $m = \frac{K}{r}$. Then, we have the following:

(i) Asymptotically equal IV sizes: There exists some $T > 0$ such that

$$\lim_{m\to\infty} T_k = T, \quad \forall k \in [K]. \quad (12)$$

(ii) Asymptotically equal number of files mapped at each node:

$$\lim_{m\to\infty} |\mathcal{M}_k| = \frac{Nr}{K}, \quad \forall k \in [K]. \quad (13)$$

Proof: Corollary 3 can be directly obtained in Section VI-B. ∎

From (12) of Corollary 3, we see that when $\frac{K}{r} = m$ is large, the IV sizes of different nodes in the network are approximately equal. In this case, the constraint of equal IV size commonly used in other designs is relaxed only slightly. Hence, an important consequence of Corollary 3 is that for small variation in IV size, FLCD can fit a much wider range of parameters since FLCD operates for any integer $r \le \frac{K}{2}$. This is opposed to the previous low complexity design [6] which only operates for integer $\frac{K}{r}$.

[5]The case of $K \le 2$ is straightforward. Hence, we do not consider this case.

[6]When $K = 3$, FLCD is designed only for $r = 2$ since the case of $r = 1$ is conventional uncoded MapReduce. When $r = 3$, each node maps the entire library and strategic map and shuffle designs are unnecessary.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3098593, IEEE Transactions on Cloud Computing

TABLE I: Flexibility and Complexity of Achievable CDC Designs

			LMYA Design [4]			KR Design [6]			FLCD		
K	r	m	L	N	G	L	N	G	L	N	G
16	3	5.33	0.27	560	1820	–	–	–	0.41	150	150
16	4	4	0.19	1820	4368	0.25	64	192	0.25	256	256
16	5	3.2	0.14	4368	8008	–	–	–	0.17	324	324
22	3	4.33	0.29	1540	7315	–	–	–	0.43	392	392
22	4	5.5	0.20	7315	26334	–	–	–	0.27	900	900
22	5	4.4	0.15	26334	74613	–	–	–	0.19	1600	1600
25	3	8.33	0.29	2300	12650	–	–	–	0.44	576	576
25	4	6.25	0.21	12650	53130	–	–	–	0.28	1512	1512
25	5	5	0.16	53130	177100	0.2	625	2500	0.2	3125	3125

C. Comparison to State-of-the-Art CDC Designs

In Table I, we list the key parameters L (communication load), N(number of files) and G (number of shuffle groups) of different CDC designs considered in this paper. First, as discussed before, the LMYA design [4] has a relatively high complexity. For example, it requires $N > 10^4$ and $G > 10^5$ for $K = 25$ and $r = 5$. As shown in the empirical evaluations (see Section VII), a large G greatly negates the promised gain of CDC. Second, the KR design [6] has the least complexity in terms of N and G, but is rather limited to network parameters with integer m. Third, the proposed FLCD scheme can operate on all (K, r) pairs of Table I and has over $10\times$ reduction in G compared to LMYA [4]. Empirical evaluations of these designs on Amazon EC2 (see Section VII) will confirm that FLCD outperforms both of the LMYA and KR designs in MapReduce total execution times.[7]

Next, in Section VI, we will introduce the general FLCD schemes that achieve (8) and (10), respectively.

VI. Description of the General FLCD Scheme

In this section, we will first present the general design of FLCD when $K = 3$ and $r = 2$ and then we will introduce the general design of FLCD for $K > 3$.

A. General FLCD Scheme for $K = 3$ and $r = 2$

Given arbitrary IV sizes $T_1, T_2, T_3 > 0$, we first present the general design of FLCD when $K = 3$. We first split the files into three non-overlapping sets $\mathcal{M}_{\{1,2\}}$, $\mathcal{M}_{\{1,3\}}$ and $\mathcal{M}_{\{2,3\}}$. The files of $\mathcal{M}_{\{1,2\}}$ are mapped at nodes 1 and 2, $\mathcal{M}_{\{1,3\}}$ are mapped at nodes 1 and 3 and $\mathcal{M}_{\{2,3\}}$ are mapped at nodes 2 and 3. Also, given the IV sizes, T_1, T_2 and T_3 bits, the file sets are defined such that

$$|\mathcal{M}_{\{1,2\}}|T_3 = |\mathcal{M}_{\{1,3\}}|T_2 = |\mathcal{M}_{\{2,3\}}|T_1, \tag{14}$$

where $|\mathcal{M}_{\{i,j\}}|$ is the number of files in $\mathcal{M}_{\{i,j\}}$. Then, we define $\mathcal{V}^3_{\{1,2\}}$ as the set of IVs for node 3's output function from the files of $\mathcal{M}_{\{1,2\}}$. The IV sets $\mathcal{V}^2_{\{1,3\}}$ and $\mathcal{V}^1_{\{2,3\}}$ are defined similarly. Each IV set $\mathcal{V}^k_{\{i,j\}}$ is split into two equal size sets $\mathcal{V}^{k,i}_{\{i,j\}}$ and $\mathcal{V}^{k,j}_{\{i,j\}}$ to be transmitted in a coded message from node i and j, respectively. In the shuffle phase, node 1 transmits $\mathcal{V}^{3,1}_{\{1,2\}} \oplus \mathcal{V}^{2,1}_{\{1,3\}}$, node 2 transmits $\mathcal{V}^{3,2}_{\{1,2\}} \oplus \mathcal{V}^{1,2}_{\{2,3\}}$ and node 3 transmits $\mathcal{V}^{2,3}_{\{1,3\}} \oplus \mathcal{V}^{1,3}_{\{2,3\}}$. Due to (14), we can see for each coded transmission, the message sets being XOR'd together have the same length in bits.

Each coded transmission successfully serves independent requests of two nodes simultaneously via the shared-link such that the receiving nodes use locally computed IVs to resolve the requested IVs from this coded transmission. For example, node 2 receives $\mathcal{V}^{3,1}_{\{1,2\}} \oplus \mathcal{V}^{2,1}_{\{1,3\}}$ from node 1. Since node 2 has already computed $\mathcal{V}^{3,1}_{\{1,2\}}$ locally, it can XOR $\mathcal{V}^{3,1}_{\{1,2\}}$ with the received message, $\mathcal{V}^{3,1}_{\{1,2\}} \oplus \mathcal{V}^{2,1}_{\{1,3\}}$, to recover $\mathcal{V}^{2,1}_{\{1,3\}}$. From the Shuffle phase, each node receives and decodes all needed IVs from files that are not locally available. For example, node 2 can resolve the IV sets $\mathcal{V}^{2,1}_{\{1,3\}}$ and $\mathcal{V}^{2,3}_{\{1,3\}}$ which collectively contain all IVs from the files $\mathcal{M}_{\{1,3\}}$ that are not available to node 2 but available at nodes 1 and 3. Hence, we can conclude the correctness of FLCD for $K = 3$ and $r = 2$.

The communication load of FLCD for $K = 3$ is shown in (8) and its proof is can be found in Appendix A.

B. General FLCD scheme for $K > 3$

Next, we present the proposed FLCD design for the general case of $K > 3$. It comprises of a strategic file mapping and shuffle design, which centers around supporting varying IV sizes, such that each node can compute its assigned output function in the reduce phase and the total number of requested IVs bits are kept the same for different nodes. Compared to prior work of [25], [26], which focus on CDC networks with heterogeneous function assignments, the FLCD proposed here takes a different approach to explore heterogeneous IV sizes instead of function assignments, assuming that only one reduced function is assigned to each node. This approach leads to a new class of asymptotic homogeneous CDC design proposed here that are amenable for practical implementations due to the reduced packetization.

Assume $2 \leq r \leq \frac{K}{2}$ where r and K are positive integers. Let $m = \frac{K}{r}$ and $\hat{m} = \lfloor m \rfloor + 1$. We split all the computing nodes into two non-overlapping sets $\mathcal{K}_1$ and $\mathcal{K}_2$. Each node in $\mathcal{K}_1$ maps a $\frac{1}{\hat{m}}$ fraction of the entire dataset and each node in $\mathcal{K}_2$ maps a $\frac{1}{\lfloor m \rfloor}$ fraction of the entire dataset. Moreover, $\mathcal{K}_1$ and $\mathcal{K}_2$ contain $K_1 = \hat{m}K - \lfloor m \rfloor \hat{m} r$ and $K_2 = \lfloor m \rfloor \hat{m} r - \lfloor m \rfloor K$ nodes, respectively. Note that $K_1 + K_2 = K$. The number of times that the nodes in $\mathcal{K}_1$ collectively map the file library is $r_1 = K - \lfloor m \rfloor r$. Similarly, the nodes in $\mathcal{K}_2$

[7]Note that, in Table I, the LMYA scheme has a lower communication load L than FLCD. While FLCD can have a lower L for $K = 3$, it is not always the case for $K > 3$. This stems from the fact that the upper bound of L is reduced by a factor of $r - 1$ for FLCD, but a factor of r for LMYA.

collectively map the entire dataset $r_2 = \hat{m}r - K$ times. Note that $r_1 + r_2 = r$. We further split $\mathcal{K}_1$ and $\mathcal{K}_2$ into r_1 and r_2, respectively, equally sized non-overlapping sets $\mathcal{K}_1^1, \ldots, \mathcal{K}_1^{r_1}$ and $\mathcal{K}_2^1, \ldots, \mathcal{K}_2^{r_2}$, where $|K_1^i| = \frac{K_1}{r_1} = \hat{m}$ and $|K_2^i| = \frac{K_2}{r_2} = \lfloor m \rfloor$. Nodes in each set $\mathcal{K}_\ell^i$ collectively map the file library exactly once. Moreover, we design the map functions such that $v_{k,n}$ is of size T_1' bits if $k \in \mathcal{K}_1$ and size T_2' bits if $k \in \mathcal{K}_2$ where $\lfloor m \rfloor T_1' = (\lfloor m \rfloor - 1)T_2'$. This design choice ensures each node requests the same number of bits of IVs from each shuffle group. When m is large (e.g., K is large and r is fixed), it can be shown that all IVs will have approximately the same size, i.e,

$$\lim_{m\to\infty} \frac{T_1'}{T_2'} = \frac{\lfloor m \rfloor - 1}{\lfloor m \rfloor} = 1, \tag{15}$$

and the number of files mapped to each node are approximately the same

$$\lim_{m\to\infty} \frac{|\mathcal{M}_{k_1}|}{|\mathcal{M}_{k_2}|} = \frac{N}{\lfloor m \rfloor + 1} \cdot \frac{\lfloor m \rfloor}{N} = 1, \tag{16}$$

for any nodes $k_1 \in \mathcal{K}_1$ and $k_2 \in \mathcal{K}_2$. This proves Corollary 3.

Map Phase: We split the dataset into $N = \hat{m}^{r_1} \times \lfloor m \rfloor^{r_2}$ files and define N groups, $\mathcal{S}_1, \ldots, \mathcal{S}_N$. Each such group is called a placement group. The placement groups $\mathcal{S}_n, n \in [N]$ consist of all possible sets with cardinality of r nodes such that each set contains exactly one node from every node set $\mathcal{K}_1^1, \ldots, \mathcal{K}_1^{r_1}, \mathcal{K}_2^1, \ldots, \mathcal{K}_2^{r_2}$. Each file, $w_n, n \in [N]$ is then placed into every node in $\mathcal{S}_n$. In this way, the library is mapped exactly r times and each node in $\mathcal{S}_n$ maps file w_n to compute the corresponding IVs $v_{1,n}, \ldots, v_{K,n}$.

Shuffle Phase: In FLCD, each placement group $\mathcal{S}_n$ also forms a shuffle group. The nodes in $\mathcal{S}_n$ shuffle IVs requested by one node and locally computed by the other $r-1$ nodes in $\mathcal{S}_n$. We define the set of IVs, $\mathcal{V}_n^k$, to be those requested by node k and locally computed by the other nodes in $\mathcal{S}_n$. Then we split $\mathcal{V}_n^k$ into $r-1$ equal size subsets $\mathcal{V}_n^{k,j}$, where $j \in \mathcal{S}_n \setminus k$ and node j is responsible for transmitting the IVs of $\mathcal{V}_n^{k,j}$. Specifically, each node $j \in \mathcal{S}_n$ broadcasts the coded message $\bigoplus_{k \in \mathcal{S}_n \setminus j} \mathcal{V}_n^{k,j}$ to the rest of nodes in $\mathcal{S}_n$. It can be seen that due to the requirement that $\lfloor m \rfloor T_1' = (\lfloor m \rfloor - 1)T_2'$, the transmitted messages from node j, $\mathcal{V}_n^{k,j}$, $k \in \mathcal{S}_n \setminus j$, have the same length in bits.

The communication load of FLCD for $K > 3$ is shown in (10). The correctness of FLCD for $K > 3$ and the proof of (10) can be found in Appendix C.

Remark 5: When $K > 3$, the size of the IVs in FLCD are exactly the same for integer m. In this case, we have $K_1 = 0$, $K_2 = K$. This means that all the computing nodes are in $\mathcal{K}_2$ and each maps a $\frac{1}{m}$ fraction of the file library. Hence, in this case, all the IVs are of size T_2' bits.

C. An Example of FLCD for $K > 3$

Example 5: Our goal is to use the FLCD on a network of $K = 18$ computing nodes with a computation load of $r = 4$. We find $\frac{K}{r} = m = \frac{9}{2}$ is not an integer and the KR design cannot be used. However, by allowing varying IV sizes in the network we can use FLCD. Define $\hat{m} = \lfloor m \rfloor + 1 = 5$, we split the nodes into two sets $\mathcal{K}_1$ and $\mathcal{K}_2$ of size $K_1 = \hat{m}K - \lfloor m \rfloor \hat{m}r = 10$ and $K_2 = \lfloor m \rfloor \hat{m}r - \lfloor m \rfloor K = 8$ nodes, respectively. In particular, the 10 nodes of $\mathcal{K}_1$ will each map $\frac{1}{\hat{m}} = \frac{1}{5}$ of the files and the 8 nodes of $\mathcal{K}_2$ will each map $\frac{1}{\lfloor m \rfloor} = \frac{1}{4}$ of the files. These node sets are each split into 2 equally sized disjoint subsets such that $\mathcal{K}_1 = \mathcal{K}_1^1 \cup \mathcal{K}_1^2$ and $\mathcal{K}_2 = \mathcal{K}_2^1 \cup \mathcal{K}_2^2$. The file library is split into $N = 5^2 \cdot 4^2 = 400$ equally sized files where a file is mapped at a set of $r = 4$ nodes, $\mathcal{S}_n$ with one node from each set of $\{\mathcal{K}_1^1, \mathcal{K}_1^2, \mathcal{K}_2^1, \mathcal{K}_2^2\}$. As an example, let $\mathcal{S}_1 = \{1, 2, 3, 4\}$ where nodes 1, 2, 3 and 4 belong to the sets $\mathcal{K}_1^1$, $\mathcal{K}_1^2$, $\mathcal{K}_2^1$, and $\mathcal{K}_2^2$, respectively. A file is mapped to these nodes that is not mapped to any other of the 14 nodes.

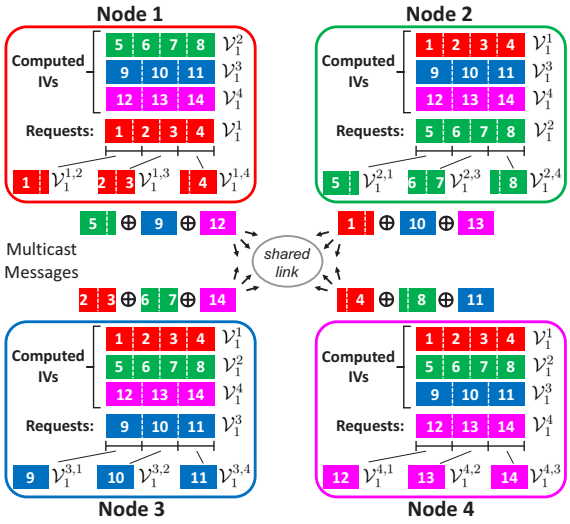

Fig. 3: Illustration of the data shuffle within a specific shuffle group $\mathcal{S}_1$ of the FLCD scheme for a CDC network with $K = 18$ and $r = 4$ of Example 5. Given the IVs computed from locally available files, each node can recover its requested IVs from the coded transmissions.

Each set $\mathcal{S}_n, n \in [N]$ also represents a shuffle group. Fig. 3 shows the IVs requested and transmitted by the nodes of $\mathcal{S}_1 = \{1, 2, 3, 4\}$. Each node requests IVs that are locally computed at the other nodes, presenting multicast opportunities. For example, the nodes of $\mathcal{S}_1 \setminus \{1\} = \{2, 3, 4\}$ also form placement groups with the 4 nodes of $\mathcal{K}_1^1 \setminus \{1\}$. Therefore, there are 4 files available to the nodes of $\{2, 3, 4\}$, but not node 1. Without loss of generality, let these files be w_1, w_2, w_3 and w_4, then node 1 requests the IVs of $\mathcal{V}_1^1 = \{v_{1,1}, v_{1,2}, v_{1,3}, v_{1,4}\}$ from the shuffle group $\mathcal{S}_1$. Similarly, node 2 requests the 4 IVs of $\mathcal{V}_1^2 = \{v_{2,5}, v_{2,6}, v_{2,7}, v_{2,8}\}$. Then, we see nodes 3 and 4 each request 3 IVs from $\mathcal{S}_1$ because the nodes of $\{1, 2, 4\}$ and $\{1, 2, 3\}$ form placement groups with the 3 nodes of $\mathcal{K}_2^1 \setminus \{3\}$ and $\mathcal{K}_2^2 \setminus \{4\}$, respectively. Again, without loss of generality, node 3 requests the IVs of $\mathcal{V}_1^3 = \{v_{3,9}, v_{3,10}, v_{3,11}\}$ and node 4 requests the IVs of $\mathcal{V}_1^4 = \{v_{4,12}, v_{4,13}, v_{4,14}\}$ from $\mathcal{S}_1$. Since the IVs requested by nodes 1 and 2 are T_1' bits each and the IVs requested by nodes 3 and 4 are T_2' bits each, we see each node requests the same number of bits from this shuffle group $\mathcal{S}_1$ since $\lfloor m \rfloor T_1' = 4T_1' = 3T_2' = (\lfloor m \rfloor - 1)T_2'$.

Fig. 3 depicts the IVs of $\mathcal{V}_1^1$ (red squares), $\mathcal{V}_1^2$ (green

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3098593, IEEE Transactions on Cloud Computing

squares), $\mathcal{V}_1^3$ (blue rectangles), and $\mathcal{V}_1^4$ (magenta rectangles). In particular, the width of the IVs reflect their relative size. In practice, the IVs requested by a particular node will be concatenated as shown in Fig. 3 where the IVs are lined up side-by-side. We visualize that each node requests the same amount because the width of the concatenated messages are the same. Then, each concatenated IV set is split into $r-1=3$ messages to be transmitted by 3 different nodes. For example, $\mathcal{V}_1^1$ is split into $\mathcal{V}_1^{1,2}$, $\mathcal{V}_1^{1,3}$, and $\mathcal{V}_1^{1,4}$ to be transmitted by nodes 2, 3 and 4, respectively. Note that, in practice, $\mathcal{V}_1^{1,i}$, $i \in \{2,3,4\}$ each contain fractions of IVs and not necessarily whole IVs in order to split $\mathcal{V}_1^1$ into 3 equal size subsets.

Each node $i \in \mathcal{S}_1 = \{1,2,3,4\}$ transmits $\bigoplus_{j \neq i} \mathcal{V}_1^{j,i}$ to the other nodes of $\mathcal{S}_1$. For example, node 1 transmits the coded combination of $\mathcal{V}_1^{2,1}$ (green rectangle that includes $v_{2,5}$ and a fraction of $v_{2,6}$), $\mathcal{V}_1^{3,1} = \{v_{3,9}\}$ (blue rectangle with the number 9) and $\mathcal{V}_1^{4,1}$ (magenta rectangle with the number 12). The size of the transmission from each node is $\frac{4}{3}T_1' = T_2'$ bits. Accounting for each shuffle group $\mathcal{S}_n$, $n \in [N]$, the communication load is $L^{\text{FLCD}} = \frac{400 \cdot 4 \cdot T_2'}{N(|\mathcal{K}_1| \cdot T_1' + |\mathcal{K}_2| \cdot T_2')} \approx 0.2581$ where we normalize by the total bits of all IVs which is $N(|\mathcal{K}_1| \cdot T_1' + |\mathcal{K}_2| \cdot T_2')$ bits.

Algorithm 1 CDC: Master Node Processing Flow

Input: r, K, CDC design
1: Record timestamp % CodeGen start
2: Define dataset file partitions, file mappings and function assignments based on r, K and CDC design
3: Send dataset partitions and file assignments to all worker nodes
4: Wait for worker nodes to finish establishing MPI-communicators for shuffles groups
5: Record timestamp % CodeGen end, Map start
6: Wait for all worker nodes to load and map assigned files
7: Record timestamp % Mep end, Encode start
8: Wait for all worker nodes to encode IVs
9: Record timestamp % Encode end, Shuffle start
10: Wait for all worker nodes to shuffle encoding IVs
11: Record timestamp % Shuffle end, Decode start
12: Wait for all worker nodes decode received IVs
13: Record timestamp % Decode end, Reduce start
14: Wait for all worker nodes to sort locally collected IVs for assigned bin
15: Record timestamp % Reduce end, Execution complete
Output: timestamps

VII. Empirical Evaluation on Amazon EC2

A. Experiment Setup

In order to evaluate the effectiveness of the proposed FLCD approach, we perform a TeraSort algorithm [28] on Amazon EC2 with $K = 16, 22, 25$ worker nodes and an additional master node. Each computing node is a `t2.large` EC2 instance with 2 vCPUs, 8 GiB of RAM and 24 GB of solid-state drive (SSD) storage. We developed Python software to implement a TeraSort algorithm using the proposed FLCD, LMYA [21], KR [6], and the conventional uncoded design. Nodes sort 12 GB of data comprised of 6×10^8 key-value pairs (KVs) in total. Each key is a 16-bit unsigned integer (`uint16`) and each value a length-9 array of 16-bit unsigned integers. Each node is assigned an output function to sort KVs with keys in a specific range. We design the map and reduce functions using the method outlined in Examples 1 and 2 so that the length of the IVs satisfy the requirement of FLCD and $\lfloor m \rfloor T_1' = (\lfloor m \rfloor - 1)T_2'$ for non-integer m. as well as for the homogeneous requirements of the LMYA and KR designs. The map functions hash the KVs by placing KVs in bins based on their keys. The bins correspond to the specific range of keys each node is assigned to sort. We use the open Message Passing Interface (MPI) library to facilitate the inter-node communications. To prevent bursty communication rates, the incoming and outgoing traffic rate of each computing node is limited to 100 Megabits per second (Mbps) using the Linux `tc` command. The execution is split into 6 steps described as follows.

1) *CodeGen*: The worker nodes define placement and shuffle groups and reduce function assignments. The placement groups define partitions of the data and the set of KVs that each node will map based on the specific CDC design. The shuffle groups are defined using the MPI `Create` function to create a new MPI-communicator and facilitate the shuffle phase.
2) *Map*: The worker nodes load data from the solid-state drive (SSD) and use map functions to hash KVs into bins defined by the reduce functions, or the range of values the nodes are responsible for sorting.
3) *Encode*: Based on the CDC design, the worker nodes form the coded messages of IVs that will be used for the multicast transmissions. The IVs are combined using bit-wise XOR and concatenation operations. Note that this step does not apply to the corresponding uncoded design.
4) *Shuffle*: Nodes sequentially transmit the (coded) messages to the other nodes in the same shuffle groups based on the shuffle design of the specific CDC design. For data transmission, the coded designs use the MPI `bcast` function and the uncoded design uses the MPI `scatter` function.
5) *Decode*: Using the received and locally computed coded messages, the nodes resolve the necessary IVs for their assigned reduce functions. Note that this step does not apply to the uncoded design.
6) *Reduce*: The nodes execute their assigned reduce functions to sort the IVs within their corresponding assigned range. In this way, the data set is sorted across the computing network.

We provide the developed Python code for this evaluation on the Github page https://github.com/C3atUofU/Coded-Distributed-Computing-over-AWS.

Algorithm 2 CDC: Worker Node Processing Flow

Input: r, K, rank, CDC design
1: Receive dataset partitions, file mappings and function assignments from master node
2: **for** each participating shuffle group **do**
3: Create MPI-communicator for shuffle group
4: **end for**
5: **for** each assigned file **do**
6: Load file into local memory from storage
7: Hash all elements of files to appropriate bins (compute IVs)
8: **end for**
9: **for** each participating shuffle group **do**
10: **for** node k in shuffle group **do**
11: **if** $k ==$ rank **then**
12: Split and encode IVs into message that serves shuffle group
13: **else**
14: Split and encode IVs into message used for decoding
15: **end if**
16: **end for**
17: **end for**
18: **for** node $k \in [K]$ **do**
19: **if** $k ==$ rank **then**
20: **for** each participating shuffle group **do**
21: broadcast encoded messages to MPI-communicator of associated shuffle group
22: **end for**
23: **else**
24: receive messages from transmitting worker node
25: **end if**
26: **end for**
27: Use decoding and received messages to resolve requested IVs
28: Sort elements from assigned bin
Output: Sorted data subset

B. Processing Flow

The processing flow for our empirical evaluations is outlined in Algorithm 1 and 2 for the master node and worker nodes, respectively. For the experiment, the master node comes up with explicit file mapping and function assignments based on r, K and the CDC design being used and transmits this information to worker nodes. Then, the master node keeps timing for each of the 6 steps in the execution by monitoring the progress of worker nodes and recording the time when workers nodes complete a step.

In Algorithm 2, the rank indicates the specific numerical label of the worker node. Lines 1 through 4 outline the CodeGen step. Worker nodes receive file mapping and function assignments from the master node. From this, worker nodes form shuffle groups in the network and create MPI-communicator for each shuffle group they participate in. Then, lines 5 through 8 outline the map phase where nodes load files and hash elements from the loaded files creating the IVs. Note that, the bins used for hashing are defined by the reduce function design, or the range of keys each worker node is responsible for sorting in the reduce phase. Lines 9 through 17 outline the encoding where each worker node cycles through the shuffle groups that it is a part of and encodes IVs to make messages for decoding and broadcasting to other nodes of shuffle group. Lines 18 through 26 outline the shuffle phase where nodes sequentially broadcast messages over the network. Lines 27 and 28 outline the decoding step and reduce phase, respectively. After Algorithm 2 is done, the dataset is sorted across worker nodes in the network.

C. Results

Evaluation results are shown in Fig. 4 and Table II ($K = 16$), where shuffle times for different K are shown in Fig. 4(a) to Fig. 4(c) and total times are shown in Fig. 4(d) to Fig. 4(f). In addition, the "Speedup" column in Table II refers to the factor speed-up compared to conventional uncoded MapReduce. The following observations are made based on these results.

- For most points in Fig. 4(a) to Fig. 4(c), the shuffle time decreases proportionally to r and almost coincide with the theoretical results (10). This is the first time that theoretical predictions of the shuffle time of a CDC design are validated by empirical evaluations for a large range of r.[8] There are a few points in (b) and (c) where the shuffle times lie above (10), possibly due to the underlying topology of EC2 and the MPI protocol. For instance, the efficiency and overhead of the multicast changes depending on the number of nodes in the multicast group, whereas this is assumed to be constant in calculating the theoretic prediction (10).
- In Fig. 4(d) to Fig. 4(f), for most of the points, total time decreases significantly with increasing r despite the time of Map Phase increasing greatly as r grows due to increased computations at each node. This demonstrates the multiplicative gain of CDC holds even for the total time.
- The proposed FLCD scheme outperforms LMYA when comparing total time. From Fig. 4(d) to Fig. 4(f), for each value of K, with the choice of r that minimizes the total time, FLCD has a total time $12\% \sim 52\%$ lower than LMYA.
- While the FLCD and KR have similar shuffle and total time, the FLCD has greater flexibility. Table II shows that when $K = 16$, the scenario of $r = 5$ cannot be achieved by the KR scheme, and the gain in terms of the total time of FLCD is 19% compared to KR scheme ($r = 4$). This observation is important because in practical networks may be storage limited and $r = 5$ may be an upper limit for example. Note that for the case of $K = 25$ and $r = 5$, the KR scheme has the lowest total time, possibly because the KR scheme requires a much smaller N than that of the FLCD for this setting (see Table 1) and a smaller CodeGen time and Map time (see Table IV).

[8]The communication rate in the experiment is set to be 100 Mbps. The theoretical curve is computed using the total traffic load divided by 100 Mbps. Therefore, the bandwidth occupation is nearly 100%

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3098593, IEEE Transactions on Cloud Computing

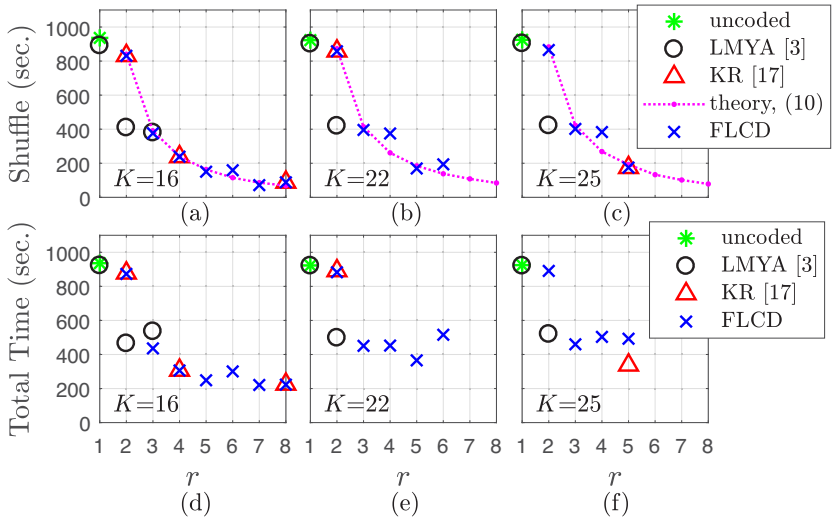

Fig. 4: Empirical evaluations of the proposed FLCD, LMYA [4], KR [6] on Amazon EC2 for implementing the TeraSort Algorithm using $K = 16, 22, 25$ computing nodes. In the first row, (a)-(c) show shuffle time versus computation load r for the three schemes and the theoretical prediction of shuffle time from (10). In the second row, (c)-(d) show total time versus r.

- From Table II, the FLCD ($r = 5$) has a 47% reduction in total time compared to the LMYA scheme with $r = 2$. Due to the high complexity of the LMYA, the maximum implementable r is limited to 3.
- Table II shows a $2.15 \sim 4.24\times$ speed-up of the FLCD design compared to the conventional uncoded MapReduce approach.

Additional evaluation results are also provided in Tables III and IV, which include a detailed break down of the times of each step for $K = 22, 25$ worker nodes similar to the case for $K = 16$. These evaluations show similar behavior of all the schemes considered in this paper and demonstrate the significant advantage of the proposed FLCD. For example, in Table III, we see that the KR and LMYA scheme are only feasible for $r = 2$ and $r = 1$, respectively, but the FLCD scheme allows for up to $r = 6$. In addition, we observe from Tables III and IV a clear trend that the IV ratio approaches 1 as $\frac{K}{r}$ increases (or equivalently r decreases). This confirms that the proposed design leads to asymptotic homogeneous systems for which the reduced communication load and implementation complexity are achieved with only small variations in IV sizes.

VIII. Discussion

This work and others [4], [6] have demonstrated a significant speed-up for the distributed computing application of TeraSort. The proposed FLCD is specifically designed for the TeraSort application. The application of FLCD to other computing problems is unknown. Hence, it will be interesting to consider other applications that can benefit from a similar speed-up using FLCD. The target applications will need to have a communication bottleneck in the Shuffle phase. CDC works by trading computation load for communication load. If there is a computation bottleneck (i.e. Map computations take up most of the execution time), increasing the computation load will certainly not reduce the overall execution time, even if the communication load is reduced. Other applications that are shuffle-heavy, such as Grep, InvIndex, RankInvIndex and SelfJoin [3], may be good candidates for implementation using FLCD. This will be considered as our future work.

For general applications, the following approach can be used to control the IV size. When there are many functions to be computed on a dataset, we will assign multiple functions to each node. We have the flexibility of defining and assigning these function groups. In this work, we can think of a reduce function as a cumulative set of functions. In this way, each node is assigned one reduce function. Furthermore, we can think of an IV as a cumulative set of map computations, where each computation is for a function assigned to some node. In this way, the size of the IV is controlled by the function grouping and assignment. For example, if all functions are associated with the same size IVs, the cumulative IV size is proportional to the number of functions assigned to a node. Therefore, the reduce function assignments can be designed in such a way to control the IV size.

Another consideration in the implementation of CDC is the cost of duplicating a dataset r times over the network. Moreover, the dataset needs to be strategically split up and placed at specific nodes. For the experiments of this paper, each node stores the entire dataset on the hard-drive and then loads the appropriate files based on the file mapping. For some systems it may be unrealistic to duplicate the data many times. Or, it may not be possible to realize a specific CDC file mapping given some predefined storage on the computing nodes. Sending files to the nodes before the Map phase can take lots of time, which may outweigh the benefits of reduced communication load in the shuffle phase. However, there are some interesting results in the literature which begin to tackle this problem. For example, [29] considers the problem of coded caching whose designs are generally "translatable" to

TABLE II: Empirical Evaluation with $K = 16$ worker nodes

Design	r	IV size ratio	CodeGen (sec.)	Map (sec.)	Encode (sec.)	Shuffle (sec.)	Decode (sec.)	Reduce (sec.)	Total Time (sec.)	Speedup
Uncoded	1	1	0.05	14.94	–	906.46	–	14.63	936.07	–
LMYA [4]	1	1	0.79	15.14	0.81	891.69	0.79	13.55	922.76	1.01×
LMYA [4]	2	1	15.23	27.34	1.10	409.37	0.58	11.40	465.01	2.01×
LMYA [4]	3	1	101.37	39.72	1.16	379.90	0.62	13.143	535.91	1.75×
KR [6]	2	1	0.437	30.45	0.77	831.83	0.76	13.65	877.89	1.07×
KR [6]	4	1	1.83	55.32	0.74	238.13	0.44	10.74	307.21	3.05×
KR [6]	8	1	1.08	122.46	1.13	88.50	0.32	11.81	225.31	4.15×
FLCD	2	1	0.29	30.85	1.10	831.12	0.63	9.16	873.14	1.07×
FLCD	3	4 : 5	0.89	45.51	1.53	376.31	0.63	10.89	435.75	2.15×
FLCD	4	1	2.04	49.09	1.60	238.66	0.49	14.02	305.91	3.06×
FLCD	5	2 : 3	5.08	75.28	1.90	150.34	0.49	15.44	248.52	3.77×
FLCD	6	1 : 2	3.60	109.86	2.22	159.08	0.56	25.32	300.65	3.11×
FLCD	7	1 : 2	3.03	125.84	1.96	71.51	0.46	18.18	220.98	4.24×
FLCD	8	1	3.43	115.23	1.89	88.14	0.39	13.89	222.97	4.20×

TABLE III: Empirical Evaluation with $K = 22$ worker nodes

Design	r	IV size ratio	CodeGen (sec.)	Map (sec.)	Encode (sec.)	Shuffle (sec.)	Decode (sec.)	Reduce (sec.)	Total Time (sec.)	Speedup
Uncoded	1	1	0.02	12.20	1	903.78	–	6.96	922.95	–
LMYA [4]	1	1	2.92	7.79	0.61	901.73	0.41	8.56	921.40	1.00×
LMYA [4]	2	1	44.00	24.28	0.89	419.50	0.43	8.83	497.93	1.85×
KR [6]	2	1	5.17	16.85	0.58	858.61	0.56	9.71	891.46	1.04×
FLCD	2	1	0.74	15.82	0.86	857.72	0.51	7.14	882.788	1.05×
FLCD	3	6 : 7	3.26	40.22	1.25	395.67	0.45	9.93	450.76	2.05×
FLCD	4	5 : 6	16.88	48.11	1.60	374.86	0.53	10.49	452.37	2.04×
FLCD	5	4 : 5	107.76	75.53	2.05	169.41	0.68	10.03	365.46	2.53×
FLCD	6	3 : 4	187.30	118.03	3.12	193.69	1.08	12.62	515.83	1.79×

TABLE IV: Empirical Evaluation with $K = 25$ worker nodes

Design	r	IV size ratio	CodeGen (sec.)	Map (sec.)	Encode (sec.)	Shuffle (sec.)	Decode (sec.)	Reduce (sec.)	Total Time (sec.)	Speedup
Uncoded	1	1	0.06	13.94	–	904.55	–	6.24	924.78	–
LMYA [4]	1	1	3.24	6.79	0.52	903.93	0.37	6.26	921.10	1.00×
LMYA [4]	2	1	74.66	13.82	0.82	421.94	0.43	8.47	520.14	1.78×
KR [6]	5	1	125.21	28.37	0.94	174.34	0.56	8.06	337.50	2.74×
FLCD	2	11 : 12	1.04	15.13	0.78	865.01	0.39	8.46	890.82	1.04×
FLCD	3	7 : 8	9.99	37.96	1.19	401.59	0.44	8.67	459.83	2.01×
FLCD	4	5 : 6	57.57	51.24	1.52	383.38	0.53	9.16	503.51	1.84×
FLCD	5	1	212.71	93.77	2.17	174.78	0.79	8.39	492.62	1.88×

CDC. Specifically, the authors of [29] consider a random file placement and demonstrate that multicasting opportunities still exist in this setting. In fact, an order optimal trade-off still exists. Alternatively, it would be interesting to study a network with some storage constraints such that not all file mappings are possible. Perhaps there exists some hybrid design using concepts of random placement and deterministic placement that achieves a significant communication load improvement.

Furthermore, an interesting direction for future work is to explore more communication efficient CDC designs with flexible IV sizes that can serve r nodes within each shuffle group, as opposed to serving only $r-1$ nodes as in the present FLCD design for $K > 3$. This has the potential to generalize the proposed FLCD design for the special case of $K = 3$ to arbitrary K, and possibly yield a better communication-computation trade off in this general MapReduce framework.

IX. Conclusions

In this work, we developed a new flexible, low complexity design (FLCD) to expedite computing platforms such as MapReduce and Spark by trading increased local computation with reduced communication across the network. Built upon a combinatorial design for the Map and Shuffle phase, the FLCD schemes utilize the design freedom in defining map and reduce functions to facilitate varying IV sizes under a general MapReduce framework. This new approach led to an interesting class of asymptotic homogeneous CDC systems that can adapt to a wide range of network parameters and facilitate low complexity implementation, while requiring only small variations in the IV sizes. We provided the most comprehensive empirical evaluations to date on Amazon EC2 for the comparisons of the CDC schemes. Our evaluations of the

FLCD covered noticeably more network configurations than previous designs permitted and showed substantial reductions of 12%-52% in total time under the same network parameters. These successfully validated the flexibility and low complexity of the FLCD schemes.

Acknowledgement

This work was supported through the National Science Foundation grants CCF-1817154. We thank Aaron Goh for the help on the Amazon EC2 implementations of the algorithms used in this paper.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets.," *HotCloud*, vol. 10, no. 10-10, pp. 95, 2010.

[3] Zhuoyao Zhang, Ludmila Cherkasova, and Boon Thau Loo, "Performance modeling of mapreduce jobs in heterogeneous cloud environments," in *2013 IEEE Sixth International Conference on Cloud Computing*. IEEE, 2013, pp. 839–846.

[4] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Transactions on Information Theory*, vol. 64, no. 1, pp. 109–128, 2017.

[5] K. Konstantinidis and A. Ramamoorthy, "Leveraging coding techniques for speeding up distributed computing," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–6.

[6] K. Konstantinidis and A. Ramamoorthy, "Resolvable designs for speeding up distributed computing," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2020.

[7] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *2016 IEEE Globecom Workshops (GC Wkshps)*, Dec 2016, pp. 1–6.

[8] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Compressed coded distributed computing," in *2018 IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 2032–2036.

[9] M. Kiamari, C. Wang, and A. S. Avestimehr, "On heterogeneous coded distributed computing," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–7.

[10] Y. H. Ezzeldin, M. Karmoose, and C. Fragouli, "Communication vs distributed computation: An alternative trade-off curve," in *2017 IEEE Information Theory Workshop (ITW)*, Nov 2017, pp. 279–283.

[11] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "A scalable framework for wireless distributed computing," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2643–2654, 2017.

[12] N. Woolsey, R. Chen, and M. Ji, "A new combinatorial design of coded distributed computing," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 726–730.

[13] N. Woolsey, R. Chen, and M. Ji, "Cascaded coded distributed computing on heterogeneous networks," *arXiv preprint arXiv:1901.07670*, 2019.

[14] N. Woolsey, R. Chen, and M. Ji, "Coded distributed computing with heterogeneous function assignments," *arXiv preprint arXiv:1902.10738*, 2019.

[15] S. R. Srinivasavaradhan, L. Song, and C. Fragouli, "Distributed computing trade-offs with random connectivity," in *2018 IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 1281–1285.

[16] K. Konstantinidis and A. Ramamoorthy, "Camr: Coded aggregated mapreduce," in *2019 IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 1427–1431.

[17] F. Xu and M. Tao, "Heterogeneous coded distributed computing: Joint design of file allocation and function assignment," *arXiv preprint arXiv:1908.06715*, 2019.

[18] K. Wan, M. Ji, and G. Caire, "Topological coded distributed computing," *arXiv preprint arXiv:2004.04421*, 2020.

[19] J. Jiang and L. Qu, "Coded distributed computing schemes with smaller numbers of input files and output functions," *arXiv preprint arXiv:2001.04194*, 2020.

[20] Z. Bar-Yossef, Y. Birk, T.S. Jayram, and T. Kol, "Index coding with side information," *Information Theory, IEEE Transactions on*, vol. 57, no. 3, pp. 1479–1494, 2011.

[21] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *Information Theory, IEEE Transactions on*, vol. 60, no. 5, pp. 2856–2867, 2014.

[22] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *Networking, IEEE/ACM Transactions on*, vol. 23, no. 4, pp. 1029–1040, Aug 2015.

[23] K. Wan, D. Tuninetti, M. Ji, G. Caire, and P. Piantanida, "Fundamental limits of decentralized data shuffling," *IEEE Transactions on Information Theory*, 2020.

[24] M. Ji, G. Caire, and A. F. Molisch, "Fundamental limits of caching in wireless d2d networks," *IEEE Transactions on Information Theory*, vol. 62, no. 2, pp. 849–869, Feb 2016.

[25] N. Woolsey, R.-R. Chen, and M. Ji, "A new combinatorial coded design for heterogeneous distributed computing," *arXiv preprint arXiv:2007.11116*, 2020.

[26] N. Woolsey, R.-R. Chen, and M. Ji, "A combinatorial design for cascaded coded distributed computing on general networks," *arXiv preprint arXiv:2008.00581*, 2020.

[27] S. Li, S. Supittayapornpong, M. A. Maddah-Ali, and S. Avestimehr, "Coded terasort," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2017, pp. 389–398.

[28] M. Noll, "Benchmarking and stress testing an hadoop cluster with terasort, testdfsio & co," *Online: http://www. michael-noll.com/blog/2011/04/09/benchmarking-andstress-testing-an-hadoopcluster-with-terasort-testdfsio-nnbench-mrbench*, 2011.

[29] M. A. Maddah-Ali and U. Niesen, "Decentralized caching attains order-optimal memory-rate tradeoff," *arXiv preprint arXiv:1301.5848*, 2013.

Nicholas Woolsey (S'17) is Ph.D. student of the Department of Electrical and Computer Engineering at University of Utah and defended his Ph.D. in December, 2020 receiving the 2020 Best ECE Dissertation Award. He is currently with Trabus Technologies, San Diego, CA as a signal processing engineer developing decentralized wireless network technologies. His research interests include combinatorial designs and algorithms for resource allocation, coding and efficient communications in distributed computing, private and caching networks. From 2014 to 2017, he was an Electrical Engineer at Northrop Grumman Corporation (NGC), Ogden, UT developing test and evaluation methods, modernization solutions and signal processing algorithms for the sustainment of aging aircraft and ground communication systems. While at NGC, he received the "Outside the Box" Grant to investigate the design of a modern receiver that interfaces aging technology and the 2016 Brent Scowcroft Team Award for performing exceptional systems engineering work. He received a B.S. degree in Biomedical Engineering from University of Connecticut in 2012 and M.Eng. degree in Bioengineering from University of Maryland, College Park in 2015 with a focus on signal processing, imaging and optics.

Rong-Rong Chen received the B.S. degree in Applied Mathematics from Tsinghua University, P.R. China in 1993, and the M.S. degree in Mathematics and the Ph.D. degree in Electrical and Computer Engineering from the University of Illinois at Urbana-Champaign in 1995, and 2003, respectively. She was an Assistant Professor at the University of Utah from 2003-2011 and has been an Associate Professor from 2011 to present. Her main research interests are in the area of communication systems and networks, with current emphasis on distributed computing, machine learning, caching networks, statistical signal processing, image reconstructions, and channel coding. She was the recipient of the M. E. Van Valkenburg Graduate Research Award for excellence in doctoral research in the ECE department at the University of Illinois at Urbana-Champaign in 2003. She was a recipient of the prestigious National Science Foundation Faculty Early Career Development (CAREER) award in 2006. She was rated among the Top 15% Instructors of College of Engineering at University of Utah in 2017 and 2018. She has served as an associate editor for IEEE Transactions on Signal Processing and as a guest editor of IEEE Journal on Selected Topics in Signal Processing. She has served on the technical program committees of leading international conferences in wireless communication and networks.

Mingyue Ji (S'09-M'15) received the B.E. in Communication Engineering from Beijing University of Posts and Telecommunications (China), in 2006, the M.Sc. degrees in Electrical Engineering from Royal Institute of Technology (Sweden) and from University of California, Santa Cruz, in 2008 and 2010, respectively, and the PhD from the Ming Hsieh Department of Electrical Engineering at University of Southern California in 2015. He subsequently was a Staff II System Design Scientist with Broadcom Corporation (Broadcom Limited) in 2015-2016. He is now an Assistant Professor of Electrical and Computer Engineering Department and an Adjunct Assistant Professor of School of Computing at the University of Utah. He received the IEEE Communications Society Leonard G. Abraham Prize for the best IEEE JSAC paper in 2019, the best paper award in IEEE ICC 2015 conference, the best student paper award in IEEE European Wireless 2010 Conference and USC Annenberg Fellowship from 2010 to 2014. He has served as an Associate Editor of IEEE Transactions on Communications from 2020. He is interested the broad area of information theory, coding theory, concentration of measure and statistics with the applications of caching networks, wireless communications, distributed storage and computing systems, distributed machine learning, and (statistical) signal processing.

APPENDIX A
THE PROOF OF PROPOSITION 1

We consider the FLCD scheme for $K = 3$ and $r = 2$. It can be seen directly that the FLCD scheme is correct from its description in Section VI-A. Here, we will derive the communication load (8).

From the FLCD description in Section VI-A, it can be seen that this scheme is correct straightforwardly. Note that, by (14), the number of total bits of each IV set $\mathcal{V}^k_{\{i,j\}}$ is the same since it contains $|\mathcal{M}_{\{i,j\}}|$ IVs of size T_k bits each. Let the number of bits in each IV set $\mathcal{V}^k_{\{i,j\}}$ be B, then $|\mathcal{M}_{\{i,j\}}|T_k = B$ and we obtain

$$L^{\mathrm{FLCD}}(2) = \frac{3(B/2)}{N(T_1+T_2+T_3)}$$

$$= \frac{3B}{2(|\mathcal{M}_{\{1,2\}}| + |\mathcal{M}_{\{1,3\}}| + |\mathcal{M}_{\{2,3\}}|)(T_1+T_2+T_3)}$$
$$= \frac{3B}{2\left(\frac{B}{T_3}+\frac{B}{T_2}+\frac{B}{T_1}\right)(T_1+T_2+T_3)}$$
$$= \frac{3T_1T_2T_3}{2(T_1T_2+T_1T_3+T_2T_3)(T_1+T_2+T_3)}. \tag{17}$$

Hence, we finish the proof of Proposition 1.

APPENDIX B
PROOF OF COROLLARY 1

In this section, we will prove Corollary 1, which states that $L^{\mathrm{FLCD}}(2) < \frac{1}{6} = L^{\mathrm{LMYA}}(2)$ when T_1, T_2 and T_3 are not all equal. Here, $L^{\mathrm{FLCD}}(2)$ and $L^{\mathrm{LMYA}}(2)$ refer to equations (8) and (4), respectively. Note that, when $K = 3$ and $r = 2$, we obtain that $L^{\mathrm{LMYA}} = \frac{1}{2}\left(1-\frac{2}{3}\right) = \frac{1}{6}$. Then, by using the Arithmetic Mean-Geometric Mean (AM-GM) Inequality twice to obtain

$$\frac{T_1T_2+T_1T_3+T_2T_3}{3} \geq \sqrt[3]{(T_1T_2T_2)^2}, \tag{18}$$

and

$$\frac{T_1+T_2+T_3}{3} \geq \sqrt[3]{T_1T_2T_2}. \tag{19}$$

In both (18) and (19), equality holds only when $T_1 = T_2 = T_3$. By using (18) and (19), we can obtain that

$$(T_1T_2+T_1T_3+T_2T_3)(T_1+T_2+T_3)$$
$$\geq \sqrt[3]{(T_1T_2T_2)^2}\cdot\sqrt[3]{T_1T_2T_2} = 9T_1T_2T_2. \tag{20}$$

Therefore

$$L^{\mathrm{FLCD}}(2) = \frac{3T_1T_2T_2}{2(T_1T_2+T_1T_3+T_2T_3)(T_1+T_2+T_3)}$$
$$\leq \frac{3T_1T_2T_2}{2\cdot 9T_1T_2T_2} = \frac{1}{6}, \tag{21}$$

where equality holds only if $T_1 = T_2 = T_3$. Hence, we complete the proof of Corollary 1.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3098593, IEEE Transactions on Cloud Computing

APPENDIX C
THE PROOF OF THEOREM 1

Here, we will provide the correctness proof of the general FLCD scheme for $K > 3$ and prove the communication-computation tradeoff shown in (10).

In this case, we will first prove (10) in Theorem 1 and then prove the correctness of the FLCD scheme. To derive the communication load, we will need to count the number of bits transmitted. By the FLCD design, the number of bits in each IV set $\mathcal{V}_n^{k,j}$ is $\lfloor m \rfloor T_1'$ bits. The reason for it is as follows. If $k \in \mathcal{K}_1^i \subseteq \mathcal{K}_1, k \in [K]$, there are $|\mathcal{K}_1^i| - 1 = \hat{m} - 1 = \lfloor m \rfloor$ files that the nodes in $\mathcal{S}_n \setminus k$ have the access to but node k does not. These files are defined by the files mapped to the nodes $\mathcal{S}_n \setminus k$ and a node $k' \in \mathcal{K}_1^i \setminus k$. Therefore, node k requests $\lfloor m \rfloor$ IVs, each of size T_1' bits, from the nodes of $\mathcal{S}_n \setminus k$. Similarly, if $k \in \mathcal{K}_2^i \subseteq \mathcal{K}_2, k \in [K]$, the number of bits in each IV set $\mathcal{V}_n^{k,j}$ is $(|\mathcal{K}_2^i| - 1)T_2 = (\lfloor m \rfloor - 1)T_2'$ bits that node k requests from the nodes of $\mathcal{S}_n \setminus k$. Since $\lfloor m \rfloor T_1' = (\lfloor m \rfloor - 1)T_2'$, each IV set $\mathcal{V}_n^{k,j}$ is $\lfloor m \rfloor T_1'$ bits. Consider all shuffle groups $\mathcal{S}_n, n \in [N]$. Each of the r nodes of $\mathcal{S}_n$, sends a message of size $\frac{\lfloor m \rfloor T_1'}{r-1}$ bits. Hence, the communication load is given by

$$
\begin{aligned}
L^{\text{FLCD}} &\overset{\text{(a)}}{=} \frac{1}{N(K_1 T_1' + K_2 T_2')} \cdot N \cdot r \cdot \frac{\lfloor m \rfloor T_1'}{r-1} \\
&= \frac{r\lfloor m \rfloor T_1'/(r-1)}{(\hat{m}K - \lfloor m \rfloor \hat{m} r)T_1' + (\lfloor m \rfloor \hat{m} r - \lfloor m \rfloor K) \cdot \frac{\lfloor m \rfloor}{\lfloor m \rfloor - 1} \cdot T_1'} \\
&= \frac{r\lfloor m \rfloor (\lfloor m \rfloor - 1)/(r-1)}{(\hat{m}K - \lfloor m \rfloor \hat{m} r)(\lfloor m \rfloor - 1) + (\lfloor m \rfloor \hat{m} r - \lfloor m \rfloor K)\lfloor m \rfloor} \\
&= \frac{r\lfloor m \rfloor (\lfloor m \rfloor - 1)/(r-1)}{r(\lfloor m \rfloor \hat{m} - \lfloor m \rfloor^2 \hat{m} + \lfloor m \rfloor^2 \hat{m}) + K(\hat{m}(\lfloor m \rfloor - 1) - \lfloor m \rfloor^2)} \\
&= \frac{1}{r-1} \cdot \frac{r\lfloor m \rfloor (\lfloor m \rfloor - 1)}{r\lfloor m \rfloor \hat{m} + K((\lfloor m \rfloor + 1)(\lfloor m \rfloor - 1) - \lfloor m \rfloor^2)} \\
&= \frac{1}{r-1} \cdot \frac{r\lfloor m \rfloor (\lfloor m \rfloor - 1)}{r\lfloor m \rfloor \hat{m} - K} \\
&= \frac{1}{r-1} \left(\frac{\lfloor m \rfloor^2 - \lfloor m \rfloor}{\lfloor m \rfloor \hat{m} - m} \right),
\end{aligned} \tag{22}
$$

where (a) is because $\mathcal{V}_n^{k,j}$ contains $\lfloor m \rfloor T_1'$ bits. Hence, we obtain (10) in Theorem 1.

It remains to prove the correctness of the FLCD scheme when $K > 3$. In order to show this, we will need to verify that every node k collects all IVs $V_{k,1}, \ldots, v_{k,N}$. This can be seen because node k will receive every IV set $\mathcal{V}_n^k$ for all n such that $k \in \mathcal{S}_n$. Moreover, $\mathcal{V}_n^k$ contains every IV computed by the nodes of $\mathcal{S}_n \setminus k$ but not at node k. This includes all IVs from files mapped at nodes $\mathcal{S}_n \setminus k$ and at one node from $\mathcal{K}_j^i \setminus k$ where $k \in \mathcal{K}_j^i$. By considering all N node groups, this covers all files not available to node k. Therefore, node k will receive all requested IVs that are not locally computed.

APPENDIX D
PROOF OF COROLLARY 2

In this section, we prove the Corollary 2 which states that $L^{\text{FLCD}}(r) < \frac{1}{r-1}\left(1 - \frac{r}{K}\right)$ when m is not an integer and $K > 3$. $L^{\text{FLCD}}(r)$ is given in (10). Assume that $m = \lfloor m \rfloor + a > 1$ and $0 < a < 1, a \in \mathbb{R}$ such that m is not an integer. Then, using the fact that $a - a^2 > 0$, it can be seen that

$$
\begin{aligned}
&\lfloor m \rfloor^3 - \lfloor m \rfloor^2 + a\lfloor m \rfloor^2 - a\lfloor m \rfloor \\
&< \lfloor m \rfloor^3 - \lfloor m \rfloor^2 + a\lfloor m \rfloor^2 - a\lfloor m \rfloor + a - a^2.
\end{aligned} \tag{23}
$$

Then, we obtain

$$
(\lfloor m \rfloor + a)(\lfloor m \rfloor^2 - \lfloor m \rfloor) < (\lfloor m \rfloor + a - 1)(\lfloor m \rfloor^2 - a). \tag{24}
$$

Using the fact that $\lfloor m \rfloor^2 \geq 1 > a$, (24) implies

$$
\frac{\lfloor m \rfloor^2 - \lfloor m \rfloor}{\lfloor m \rfloor^2 - a} < \frac{\lfloor m \rfloor + a - 1}{\lfloor m \rfloor + a}, \tag{25}
$$

Since $m = \lfloor m \rfloor + a$, we obtain

$$
\frac{\lfloor m \rfloor^2 - \lfloor m \rfloor}{\lfloor m \rfloor^2 + \lfloor m \rfloor - m} < \frac{m-1}{m}, \tag{26}
$$

which implies

$$
\frac{\lfloor m \rfloor^2 - \lfloor m \rfloor}{\lfloor m \rfloor(\lfloor m \rfloor + 1) - m} < 1 - \frac{1}{m}. \tag{27}
$$

Finally, since $\hat{m} = \lfloor m \rfloor + 1$ and $m = \frac{K}{r}$, we obtain

$$
\frac{\lfloor m \rfloor^2 - \lfloor m \rfloor}{\lfloor m \rfloor \hat{m} - m} < 1 - \frac{r}{K}. \tag{28}
$$

Hence,

$$
L^{\text{FLCD}}(r) = \frac{1}{r-1}\left(\frac{\lfloor m \rfloor^2 - \lfloor m \rfloor}{\lfloor m \rfloor \hat{m} - m}\right) < \frac{1}{r-1}\left(1 - \frac{r}{K}\right). \tag{29}
$$

Therefore, we complete the proof of Corollary 2.