Pipelined Regeneration with Regenerating Codes for Distributed Storage Systems

Jun Li[1], Xin Wang[1], Baochun Li[2]
[1]School of Computer Science, Fudan University, China
[2]Department of Electrical and Computer Engineering, University of Toronto, Canada

***Abstract*—Distributed storage systems store a substantial amount of data and maintain data integrity by storing redundancy in a large number of storage nodes. When storage nodes fail, the lost data should be regenerated on replacement nodes. Regenerating codes minimize the volume of network traffic brought by the regeneration process. However, regenerating codes consume less traffic by requiring a larger number of nodes participating in the regeneration process. In this paper, we propose a new pipelined regeneration process for regenerating codes, that enables the system to consume a smaller amount of network traffic during the regeneration without engaging a large number of participating nodes. Moreover, the time spent during the regeneration can be further saved, while the data integrity can still be maintained no worse than the conventional regeneration process with regenerating codes.**

***Keywords*—pipelined regeneration, regenerating codes, distributed storage**

I. Introduction

Distributed storage systems store a substantial amount of data by utilizing a large number of storage nodes that may fail to work effectively and cause data loss. As a result, large-scale distributed storage systems are designed to treat storage node failures as the *rule*, rather than the *exception*. To compensate data losses incurred by such failures, these systems need to store redundancy so that a certain number of storage failures can be tolerated.

To maintain such tolerance, when a node failure occurs, the redundancy that is lost due to the failure should be regenerated on a replacement node, called a *newcomer*. Costing time and bandwidth, the newcomer needs to fetch data from certain active storage nodes, called *providers*, to regenerate the lost data. Storing redundancy as MDS codes, which encode the original file into some coded blocks, can guarantee the property that any k coded blocks can recover the original file, achieving a higher data integrity than replication [1]. Dimakis *et al.* have proposed a family of MDS codes called regenerating codes [2] that achieve the optimal trade-off between the storage cost and the communication cost to regenerate one coded block. If the size of the original file is M bits, minimum-storage regenerating codes require the newcomer to receive $\frac{M}{k} \cdot \frac{d}{d-k+1}$ bits from d providers ($d \geq k$), which approximately equal the size of the coded block ($\frac{M}{k}$ bits) to be regenerated if d is large enough.

To achieve the traffic consumption that converges to $\frac{M}{k}$ bits, regenerating codes always require much more than k participating nodes during regeneration. In other words, the more nodes participating, the less network traffic will be consumed during regeneration. However, in practical distributed storage systems, it is not favorable to let a large number of nodes work cooperatively. The resistance to node churns decreases with the increasing of participating nodes. In addition, since some systems may let nodes sleep and wake up on demand, it is not energy-saving to utilize a large number of nodes simultaneously.

In order to avoid using a large number of participating nodes, some techniques (*e.g.*, [3], [4]) have been proposed. However, none of them can maintain the property that any k coded blocks can recover the original file. In this paper, we propose a pipelined regeneration process that can drastically reduce the number of participating nodes required by regenerating codes without sacrificing data integrity. In other words, with utilizing the same number of participating nodes, pipelined regeneration can achieve the same performance of regenerating codes that would have required for much more participating nodes. For example, When $k = 3$, the newcomer will receive $\frac{2}{3}M$ bits from 4 providers in the conventional regeneration process with regenerating codes. However, in the pipelined regeneration process, with the same number of participating nodes, only $\frac{4}{9}M$ bits of data will be transferred, which would only have been achieved when there were 8 providers in the conventional regeneration process. Therefore, both the consumption of bandwidth and time can be saved by the pipelined regeneration.

To achieve this goal, we introduce a new type of nodes participating in the pipelined regeneration process, called *apprentices*. As shown in Fig. 1, if d providers are required

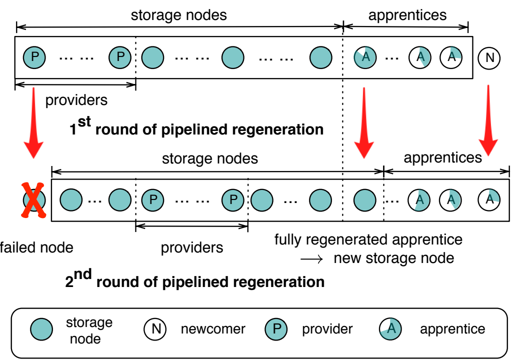


Fig. 1. Two consecutive rounds of pipelined regeneration.

to regenerate one coded block by using regenerating codes, in each round of pipelined regeneration, both the newcomer and apprentices are partially regenerated, such that there are fewer than d providers. The newcomer becomes an apprentice after pipelined regeneration. A new apprentice accumulates more and more data to regenerate a coded block in upcoming rounds of pipelined regeneration. When it has received sufficient data to be fully regenerated, it "graduates" to become a new storage node.

In this paper, we show that the pipelined regeneration process is able to consume a smaller amount of traffic that would have been achieved by the conventional regeneration process with a much higher number of participating nodes, while maintaining the data integrity as well as regenerating codes and other MDS codes. Meanwhile, the time spent during regeneration can also be reduced.

The remainder of this paper is organized as follows. We introduce the preliminaries in Sec. II. We propose the pipelined regeneration process in Sec. III, describing its transmission and encoding scheme, and proving that it can maintain the data integrity as well as regenerating codes. In Sec. IV, we analyze the performance of pipelined regeneration, showing its requirement for the amount of redundancy and analyzing the consumption of network traffic and time. We conclude the paper in Sec. V.

II. Preliminaries

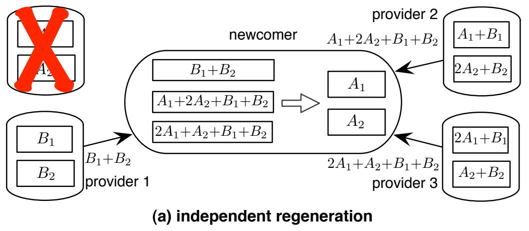
A. Regenerating codes

Given a file of size M bits, MDS codes encode it into more than k coded blocks of size $\frac{M}{k}$ bits, such that any k coded blocks are sufficient to recover the original file. However, among MDS codes, the efficiency of different codes in the regeneration process varies, and *regenerating codes* result in the optimal amount of traffic needed.

Suppose that d providers ($d \geq k$) are required in the regeneration process. Minimum-storage regenerating codes divide the original file into $k(d-k+1)$ segments and encode them into coded segments. One coded block will contain $d-k+1$ coded segments and their coefficients such that any k coded blocks can recover the original file. For example in Fig. 2(a), when $k = 2$ and $d = 3$, each storage node stores one coded blocks containing two coded segments.

Regenerating codes encode data on a finite field GF(2^q). In each segment, every continuous q bits can be regarded as a symbol on the GF(2^q), such that each segment can be regarded as a vector containing $\frac{M}{kq(d-k+1)}$ symbols on GF(2^q). A coded segments is a linear combination of the original segments on GF(2^q). To recover the original file, $k(d-k+1)$ coded segments and corresponding coefficients should be collected from at least k coded blocks, and then the original file can be decoded by solving a linear system containing $k(d-k+1)$ unknowns and $k(d-k+1)$ equations.

One storage node stores at most one coded block. When a storage node fails, to regenerate the lost coded block, a newcomer needs to contact d providers, each of which sends a linear combination of its corresponding coded segments



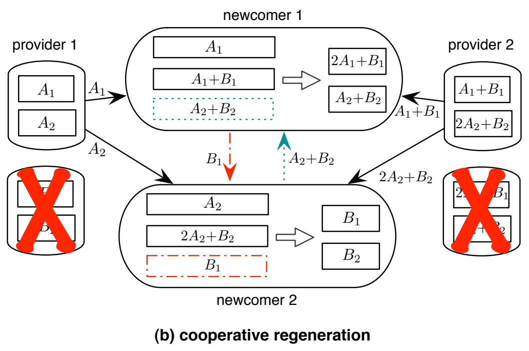


Fig. 2. Two examples of independent regeneration and cooperative regeneration with regenerating codes [5].

and the corresponding coefficients to the newcomer. Then the newcomer encodes the received segments into $d-k+1$ coded segments and groups them as a coded block. In Fig. 2(a), the newcomer receives three coded segments from three providers and then encodes them into two segments. The coded block that is regenerated still maintains the property that any k coded blocks can recover the original file [2]. In this paper, we do not consider the size of coefficients since it is much smaller than that of coded segments in the distributed storage system. Thus, a total of $\frac{M}{k(d-k+1)}$ bits will be conveyed during regeneration.

The regeneration that storage nodes are regenerated in a one-by-one fashion is called *independent regeneration*, as shown in Fig. 2(a). However, in some practical distributed storage systems, such as Total Recall [6], the regeneration process will not be triggered until the number of failed storage node has reached a certain threshold, and they can be regenerated in batches, *i.e.*, there are more than one newcomer in each round of regeneration. Hu *et al.* [7] have shown that even less bandwidth will be consumed if newcomers can cooperate in the regeneration process called *cooperative regeneration*, rather than regenerating lost data in different independent regeneration processes. Shum [5] and Kermarrec *et al.* [8] have independently proposed cooperative regenerating codes that allow newcomers to coordinate during cooperative regeneration to achieve optimal bandwidth consumption. The minimum-storage cooperative regenerating codes incur only $\frac{M}{k} \cdot \frac{d-1+r}{d-k+r} \cdot r$ bits in total in each round of cooperative regeneration, in which there are d providers and r newcomers. Similar to regenerating codes in independent regeneration, the bandwidth consumption to regenerate one storage node converges to $\frac{M}{k}$ bits when $d+r$ is large enough.

Suppose that d providers and r newcomers are required in one round of cooperative regeneration, the original file should be divided into $k(d-k+r)$ segments and each coded block should contain $d-k+r$ coded segments. Thus, k coded blocks can still recover the original file. To regenerate r coded blocks together, r newcomers first receive a linear combination of coded segments from each provider, respectively. Then each newcomer sends $r-1$ different linear combinations of its d received coded segments to all other providers. Finally, each newcomer encodes all of its received coded segments into $d-k+r$ coded segments and groups them as one coded block, which still maintains the property that any k coded blocks can recover the original file [5], [8]. When $k=2$, Fig. 2(b) shows an example of cooperative regeneration with two providers and two newcomers ($d=r=2$). Kermarrec *et al.* [8] have shown that if each coded block contains $d-k+r$ coded segments, both independent regeneration ($r=1$) and cooperative regeneration ($r>1$) with $d+r$ participating nodes can be supported adaptively. In this paper, we propose pipelined (independent) regeneration that engages much fewer participating nodes by dividing one round of cooperative regeneration into several rounds of pipelined regeneration.

B. System model

In this paper, we assume that any k coded blocks of the same file can recover the original data. As for one file, each storage node stores at most one of its coded blocks. All data stored are produced by randomized regenerating codes, *i.e.*, all coded segments are random linear combinations of the original segment. Wu *et al.* [9] and Wang *et al.* [10] have shown that both randomized regenerating codes and randomized cooperative regenerating codes can both maintain the property that any k coded blocks can recover the original file with a very high probability if the size of the finite field is large enough. Thus, we do not consider the issue of linear dependence in this paper.

We consider one file with a size of M bits, and each coded block has a size of $\frac{M}{k}$ bits. Each coded block contains $N-k$ coded segments, such that N participating nodes are required by conventional independent or cooperative regeneration process [8]. We assume that the available bandwidth between any pair of two nodes in the system satisfies an independent distribution. Suppose that at most n nodes ($n<N$) are allowed to participate during regeneration, which can not be supported by the conventional regeneration process. We show that a pipelined regeneration process that allows each coded block contains $N-k$ coded segments, such that less network traffic will be incurred during regeneration, while still maintaining data integrity the same as conventional regeneration process.

III. Pipelined Regeneration

A. Transmission

There are three types of nodes in the pipelined regeneration process: one newcomer, ν providers ($P_1,\dots,P_\nu$, $\nu<n-1$), and α apprentices ($A_1,\dots,A_\alpha$). The newcomer and apprentices receive coded segments from providers and some other apprentices during pipelined regeneration. We define $\text{rank}(A_i)$ as the number of coded segments A_i has received from other nodes in previous rounds of pipelined regeneration, $i=1,\dots,\alpha$. Without loss of generality, we assume that $0<\text{rank}(A_1)<\dots<\text{rank}(A_\alpha)$. A_α is also referred to as the *root*. After one round of pipelined regeneration, the newcomer will become an apprentice. Each apprentice is partially regenerated since $\nu<n-1$. α and ν are selected such that the root becomes fully regenerated after each round of pipelined regeneration. Therefore, a newcomer will be fully regenerated after $\alpha+1$ rounds of pipelined regeneration.

During pipelined regeneration, all providers send $\alpha+1$ coded segments, which are random linear combinations of its coded segments with distinct coefficients, to α apprentices and the newcomer. The root then sends α random linear combinations of its coded segments that have been received so far to all other apprentices and the newcomer. In this process, the root will get ν coded segments, and all non-root apprentices and the newcomer will get $\nu+1$ coded segments. If each coded block is supposed to contain $N-k$ coded segments, the root encodes all of its received coded segments to $N-k$ segments and groups them as one coded block. Fig. 3 shows an example of the independent pipelined regeneration where $k=3$ and $\alpha=\nu=2$. In the conventional regeneration, at least $\frac{2}{3}M$ bits will be transferred if there are at most 4 providers. In pipelined regeneration, however, we can let $N=9$ and thus only $\frac{4}{9}M$ will be transferred, saving 33.3% of network traffic. We will discuss how to select α and ν, and investigate the encoding scheme in Sec. III-B.

To guarantee the property that any k coded blocks can recover the original file, in $\alpha+1$ consecutive rounds of pipelined regeneration, it is required that one storage node should be used as a provider once at most. Specifically, if an apprentice that has accomplished τ rounds of pipelined regeneration ($1\le\tau<\alpha+1$) fails, during the next $(1+\alpha-\tau)^{\text{th}}$ pipelined regeneration when this apprentice would have acted as a root if it had not failed, one additional provider should be selected to replace this failed apprentice as the root. The number of participating nodes thus remains the same and the newcomer and apprentices can still obtain $\nu+1$ coded segments after this round of pipelined regeneration. Moreover, the provider selected to replace the failed apprentice just

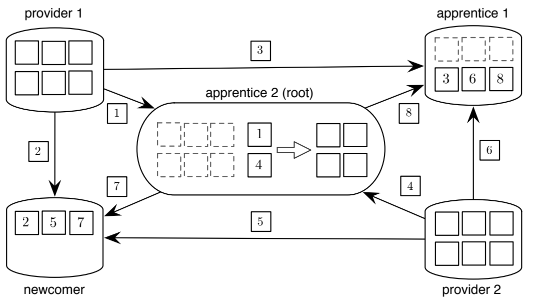


Fig. 3. An example of pipelined regeneration ($k=3, N=9, \alpha=\nu=2$). We omit linear coefficients of each segment and use indices to represent different coded segments. Each arrow represents one connection between two participating nodes that conveys one coded segment. Coded segments that have been received before in apprentices are indicated with dashed lines.

receives, encodes and sends out data, but does not store received data or update its own coded block, since its block has already been fully regenerated before. This provider should not be used again in recent $\alpha+1$ rounds of pipelined regeneration as other providers as well.

We show that if ν and α are large enough, we can guarantee that the root can always be fully regenerated after each round of pipelined regeneration, and the coded block regenerated can maintain the property that any k coded blocks can recover the original file.

Theorem 1: If $(\alpha+1)(\nu+1) \geq N$ and $(\alpha+1)\nu \geq k$, all roots can be fully regenerated after pipelined regeneration processes, and coded blocks they store can maintain the property that any k coded blocks can recover the original file.

Proof: A newcomer can be fully regenerated after at least $\alpha+1$ rounds of pipelined regeneration. Thus, we prove this theorem by mapping $\alpha+1$ consecutive rounds of pipelined regeneration to one round of conventional cooperative regeneration with $(\alpha+1)\nu$ providers and $\alpha+1$ newcomers.

Given one root, denoted as A in one round of pipelined regeneration, we consider this round and the previous α rounds of pipelined regeneration. We map all roots in these $\alpha+1$ rounds of pipelined regeneration to $\alpha+1$ newcomers in conventional cooperative regeneration. In particular, the newcomer mapped from A is referred to as A'. All $(\alpha+1)\nu$ providers that the A received data from are mapped as providers in cooperative regeneration, denoted as $P'_1, \ldots, P'_{(\alpha+1)\nu}$.

In the cooperative regeneration process, a newcomer should receive one coded segment from $(\alpha+1)\nu$ providers and other α newcomers, respectively. A' does so in the mapped cooperative regeneration process. However, the mapped newcomers other than A' may receive data from other providers that are not mapped in the pipelined regeneration process. Assume that one of these newcomers receives coded segments from $P''_1, \ldots, P''_{(\alpha+1)\nu}$ in its $\alpha+1$ rounds of pipelined regeneration. Since $(\alpha+1)\nu \geq k$, coded segments stored in these providers can all be represented as linear combinations of coded segments stored in $\{P'_1, \ldots, P'_{(\alpha+1)\nu}\}$. Because all coded segments a newcomer receives are random linear combinations of coded segments of providers, it is equivalent with that these segments are random linear combinations of coded segments in $\{P'_1, \ldots, P'_{(\alpha+1)\nu}\}$. Therefore, the coded block in A is regenerated equivalently with the block regenerated in a cooperative regeneration with $(\alpha+1)\nu$ providers and $\alpha+1$ newcomers. Since there are no less than k providers and no less than N participating nodes in the mapped cooperative regeneration process, the coded block regenerated at A' can maintain the property that any k coded blocks can recover the original file [5], [8]. Therefore, A can be fully regenerated and its coded block still maintains this property as well. ■

B. Encoding

Theorem 1 shows the condition of α and ν that the root can be fully regenerated after pipelined regeneration. Now we discuss the choices of α and ν to achieve the minimum bandwidth consumption in the pipelined regeneration process.

Theorem 2: To regenerate a coded block containing $N-k$ coded segments, the minimum number of participating nodes is $2\sqrt{N}-1$, while $\alpha=\nu=\sqrt{N}-1$.

Proof: Let $r=1+\alpha$. By Theorem 1, $N \leq (1+\alpha)(1+\nu) = (1+\nu)r$. Therefore, $\nu \geq \frac{N}{r}-1$, and the total number of participating nodes in the pipelined regeneration process is $\alpha+1+\nu = \nu+r \geq \frac{N}{r}+r-1 \geq 2\sqrt{N}-1$. The equations are achieved when $r=\sqrt{N}$ and $\nu=\frac{N}{r}-1$, *i.e.*, $\alpha=\nu=\sqrt{N}-1$. ■

When at most n nodes are allowed to participate in the regeneration process, by Theorem 2 we can let $\alpha=\nu=\lfloor\frac{n-1}{2}\rfloor$, such that $\alpha+\nu+1 \leq n$. Thus, the maximum number of N is $N_{\max} = (\alpha+1)^2 = (\lfloor\frac{n-1}{2}\rfloor+1)^2 = (\lfloor\frac{n+1}{2}\rfloor)^2$. The more N is, the less traffic will be incurred. Thus, the minimum traffic is achieved when $N=(\lfloor\frac{n+1}{2}\rfloor)^2$. Therefore, even though $N_{\max}$ is very large, which can achieve the bandwidth consumption converging very closely to $\frac{M}{k}$ bits, a much smaller number of nodes are required to participate during pipelined regeneration. For example, only 17 participating nodes are required to support the encoding scheme that $N=100$. This property is very useful for some systems that wake up nodes on demand (*e.g.*, archival file system) or have a high node churn (*e.g.*, P2P file system).

IV. Performance Analysis and Evaluation

A. The minimum amount of redundancy

If the encoding scheme of regenerating codes in the distributed storage system has been determined, we can also determine the minimum amount of redundancy that should be stored to compensate at least one node failure. In the conventional regeneration process, if a coded block is composed of $N-k$ coded segments, a newcomer needs to contact at least $N-1$ providers. Therefore, each file stored in the system requires at least N storage nodes to compensate at least one node failure. Thus, the minimum amount of redundancy of conventional regeneration is $\frac{N}{k}$ times of the original file.

As for pipelined regeneration, however, though it requires fewer providers, the minimum amount of redundancy is more than that required by conventional regeneration, due to the constraint that a storage node can act as a provider at most once in $\alpha+1$ consecutive rounds of pipelined regeneration.

Theorem 3: The minimum amount of redundancy required by the pipelined regeneration is $\frac{N+\alpha}{k}$ times of the original file.

Proof: Though there are only ν providers during pipelined regeneration, the newcomer still needs to receive coded segments from at least $N-1$ providers within $1+\alpha$ rounds of pipelined regeneration. Thus, the amount of redundancy is enough if and only if we can always find ν providers that do not appear in previous α rounds of pipelined regeneration.

If the failed node is a storage node that has been used during the recent α rounds of pipelined regeneration, the required number of available storage nodes is ν.

If the failed node is an available storage node, there should be at least $\nu+1$ storage nodes available before this failure.

We last consider the case that the failed node is an apprentice. If the apprentice should have been the root during the next

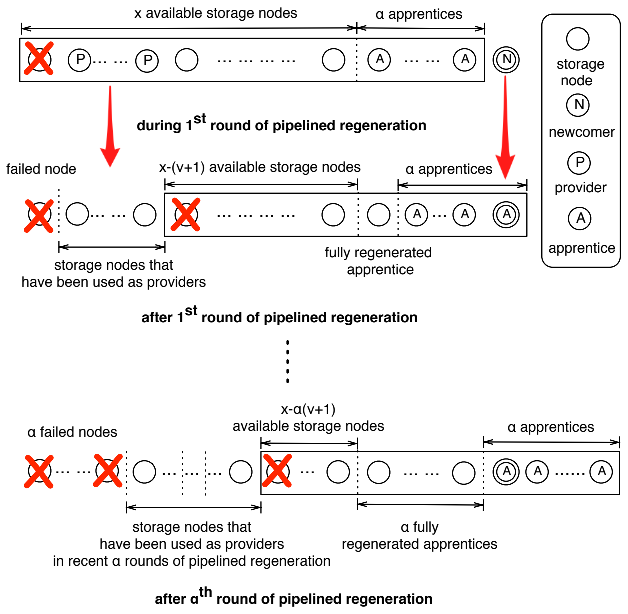

x available storage nodes
α apprentices
during 1st round of pipelined regeneration
failed node
x-(v+1) available storage nodes
α apprentices
storage nodes that have been used as providers
fully regenerated apprentice
after 1st round of pipelined regeneration
α failed nodes
x-α(v+1) available storage nodes
α apprentices
storage nodes that have been used as providers in recent α rounds of pipelined regeneration
α fully regenerated apprentices
after αth round of pipelined regeneration
storage node
newcomer
provider
apprentice

Fig. 4. The required number of available storage nodes in $1+\alpha$ consecutive rounds of pipelined regeneration, when failed nodes are available storage nodes.

round of pipelined regeneration, the regeneration will require $\nu+1$ providers to compensate for this loss. On the other hand, if the apprentice has just accomplished τ rounds of pipelined regeneration ($1 \leq \tau < 1+\alpha$), the required number of available storage nodes in the next round of pipelined regeneration is ν, but in the next $(1+\alpha-\tau)^{\text{th}}$ round of pipelined regeneration, $\nu+2$ available storage nodes will be required. Therefore, the total number of required available storage nodes remains within $\alpha+1$ consecutive rounds of pipelined regeneration.

Since within $\alpha+1$ consecutive rounds of pipelined regeneration, the average number of required available storage nodes is at most $\nu+1$, without loss of generality, we assume that each regeneration is triggered by the failure of an available storage node. As shown in Fig. 4, $\nu+1$ available storage nodes become unavailable to the double-circled apprentice after the first round of pipelined regeneration. The fully regenerated apprentice can not be regarded as available because the double-circled apprentice can only produce its fully regenerated block after this round of pipelined regeneration. Assume that there are x storage nodes available to the double-circled newcomer before its first round of pipelined regeneration. After α rounds of pipelined regeneration, there are $x-\alpha(\nu+1)$ nodes still available. The required number of available storage nodes before the $(\alpha+1)^{\text{th}}$ round of pipelined regeneration is $\nu+1$. Thus, $x-\alpha(\nu+1) \geq \nu+1$.

The minimum required number of nodes is $x+\alpha$, since there are α partially regenerated apprentices between two rounds of pipelined regeneration. By Theorem 1, $x+\alpha \geq (1+\alpha)(1+\nu)+\alpha \geq N+\alpha$. The equations can be achieved when $\alpha = \nu = \sqrt{N}-1$. Therefore, the minimum required amount of redundancy is $\frac{N+\alpha}{k}$. ∎

Theorem 3 shows that only the storage space for additional apprentices is required in the distributed storage system to support pipelined regeneration. As shown by Fig. 5, even though a small number of nodes participate during regeneration, pipelined regeneration is able to support a very high value of $N_{\max}$, with only a small fraction of additional storage space (13.9% when $n = 9$).

B. Network traffic

Now we analyze the network traffic transferred in the pipelined regeneration process. Since there are ν providers and α apprentices during pipelined regeneration, there are $(\alpha+1)\nu$ connections from providers to apprentices or the newcomer, and α connections from the root to other apprentices or the newcomer. Thus, there are totally $(1+\alpha)(1+\nu)-1 = N_{\max}-1$ connections. Since one coded segment is conveyed in each connection, the total amount of traffic is $\frac{M(N_{\max}-1)}{k(N_{\max}-k)}$. On the other hand, if n nodes are allowed to participate, conventional regeneration will cost $\frac{M(n-1)}{k(n-k)}$ when $n \geq k+1$. Thus, with the same number of participating nodes pipelined regeneration can save bandwidth consumption by $\frac{(N_{\max}-n)(k-1)}{(n-1)(N_{\max}-k)}$. Fig. 6 compares the bandwidth consumption of pipelined regeneration and conventional regeneration when $k = 3$. We can see that up to 60% of the network traffic can be saved by pipelined regeneration ($n = 6$), and a smaller n can save more network traffic. Notice that n can even be smaller than $k+1$ as long as $(\alpha+1)\nu \geq k$, by Theorem 1. Thus, it is very promising for the distributed storage system that can use a limited number of participating nodes simultaneously.

C. Regeneration time

We analyze the regeneration time from the perspective of bottleneck bandwidth — the bandwidth available over the bottleneck connection during regeneration. In the pipelined regeneration process, we use a fine-grained pipelined transmission at a symbol level. Notice that each coded segment can be regarded as a series of symbols on a finite field. When the root has received the first symbols of coded segments from all providers, it has got all the first symbols of coded segments that make it fully regenerated. Then it can produce all the first symbols of coded segments in its coded blocks. At the

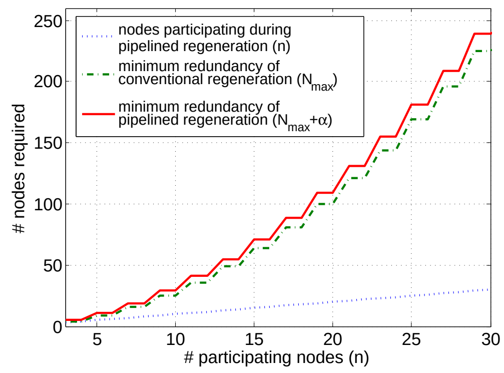


Fig. 5. The minimum number of storage nodes required by conventional regeneration and pipelined regeneration.

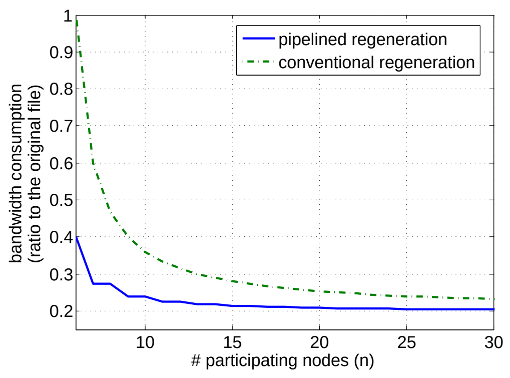


Fig. 6. Bandwidth consumption of pipelined regeneration and conventional regeneration ($k = 3$).

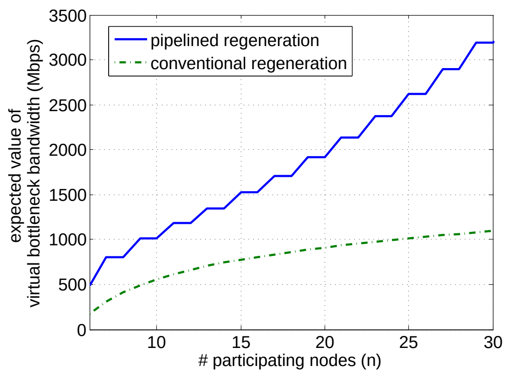


Fig. 7. Virtual bottleneck bandwidth of pipelined regeneration and conventional regeneration ($k = 3$).

same time, it produces the first symbols of coded segments to be sent to other apprentices and the newcomer, and send them immediately. After that, the root does the same thing of the second symbols, and so on. Since the block size in the distributed storage system is usually very large, we do not consider the initial delay of first symbols, and thus the regeneration time can be regarded as inversely proportional to the bottleneck bandwidth.

Since there are $N_{\max} - 1$ connections during pipelined regeneration, the probability density function of the minimum bandwidth available over $N_{\max} - 1$ connection is $f_{(N_{\max}-1)}(x) = (N_{\max} - 1)[1 - F(x)]^{(N_{\max}-2)} f(x)$ [11], where $f(x)$ and $F(x)$ are the probability density function and the cumulative distribution function of the bandwidth distribution, respectively. Thus, the expected value of bottleneck bandwidth of pipelined regeneration and conventional regeneration is $E[f_{(N_{\max}-1)}(x)]$ and $E[f_{(n-1)}(x)]$, respectively.

Since the size of a coded segment is different between pipelined regeneration and conventional regeneration with the same number of participating nodes, we compare the virtual bottleneck bandwidth that equals the number of coded segments in a coded block times the bottleneck bandwidth, such that the regeneration time can be regarded as the ratio of the size of a coded block to the virtual bottleneck bandwidth. We assume the bandwidth distribution satisfy a uniform distribution $U[10 \text{ Mbps}, 1024 \text{ Mbps}]$ and $k = 5$. Even though pipelined regeneration incurs more connections, we can see from Fig. 7 that it still can improve the expected value of virtual bottleneck bandwidth since the coded segment conveyed in each connection is much smaller than conventional regeneration, leading to less regeneration time.

V. Conclusion

In this paper, we propose a pipelined regeneration process for regenerating codes that enables the system to achieve the performance with a limited number of participating nodes during regeneration that can only be achieved by conventional regeneration with a much larger number of participating nodes. Preserving the data integrity as well as conventional regeneration, pipelined regeneration can reduce the bandwidth consumption without increasing the number of participating nodes. Meanwhile, it can save the time spent during regeneration as well. We will discuss pipelined cooperative regeneration for regenerating codes in our future work.

Acknowledgment

This work is supported in part by 863 program of China under Grant No. 2009AA01A348, National Key S&T Project under Grant 2010ZX03003-003-03, and Shanghai Municipal R&D Foundation under Grant No. 09511501200. Xin Wang (Email: xinw@fudan.edu.cn) is the corresponding author.

References

[1] R. Rodrigues and B. Liskov, "High Availability in DHTs: Erasure Coding vs. Replication," in *Proc. 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005.

[2] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network Coding for Distributed Storage Systems," *IEEE Tran. Inform. Theory*, vol. 56, no. 9, Sept. 2010.

[3] 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*, 2008, pp. 89–98.

[4] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Enabling Node Repair in Any Erasure Code for Distributed Storage," arXiv: 1101.0133v1, Dec. 2010. [Online]. Available: http://arxiv.org/pdf/1101.0133v1

[5] K. W. Shum, "Cooperative Regenerating Codes for Distributed Storage Systems," *to appear in Proc. IEEE International Conference on Communications (ICC)*, 2011.

[6] 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, p. 25.

[7] Y. Hu, Y. Xu, X. Wang, C. Zhan, and P. Li, "Cooperative Recovery of Distributed Storage from Multiple Losses with Network Coding," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 2, pp. 268–276, Feb. 2010.

[8] A.-M. Kermarrec, N. Le Scouarnec, and G. Straub, "Repairing Multiple Failures with Coordinated and Adaptive Regenerating Codes," INRIA, Research Report RR-7375, Sept. 2010. [Online]. Available: http://hal.inria.fr/inria-00516647/PDF/RR-7375.pdf

[9] Y. Wu, A. Dimakis, and K. Ramchandran, "Deterministic Regenerating Codes for Distributed Storage," in *Proc. 45th Annual Allerton Conference on Communication, Control, and Computing*, 2007.

[10] X. Wang, Y. Xu, Y. Hu, and K. Ou, "MFR: Multi-Loss Flexible Recovery in Distributed Storage Systems," in *Proc. IEEE International Conference on Communications (ICC)*, 2010.

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