Multi-Cluster Interleaving on Linear Arrays and Rings

Anxiao (Andrew) Jiang and Jehoshua Bruck
California Institute of Technology
Electrical Engineering Department
MC 136-93
Pasadena, CA 91125, U.S.A.
E-mail: {jax,bruck}@paradise.caltech.edu

***Abstract*— Interleaving codewords is an important method not only for combatting burst-errors, but also for flexible data-retrieving. This paper defines the Multi-Cluster Interleaving (MCI) problem, an interleaving problem for parallel data-retrieving. The MCI problems on linear arrays and rings are studied. The following problem is completely solved: how to interleave integers on a linear array or ring such that any m ($m \geq 2$) non-overlapping segments of length 2 in the array or ring have at least 3 distinct integers. We then present a scheme using a 'hierarchical-chain structure' to solve the following more general problem for linear arrays: how to interleave integers on a linear array such that any m ($m \geq 2$) non-overlapping segments of length L ($L \geq 2$) in the array have at least $L+1$ distinct integers. It is shown that the scheme using the 'hierarchical-chain structure' solves the second interleaving problem for arrays that are asymptotically as long as the longest array on which an MCI exists, and clearly, for shorter arrays as well.**

I. INTRODUCTION

Interleaving codewords is an important method for both data-retrieving and error-correction. Its application in error-correction is well-known. The most familiar example is the interleaving of codewords on a linear array, which has the form '$-1-2-3-\cdots-n-1-2-3-\cdots-n-$', for combatting one-dimensional burst-errors of length up to n. Other interesting examples include [1] [2] [3] [6] [7] [14], which are mainly for correcting burst-errors of different shapes on two- or three-dimensional arrays.

The applications of codeword interleaving in data-retrieving, although maybe less well-known, are just as broad. Data streaming and broadcast schemes using forward-error-correcting codes have received extensive interest in both academia and industry, where interleaved components of a codeword are transmitted in sequence and every client can listen to this data stream for a while until a sufficiently large subset of the codeword components are received for recovering the information in the codeword [4] [10]. Interleaving is also studied in the scenario of file storage, where a file is encoded into a codeword and components of the codeword are interleavingly placed on a network, such that every node in the network can retrieve enough distinct codeword components from its proximity for recovering the file [8] [11]. In all those cases, the codeword components are interleaved on some graph structure. For example, in the data streaming and broadcast case, the codeword components can be seen as interleaved on a linear array, because they are sequentially transmitted along the time axis. (If the sequence of data are transmitted repeatedly — e.g., using a broadcast disk — then they can be seen as interleaved on a ring.) For file storage schemes as those in [8] and [11], the codeword components are interleaved (placed) on more general graphs, with the graphs' vertices representing network-nodes and edges representing network-links. What's more, most of the time, retrieving data corresponds to retrieving the interleaved codeword components on a connected subgraph — for example, in data streaming/broadcast a client usually listens to the data in one time period, which form a segment of the array (or ring); and in file storage [8] [11] the proximity of each node is a subgraph. We call every such connected subgraph a *cluster*.

By using interleaving, the above schemes all enable 'flexible' data-retrieving, in the sense that the original information contained in the interleaved data can be recovered by accessing *any* sufficiently large cluster. The data-retrieving performance can be further improved if multiple clusters can be accessed in parallel. Accessing data placed in different parts of a graph in parallel has the benefits of balancing load and reducing access time, and has already been studied [5] [13]. In fact, even the RAID system [12] can be seen as an example of it. Then it's natural to ask the following question: what is the appropriate form of interleaving for parallel data-retrieving?

If it is required that for any m ($m \geq 2$) non-overlapping clusters, the interleaved codeword components on them are all distinct, then each codeword component can be placed only once on the graph, even if m is as small as 2. Such an interleaving scheme, although minimizes the sizes of clusters that a client needs to access to retrieve enough dis-

[1] This work was supported in part by the Lee Center for Advanced Networking at the California Institute of Technology.

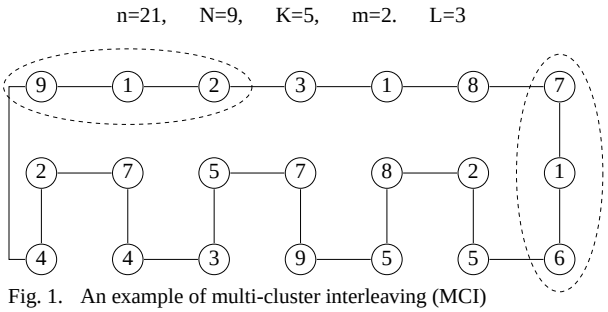

Fig. 1. An example of multi-cluster interleaving (MCI)

tinct codeword components, is not scalable because it requires the number of components in the codeword to equal the size of the graph, which would imply very high encoding/decoding complexity or even non-existence of the code if the graph is huge. So a tradeoff is needed between the scheme's scalability and the amount of overlapping among codeword components on different clusters.

In this paper we only study interleaving on linear arrays and rings. We define the following general interleaving problem for parallel data-retrieving:

Definition 1: Let $G = (V, E)$ be a linear array (or ring) of n vertices. Let N, K, m and L be positive integers such that $N \geq K > L$ and $m \geq 2$. A *cluster* is defined to be a connected subgraph of the array (or ring) containing L vertices. Assign one number in the set $\{1, 2, \cdots, N\}$ to each vertex. Such an assignment is called a *Multi-Cluster Interleaving* (MCI) if and only if any m clusters that are non-overlapping are assigned no less than K distinct numbers. □

Note that the N numbers in $\{1, 2, \cdots, N\}$ assigned to the array (or ring) represent the N components in a codeword decoding which needs K distinct components. Clearly if we let $m = 1$ in the above definition (and then let $K = L$), then it becomes the traditional interleaving. And if an interleaving on a linear array (or ring) is an MCI for some given value of m, then it is an MCI for larger values of m as well.

The following is an example of MCI.

Example 1: A ring $G = (V, E)$ of $n = 21$ vertices is shown in Fig. 1. The parameters are $N = 9$, $K = 5$, $m = 2$ and $L = 3$. An interleaving is shown in the figure, where the number on every vertex is the number assigned to it. It can be verified that any 2 clusters that don't overlap have at least 5 distinct numbers. For example, the two clusters in circle in Fig. 1 have numbers '9, 1, 2' and '7, 1, 6' respectively, so they together have no less than 5 distinct numbers. So the interleaving is a multi-cluster interleaving on the ring G.

If we remove an edge in the ring, then G will become a linear array. Clearly if all other parameters remain the same, the interleaving shown in Fig. 1 will be a multi-cluster interleaving on the array. □

The general MCI problem can be divided into smaller problems according to the values of the parameters. Our main results in this paper are:

- The family of problems with constraints that $L = 2$ and $K = 3$ are solved completely for both arrays and rings. In this case, structural properties of MCI are revealed, and algorithms are presented which output MCI on arrays or rings as long as the MCI exists.
- The family of problems with the constraint that $K = L + 1$ are studied for arrays. A scheme using a 'hierarchical-chain' structure is presented for constructing MCI on arrays. It is shown that the scheme solves the MCI problem for arrays that are asymptotically as long as the longest array on which an MCI exists, and clearly, for shorter arrays as well.

Due to the space limitation, we skip or present only sketches of the proofs for the results in this paper. For detailed proofs, please refer to [9].

The multi-cluster interleaving on arrays and rings seems to have natural applications in data-streaming and broadcasting. Imagine that the interleaved codeword components are transmitted in several channels, and the data in each channel have a different time-offset. Then a client can simultaneously listen to multiple channels in order to get data faster, which is equivalent to retrieving data from multiple clusters. Another possible application is data storage on disks, where we assume multiple instruments can read different parts of a disk in parallel to accelerate I/O speed.

II. MCI WITH CONSTRAINTS $L = 2$ AND $K = 3$

In this section we study the MCI on linear arrays and rings with constraints that $L = 2$ and $K = 3$.

A. Linear Arrays

The following notations will be used throughout this paper. We denote the n vertices in the linear array $G = (V, E)$ by v_1, v_2, $\cdots$, v_n. For $2 \leq i \leq n-1$, the two vertices adjacent to v_i are v_{i-1} and v_{i+1}. A connected subgraph of G induced by vertices $v_i, v_{i+1}, \cdots, v_j$ ($j \geq i$) is denoted by $(v_i, v_{i+1}, \cdots, v_j)$. If G has an interleaving on it, then $c(v_i)$ denotes the number assigned to vertex v_i. The numbers assigned to a connected subgraph of G, $(v_i, v_{i+1}, \cdots, v_j)$, are denoted by $[c(v_i) - c(v_{i+1}) - \cdots - c(v_j)]$.

For any fixed parameters N, K, m and L, there is a corresponding number n_{max} of finite value such that an MCI exists on an array G only if G's length n is no greater than n_{max}. That's because in an MCI, for any set of L distinct numbers, there can be at most $m-1$ non-overlapping clusters each of which is assigned those L numbers (including a subset of those L numbers) only. There are totally $\binom{N}{L}$ such sets containing L distinct numbers; and each cluster is assigned at most L distinct numbers. So n_{max} can't be infinite. Below we study the relationship between the structure of the MCI and the length of the array.

Lemma 1: Let the values of N, K, m and L be fixed, where $N \geq 4$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on an array of n vertices. Then in any MCI on an array $G = (V, E)$ of n_{max} vertices, no two adjacent vertices are assigned the same number.

Proof: Please refer to [9]. □

Lemma 2: Let the values of N, K, m and L be fixed, where $N \geq 4$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on an array of n vertices. Then $n_{max} \leq (N-1)[(m-1)N - 1] + 2$.

Sketch of Proof: Let $G = (V, E)$ be a linear array of n_{max} vertices. And say there is an MCI on G. Then we color the vertices in G with three colors — 'red', 'yellow' and 'green' — through the following three steps: Step 1, for $2 \leq i \leq n_{max} - 1$, if $c(v_{i-1}) = c(v_{i+1})$, then we color v_i with the 'red' color; Step 2, for $2 \leq i \leq n_{max}$, we color v_i with the 'yellow' color if v_i is not colored 'red' and there exists j such that these four conditions are satisfied: (1) $1 \leq j < i$, (2) v_j is not colored 'red', (3) $c(v_j) = c(v_i)$, (4) the vertices between v_j and v_i — that is, v_{j+1}, v_{j+2}, $\cdots$, v_{i-1} — are all colored 'red'; Step 3, for $1 \leq i \leq n_{max}$, if v_i is neither colored 'red' nor colored 'yellow', then color v_i with the 'green' color.

If we arbitrarily pick two different numbers — say 'i' and 'j' — from the set $\{1, 2, \cdots, N\}$, then we get a pair $[i, j]$. There are totally $\binom{N}{2}$ such un-ordered pairs. We divide those $\binom{N}{2}$ pairs into four groups 'A', 'B', 'C' and 'D' in the following way:

(1) A pair $[i, j]$ is placed in group A if and only if the following two conditions are satisfied: (i) at least one 'green' vertex is assigned number 'i' and at least one 'green' vertex is assigned number 'j', (ii) for any two 'green' vertices that are assigned numbers 'i' and 'j' respectively, there is at least one 'green' vertex between them.

(2) A pair $[i, j]$ is placed in group B if and only if the following two conditions are satisfied: (i) at least one 'green' vertex is assigned number 'i' and at least one 'green' vertex is assigned number 'j', (ii) there exist two 'green' vertices that are assigned numbers 'i' and 'j' respectively such that there is no 'green' vertex between them.

(3) A pair $[i, j]$ is placed in group C if and only if one of the following two conditions is satisfied: (i) at least one 'green' vertex is assigned number 'i' and no 'green' vertex is assigned number 'j', (ii) at least one 'green' vertex is assigned number 'j' and no 'green' vertex is assigned number 'i'.

(4) A pair $[i, j]$ is placed in group D if and only if no 'green' vertex is assigned number 'i' or 'j'.

A detailed analysis shows that there are at most $2m - 2$ edges in G whose two endpoints form a pair in group A or C, and there are at most $2m - 3$ (respectively, 0) edges in G whose two endpoints form a pair in group B (respectively, D). By Lemma 1, any two adjacent vertices in G are assigned different numbers. Let the number of *distinct* numbers assigned to 'green' vertices be denoted by 'x'. It can be seen that exactly $\binom{x}{2}$ pairs $[i, j]$ are in group A and group B, among which at least $x - 1$ pairs are in group B; and exactly $x(N - x)$ pairs are in group C and exactly $\binom{N-x}{2}$ pairs are in group D. Therefore the number of edges in G is at most $[\binom{x}{2} - (x-1)] \cdot (2m-2) + (x-1) \cdot (2m-3) + x(N-x) \cdot (2m-2) + \binom{N-x}{2} \cdot 0 = (1-m)x^2 + (2mN - 2N - m)x + 1$, whose maximum value (at integer solutions) is achieved when $x = N - 1$ — and that maximum value is $(N-1)[(m-1)N-1]+1$. So n_{max}, the number of vertices in G, is at most $(N-1)[(m-1)N - 1] + 2$.

□

Lemma 3: Let the values of N, K, m and L be fixed, where $N = 3$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on an array of n vertices. Then $n_{max} \leq (N-1)[(m-1)N - 1] + 2$.

Proof: Please refer to [9]. □

Below we present the algorithm for computing an MCI on a linear array.

Algorithm 1: MCI on linear array with constraints $L = 2$ and $K = 3$

Input: A linear array $G = (V, E)$ of n vertices. Parameters N, K, m and L, where $N \geq 3$, $K = 3$, $m \geq 2$ and $L = 2$.

Output: An MCI on G.

Algorithm:

1. If $n > (N-1)[(m-1)N - 1] + 2$, there doesn't exist an MCI, so exit the algorithm.

2. If $n \leq N$, select n numbers in $\{1, 2, \cdots, N\}$ and assign each number to one vertex, and exit the algorithm.

3. If $N < n \leq (N-1)[(m-1)N - 1] + 2$ and $n - \{(N-1)[(m-1)N - 1] + 2\}$ is even, then select a set of integers $\{x_{i,j} | i = 1, 2, \cdots, N-1; j = 2, 3, \cdots, N; i < j\}$ that satisfy the following four requirements: (1) for $1 \leq i \leq N-1$, $x_{i,N}$ is even and $0 \leq x_{i,N} \leq 2m-2$; (2) for $1 \leq i \leq N-2$ and $j = i+1$, $x_{i,j}$ is odd and $1 \leq x_{i,j} \leq 2m-3$; (3) for $1 \leq i \leq N-3$ and $i+2 \leq j \leq N-1$, $x_{i,j}$ is even and $0 \leq x_{i,j} \leq 2m-2$; (4) if we define S as $S = \{x_{i,j} | i = 1, 2, \cdots, N-1; j = 2, 3, \cdots, N; i < j\}$, then $\sum_{x \in S} x = n - 1$.

Let $H = (V_H, E_H)$ be such a multi-graph: the vertex set $V_H = \{u_1, u_2, \cdots, u_N\}$; and for any $1 \leq i < j \leq N$, there are $x_{i,j}$ undirected edges between u_i and u_j.

Find a walk in H, $u_{k_1} \rightarrow u_{k_2} \rightarrow \cdots \rightarrow u_{k_n}$, that satisfies the following conditions: (1) the walk starts with u_1 and ends with u_{N-1} — namely, $u_{k_1} = u_1$ and $u_{k_n} = u_{N-1}$ — and passes every edge in H exactly once; (2) for any $1 \leq i < j \leq N$, the walk passes all the $x_{i,j}$ edges between u_i and u_j *consecutively*.

For $i = 1, 2, \cdots, n$, assign the number 'k_i' to the vertex v_i in G, and we get an interleaving on G. Exit the algorithm.

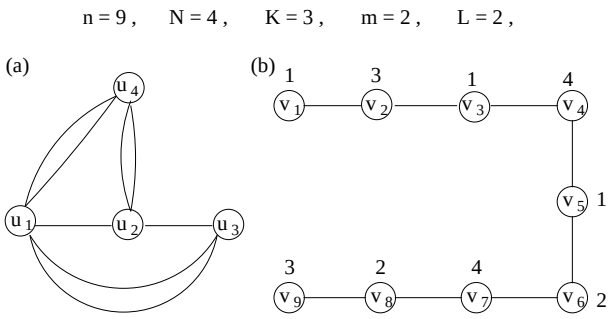

Fig. 2. (a) The graph $H = (V_H, E_H)$ (b) MCI on the array $G = (V, E)$

4. If $N < n \leq (N-1)[(m-1)N-1]+2$ and $n - \{(N-1)[(m-1)N-1]+2\}$ is odd, then select a set of integers $\{x_{i,j} | i = 1, 2, \cdots, N-1; j = 2, 3, \cdots, N; i < j\}$ that satisfy the following three requirements: (1) for $1 \leq i \leq N-1$ and $j = i+1$, $x_{i,j}$ is odd and $1 \leq x_{i,j} \leq 2m-3$; (2) for $1 \leq i \leq N-2$ and $i+2 \leq j \leq N$, $x_{i,j}$ is even and $0 \leq x_{i,j} \leq 2m-2$; (3) if we define S as $S = \{x_{i,j} | i = 1, 2, \cdots, N-1; j = 2, 3, \cdots, N; i < j\}$, then $\sum_{x \in S} x = n-1$.

Let $H = (V_H, E_H)$ be such a multi-graph: the vertex set $V_H = \{u_1, u_2, \cdots, u_N\}$; and for any $1 \leq i < j \leq N$, there are $x_{i,j}$ undirected edges between u_i and u_j.

Find a walk in H, $u_{k_1} \rightarrow u_{k_2} \rightarrow \cdots \rightarrow u_{k_n}$, that satisfies the following conditions: (1) the walk starts with u_1 and ends with u_N — namely, $u_{k_1} = u_1$ and $u_{k_n} = u_N$ — and passes every edge in H exactly once; (2) for any $1 \leq i < j \leq N$, the walk passes all the $x_{i,j}$ edges between u_i and u_j *consecutively*.

For $i = 1, 2, \cdots, n$, assign the number 'k_i' to the vertex v_i in G, and we get an interleaving on G. Exit the algorithm.

□

Algorithm 1 has complexity $O(n)$. The following is an example of the algorithm.

Example 2: Assume $G = (V, E)$ is a linear array of $n = 9$ vertices, and the parameters are $N = 4$, $K = 3$, $m = 2$ and $L = 2$. Therefore $N < n \leq (N-1)[(m-1)N-1]+2$ and $n - \{(N-1)[(m-1)N-1]+2\} = -2$ is even. So Algorithm 1's step 3 is used to compute the interleaving. We can very easily choose the following values for $x_{i,j}$: $x_{1,2} = x_{2,3} = 1$, $x_{1,3} = x_{1,4} = x_{2,4} = 2$. Then the graph $H = (V_H, E_H)$ is as shown in Fig. 2(a). We can easily find the following walk that passes every edge once: $u_1 \rightarrow u_3 \rightarrow u_1 \rightarrow u_4 \rightarrow u_1 \rightarrow u_2 \rightarrow u_4 \rightarrow u_2 \rightarrow u_3$. Corresponding to that walk, we get the MCI as shown in Fig. 2(b).

Generally speaking, when $N < n \leq (N-1)[(m-1)N-1]+2$ and $n - \{(N-1)[(m-1)N-1]+2\}$ is even, for $1 \leq i \leq N-3$, the walk in graph H that Algorithm 1 needs to find passes all the edges between u_i and u_{i+1} before passing any edge between u_{i+1} and u_{i+2}. The walk contains many 'ears' (small cycles) of the form '$i \rightarrow j \rightarrow i$'. If we delete all the 'ears' from the walk, the remaining walk is simply '$u_1 \rightarrow u_2 \rightarrow \cdots \rightarrow u_{N-1}$'. It's clear that such a walk in H can be easily found based on the above observation. The case where $N < n \leq (N-1)[(m-1)N-1]+2$ and $n - \{(N-1)[(m-1)N-1]+2\}$ is odd can be analyzed in similar ways. □

Theorem 1: Algorithm 1 is correct.

Proof: Please refer to [9]. □

Theorem 2: Let the values of N, K, m and L be fixed, where $N \geq 3$, $K = 3$, $m \geq 2$ and $L = 2$. Then there exists an MCI on a linear array of n vertices if and only if $n \leq (N-1)[(m-1)N-1]+2$.

Sketch of Proof: By using Lemma 2, Lemma 3, and Theorem 1. □

B. Rings

Lemma 4: Let the values of N, K, m and L be fixed, where $N \geq 4$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on a ring of n vertices. Then in any MCI on a ring $G = (V, E)$ of n_{max} vertices, no two adjacent vertices are assigned the same number.

Lemma 5: Let the values of N, K, m and L be fixed, where $N \geq 4$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on a ring of n vertices. Then $n_{max} \leq (N-1)[(m-1)N-1]$.

Lemma 6: Let the values of N, K, m and L be fixed, where $N = 3$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on a ring of n vertices. Then $n_{max} \leq (N-1)[(m-1)N-1]$.

Below we present the algorithm for computing an MCI on a ring.

Algorithm 2: MCI on ring with constraints $L = 2$ and $K = 3$

Input: A ring $G = (V, E)$ of n vertices. Parameters N, K, m and L, where $N \geq 3$, $K = 3$, $m \geq 2$ and $L = 2$.

Output: An MCI on G.

Algorithm: Please refer to [9]. □

Algorithm 2 has complexity $O(n)$.

Theorem 3: Algorithm 2 is correct.

Theorem 4: Let the values of N, K, m and L be fixed, where $N \geq 3$, $K = 3$, $m \geq 2$ and $L = 2$. Then there exists an MCI on a ring of n vertices if and only if $n \leq (N-1)[(m-1)N-1]$.

For detailed proofs of Lemma 4 to 6 and Theorem 3 to 4, please refer to [9].

III. MCI WITH CONSTRAINT $K = L + 1$

In this section we study the MCI problem on linear arrays with the constraint that $K = L+1$. It covers the MCI problem with constraints that $L = 2$ and $K = 3$, which is studied in the previous section, as a special case.

We define three operations on arrays — 'remove a vertex', 'insert a vertex' and 'combine two arrays'. Let G be an array of n vertices: v_1, v_2, $\cdots$, v_n. By 'removing the vertex v_i' from G ($1 \leq i \leq n$), we get a new array '$v_1 - v_2 - \cdots - v_{i-1} - v_{i+1} - \cdots - v_n$'. By 'inserting a vertex $\hat{v}$' in front of the vertex v_i in G ($1 \leq i \leq n$), we get a new array '$v_1 - v_2 - \cdots - v_{i-1} - \hat{v} - v_i - \cdots - v_n$'. (Similarly we can define 'inserting a vertex $\hat{v}$ behind the vertex v_i in G' and 'inserting a vertex $\hat{v}$ between the vertices v_i and v_{i+1} in G'.) Let H be an array of n' vertices: u_1, u_2, $\cdots$, $u_{n'}$. Assume for $1 \leq i \leq n$, v_i is assigned the number $c(v_i)$; and assume for $1 \leq i \leq n'$, u_i is assigned the number $c(u_i)$. Also let l be a positive integer between 1 and $min(n, n')$, and assume for $1 \leq i \leq l$, $c(v_i) = c(u_{n'-l+i})$. Then by saying 'combining H with G such that the last l vertices of H overlap the first l vertices of G', we mean to construct an array of $n' + n - l$ vertices whose assigned numbers are $[c(u_1) - c(u_2) - \cdots - c(u_{n'}) - c(v_{l+1}) - c(v_{l+2}) - \cdots - c(v_n)]$.

Now we present an algorithm which computes an MCI on a linear array. Different from Algorithm 1 and Algorithm 2, in this algorithm the length of the array is unknown. Instead, the algorithm tries to find the longest array that has an MCI, and compute an MCI on it. Thus the output of this algorithm not only provides an MCI solution, but also gives a lower bound on the maximum length of the array on which an MCI exists.

Algorithm 3: MCI on linear array with the constraint $K = L + 1$

Input: Parameters N, K, m and L, where $N \geq K = L + 1 \geq 3$ and $m \geq 2$.

Output: An MCI on a linear array $G = (V, E)$ of n vertices, with the value of n as large as possible.

Algorithm:

1. If $L = 2$, then let $G = (V, E)$ be an array of $n = (N-1)[(m-1)N - 1] + 2$ vertices, and use Algorithm 1 to find an MCI on G. Output G and the MCI on it, and return. (So step 2 to step 4 will be executed only if $L \geq 3$.)

2. Find a linear array B_{L+1} as long as possible that satisfies the following two conditions: (1) each vertex of B_{L+1} is assigned a number in $\{1, 2, \cdots, L\}$, namely, there is an interleaving of the numbers in $\{1, 2, \cdots, L\}$ on B_{L+1}; (2) any m non-overlapping connected subgraphs in B_{L+1} each of which contains $L-1$ vertices are assigned at least L distinct numbers. To find the array B_{L+1}, (recursively) call Algorithm 3 by replacing the inputs of the algorithm — N, K, m and L — respectively with L, L, m and $L - 1$.

Scan the vertices in B_{L+1} backward (from the last vertex to the first vertex), and insert a new vertex after every $L - 1$ vertices in B_{L+1}. (In other words, if the vertices in B_{L+1} are v_1, v_2, $\cdots$, $v_{\hat{n}}$, then by inserting vertices into B_{L+1}, we get a new array of $\hat{n} + \lfloor \frac{\hat{n}}{L-1} \rfloor$ vertices; and if we look at the new array in the reverse order — from the last vertex to the first vertex — then the array is of the form '$v_{\hat{n}} - v_{\hat{n}-1} - \cdots - v_{\hat{n}+1-(L-1)}$−(new vertex)$-v_{\hat{n}-(L-1)} - v_{\hat{n}-(L-1)-1} - \cdots - v_{\hat{n}+1-2(L-1)}$−(new vertex)$-v_{\hat{n}-2(L-1)} - v_{\hat{n}-2(L-1)-1} - \cdots - v_{\hat{n}+1-3(L-1)}$−(new vertex)$-\cdots\cdots$'. In this new array, every connected subgraph of L vertices contains exactly one newly inserted vertex.) Assign the number '$L + 1$' to every newly inserted vertex in the new array, and denote this new array by 'A_{L+1}'.

3. for $i = L + 2$ to N do

{ Find a linear array B_i as long as possible that satisfies the following three conditions: (1) each vertex of B_i is assigned a number in $\{1, 2, \cdots, i-1\}$, namely, there is an interleaving of the numbers in $\{1, 2, \cdots, i-1\}$ on B_i; (2) any m non-overlapping connected subgraphs in B_i each of which contains $L - 1$ vertices are assigned at least L distinct numbers; (3) for $j = 1$ to $L - 1$, the j-th last vertex of B_i is assigned the same number as the $(L - j)$-th vertex of A_{i-1}. To find the array B_i, (recursively) call Algorithm 3 by replacing the inputs of the algorithm — N, K, m and L — respectively with $i - 1$, L, m and $L - 1$.

Scan the vertices in B_i backward (from the last vertex to the first vertex), and insert a new vertex after every $L - 1$ vertices in B_i. Assign the number 'i' to every newly inserted vertex in the new array, and denote this new array by 'A_i'.

}

4. Combine A_N with A_{N-1}, combine A_{N-1} with A_{N-2}, $\cdots$, and combine A_{L+2} with A_{L+1} such that the last $L - 1$ vertices of A_N overlap the first $L - 1$ vertices of A_{N-1}, the last $L - 1$ vertices of A_{N-1} overlap the first $L - 1$ vertices of A_{N-2}, $\cdots$, and the last $L - 1$ vertices of A_{L+2} overlap the first $L - 1$ vertices of A_{L+1}. (In other words, if we denote the number of vertices in A_i by l_i, for $L + 1 \leq i \leq N$, then the new array we get has $\sum_{i=L+1}^{N} l_i - (L-1)(N-L-1)$ vertices.) Let this new array be $G = (V, E)$. Output G and the interleaving (which is an MCI) on it, and return.

□

Algorithm 3 outputs an array G, which is as long as the algorithm can find, and an MCI on G. Note that Algorithm 3 is recursive. The MCI on G has a 'hierarchical-chain' structure, because G is a 'chain' of sub-arrays A_{L+1}, A_{L+2}, $\cdots$, A_N of increasing lengths, while each sub-array is recursively made from more smaller sub-arrays — so they form both a 'horizontal hierarchy' and a 'vertical hierarchy'. G's length, n, is unknown before Algorithm 3 ends. But if we can use to n to evaluate the complexity of Algorithm 3, then Algorithm 3 can be easily seen to have complexity $O(n)$. If an interleaving on a long array is an MCI, then the interleaving on a connected subgraph of it (a sub-array) is also an MCI; and Algorithm 3 constructs the array G piece by piece. So it's simple to see that the algorithm can be easily modified to compute the MCI on any array of less than n vertices.

The following is an example of Algorithm 3.

Example 3: Given the parameters $N = 6$, $K = 4$, $m = 2$ and $L = 3$, we use Algorithm 3 to compute an MCI on an array as long as possible. Three arrays — B_4, B_5

and B_6 — need to be found. To compute each B_i ($4 \leq i \leq 6$), Algorithm 3 is (recursively) called; and for this example we're considering now, B_i is eventually computed by Algorithm 1. For $4 \leq i \leq 6$, B_i has $(i-2)[(m-1)(i-1)-1]+2$ vertices. As a possible result, let's say that the numbers on B_4 are $[1-3-1-2-3-2]$; and therefore the numbers on A_4 are $[4-1-3-4-1-2-4-3-2]$. Then the numbers on B_5 can be made to be $[3-4-3-1-3-2-4-2-1-4-1]$. (Note that the last $L-1=2$ vertices of B_5 are assigned numbers '4-1', the same as the first $L-1=2$ vertices of A_4. That can be done easily by permuting numbers on B_5.) Then the numbers on A_5 are $[3-5-4-3-5-1-3-5-2-4-5-2-1-5-4-1]$. Then the numbers on B_6 can be made to be $[1-3-1-4-1-5-1-2-3-2-5-2-4-3-4-5-3-5]$. (Note that the last $L-1=2$ vertices of B_6 are assigned numbers '3-5', the same as the first $L-1=2$ vertices of A_5.) Then the numbers on A_6 are $[6-1-3-6-1-4-6-1-5-6-1-2-6-3-2-6-5-2-6-4-3-6-4-5-6-3-5]$. Finally, A_6 is combined with A_5 with $L-1=2$ overlapping vertices, and A_5 is combined with A_4 with $L-1=2$ overlapping vertices, and we get the array $G=(V,E)$ which are assigned numbers $[6-1-3-6-1-4-6-1-5-6-1-2-6-3-2-6-5-2-6-4-3-6-4-5-6-3-5-4-3-5-1-3-5-2-4-5-2-1-5-4-1-3-4-1-2-4-3-2]$. G has 48 vertices. It can be verified that the interleaving on G is an MCI. □

Theorem 5: Algorithm 3 is correct.

Sketch of Proof: Do induction on the parameter L (one of the inputs of Algorithm 3). The induction assumption is: Algorithm 3 correctly outputs an MCI on an array for any given valid set of inputs N, K, m and L; and in that MCI, any L consecutive vertices are assigned L different numbers. □

The length of the longest array on which an MCI exists increases when N, the number of integers that are interleaved, increases. The performance of Algorithm 3 can be evaluated by the difference between the length of the array constructed by Algorithm 3 and the length of the longest array on which an MCI exists. We're interested in studying how the difference goes when N increases. The following theorem shows the result.

Theorem 6: Fix the values of the parameters K, m and L, where $K=L+1\geq 3$ and $m\geq 2$, and let N be a variable ($N\geq K$). Then the longest linear array on which an MCI exists has $\frac{m-1}{(L-1)!}N^L+O(N^{L-1})$ vertices. And the array output by Algorithm 3 also has $\frac{m-1}{(L-1)!}N^L+O(N^{L-1})$ vertices.

Proof: Please refer to [9]. □

Theorem 6 shows that the array output by Algorithm 3 is asymptotically as long as the longest array on which an MCI exists. As mentioned before, clearly the algorithm can be easily modified to computer MCI on shorter arrays as well.

IV. CONCLUSION

This paper defines the multi-cluster interleaving (MCI) problem for flexible parallel data-retrieving. Two families of MCI problems are solved for arrays/rings. MCI seems to have natural applications in data-streaming, broadcasting, etc. We expect that the techniques for solving the MCI problems presented in this paper may provide valuable insights into solving more general MCI problems.

REFERENCES

[1] K. A. S. Abdel-Ghaffar, R. J. McEliece, and H. C. A. van Tilborg. Two-dimensional burst identification codes and their use in burst correction. *IEEE Transactions on Information Theory*, 34:494–504, 1988.

[2] C. Almeida and R. Palazzo. Two-dimensional interleaving using the set partition technique. In *Proc. IEEE International Symposium on Information Theory*, page 505, Trondheim, Norway, 1994.

[3] M. Blaum, J. Bruck, and A. Vardy. Interleaving schemes for multidimensional cluster errors. *IEEE Transactions on Information Theory*, 44(2):730–743, 1998.

[4] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proc. ACM SIGCOMM'98*, Vancouver,Canada, 1998.

[5] J. W. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: using Tornado codes to speed up downloads. In *Proc. IEEE Infocom'99*, pages 275–283, Boston Univ.,MA,USA, 1999.

[6] P. Delsarte. Bilinear forms over a finite field, with applications to coding theory. *J. Combin. Theory*, 25-A:226–241, 1978.

[7] T. Etzion and A. Vardy. Two-dimensional interleaving schemes with repetitions: constructions and bounds. *IEEE Transactions on Information Theory*, 48(2):428–457, 2002.

[8] A. Jiang and J. Bruck. Diversity coloring for information storage in networks. In *Proceedings of the 2002 IEEE International Symposium on Information Theory*, page 381, Lausanne, Switzerland, 2002.

[9] A. Jiang and J. Bruck. Multi-cluster interleaving on linear arrays and rings. Technical Report ETR 051, Parallel and Distributed Systems Lab, California Institute of Technology, http://www.paradise.caltech.edu/papers/etr051.pdf, 2003.

[10] A. Mahanti, D. L. Eager, M. K. Vernon, and D. Sundaram-Stukel. Scalable on-demand media streaming with packet loss recovery. In *Proc. ACM SIGCOMM 2001*, pages 97–108, San Diego, 2001.

[11] M. Naor and R. M. Roth. Optimal file sharing in distributed networks. *SIAM J. Comput.*, 24(1):158–183, 1995.

[12] D. A. Patterson, G. A. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks. In *Proc. SIGMOD Int. Conf. Data Management*, pages 109–116, 1988.

[13] P. Rodriguez and E. W. Biersack. Dynamic parallel access to replicated content in the Internet. *IEEE/ACM Transactions on Networking*, 10(4):455–465, 2002.

[14] R. M. Roth. Maximum-rank array codes and their application to crisscross error correction. *IEEE Transactions on Information Theory*, 37:328–336, 1991.