Network Coding for Joint Storage and Transmission with Minimum Cost

Anxiao (Andrew) Jiang

Department of Computer Science, Texas A&M University, College Station, TX 77843-3112. ajiang@cs.tamu.edu.

Abstract— **Network coding provides elegant solutions to many data transmission problems. The usage of coding for distributed data storage has also been explored. In this work, we study a joint storage and transmission problem, where a source transmits a file to storage nodes whenever the file is updated, and clients read the file by retrieving data from the storage nodes. The cost includes the transmission cost for file update and file read, as well as the storage cost. We show that such a problem can be transformed into a pure flow problem and is solvable in polynomial time using linear programming. Coding is often necessary for obtaining the optimal solution with the minimum cost. However, we prove that for networks of generalized tree structures, where adjacent nodes can have asymmetric links between them, file splitting — instead of coding — is sufficient for achieving optimality. In particular, if there is no constraint on the numbers of bits that can be stored in storage nodes, there exists an optimal solution that always transmits and stores the file as a whole. The proof is accompanied by an algorithm that optimally assigns file segments to storage nodes.**

I. Introduction

Coding provides elegant solutions to both data transmission and data storage in networks. Its power comes from the improved flexibility that codeword symbols have in expressing a file, which can help us obtain the maximum amount of information from a set of data flows or stored data symbols.

The usage of coding for data transmission, commonly known as *network coding*, has been studied extensively. In particular, multicast using network coding achieves the network capacity, and the result extends the well known max-flow min-cut theorem [2]. Linear coding and random linear coding [4] for network coding attract lots of interest due to their simplicity and optimal performance, both of great importance for applications. Network coding can lower the complexity of data flow problems and lead to solutions that are decentralized or of improved performance in various aspects [9]. Both multicast and non-multicast [10], block codes and convolutional codes, linear coding and non-linear coding have been studied for network coding, and the complexity classification of those problems have been gaining attention [12].

Coding for data storage in distributed networks has also been studied. In [11], the following problem is studied: how to store a file distributedly such that every node can reconstruct the file by accessing the data stored on itself and its direct neighbors. The objective is to minimize the total number of bits stored in the network. In [5], [6], interleaving techniques are presented for placing codeword symbols on network nodes for optimized distributed file retrieval performance. In [1], random linear coding is used for storage, and the scheme is shown to have high success rate for data retrieval.

In this paper, we study a joint storage and transmission problem. It follows a commonly used storage-retrieval model in many network applications. In this model, the network consists of three types of nodes: a source node, storage nodes and client nodes. For generality, every node can be a storage node and a client node. The data of a file are stored in the storage nodes, each having an upper bound on its memory size. Every time the file is updated, the source node re-sends the data to the storage nodes. Each client node reads the file – with a certain frequency – by accessing the storage nodes. (A client can be thought of as a node running programs that need to read the latest file version, or a network access node that forwards consumers' file retrieval requests.) The total cost, which we seek to minimize, includes the transmission cost associated with file update/read and the storage cost. As the first result in the paper, we shall show that the joint storage and transmission problem can, in fact, be transformed to a pure network flow problem, and can be solved in polynomial time by linear programming.

In many cases, coding is necessary for minimizing cost. The problem under study here includes multicast as a special case, where coding is in general beneficial. However, the necessity of coding heavily depends on the network structure. As the main result of this paper, we prove that when the network has a generalized tree structure – a tree where between every pair of adjacent nodes, there are two opposite-direction links with asymmetric attributes, – file splitting, instead of coding, is sufficient for achieving optimality. (The network in Fig. 1 (a) is an example of generalized trees.) In particular, if there is no constraint on the memory sizes of storage nodes, then there exists an optimal solution where the file is stored and transmitted as a whole.

The main result here improves the current knowledge on the performance gap between approaches respectively using and not using coding [3], [8]. The findings can potentially lead to solutions with substantially lower complexity and simpler forms, both of which are important for network applications.

The rest of the paper is organized as follows. In Section II, we formally define the problem, and present its transformation into a pure flow problem. In Section III, we present an interleaving algorithm for the transmission and storage of file segments in a generalized tree network, which achieves the information-theoretic bound for the file retrieval performance.

1-4244-0504-1/06/$20.00 ©2006 IEEE

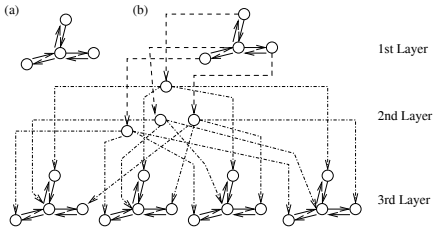

Fig. 1. (a) $G = (V, E)$ (b) new graph $H = (V_H, E_H)$

In Section IV, we prove that file splitting is sufficient for achieving optimality in generalized tree networks. In Section V, we conclude the paper.

II. Joint Storage and Transmission Problem

A. Definition of The Problem

We model the network as a directed graph $G = (V, E)$. An edge $e \in E$ from vertex u to vertex v is also denoted by (u, v). Each edge $e = (u, v) \in E$ has a length $l_e = l_{u,v} \geq 0$, which is the cost of transmitting one bit from u to v via the edge. Each vertex $v \in V$ has an associated value $m_v \geq 0$, which is the cost of storing one bit on v per unit time, and a second associated value, M_v, which is the maximum number of bits that can be stored on v (that is, its memory size). There is one particular source vertex $v_{root} \in V$, which generates an updated version of a file on average f_{update} times per unit time. The file is always of L_{file} bits. Every vertex $v \in V$ reads the file on average $f_{read}(v)$ times per unit time.

The storage and retrieval process is as follows. Every time the file is updated, v_{root} sends encoded data of the file to other vertices. Every vertex v may store some encoded data, which cannot exceed v's memory size M_v. The data that a vertex sends to its neighbor or stores on itself can be any encoding of the data that it receives. Every time a vertex v needs to read the file, it retrieves data from the network and recovers the file via decoding. The objective is to minimize the total cost, which includes the transmission cost for file update and read, as well as the storage cost. More specifically, if p_e (resp., $q_{v,e}$) bits are transmitted through the edge e when the file is updated (resp., when vertex v reads the file), and each vertex v stores x_v bits, then the objective is to obtain a storage-transmission solution that minimizes: $\sum_{e \in E} f_{update} p_e l_e + \sum_{v \in V} x_v m_v + \sum_{v \in V, e \in E} f_{read}(v) q_{v,e} l_e$.

We assume that the file update and read operations are asynchronous. So coding schemes can be used for each individual file update or read session, but not jointly for multiple sessions. We assume that the network uses standard lock/unlock mechanisms to prevent concurrent read and write (update) operations. We also assume that the entire scheme — how data are encoded, transmitted and stored — are determined beforehand, and the cost for vertices to remember it is negligible compared to the cost for constantly transmitting and storing the file data. Note that undirected networks can also be incorporated into this problem by replacing an undirected edge with two directed edges of opposite directions.

B. Transformation and Complexity

We shall show that the joint storage and transmission problem for $G = (V, E)$ can be transformed into a pure flow problem for a new graph $H = (V_H, E_H)$. An example of the transformation is shown in Fig. 1. The vertices of H are placed in three layers:

- The vertices and edges in the first layer are an exact copy of the graph G. We denote the vertices in G by $v_1, v_2, \cdots, v_n$, and correspondingly, denote the vertices in the first layer of H by $v_1^1, v_2^1, \cdots, v_n^1$. ($n$ is the number of vertices in G.) Each edge (v_i^1, v_j^1) has length $l_{(v_i,v_j)} \cdot f_{update}$ and no capacity constraint.
- The second layer of H has n vertices — $v_1^2, v_2^2, \cdots, v_n^2$ — and no edges. However, for $i = 1, 2, \cdots, n$, there is a directed edge (v_i^1, v_i^2) with length m_{v_i} and a link capacity of M_{v_i} bits per unit time.
- The third layer of H has n exact copies of G. (The *exactness* is in terms of topology, same as for the first layer.) For the i-th copy ($1 \leq i \leq n$), we denote its vertices by $u_1^i, u_2^i, \cdots, u_n^i$. Each edge (u_i^k, u_j^k) has length $l_{(v_i,v_j)} \cdot f_{read}(v_k)$ and no capacity constraint. For $i = 1, 2, \cdots, n$ and $j = 1, 2, \cdots, n$, there is a directed edge (v_j^2, u_j^i) with length 0 and no capacity constraint.

The joint storage and transmission problem for G is equivalent to a multicast problem for H, where the source is the vertex in the first layer of H corresponding to v_{root}, the sinks are $u_1^1, u_2^2, \cdots, u_n^n$, and the objective is to minimize the total link cost for the multicast. Clearly, the transmission in G for updating the file corresponds to the same transmission in the first layer of H, the storage cost paid by each vertex v_i in G is the same as the transmission cost in H from v_i^1 to v_i^2, and the file retrieval transmission in G initiated by v_i corresponds to the same transmission in the i-th copy of G in the third layer of H.

The multicast flow problem can be formulated as a linear programming problem [9], which can be solved in polynomial time. For simplicity, we skip the details of the LP formulation. Interested readers may refer to [9] for details.

III. Optimal Data Assignment with File Splitting

We proceed to study the joint storage and transmission problem for networks of generalized tree structures.

A *generalized tree* is a tree with two directed edges of opposite directions between every pair of adjacent vertices, where the two directed edges can have different lengths. An example of a generalized tree with four vertices is shown in Fig. 1 (a). We let the source v_{root} be the root of tree G. The concepts of leaf, parent, child, ancestor and descendant are defined for the vertices in the same way as in classic graph theory. We call each edge from a parent to a child a *down link*, and call each edge from a child to a parent an *up link*. We call the transmission of data from v_{root} to other vertices when the file is updated the *file update operation*, and call the transmission of data from certain vertices to a vertex $v \in V$ that is retrieving the file the *file read operation by* v.

We shall prove that file splitting is sufficient for achieving optimality. That is, by just transmitting and storing truthful pieces of the file, the same minimum cost can be obtained as by any optimal solution that uses coding. In particular, if there is an optimal solution based on coding that transmits $C_{update}(u,v)$ bits over each down link (u,v) in the file update operation and stores $D_{store}(v)$ bits in each vertex v, there exists a solution based only on file splitting that transmits at most (actually, equal to) $C_{update}(u,v)$ bits over each down link (u,v) in the file update operation, stores $D_{store}(v)$ bits in each vertex v, and incurs no more (actually, the same) cost for each file read operation than the optimal solution based on coding does. (Clearly, an optimal solution does not use *up links* in the file update operation, because a vertex cannot get any new information from its descendants during the operation.) In the following, we present such a file splitting scheme.

A. Notations and Basic Properties

We assume that the file is split into K equally long file segments, and that a file segment is the minimum unit for all transmission and storage operations. Note that a file segment can be made as short as just one bit, so no generality is lost. However, it is prevailing practice in network applications to operate at the file segment level instead of the bit level. We denote the file segments by ρ_1, ρ_2, $\cdots$, ρ_K. With a small modification of terms, we require that at most $C_{update}(u,v)$ file segments (in stead of bits) are transmitted over each down link (u,v) in the file update operation, and that each vertex v stores $D_{store}(v)$ file segments (in stead of bits). We look for the optimal solution given those two constraints.

We define a term, $D_{update}(v)$, as follows: if $v \in V$ has a parent u, then $D_{update}(v) = C_{update}(u,v)$; otherwise, v is the root v_{root}, and we let $D_{update}(v) = K$. For each vertex $v \in V$, T_v denotes the subtree of G rooted at v. We denote the *least common ancestor* of vertices u and v by $LCA(u,v)$.

The distance from $u \in V$ to $v \in V$ is the length of the path from u to v and is denoted by $\delta(u \to v)$.

Since every vertex v is to store $D_{store}(v)$ file segments, we think of v as having $D_{store}(v)$ *memory-slots*, one for each file segment. We define ϖ as $\varpi = \sum_{v \in V} D_{store}(v)$, that is, the total number of memory-slots in the tree G. We use $host(s)$ to denote the vertex that a memory slot s belongs to. We label all the memory-slots in G as s_1, s_2, $\cdots$, s_ϖ following this rule: if $\delta(host(s_i) \to v_{root}) < \delta(host(s_j) \to v_{root})$, then $i < j$.

For a memory slot s_i and a vertex v, we define the *generic distance from s_i to v*, $d(s_i \to v)$, as follows:

- $\forall$ s_i, s_j and vertex v, $d(s_i \to v) < d(s_j \to v)$ if and only if "$\delta(host(s_i) \to v) < \delta(host(s_j) \to v)$" or "$\delta(host(s_i) \to v) = \delta(host(s_j) \to v)$ and $i < j$."

There is no need to assign concrete values to generic distance. All we need is the ability to compare the generic distance from two memory slots to a vertex. If $d(s_i \to v) < d(s_j \to v)$, we say that *$s_i$ is generically closer to v than s_j is*.

For a file segment ρ_i and a vertex v, we define the *generic distance from ρ_i to v*, $d(\rho_i \to v)$, as follows:

- Assume that part of the memory slots of G have been assigned file segments. For each file segment ρ_i, we use A_i to denote the set of memory slots that are assigned ρ_i. If $A_i \neq \emptyset$, then for any vertex v, we use $\mu(\rho_i, v)$ to denote the memory slot in A_i that is the generically closest to v. Then, (1) if $A_i \neq \emptyset$ and $A_i \neq \emptyset$, $d(\rho_i \to v) < d(\rho_j \to v)$ iff $d(\mu(\rho_i, v) \to v) < d(\mu(\rho_j, v) \to v)$; (2) if $A_i \neq \emptyset$ and $A_j = \emptyset$, then $d(\rho_i \to v) < d(\rho_j \to v)$; (3) if $A_i = A_j = \emptyset$, then $d(\rho_i \to v) < d(\rho_j \to v)$ iff $i < j$.

There is again no need to assign concrete values to the generic distance that is defined above.

For i memory slots that belong to a subtree T_v, if $i > D_{update}(v)$, then we say that those i memory slots are *dependent*, because the data stored in them must be redundant, even if coding is used for both data transmission and storage. (Note that all the information stored in the i memory slots came from vertex v, which receives only $D_{update}(v)$ segments from the source.) More generally, we call a set of memory slots *dependent* if it contains a subset that is *dependent*; otherwise, the set is *independent*.

Given a set of memory slots A, we define its *dimension* to be the maximum number of independent memory slots that exist in A. It indicates the maximum amount of information we can possibly get by reading the data stored in A when coding is used. (Note that coding-based schemes include file splitting as a special case). Similar to vectors in linear space, the maximum set of independent memory slots in A can be found in a greedy way. Also, similar to vectors, we have the following observation.

Lemma 3.1. *If a set of memory slots has dimension i, then with any coding-based solution, the data stored in them contains no more than $i \cdot \frac{L_{file}}{K}$ bits of information. (Here $\frac{L_{file}}{K}$ is the number of bits in a file segment.) In particular, if only file splitting is used, then there are at most i different file segments stored in those memory slots.*

We define $B_v(i)$ to be the set of i memory slots generically closest to vertex v. That is, $|B_v(i)| = i$ and for any $s_j \in B_v(i)$ and $s_k \notin B_v(i)$, $d(s_j \to v) < d(s_k \to v)$.

B. Algorithm for Assigning File Segments

We shall present an algorithm that stores $D_{store}(v)$ file segments in each vertex v and has the property that $\forall\ u \in V$, the number of distinct file segments stored in the memory slots in T_u is at most $D_{update}(u)$. With such an assignment of file segments, at most $D_{update}(u)$ file segments need to reach $u \in V$ in the file update operation, so file splitting is sufficient for its implementation.

The algorithm assigns file segments to memory slots one by one. To decide which file segment is to be assigned to memory slot s_i, the algorithm checks if there is a subtree T_v containing s_i that has already been assigned $D_{update}(v)$ distinct file segments. If so, only those file segments serve as candidates, and the algorithm chooses the generically furthest one for s_i; otherwise, the algorithm chooses the generically

furthest file segment among all the K file segments. It will be proved that such an assignment has the nice property that any $v \in V$ can find K distinct file segments in the K closest independent memory slots, which is the best possible.

Algorithm 1. [Data Assignment for Tree $G = (V, E)$]

1. $\forall\ v \in V$, let A_v denote the set of file segments that have been assigned to memory slots in T_v. Initially, $A_v = \emptyset$.

2. for $i = 1$ to ϖ do

{ 2.1 Check the vertices in the path from $host(s_i)$ to v_{root} in order, starting with $host(s_i)$. If v is the first vertex we find such that $|A_v| = D_{update}(v)$, assign to s_i the file segment $\tau \in A_v$ such that $d(\tau \rightarrow host(s_i)) = max_{\varrho \in A_v} d(\varrho \rightarrow host(s_i))$. If no such v is found, assign to s_i the file segment τ such that $d(\tau \rightarrow host(s_i)) = max_{1 \le j \le K} d(\rho_j \rightarrow host(s_i))$.

2.2 For every vertex v in the path from $host(s_i)$ to v_{root}, let $A_v \leftarrow A_v \cup \{\tau\}$.

}

□

The above algorithm has time complexity $O(|V|^2 + \varpi|V| + \varpi K)$.

C. Optimality of Algorithm 1

In this subsection, we shall prove this following theorem, which shows that Algorithm 1 minimizes the file read cost for *all vertices* simultaneously, even when compared to optimal solutions based on coding.

Theorem 3.2. *With the file segment assignment output by Algorithm 1, for every vertex $v \in V$, the K generically closest independent memory slots are assigned K different file segments.*

We first present two lemmas.

Lemma 3.3. *If $s_i \in B_v(j)$, then any memory slot in $\{s_t | t < i\} \cap B_v(j)$ is generically closer to $host(s_i)$ than any memory slot in $\{s_t | t < i\} - B_v(j)$ is.*

Proof: Let $s_r \in \{s_t | t < i\}$ be a memory slot in the subtree rooted at $LCA(v, host(s_i))$. Since $r < i$, $\delta(host(s_r) \rightarrow v_{root}) \le \delta(host(s_i) \rightarrow v_{root})$, so $\delta(host(s_r) \rightarrow LCA(v, host(s_i))) \le \delta(host(s_i) \rightarrow LCA(v, host(s_i)))$. So $\delta(host(s_r) \rightarrow v) \le \delta(host(s_r) \rightarrow LCA(v, host(s_i))) + \delta(LCA(v, host(s_i)) \rightarrow v) \le \delta(host(s_i) \rightarrow v)$. So $d(s_r \rightarrow v) < d(s_i \rightarrow v)$. $s_i \in B_v(j)$, so $s_r \in B_v(j)$. Let s_q be any memory slot in $\{s_t | t < i\} - B_v(j)$. Then we know that s_q is not in the subtree rooted at $LCA(v, host(s_i))$.

Let s_p be any memory slot in $\{s_t | t < i\} \cap B_v(j)$. We shall prove that $d(s_p \rightarrow host(s_i)) < d(s_q \rightarrow host(s_i))$. First we assume that s_p is in the subtree rooted at $LCA(v, host(s_i))$. Then $\delta(host(s_p) \rightarrow LCA(v, host(s_i))) \le \delta(host(s_i) \rightarrow LCA(v, host(s_i)))$. Since $s_i \in B_v(j)$, $i > q$ and $s_q \notin B_v(j)$, we must have $\delta(host(s_q) \rightarrow v) > \delta(host(s_i) \rightarrow v)$, which leads to $\delta(host(s_q) \rightarrow LCA(v, host(s_i))) > \delta(host(s_i) \rightarrow LCA(v, host(s_i)))$. So $\delta(host(s_p) \rightarrow host(s_i)) \le \delta(host(s_p) \rightarrow LCA(v, host(s_i))) + \delta(LCA(v, host(s_i)) \rightarrow host(s_i)) \le \delta(host(s_i) \rightarrow LCA(v, host(s_i))) + \delta(LCA(v, host(s_i)) \rightarrow host(s_i)) < \delta(host(s_q) \rightarrow LCA(v, host(s_i))) + \delta(LCA(v, host(s_i)) \rightarrow host(s_i)) = \delta(host(s_q) \rightarrow host(s_i))$. So $d(s_p \rightarrow host(s_i)) < d(s_q \rightarrow host(s_i))$.

We now assume that s_p is not in the subtree rooted at $LCA(v, host(s_i))$. Since $s_p \in B_v(j)$ and $s_q \notin B_v(j)$, we have $d(s_p \rightarrow v) < d(s_q \rightarrow v)$. Both v and s_i are in the subtree rooted at $LCA(v, host(s_i))$, and both s_p and s_q are out of that subtree, so it is simple to see that again, $d(s_p \rightarrow host(s_i)) < d(s_q \rightarrow host(s_i))$. □

Lemma 3.4. *With the file segment assignment output by Algorithm 1, for any $v \in V$ and any i, j, the number of different file segments assigned to the memory slots in $\{s_t | t \le i\} \cap B_v(j)$ equals the dimension of $\{s_t | t \le i\} \cap B_v(j)$.*

Proof: We prove this lemma by induction on i for $i = 1, 2, \cdots, \varpi$. When $i = 1$, the lemma clearly holds. That serves as the base case. Now assume that the lemma holds for any $i < I$. ($1 < I \le \varpi$.) We shall prove that it also holds when $i = I$.

Let $i = I$. If the dimension of $\{s_t | t \le i\} \cap B_v(j)$ equals the dimension of $\{s_t | t \le i - 1\} \cap B_v(j)$, by the induction assumption and Lemma 3.1, the number of different file segments assigned to $\{s_t | t \le i\} \cap B_v(j)$ must equal the dimension of $\{s_t | t \le i - 1\} \cap B_v(j)$, so the lemma holds. In the rest of the proof, we assume that the dimension of $\{s_t | t \le i\} \cap B_v(j)$ is one more than the dimension of $\{s_t | t \le i-1\} \cap B_v(j)$. The assumption immediately implies that $s_i \in B_v(j)$, and that the dimension of $\{s_t | t \le i - 1\} \cap B_v(j)$ is less than K.

Let's recall Algorithm 1. Consider the moment when Algorithm 1 is to assign a file segment to s_i. The algorithm checks the vertices in the path from $host(s_i)$ to v_{root} in order, starting with $host(s_i)$. There are two cases.

- *Cases 1. The algorithm finds no vertex u in the path such that the subtree T_u has been assigned $D_{update}(u)$ different file segments.* So it assigns to s_i the file segment that is generically furthest to $host(s_i)$. At that moment, only memory slots in $\{s_t | t < i\}$ have been assigned file segments, and Lemma 3.3 shows that any memory slot in both $\{s_t | t < i\}$ and $B_v(j)$ is generically closer to $host(s_i)$ than any memory slot in $\{s_t | t < i\}$ but not in $B_v(j)$. Memory slots in $\{s_t | t \le i - 1\} \cap B_v(j)$ are assigned less than K different file segments, so the file segment that is generically furthest from $host(s_i)$ — which is now assigned to s_i — must be outside the set $\{s_t | t \le i - 1\} \cap B_v(j)$. So we can see that the lemma holds here.
- *Cases 2. u is the first vertex that the algorithm finds in the path such that the subtree T_u has been assigned $D_{update}(u)$ different file segments.* So the algorithm assigns to s_i the file segment in the following way: among the $D_{update}(u)$ file segments that have been assigned to T_u, select the one that is generically furthest from $host(s_i)$.

  Define F as $F = \{w | w$ is a vertex in the path from $host(s_i)$ to v_{root}, T_w has been assigned $D_{update}(w)$ different file segments at the moment right before s_i is assigned a file segment.$\}$. $F \neq \emptyset$ since $u \in F$. $\forall w \in$

F, we partition the $D_{update}(w)$ different file segments assigned to T_w into two sets P_w and Q_w, where P_w contains those segments that have also been assigned to $B_v(j) \cap \{s_t | t \leq i-1\} \cap \{\text{memory slots in } T_w\}$, and Q_w contains the rest.

$\forall w \in F$, as s_i is in T_w, the dimension of $\{s_t | t \leq i\} \cap B_v(j)$ is one more than the dimension of $\{s_t | t \leq i-1\} \cap B_v(j)$, the dimension of $B_v(j) \cap \{s_t | t \leq i-1\} \cap \{\text{memory slots in } T_w\}$ must be less than $D_{update}(w)$. So $Q_w \neq \emptyset$.

We now use contradiction to prove that $\forall w \in F$, there is a file segment in Q_w that is not assigned to $\{s_t | t \leq i-1\} \cap B_v(j)$. Assume no such file segment exists. There is a set of memory slots in T_w with indices smaller than i that are assigned file segments in Q_w. Let s_x be the one among those memory slots with the smallest index. (That is, x is the smallest.) Since $s_x \notin B_v(j)$, it is not difficult to see that v is a descendant of w. All the memory slots in $Y = B_v(j) \cap \{s_t | t \leq i-1\} - \{\text{memory slots in } T_w\}$ have indices smaller than x; and for the memory slots not in T_w, those in Y are the generically closest to v, $host(s_i)$ and $host(s_x)$. By our assumption, the file segment assigned to s_x is also assigned to memory slots in Y. $B_v(j) \cap \{s_t | t \leq i-1\}$ has less than K different file segments, so w must have an ancestor $w' \in F$, and while s_x was assigned its file segment, it was assigned the generically furthest one among those $D_{update}(w')$ file segments that had already been assigned to $T_{w'}$– and all those $D_{update}(w')$ segments are also assigned to $B_v(j) \cap \{s_t | t \leq i-1\}$. Now replace w with w' and repeat the above argument, and a contradiction will appear.

$u \in F$. So there is a file segment in Q_u that has not been assigned to $\{s_t | t \leq i-1\} \cap B_v(j)$. Such a file segment is generically further from $host(s_i)$ than any file segment assigned to $B_v(j) \cap \{s_t | t \leq i-1\}$. So the algorithm will assign to s_i a file segment different from all those assigned to $B_v(j) \cap \{s_t | t \leq i-1\}$. So the lemma holds. □

Proof of Theorem 3.2: By using Lemma 3.4 where we let $i = \varpi$, we see that the number of different file segments assigned to $B_v(j)$ equals its dimension. Increase j from 1 to ϖ and we see that the number of different file segments in $B_v(j)$ becomes K as soon as $B_v(j)$ has included the K-th generically closest independent memory slot. □

IV. Optimal Solution with File Splitting

The cost of an optimal solution based on coding to the joint storage and transmission problem is no more than the cost of a solution based on file splitting. However, given an optimal solution based on coding, we can use Algorithm 1 to find a solution based on file splitting that has the same amount of flow on each link in the file update process, stores the same number of bits in each vertex, and enables each vertex to retrieve exactly L_{file} bits of data from the closest set of independent memory slots for file read. Such a solution is clearly also optimal. Thus we have:

Theorem 4.1. *For the joint storage and transmission problem for generalized tree networks, file splitting is sufficient for achieving optimality.*

In a solution based on file splitting, the total cost is the summation of the individual cost of updating, storing and retrieving each single bit in the file. If vertices do not have constraints on their memory sizes, we can apply the same solution for the bit associated with the minimum cost to all the bits without increasing the total cost. Therefore,

Theorem 4.2. *For the joint storage and transmission problem for generalized tree networks, if vertices do not have constraints on their memory sizes, there exists an optimal solution that always transmits and stores the file as a whole piece.*

The knowledge on the optimality of file splitting can help us reduce the complexity of finding a solution. Especially, when the file is transmitted and stored as a whole, the problem reduces to the traditional file assignment problem and can be solved in low degree polynomial time [7]. Also, file splitting removes the complexity of encoding and decoding.

V. Conclusions

We have studied a joint storage and transmission problem, and shown that it can be transformed into an LP multicast problem. We have constructively proved that for generalized tree networks, file splitting is sufficient for optimality. The results deepen the current understanding on the performance between coding and no coding. As future work, we are interested in finding efficient and decentralized algorithms for general networks and study the robustness of the solutions.

References

[1] S. Acedanski, S. Deb, M. Medard, and R. Koetter. How good is random linear coding based distributed networked storage. In *Proc. NetCod*, 2005.

[2] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. on Information Theory*, 46(4):1204–1216, 2000.

[3] M. Charikar and A. Argawal. On the advantage of network coding for improving network throughput. In *Proc. IEEE Information Theory Workshop*, 2004.

[4] T. Ho, M. Medard, J. Shi, M. Effros, and D. R. Karger. On randomized network coding. In *Proc. the 41st Allerton Annual Conference on Communication, Control and Computing*, 2003.

[5] A. Jiang and J. Bruck. Multicluster interleaving on paths and cycles. *IEEE Transactions on Information Theory*, 51(2):597–611, Feb. 2005.

[6] A. Jiang and J. Bruck. Network file storage with graceful performance degradation. *ACM Transactions on Storage*, 1(2):171– 189, 2005.

[7] K. Kalpakis, K. Dasgupta, and O. Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):628–637, 2001.

[8] Z. Li and B. Li. Network coding in undirected networks. In *Proceedings of CISS*, 2004.

[9] D. S. Lun, N. Ratnakar, R. Koetter, M. Medard, E. Ahmed, and H. Lee. Achieving minimum-cost multicast: A decentralized approach based on network coding. In *Proc. the 24th IEEE INFOCOM*, March 2005.

[10] M. Medard, M. Effros, T. Ho, , and D. Karger. On coding for non-multicast networks. In *Proc. the 41st Allerton Annual Conference on Communication, Control, and Computing*, 2003.

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

[12] A. Rasala-Lehman and E. Lehman. Complexity classification of network information flow problems. In *Proc. 41st Allerton Conference on Communication, Control and Computing*, 2003.