Building Parallel Regeneration Trees in Distributed Storage Systems with Asymmetric Links

Jun Li, Shuang Yang, Xin Wang

***Abstract*—Distributed storage systems provide reliable storage service by storing data, with a certain amount of redundancy, into a substantial number of storage nodes. In order to compensate the data loss incurred by node failures, the lost data should be regenerated. Tree-structured regeneration, during which storage nodes may relay the network traffic, has shown its potential to improve the efficiency of the regeneration process in the network with symmetric links. In this paper, we consider tree-structured regeneration in the network with asymmetric links, and analyze its expected time spend during the regeneration. Moreover, we further reduce the regeneration time by constructing multiple parallel regeneration trees. We proposed two optimal algorithms with polynomial time complexity, to construct multiple edge-disjoint and multiple edge-sharing parallel regeneration trees, respectively. We evaluate our algorithms by the simulation using real data measured in PlanetLab. The simulation results show that multiple parallel regeneration trees can reduce the regeneration time by 75% and keep the file availability more than 98%.**

I. Introduction

The distributed storage system (*e.g.,* [1]), designed to provide a large-scale and reliable data storage service, stores data into a large number of storage nodes in a network. Depending on application scenarios, storage nodes may vary from cluster servers in modular data centers [2], to even ordinary computers in peer-to-peer networks [1]. The storage nodes are subject to fail, resulting in the data loss. Thus, the system needs to store a certain amount of redundancy, to guarantee that a subset of storage nodes can recover the data. MDS codes, such as Reed-Solomon codes, maintain the *recoverability property* such that any subset including at least k nodes can recover the original data.

When a storage node fails, a replacement node, called *newcomer*, should regenerate the lost data. In this process of regeneration, if any k nodes suffice to recover the original data, to maintain the recoverability property, the newcomer should receive data from at least k storage nodes, called *providers* in the regeneration process. Thus, the regeneration process is bottlenecked by the *slowest* end-to-end link from one provider to the newcomer, *i.e.,* the bottleneck link. As a matter of fact, the bottleneck link can be bypassed by the tree-structured regeneration [3], [4] using network coding, such that providers may encode and relay the traffic from other providers to the newcomer, as shown in Fig. 1(a). We proposed the tree-structured regeneration in the network with symmetric links [3] and incorporated it with regenerating codes [5] in [4].

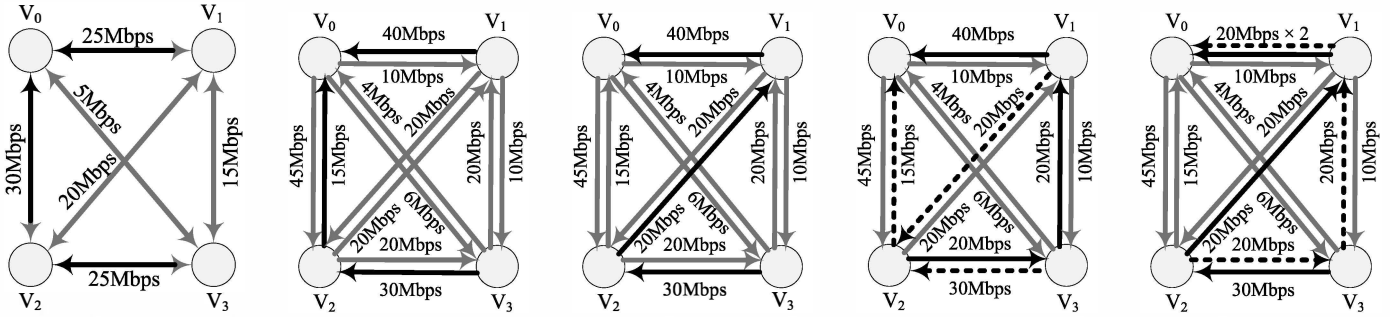
In this paper, we step forward to consider the asymmetric links in the network, *i.e.,* the bandwidth in one direction of the link is not as available as that in the other one. Lee *et al.* [6] measured the bandwidth capacity between nodes in PlanetLab [7]. The measurement results show that only 21.49% of the measured node pairs have a symmetric end-to-end link connecting them. Without the knowledge of link asymmetry, the available bandwidth of the regeneration tree will not be as good as expected, if the available bandwidth is measured in one direction or in a round trip. For example, the available bandwidth of the regeneration tree, *i.e.,* the available bandwidth of the bottleneck link, is expected to be 30Mbps in Fig. 1(a) as links are supposed to be symmetric, but in fact it is only 15Mbps if links in the network are asymmetric in Fig. 1(b). In this paper, we discuss the algorithm to construct the regeneration tree with the optimal available bandwidth. For example, the optimal regeneration tree in the network model shown in Fig. 1(c) achieves the available bandwidth of 20Mbps in the regeneration process.

In addition, we further try to utilize links more effectively during the regeneration by constructing multiple parallel regeneration trees. For example, the available bandwidth in the regeneration process can be further improved to 30Mbps in Fig. 1(d), if we construct t regeneration trees $(t > 1)$ which transmit $\frac{1}{t}$ of the whole regeneration traffic in parallel, respectively. Though the minimum available bandwidth of one parallel regeneration tree may be worse than the that of the single regeneration tree, the parallel transmission can help to reduce the time spent during the regeneration by 33%. Moreover, if some edges of several trees may share the same end-to-end link, the regeneration time can be further reduced by 25%, as shown in Fig. 1(e). In this paper, we propose two optimal polynomial algorithms to construct t edge-disjoint and edge-sharing parallel regeneration trees, respectively.

We run extensive simulations to evaluate the performance of our algorithms. We simulate a distributed storage system based on real data of available bandwidth and node behaviors measured in PlanetLab. Our simulation results show that using multiple parallel regeneration trees can reduce regeneration time to 19% of that spent by the conventional star-structured regeneration, and to 45% of that using a single regeneration tree. In the simulation of a dynamic network environment, we observe that multiple parallel regeneration trees can keep the file availability of at least 98%.

The remainder of this paper is organized as follows. In

Jun Li, Shuang Yang, and Xin Wang (contacting author: xinw@fudan.edu.cn) are with School of Computer Science, Fudan University, China. This work is supported in part by NSFC under Grant No. 60702054, 863 program of China under Grant No. 2009AA01A348, Shanghai Municipal R&D Foundation under Grant No. 09511501200, and Shanghai Rising-Star Program under Grant No. 08QA14009.



(a) An optimal regeneration tree in a network with symmetric links.

(b) A regeneration tree in a network with asymmetric links.

(c) An optimal regeneration tree in a network with asymmetric links.

(d) Two parallel regeneration trees in a network with asymmetric links.

(e) Two edge-sharing parallel regeneration trees in a network with asymmetric links.

Fig. 1. Examples of regeneration trees in the networks with symmetric and asymmetric links.

Sec. II, we introduce the background, and describe in detail the problem of the ignorance of bandwidth asymmetry and the advantage of multiple regeneration trees. Sec. III introduces the network model and discusses the construction and the expected regeneration time of the optimal regeneration tree. In Sec. IV, we discuss the construction algorithms of t edge-disjoint and t edge-sharing parallel regeneration trees. We introduce the simulation results in Sec. V. Finally, Sec. VI concludes this paper.

II. Preliminaries

A. Redundancy and linear network coding

In the distributed storage system, data are stored in a number of storage nodes. A storage node may save a whole file or a part of a file. Since the nodes may leave the network temporarily or even fail, data should be stored with redundancy such that a subset of storage nodes can recover the original data. If the size of a file is $\mathcal{M}$ bits, the storage nodes in the network may store totally $\frac{3\mathcal{M}}{2}$ bits.Thus, $\frac{\mathcal{M}}{2}$ bits are stored as the redundancy and the redundancy rate r is 1.5.

Compared with storing replicas of the original data in the storage nodes, encoding the original data into coded blocks can improve the data availability [8]. Maintaining the recoverability property with a high probability, Acedanński *et al.* [9] introduced randomized linear network coding as a way of achieving coded blocks. Dimakis *et al.* [10] presented deterministic linear network coding to achieve the redundancy. If a file is divided into k blocks, $B_1, B_2, \ldots, B_k$, a coded block F is a linear combination of the k blocks on a Galois field $\mathbb{F}_{2^q}$. q bits are regarded as a symbol on $\mathbb{F}_{2^q}$ and a block is then a sequence of symbols on $\mathbb{F}_{2^q}$. Thus we have

$$F = \sum_{i=1}^{k} a_i B_i, \tag{1}$$

where $(a_1, a_2, \ldots, a_k)^T$ is the randomly-generated encoding vector, $a_i \in \mathbb{F}_{2^q}, i = 1, 2, \ldots, k$. If q is large enough, any k coded blocks are sufficient to recover the original blocks with high probability. Given k coded blocks and their encoding vectors, we can reconstruct the original blocks by solving a linear system of k equations.

Without loss of generality, we assume that each storage node stores one coded block, such that any k storage nodes can recover the original data. Apart from the coded block, a storage node has to store the corresponding encoding vector. However, this overhead can be ignored if the block is large enough. Since the size of one coded block is equal to the size of one original block, the redundancy rate is $\frac{n}{k}$ if there are n coded blocks in the network.

B. Tree-structured data regeneration

Nodes may fail in the distributed storage system and thus the data loss may be incurred. The redundancy rate will decrease or even turn to zero if node failures are not handled. In order to regenerate the lost coded block, the system should select a node, called *newcomer*, to replace the failed storage node. The newcomer should receive k' coded blocks, $F_1, F_2, \ldots, F_{k'}$, from k' active storage nodes, called *providers*, and get a new coded block F_0 as a linear combination of the received blocks:

$$F_0 = \sum_{i=1}^{k'} a'_i F_i, \tag{2}$$

$a'_i \in \mathbb{F}_{2^q}, i = 1, 2, \ldots, k$. To keep the recoverability property, k' should be at least k. However, considering the bandwidth cost and the node behaviors, a small number of providers is preferred in the distributed storage system [11], so in this paper we only considers the scenario that $k' = k$. Wu *et al.* [5] showed that the minimum regeneration traffic when $k' = k$ is $\mathcal{M}$ bits, the size of the original file. This optimal traffic can be achieved easily when the newcomer receives k coded blocks from k storage nodes.

Even though the minimum regeneration traffic has been achieved, we can further reduce the regeneration time by exploiting the bandwidth diversity. In an overlay storage network, links between storage nodes usually enjoy different available bandwidth. Conventionally the regeneration is carried out in a manner called star-structured regeneration. For example, Fig. 1(a) shows a regeneration in a network with symmetric

links, in which V_0 is the newcomer and other three nodes are providers, *i.e.,* $k = 3$. If the newcomer receives the coded blocks directly from the three providers, the transmission will finish when the newcomer has received the coded block from V_3, and the regeneration time is thus $\frac{\mathcal{M} \text{ bits}}{3\times 5 \text{ Mbps}}$.

Moreover, we can allow providers to relay the regeneration traffic. The darkened directed edges in Fig. 1(a) indicate how data are transmitted in the network. For example, V_2 receives data from V_3, encodes the received blocks with the coded blocks stored by itself, and then sends the coded data to the newcomer as long as there are data available to send, rather than after the whole block is encoded. The edges used in the transmission form a spanning tree over the newcomer and the providers, called *regeneration tree*. The bottleneck edge of a regeneration tree is the edge with the minimum available bandwidth, and the available bandwidth of a regeneration tree is thus the available bandwidth of the bottleneck edge. In Fig. 1(a), (V_1, V_0) and (V_3, V_2) are bottleneck edges and the regeneration time becomes one forth of the star-structured regeneration.

C. Regeneration in the network with asymmetric links

The links in the network model in Fig. 1(a) are all symmetric, *i.e.,* the available bandwidth in one direction is the same as the that in the other direction, but in reality, this is improbable. The measurement in PlanetLab [6] shows that at least 78.51% of links are asymmetric. Fig. 1(b) shows a network model with asymmetric links. The available bandwidth of each edge in Fig. 1(a) is the average available bandwidth of corresponding edges in two directions in Fig. 1(b). If the link asymmetry is neglected and the bandwidth is measured in a round trip, the network model will still be the same as Fig. 1(a), but the available bandwidth of the regeneration tree will be reduced to 15Mbps, as shown in Fig. 1(c).

In addition, multiple parallel links are unlikely to saturate the available bandwidth of the node in the Internet, because the end-to-end bottleneck usually occurs in the intra-AS or inter-AS link [12]. Thus we can further use the remaining available links and even their remaining available bandwidth to construct more regeneration trees in the network. For example, two parallel regeneration trees are constructed in Fig. 1(d). The two trees have the available bandwidth of 20Mbps and 15Mbps, respectively. If we let the first tree regenerates 57.1% of the coded block and the second tree regenerates the other 42.9% in parallel, the regenerate time will be reduced to $\frac{\mathcal{M} \text{ bits}}{3\times 35 \text{ Mbps}}$. However, this way of constructing multiple parallel trees by the unequal partition of coded blocks requires each tree to know the available bandwidth, which changes frequently in the network, of all other trees. If we just divide the coded block into two equal parts, the regeneration time will still be $\frac{\mathcal{M} \text{ bits}}{3\times 2\times 15 \text{ Mbps}}$. Moreover, since the actual transmission rate in a regeneration tree is the available bandwidth of its bottleneck edge, there is spare bandwidth in most edges. The two edge-sharing parallel regeneration trees in Fig. 1(e) achieve the regeneration time as little as $\frac{\mathcal{M} \text{ bits}}{3\times 2\times 20 \text{ Mbps}}$, even though the coded blocks are equally partitioned for the two trees. Thus in this paper, we first discuss how to construct the optimal regeneration tree in the network with asymmetric links and analyze its expected regeneration time in Sec. III. Then in Sec. IV, we discuss the construction of edge-disjoint and edge-sharing parallel regeneration trees with the equal partition of coded blocks.

III. Tree-structured regeneration

A. Network model and the regeneration tree

Assume linear network coding is used to produce redundancy in a distributed storage system. The original file is divided into k blocks and encoded into more than k coded blocks. Each storage node stores one coded block. In a regeneration, a newcomer receives k coded blocks from k providers. We define the node set $V(k) = \{V_0, V_1, \ldots, V_k\}$, where V_0 denotes the newcomer and other nodes denote the providers.

In the network, links between nodes are asymmetric. We define two directed edges between two nodes V_i and V_j, $i \neq j$. The edge (V_i, V_j) denotes the end-to-end link from V_i to V_j. The edge set $E(k) = \{(V_i, V_j) | i, j \in [0..k], i \neq j\}$. The weight of (V_i, V_j), $\omega(V_i, V_j)$, denotes the available bandwidth from V_i to V_j.

In this paper, the network model is represented as a directed complete graph $G(k) = (V(k), E(k), \omega)$. We assume that for each node in $V(k)$, multiple connections with some other nodes can not saturate its available bandwidth capacity. Fig. 1(b) is an example of the network model $G(3)$.

We can construct a regeneration tree in a network model $G(k)$. In a symmetric network model such as Fig. 1(a), a regeneration tree is a spanning tree rooted at the newcomer [3], as the newcomer should receive coded blocks from all the providers. Similarly, we give the definition of the regeneration tree in $G(k)$.

Definition 1: In a network model $G(k)$, a regeneration tree is a reverse arborescence rooted at the newcomer, *i.e.,* a directed spanning tree rooted at the newcomer such that the edges are directed towards the newcomer.

Since every edge in a regeneration tree in $G(k)$ is directed towards the newcomer, every provider has a directed path to the newcomer. Thus the newcomer can receive coded data from all the providers in $G(k)$. Moreover, as we show in Fig. 1(b) and Fig. 1(c), the regeneration time is bottlenecked by the minimum edge in the regeneration tree. We do not count the initial delay, *i.e.,* the time from establishing connections to the arrival of the first byte at the newcomer, since the size of the coded block is usually large enough in distributed storage systems.

Definition 2: The available bandwidth of a regeneration tree T in $G(k)$ is the weight of the minimum edge in T.

In order to reduce the regeneration time, our objective is to find the optimal regeneration tree in $G(k)$, *i.e.,* the regeneration tree with the maximum available bandwidth. This is equivalent to finding the maximum bottleneck spanning tree in a graph. Gabow and Tarjan [13] proposed an algorithm of this problem by the dichotomic search, shown in Algorithm 1. The time complexity of Algorithm 1 is $O(|E(k)| \log(V(k))) = O(k^2 \log k)$.

Algorithm 1 Splitting algorithm for the problem of the optimal regeneration tree in $G(k)$ [13]. Let $E(k) = \{e_1, e_2, \ldots, e_{k(k+1)}\}$, where $\omega(e_1) \geq \omega(e_2) \geq \ldots \geq \omega(e_{k(k+1)})$.

$\theta_1 \leftarrow 1$, $\theta_2 \leftarrow |E(k)|$
while $\theta_1 \neq \theta_2$ **do**
  $m \leftarrow \lfloor \frac{1}{2}(\theta_1 + \theta_2) \rfloor$
  $E_1 \leftarrow \{e | \omega(e_m) \leq \omega(e), e \in E(k)\}$
  if every node in $G(V(k), E_1)$ has a directed path towards V_0 **then**
    $\theta_2 \leftarrow m$
  else
    $\theta_1 \leftarrow m + 1$
  end if
end while
$\lambda^* \leftarrow \omega(e_{\theta_1})$
return a spanning tree in $G(V(k), \{e | \omega(e) \geq \lambda^*\})$

B. Analysis of regeneration time

In this section, we investigate the gain of the optimal regeneration tree, from the perspective of the regeneration time. For the tree-structured regeneration, we have shown in Sec. III-A that the regeneration time is bottlenecked by the minimum edge in the regeneration tree.

Definition 3: The regeneration time is the ratio of the size of a coded block to the available bandwidth of the regeneration tree.

Given a network model $G(k) = \{V(k), E(k), \omega\}$, if we have known the distribution function of ω, we can know the expected value of the regeneration time. If the size of the original file is $\mathcal{M}$ bits, the size of a coded block is thus $\frac{\mathcal{M}}{k}$ bits, so the regeneration time is $\frac{\mathcal{M}}{k \cdot \omega(e_b)}$, where e_b is the bottleneck edge of the optimal regeneration tree.

Property 1: If $e_b = e_i$ in $G(k)$, *i.e.,* e_b is the i-th maximum edge in $E(k)$, then $k \leq i \leq k^2 + 1$.

Proof: In $G(k)$ there are totally $k + 1$ nodes and a regeneration tree thus has k edges, so e_b is at most the k-th maximum edge. On the other hand, since $G(k)$ is a complete graph, every provider has k edge-disjoint directed paths towards the newcomer. Thus no provider will be disconnected from the newcomer until at least k edges in $G(k)$ are removed, so $i \leq k(k+1) - k + 1 = k^2 + 1$. ■

Let $p(k+1, i)$ denote the probability that $e_b = e_i$ in $G(k)$, and $E[i, k(k+1)]$ the expected weight of the i-th maximum edge in $E(k)$. The expected regeneration time of the tree-structured regeneration is

$$t_{tree} = \sum_{i=k}^{k^2+1} \frac{\mathcal{M}}{k} \cdot \frac{p(k+1, i)}{E[i, k(k+1)]} \tag{3}$$

Given the distribution of ω, we can get $E[i, k(k+1)]$ by order statistics [14]. Assume $F(x)$ and $f(x)$ are curriculum distribution function and probability density function of ω, respectively.

$$E[i, k(k+1)] = \int_0^{+\infty} x \cdot \frac{n! F^{k(k+1)-i}(x)[1 - F(x)]^{i-1} f(x)}{(k(k+1) - i)!(i-1)!} dx. \tag{4}$$

Now we discuss the distribution of $p(k+1, i)$ in $G(k)$. According to Property 1, $p(k+1, i) = 0$ when $i < k$ or $i > k^2 + 1$.

Lemma 1: Let $Q(l, j)$ denote the number of directed graphs containing l labeled nodes and j directed edges, such that every node in the graph has a directed path towards one specific root node. When $l - 1 \leq j \leq l(l-1)$,

$$Q(l, j) = \binom{l(l-1)}{j} \sum_{i=l-1}^{j} p(l, i). \tag{5}$$

Otherwise $Q(l, j) = 0$.

Proof: When $j < l-1$, such a graph can not be connected. When $j > l(l-1)$, it is impossible since there are at most $l(l-1)$ edges in the graph. Therefore, $Q(l, j) = 0$ when $j < l - 1$ or $j > l(l-1)$.

Now we discuss the case when $l - 1 \leq j \leq l(l-1)$. For one graph satisfying the condition, we let the weight of the j edges be $l(l-1)$, $l(l-1) - 1$, ..., and $l(l-1) - (j-1)$, respectively. Then we add $l(l-1) - j$ edges into the graph to make it become a complete graph, and let the weight of the added edges be 1, 2, ..., and $l(l-1) - j$, respectively. Thus we can map this graph to $j!(l(l-1) - j)!$ $G(l-1)$s, in which $e_b = e_{j'}, l - 1 \leq j' \leq j$. Since the number of the graphs satisfying the condition is $Q(l, j)$, the number of $G(k)$, in which $e_b = e_{j'}, l - 1 \leq j' \leq j$, is $Q(l, j) j!(l(l-1) - j)!$.

On the other hand, given a network model $G(k)$, the probability that $e_b = e_{j'}, l - 1 \leq j' \leq j$ is $\sum_{i=l-1}^{j} p(l, j)$, so the number of such $G(k)$s is $(l(l-1))! \sum_{i=l-1}^{j} p(l, j)$.

Connecting the two parts above, we have the following equation:

$$Q(l, j) j!(l(l-1) - j)! = (l(l-1))! \sum_{i=l-1}^{j} p(l, j) \tag{6}$$

Therefore, $Q(l, j) = \binom{l(l-1)}{j} \sum_{i=l-1}^{j} p(l, i)$. ■

Theorem 1: If $k \leq i \leq k^2 + 1$, $p(k+1, i) =$

$$\frac{\frac{1}{k+1} \sum_{l=1}^{k} \binom{k-1}{l-1} R(k, l, i) + \frac{k-1}{k+1} \sum_{l=1}^{k-1} \binom{k-2}{l-1} R(k, l, i)}{\binom{k(k+1)-1}{i-1}}, \tag{7}$$

where $R(k, l, i) =$

$$\sum_{\substack{j_1 + j_2 + j_3 = i - 1 \\ j_1, j_2, j_3 \geq 0}} \binom{l(k+1-l)}{j_3} Q(l, j_1) Q(k+1-l, j_2). \tag{8}$$

Proof: Assume in $G(k)$ $e_b = e_i, k \leq i \leq k^2 + 1$. There are two possibilities of the position of e_i.

a. e_i points to the newcomer, *i.e.,* $e_i = (V_t, V_0), 1 \le t \le k$. The probability of this case is apparently $\frac{k}{k(k+1)} = \frac{1}{k+1}$. Let $E = \{e_1, e_2, \ldots, e_i\}$. In this situation, e_i is the bottleneck edge if and only if in $G(V(k), E)$, for each nodes except V_0 and V_t, there are at least one directed path towards V_t or V_0. Assume there are l nodes, including V_t itself, having directed paths towards V_t, $1 \le l \le k$. We define $V_{(1)}$ to be the set of these l nodes and $V_{(2)}$ the set of other $k+1-l$ nodes. Apart from V_t, the number of selecting $l-1$ nodes from $k-1$ providers into $V_{(1)}$ is $\begin{pmatrix} k-1 \\ l-1 \end{pmatrix}$. To make e_i be the bottleneck edge, the edges in $E/\{e_i\}$ can be between two nodes in $V_{(1)}$, or between two nodes in $V_{(2)}$, or from one node in $V_{(2)}$ to one node in $V_{(1)}$. Assume there are j_1 edges between two nodes in $V_{(1)}$, j_2 edges between two nodes in $V_{(2)}$, and j_3 from one node in $V_{(2)}$ to one node in $V_{(1)}$. According to Lemma 1, the number of such assignments is $\begin{pmatrix} l(k+1-l) \\ j_3 \end{pmatrix} Q(l, j_1) Q(k+1-l, j_2)$. Summarizing the numbers of all possibilities of j_1, j_2 and j_3, we get the total number of the possibilities of the edge assignments and denote it by $R(k, l, i)$. Considering the number of assigning $i-1$ edges in $E - \{e_i\}$ into $k(k+1)-1$ positions is $\begin{pmatrix} k(k+1)-1 \\ i-1 \end{pmatrix}$ and summarizing all the possibilities of $V_{(1)}$ and $V_{(2)}$, we get the probability that E_i is the bottleneck edge when $e_i = (V_t, V_0)$ is

$$p_a(k+1, i) = \frac{\sum_{l=1}^{k} \begin{pmatrix} k-1 \\ l-1 \end{pmatrix} R(k, l, i)}{\begin{pmatrix} k(k+1)-1 \\ i-1 \end{pmatrix}}. \quad (9)$$

b. The other case is that e_i is from one provider to another provider, *i.e.,* $e_i = (V_{t_1}, V_{t_2}), 1 \le t_1, t_2 \le k, t_1 \ne t_2$. The probability of this case is $\frac{k(k-1)}{k(k+1)} = \frac{k-1}{k+1}$. In $G(V(k), E)$, we let $V_{(1)}$ to be the set of the nodes which have a directed path towards V_{t_1}, including V_{t_1} itself, and $V_{(2)}$ the set of other nodes in $V(k)$. Assume there are l nodes in $V_{(1)}$, $1 \le l \le k+1-2 = k-1$, so the number of the possibilities of selecting $l-1$ nodes in $V_{(1)} - \{V_{t_1}\}$ from $k+1-3 = k-2$ nodes in $V(k) - \{V_0, V_{t_1}, V_{t_2}\}$ is $\begin{pmatrix} k-1 \\ l-1 \end{pmatrix}$. Similar with the proof above, when $e_i = (V_{t_1}, V_{t_2})$, the probability that E_i is the bottleneck edge is

$$p_b(k+1, i) = \frac{\sum_{l=1}^{k-1} \begin{pmatrix} k-2 \\ l-1 \end{pmatrix} R(k, l, i)}{\begin{pmatrix} k(k+1)-1 \\ i-1 \end{pmatrix}}. \quad (10)$$

Notice that e_i is impossible to be from V_0 to another node, because the newcomer just receives data from other nodes in the regeneration process. Therefore, $p(k+1, i)$, the probability that e_i is the bottleneck edge in $G(k)$ is

$$p(k+1, i) = \frac{1}{k+1} p_a(k+1, i) + \frac{k-1}{k+1} p_b(k+1, i). \quad (11)$$

■

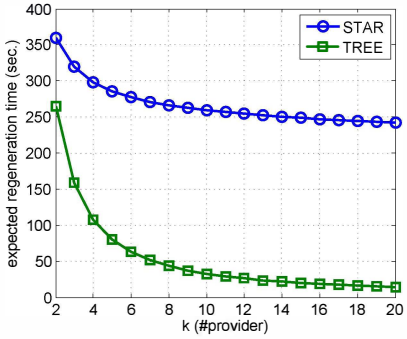


Fig. 2. Expected regeneration time of STAR and TREE in $G(k)$. The available bandwidth of the link satisfies the uniform distribution U[0.3Mbps, 140Mbps].

For the star-structured regeneration in a network model $G(k)$, however, it only uses the k edges pointing to the newcomer V_0, so the bottleneck edge is clearly the minimum edge among the k edges. Thus the expected regeneration time is

$$t_{star} = \frac{\mathcal{M}}{k} \cdot \frac{1}{E[k, k]} \quad (12)$$

Fig. 2 shows the expected regeneration time of the tree-structured regeneration (TREE) and the star-structured regeneration (STAR). We assume the available bandwidth of links satisfies a uniform distribution of U[0.3Mbps, 140Mbps]. The size of the original file is 4GB. We calculate the expected regeneration time of TREE and STAR by (3) and (12). We notice that compared with STAR, the regeneration time can be reduced by more than 90% when $k \ge 12$.

IV. Multiple Regeneration Trees

In this section, we further reduce the regeneration time by utilizing more links in the network to construct multiple regeneration trees, that works in parallel during the regeneration. For example, if we construct two parallel regeneration trees in a network model $G(k)$, we can divide the coded block in each provider into two parts. Since every regeneration tree is a spanning tree, every provider should belong to both trees. Therefore, the newcomer can receives the first part of the regenerated block from the first tree, and the second part from the second tree. The regeneration time is thus determined by the regeneration tree with more regeneration time.

Assume we can construct t parallel regeneration trees, $T_1, T_2, \ldots, T_t$. Let ω_i denote the available bandwidth of T_i, $i = 1, 2, \ldots, t$. In this paper, we consider the equal partition of the coded block and thus construct t regeneration trees that maximize $\min\{\omega_1, \omega_2 \ldots, \omega_t\}$. We start from the construction algorithm of the edge-disjoint parallel regeneration trees, and then extend it to construct edge-sharing parallel regeneration trees.

A. Edge-disjoint parallel regeneration trees

We start from constructing the optimal t edge-disjoint parallel regeneration trees in $G(k) = (V(k), E(k), \omega)$. Given a constant t, the goal is to find t edge-disjoint trees in $G(k)$, denoted by $T_1, T_2, \ldots, T_t$. $T_i = (V(k), E_i, \omega), E_i \subset E(k), i =$

$1, 2, \ldots, t$. For each tree T_i, $i = 1, 2, \ldots, t$, it is a regeneration tree such that every provider in $G(k)$ has a directed path towards the newcomer V_0. $E_i \cap E_j = \emptyset$ when $i \neq j$. Define ω_i as the available bandwidth of T_i. Since the t trees are edge-disjoint, the available bandwidth of each tree will not interfere with other trees. Since each regeneration tree is responsible for regenerating one equal part of the coded block, the regeneration time of the whole coded block is thus the regeneration time of the regeneration tree with the minimum available bandwidth, *i.e.,*

$$t_{mult} = \frac{\mathcal{M}}{kt} \cdot \frac{1}{\min\limits_{1 \leq i \leq t} \{\omega_i\}}. \tag{13}$$

The problem of achieving the optimal regeneration time is equivalent to that of maximizing $\min\limits_{1 \leq i \leq t} \{\omega_i\}$. Because $\omega_i = \min\limits_{e \in E_i} \{\omega(e)\}$, it is further equivalent to the problem of maximizing $\min\limits_{e \in E_i, 1 \leq i \leq t} \{\omega(e)\}$.

Definition 4: Let $E^* = \bigcup\limits_{i=1}^{t} E_i$ be the edge set of the t edge-disjoint parallel regeneration trees. The available bandwidth λ^* is $\min\limits_{e \in E^*} \{\omega(e)\}$.

Notice that there can be many trees with the same available bandwidth. However, since the regeneration time depends on the weight of the bottleneck edge, rather than the structure of the regeneration tree, we first find the maximum λ^* and the corresponding bottleneck edge, and then construct t regeneration trees from the edges with weight larger than λ^*.

The size of feasible solution space is $|E(k)|$, since λ^* equals the weight of one edge in $E(k)$. We design a dichotomic algorithm, similar with Algorithm 1, to find the available bandwidth λ^* in the network model $G(k)$ which runs in $O(t^2k^4 \log k)$ time. Assuming t edge-disjoint parallel regeneration trees exist in $G(k)$, the t trees with the maximum λ^* are constructed in Algorithm 2.

Algorithm 2 Splitting algorithm for the problem of finding t edge-disjoint parallel regeneration trees in $G(k)$.

```
1: θ1 ← 1, θ2 ← |E(k)|
2: while θ1 ≠ θ2 do
3:     m ← ⌊(1/2)(θ1 + θ2)⌋
4:     E1 ← {e|ω(e_m) ≤ ω(e), e ∈ E(k)}
5:     if t edge-disjoint parallel regeneration trees exist in
       G(V(k), E1) then
6:         θ2 ← m
7:     else
8:         θ1 ← m + 1
9:     end if
10: end while
11: λ* ← ω(e_{θ1})
12: return t regeneration trees in G(V(k), {e|ω(e) ≥ λ*})
```

From Line 2 to Line 10, we search the possible bottleneck weight value dichotomically. Since there are altogether $|E(k)|$ different values, all steps run at most $\log |E(k)|$ times. Line 5 checks the existence of t mutually edge-disjoint spanning trees, which has been studied by Tarjan [15]. The time complexity of this step is $O(t^2|E(k)|^2)$. In a word, the dichotomic search can find the maximum λ^* in $O(t^2|E(k)|^2 \log |E(k)|) = O(t^2k^4 \log k)$ time. After getting λ^*, we can find t edge-disjoint regeneration trees with available bandwidth no less than λ^* by the algorithm proposed in [16]. The time complexity of Line 12 is $O(\log(|V(k)|t) \cdot |V(k)|^2 \cdot |E(k)| \cdot \log(|V(k)|^2/|E(k)|) = O(\log(kt)k^4)$. Thus, the overall time complexity is $O(t^2k^4 \log k)$.

Theorem 2: Algorithm 2 produces t edge-disjoint parallel regeneration trees that achieve the maximum available bandwidth.

Proof: Since the correctness of Line 12 has been investigated in [16], we will show the correctness of the dichotomic search. If λ^* is the maximum available bandwidth, it is impossible to construct t edge-disjoint parallel regeneration trees in graph $G(V(k), \{e|e \in E(k), \omega(e) \geq \omega(e_{\theta_2})\})$, where $\omega(e_{\theta_2}) > \lambda^*$, otherwise the bottleneck $\geq \omega(e_{\theta_2}) > \lambda^*$. On the other hand, it is always possible to construct t edge-disjoint parallel regeneration trees in graph $G(V(k), \{e|e \in E(k), \omega(e) \geq \omega(e_{\theta_1})\})$, where $\omega(e_{\theta_1}) \leq \lambda^*$. Therefore, λ^* is the threshold whether t parallel edge-disjoint regeneration trees exist. Since there are finite feasible solutions, the available bandwidth λ^* can be found by the dichotomic search. ■

B. Edge-sharing parallel regeneration trees

Algorithm 2 shows how to construct multiple edge-disjoint parallel regeneration trees. However, for each edge in a regeneration tree, its available bandwidth will not be used up until it is the bottleneck edge of the tree. Thus though many links in the network may be used, the available bandwidth of these links is not fully utilized. The algorithm in this section allows links to be shared by several regeneration trees, so as to fully utilize the available bandwidth.

Given t regeneration trees, $T_i = (V(k), E_i), E_i \subset E(k), i = 1, 2, \ldots, t$, in $G(k) = (V(k), E(k), \omega)$, we assume the available bandwidth of T_i is ω_i. There are t edge-sharing parallel regeneration trees if and only if $\forall e \in E(k)$, $\sum\limits_{E_i \ni e} \omega_i \leq \omega(e)$. The regeneration time is also bottlenecked by the regeneration tree with the minimum available bandwidth, so (13) still holds and our objective remains finding the maximum λ^*.

Property 2: Given the optimal available bandwidth λ^* of t edge-sharing parallel regeneration trees in $G(k) = (V(k), E(k), \omega)$, for any edge $e \in E(k)$, define $n_{\lambda^*}(e) = \lfloor \frac{\omega(e)}{\lambda^*} \rfloor$. $n_{\lambda^*}(e) \geq |\{T_i(V(k), E_i)|e \in E_i, i = 1, 2, \ldots, t\}|$.

Proof: For each $e \in E(k)$, $\frac{\omega(e)}{\lambda^*} \geq \frac{\omega(e)}{\min\limits_{E_i \ni e} \{\omega_i\}} \geq |\{T_i(V(k), E_i)|e \in E_i, 1 \leq i \leq t\}|$. Considering the size of a set is always an integer, the floor can be achieved. ■

According to Property 2, if there exist t edge-sharing parallel regeneration trees with available bandwidth λ^*, at most $n_{\lambda^*}(e)$ connections can be established in one edge e. In other words, we can split e into $n_{\lambda^*}(e)$ edges, each with available bandwidth no worse than λ^*. The problem then becomes checking whether t edge-disjoint parallel regeneration trees exist, and thus can be solved by [16].

On the other hand, we might place at most t connections into one edge in the graph, since we only construct t edge-sharing parallel regeneration trees. Define the feasible solution space $W = \bigcup_{e \in E(k)} \{\omega(e), \frac{1}{2}\omega(e), \frac{1}{3}\omega(e), \ldots, \frac{1}{t}\omega(e)\}$, as the available bandwidth of at least one edge should be used up. Clearly, $|W| \leq t|E(k)|$. Let $W = \{w_1, w_2, \ldots, w_{|W|}\}, w_1 \geq w_2 \geq \ldots, w_{|W|}\}$.

Similar to the algorithm for finding multiple edge-disjoint parallel regeneration trees, we can use a dichotomic method in Algorithm 3 to find the maximum available bandwidth λ^* in $G(k)$, which runs in $O(t^2|E(k)|^2 \log(t|E(k)|)) = O(t^2k^4 \log(tk))$ time as $O(|W|) = O(t|E(k)|)$. We assume one spanning tree exists in $G(k)$, since we can split the only one tree into t trees by splitting each of its edges into t edges.

Algorithm 3 Splitting algorithm for the problem of finding t edge-sharing parallel regeneration trees in $G(k)$.

$\theta_1 \leftarrow 1$, $\theta_2 \leftarrow |W|$
while $\theta_1 \neq \theta_2$ **do**
  $m \leftarrow \lfloor \frac{1}{2}(\theta_1 + \theta_2) \rfloor$
  Let E_1 be a multiset of $S(w_m) = \{e \times n_{w_m}(e) | e \in E(k)\}$
  if t edge-disjoint parallel regeneration trees exist in $G(V(k), E_1)$ **then**
    $\theta_2 \leftarrow m$
  else
    $\theta_1 \leftarrow m + 1$
  end if
end while
$\lambda^* \leftarrow w_{\theta_1}$
return t regeneration trees in $G(V(k), S(\lambda^*))$

Theorem 3: Algorithm 3 produces t edge-sharing parallel regeneration trees that achieve the maximum available bandwidth.

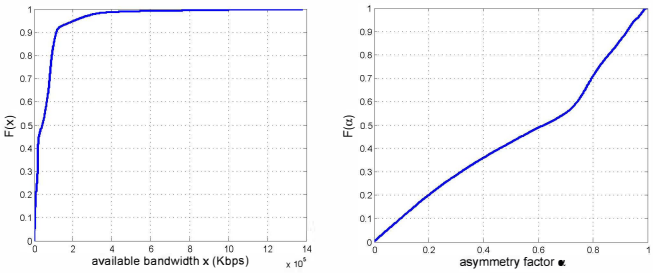
Proof: If λ^* is the maximum available bandwidth, it is impossible to construct t edge-sharing spanning trees in graph $G(V, S_{w_{\theta_2}})$, where $w_{\theta_2} > \lambda^*$, otherwise the available bandwidth $\geq w_{\theta_2} > \lambda^*$. Also, it is always possible to construct t edge-sharing parallel regeneration trees in graph $G(V, S_{w_{\theta_1}})$, where $w_{\theta_1} \leq \lambda^*$, since $S_{\lambda^*} \subseteq S_{w_{\theta_1}}$. Therefore, λ^* is the threshold whether t edge-sharing parallel regeneration trees exist. Since there are finite feasible solutions, the maximum available bandwidth λ^* can be found by the dichotomic search. ■

V. Simulation

In this section, we make extensive empirical studies to evaluate the performance of the tree-structured regeneration algorithms based on the real data of available bandwidth [17] and node behaviors [18] measured in PlanetLab [7], a global research network in which most links are asymmetry.

The main results we observe from the simulation results include:

- The regeneration time can be greatly reduced by constructing multiple parallel regeneration trees. Because of this, the file can be regarded as almost available in a dynamic network environment with the node join and departure.
- The edge-sharing algorithm performs better than the edge-disjoint algorithm, both on the available bandwidth and on the resistance to the high asymmetry factor.



(a) available bandwidth (b) asymmetry factor

Fig. 3. Curriculum distribution functions of available bandwidth and asymmetry factor in PlanetLab.

A. Static network environment

We first simulate a static network, *i.e.,* storage nodes in the network are supposed to be static and there is no node join/departure during the regeneration. The network topology is constructed based on real data of available bandwidth measured in PlanetLab by S^3 (Scalable Sensing Service) [17]. If k providers are used in the regeneration process, we construct a network containing one newcomer and k providers, and then we assign the weight of each link, using the available bandwidth measured in S^3 by the tool of pathChirp [19] on 16:28:10 (Pacific Time), Nov. 11, 2009.

We use the asymmetry factor to denote the asymmetry of links. Assume the available bandwidth in the two directions of one link is BW_1 and BW_2, the asymmetry factor is

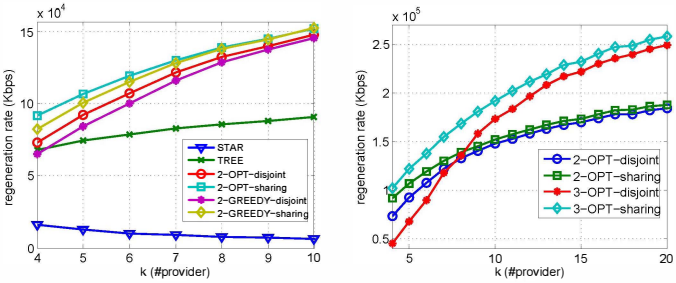
$$\alpha = \frac{|\mathrm{BW}_1 - \mathrm{BW}_2|}{\max\{\mathrm{BW}_1, \mathrm{BW}_2\}}. \quad (14)$$

It is clear that $0 \leq \alpha \leq 1$. Fig. 3 shows the curriculum distribution functions of the available bandwidth and the asymmetry factor in PlanetLab, respectively. We use the algorithms mentioned in this paper to construct regeneration tree(s) in the network and the simulation is repeated for 1000 times.

The algorithms we compare contain star-structured regeneration (STAR), tree-structured regeneration (TREE) in Sec. III, for the case of single regeneration tree. For the case of multiple parallel regeneration trees, we compare our optimal algorithms proposed in Sec. IV and two greedy algorithms. Our optimal algorithms are denoted as t-optimal-edge-disjoint algorithm (t-OPT-disjoint), and t-optimal-edge-sharing algorithm (t-OPT-sharing). The greedy algorithms are denoted as t-greedy-edge-disjoint algorithm (t-GREEDY-disjoint) and t-greedy-edge-sharing algorithm (t-GREEDY-sharing).

Now we introduce the greedy algorithms used in the simulation. Let t=2, for example. For the 2-GREEDY-disjoint algorithm, the first regeneration tree is constructed as the optimal regeneration tree by TREE, and then we remove the edges

in the first regeneration tree from the network. The second regeneration tree is then constructed by TREE on the network with remaining edges. For the 2-GREEDY-sharing algorithm, the edges of the two parallel regeneration trees can share the same link. Thus after constructing the first regeneration tree by TREE, we decrease the weight of edges in the first regeneration tree by the available bandwidth of the first tree and then construct the second tree by TREE. (13) still holds for the greedy algorithms, as we partition the coded blocks into size-equal blocks.

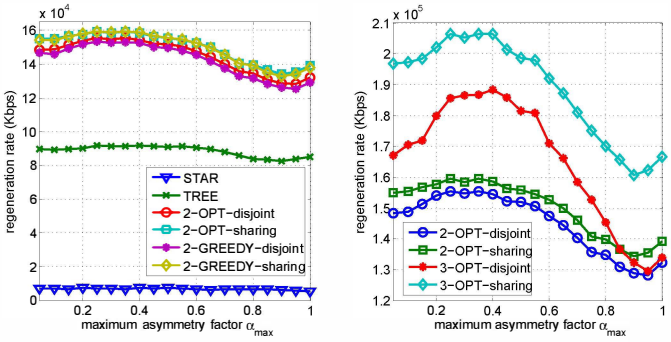


(a) 1-tree vs. 2-tree (b) 2-tree vs. 3-tree

Fig. 4. Regeneration rate with the increasing of the number of providers, when $\alpha \in (0, 0.6]$.

For fairness, we define the regeneration rate in $G(k)$ as $k \times t$ times of the available bandwidth, and thus the regeneration time is the ratio of the size of the original file to the regeneration rate. We compare the regeneration rate in Fig. 4, in which the asymmetry factor of links is selected in $(0, 0.6]$. We show in Fig. 4(a) the regeneration rate of 1-tree algorithms (STAR and TREE) and 2-tree algorithms (2-OPT/GREEDY-disjoint/sharing). We can see for all algorithms except STAR, the regeneration rate increases with k, because the tree-structured regeneration can bypass the slow links in the network, while STAR has to suffer from the link with the minimum bandwidth from one provider to the newcomer. Compared with STAR, TREE can increase the available bandwidth by a factor of 14 when $k = 10$, and the regeneration rate of 2-OPT-disjoint and 2-OPT-sharing can even achieve 22.9 and 23.5 times of STAR. On the other hand, the regeneration rate of 2-OPT-disjoint and 2-OPT-sharing is 1.6 and 1.7 times of the regeneration rate of TREE. Because we use more links by constructing two regeneration trees, the regeneration rate of the second tree can not be as good as the regeneration rate of the single tree. Therefore, 2-tree algorithms can not achieve twice of the regeneration rate of TREE.

Moreover, we notice that though our optimal algorithms perform better than the greedy algorithm, the curves of greedy algorithms converge to the curves of the corresponding optimal algorithms. However, we still think our optimal algorithms is better for practical distributed storage systems. First, considering the computational cost of encoding and decoding and the cost of establishing and maintaining connections in the network, distributed storage systems usually prefer a smaller k in practical. Second, the greedy algorithms have to construct parallel



(a) 1-tree vs. 2-tree (b) 2-tree vs. 3-tree

Fig. 5. Regeneration rate with the maximum asymmetry factor α_{max}, when $k = 10$.

regeneration trees one after another, so it may incur additional cost of regeneration time and bandwidth measurement.

We show the results in Fig. 4(b) when we employ more parallel regeneration trees during the regeneration. 3-OPT-disjoint and 3-OPT-sharing can achieve regeneration rate 1.3 and 1.4 times of 2-OPT-sharing. However, we observe interestingly that 3-OPT-disjoint performs worse than 2-OPT-sharing and even 2-OPT-disjoint when $k \leq 8$. The reason is that when k is small, the number of links in the network is also small, and the slow links in the network probably can not be bypassed. Therefore, we think it is better to share links with parallel regeneration trees, and if the distributed storage systems can not afford a large k, it is better to use only 2 parallel regeneration trees or even single regeneration tree in the regeneration process.

We also investigate how the asymmetry factor α influences the regeneration rate. We assign available bandwidth to links using the measured bandwidth data with asymmetry factor in $(0, \alpha_{max})$, and the maximum asymmetry factor α_{max} varies from 0.05 to 1. Fig. 5 shows the curves of regeneration rate with the increasing of α_{max}. For each algorithm, the average regeneration rate is related with the asymmetry factor, because the average available bandwidth in some intervals of asymmetry factor is relatively higher. Moreover, with the increasing of asymmetry factor, it is more probable that the available bandwidth is very low in one direction of the link. We thus observe that for all the algorithms, the regeneration rate changes with the asymmetry factor intervals in a similar trend and it tends to be lower with a larger asymmetry factor. In Table I, we calculate the ratio of the maximum regeneration rate of each algorithm to its minimum regeneration rate. We notice that the regeneration rate can be more affected by the asymmetry factor

TABLE I
THE RATIO OF THE MAXIMUM REGENERATION RATE TO THE MINIMUM REGENERATION RATE OF EACH ALGORITHM IN FIG. 5.

algorithm	ratio	algorithm	ratio
STAR	1.3510	TREE	1.1109
2-OPT-disjoint	1.2128	2-OPT-sharing	1.1877
2-GREEDY-disjoint	1.2249	2-GREEDY-sharing	1.1985
3-OPT-disjoint	1.4550	3-OPT-sharing	1.2854
3-GREEDY-disjoint	1.5046	3-GREEDY-sharing	1.3687

TABLE II
SIMULATION PARAMETERS

T_s	starting time (sec.)	2×10^6
T_f	finish time (sec.)	1.2×10^7
N_{all}	the number of nodes	100
N_r	repeated times	500
$\mathcal{M}$	file size (KB)	5×10^8
r	redundancy rate	1.5
t_{max}	maximum number of parallel regeneration trees	5

if there are more parallel regeneration trees. However, STAR is still very sensitive with the asymmetry factor, because it always "selects" the link with the worst available bandwidth as its bottleneck edge. We also observe that edge-sharing algorithms work better than edge-disjoint algorithms, because when the asymmetry factor is large, the bandwidth left by other parallel regeneration trees tends to be higher and thus is more likely to used by edge-sharing algorithms.

B. Dynamic network environment

We have evaluated the performance of the algorithms in a static environment, *i.e.,* there is no node join/departure in the network. However, it is improbable in reality. Thus, in this section, we introduce the node behaviors into our simulation and compare some metrics important to the practical distributed storage systems.

The trace file containing the join/departure behaviors of nodes in PlanetLab is provided by [18]. The status of the node is detected by the pings repeated every 15 minutes from Jan. 2004 to Jan. 2005. Based the trace file, we run our simulation in an event-driven simulator which simulates a practical distributed storage system in PlanetLab. Each node in the simulation is mapped to a real node in PlanetLab, and the edge in the simulation is mapped to the corresponding edge connecting the two corresponding nodes in PlanetLab in the same direction. We assign the available bandwidth of each edge according to the available bandwidth of the mapped edge.

Table II shows the parameters we use in the simulation. Time starts at 0 in the trace file and the simulation runs from T_s to T_f. We select N_{all} nodes with the most frequent join/leave behaviors from the nodes in the trace file. We assume that at the beginning of the simulation, a file has been stored in the distributed storage system. The size of the file is $\mathcal{M}$ kilobytes. The file is divided into k blocks and the coded blocks are stored in $r \times k$ storage nodes, where r is the redundancy rate. Each storage node stores one coded block.

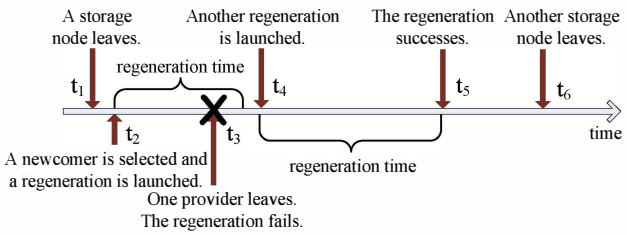


Fig. 6. Timeline of the regeneration process in the simulation.

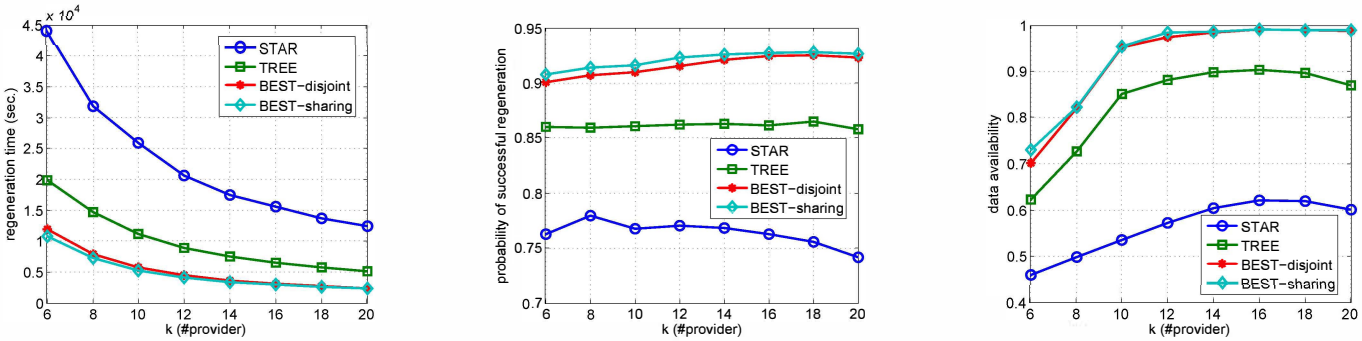
We use a timeline in Fig. 6 to describe our simulation system. After a storage node leaves the network at t_1, the system has to select a replacement node, *i.e.,* a newcomer, to regenerate the lost coded block. If there is no available replacement node, the regeneration has to be postponed until a node joins the network at t_2. When a regeneration is launched, k providers are selected from the available storage nodes with the highest available bandwidth towards the newcomer. We construct the regeneration tree(s) and calculate the corresponding regeneration time. When a storage node or the newcomer leaves the network at t_3, before the regeneration finishes, the regeneration fails and the system has to launch another regeneration at t_4. When the regeneration finishes at t_5, it is regarded as successful. The system then keeps idle until another storage node fails at t_6.

We compare four algorithms in the simulation. For 1-tree algorithms, we compare STAR and TREE. For multi-tree algorithms, we set t_{max} as the maximum number of parallel regeneration trees used during the regeneration. When t is too large, the regeneration rate of t-OPT-disjoint/sharing algorithm will decrease, as shown in Fig. 4(b). Thus we select the best t which achieves the maximum regeneration rate when $t = 1, 2, \ldots, t_{max}$. When $t = 1$, the regeneration tree is constructed by TREE, and t parallel regeneration trees are constructed by t-OPT-disjoint or t-OPT-sharing algorithm when $t > 2$. This algorithm is referred to as BEST-disjoint if the t-OPT-disjoint algorithm is used and as BEST-sharing if the t-OPT-sharing algorithm is employed. We repeat the simulation for N_r times and then get the average results.

Table II also shows the values of parameters used in our simulation. The simulation covers 10^7 seconds in PlanetLab. There are totally 100 nodes in the simulation. On average, there are 49.92 nodes available and each node joins or leaves the network 0.27 times per day during this time. We set the size of the original file to be 500GB, the size of the current mainstream commodity hard disk. The redundancy rate is set to be 1.5. At most 5 parallel regeneration trees can be constructed during the regeneration. In the simulation, k varies from 6 to 20. Thus, it is probable that there exist 5 edge-disjoint parallel regeneration trees. The simulation is repeated for 500 times.

Fig. 7 shows the simulation results. In Fig. 7(a), we can see that TREE can reduce the regeneration time by 58% and BEST-sharing further reduce the time by 55% compared with TREE. Because the number of links in the network increases with k, it is more likely to select the links with higher available bandwidth when k is larger. Thus the curves all go down with the increasing of k. Compared with Fig. 4, the curves of BEST-disjoint and BEST-sharing are quite close together. The "BEST" algorithms can select the best t to achieve the optimal regeneration time, so the gap between BEST-disjoint and BEST-sharing will not be big when k is small. When k becomes larger, the gap becomes smaller and smaller, as illustrated in Fig. 4.

We also compare another two important performance metrics of the regeneration in the distributed storage system. One metric is the probability of successful regeneration. The regeneration fails when one provider or the newcomer leaves the network



(a) regeneration time (b) probability of successful regeneration (c) data availability

Fig. 7. Simulation results in the dynamic network environment.

during the regeneration. We can see in Fig. 7(b) that both BEST-disjoint and BEST-sharing have the probability of successful regeneration more than 90%, while only about 75% of the regeneration processes using STAR and about 85% of the regeneration processes using TREE success. Moreover, when k is large, the probability of the successful regeneration process begins to reduce, because node departures are more likely to happen during the regeneration.

We notice in Fig. 7(c), the file availability of STAR is only about 60%. The file availability is the probability that the file can be recovered. If too many regeneration processes fail, the number of coded blocks can not be kept always larger than k. However, when $k > 14$, the file availability of BEST-disjoint and BEST-sharing can be more than 98%. Even for TREE, the file availability can still be almost 90%. Therefore, we can regard the file as highly available by the tree-structured regeneration algorithms.

VI. Conclusion

In this paper, we discuss the tree-structured regeneration in the network with asymmetric links. We show the construction algorithm of the optimal regeneration tree in the network with asymmetric links and analyze its performance. We then propose two algorithms to construct multiple parallel regeneration trees, so as to further reduce the regeneration time by transmitting data in parallel during the regeneration. By extensive simulations, we evaluate the performance of the tree-structured regeneration algorithms using real data measured in PlanetLab. Parallel regeneration trees can reduce the regeneration time significantly and maintain the file availability of no less than 98%.

References

[1] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total Recall: System Support for Automated Availability Management," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2004, pp. 25–25.

[2] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," in *Proc. ACM SIGCOMM 2009 conference on Data communication*, 2009, pp. 63–74.

[3] J. Li, S. Yang, X. Wang, X. Xue, and B. Li, "Tree-structured Data Regeneration with Network Coding in Distributed Storage Systems," in *Proc. 17th IEEE International Workshop on Quality of Service (IWQoS)*, 2009.

[4] J. Li, S. Yang, X. Wang, and B. Li, "Tree-structured Data Regeneration in Distributed Storage Systems with Regenerating Codes," in *Proc. INFOCOM*, 2010.

[5] Y. Wu, R. Dimakis, and K. Ramchandran, "Deterministic Regenerating Codes for Distributed Storage," in *Proc. Allerton Conference on Control, Computing, and Communication*, 2007.

[6] S.-J. Lee, P. Sharma, S. Banerjee, S. Basu, and R. Fonseca, "Measuring Bandwidth Between PlanetLab Nodes," in *Proc. Passive and Active Network Measurement (PAM)*, 2005, pp. 292–305.

[7] [Online]. Available: http://www.planet-lab.org/

[8] R. Rodrigues and B. Liskov, "High Availability in DHTs: Erasure Coding vs. Replication," 2005.

[9] S. Acedański, S. Deb, M. Médard, and R. Koetter, "How Good is Random Linear Coding based Distributed Networked Storage?" in *Proc. 1st Workshop on Network Coding (WiOpt)*, Apr. 2005.

[10] A. Dimakis, P. Godfrey, M. Wainwright, and K. Ramchandran, "Network Coding for Distributed Storage Systems," in *Proc. INFOCOM*, May 2007, pp. 2000–2008.

[11] A. Duminuco and E. Biersack, "Hierarchical Codes: How to Make Erasure Codes Attractive for Peer-to-Peer Storage Systems," in *Proc. 8th International Conference on Peer-to-Peer Computing (P2P)*, Sep. 2008, pp. 89–98.

[12] A. Akella, S. Seshan, and A. Shaikh, "An Empirical Evaluation of Wide-area Internet Bottlenecks," in *Proc. ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2003, pp. 316–317.

[13] H. N. Gabow and R. E. Tarjan, "Algorithms for Two Bottleneck Optimization Problems," *J. Algorithms*, vol. 9, no. 3, pp. 411–417, 1988.

[14] H. A. David and H. N. Nagaraja, *Order Statistics*, 3rd ed. Wiley-Interscience, Aug 2003.

[15] R. E. Tarjan, "A Good Algorithm for Edge-disjoint Branching," *Inform. Process. Lett.*, vol. 3, no. 2, pp. 51–53, 1974.

[16] H. N. Gabow and K. S. Manu, "Packing Algorithms for Arborescences (and Spanning Trees) in Capacitated Graphs," in *Proc. 4th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 1995, pp. 388–402.

[17] S. Banerjee, S.-J. Lee, P. Sharma, and P. Yalagandula. S^3 (Scalable Sensing Service). [Online]. Available: http://networking.hpl.hp.com/s-cube/PL/

[18] J. Stribling. Planetlab All Pairs Ping. [Online]. Available: http://infospect.planet-lab.org/pings

[19] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cot, "pathChirp: Efficient Available Bandwidth Estimation for Network Paths," in *Proc. Passive and Active Measurement Workshop*, 2003.