Accelerating the Reconstruction Process in Network Coding Storage System by Leveraging Data Temperature

Kai Li and Yuhui Deng*

Department of Computer Science, Jinan University, Guangzhou 510632, P.R. China
likai328@gmail.com, tyhdeng@jnu.edu.cn

Abstract. Over the past few years, network coding has been employed in data reconstruction process of storage systems to minimize the recovery bandwidth. However, the time consumption of the decoding operations incurs a significant performance degradation. In this paper, we propose a data temperature-based reconstruction optimization algorithm and integrate it into the reconstruction process of a Network-Coding-Based File System (NCFS) which adopts regenerating code as its storage coding scheme. We conduct extensive experiments to evaluate the impacts on the data reconstruction process of regenerating codes. The experimental results demonstrate that our method outperforms the conventional approach both in reconstruction time, throughput and average response time with up to 33.17%, 60.61%, 37.77% improvement, respectively.

1 Introduction

Distributed storage systems have been widely deployed in industry to provide massive data storage service [1][2][3]. In such storage systems, data is allocated into a number of nodes in a stripe manner which enhances read/write performance in parallel ways. Since node failures are common [1], when a node storing encoded information fails, in order to maintain the same level of reliability we need to create encoded information at a new node. Therefore, redundancy must be introduced to reconstruct data. The simplest redundancy method is replication, which places several copies of same data in different nodes..

Erasure coding provides the same reliability as replication but requiring much less storage space [4][5]. This technique departs one piece of data into d pieces and then encodes them into n pieces $(n>d)$, then stripe encoded data. Such that any d of them are sufficient to reconstruct the original data. So, when we want to reconstruct a failed node, at least k times size data must be read from surviving nodes to participate in the decoding process, while in replication the repair of one replica needs that only one other replica is read.

Dimakis [6] proposed Regenerating Codes that stem from the concept of network coding [7] and minimize the repair traffic among storage nodes. They exploit the optimal trade-off between storage cost and repair traffic, and there are two optimal points. One optimal point refers to the minimum storage regenerating (MSR) codes, which minimize the repair bandwidth subject to the condition that each node stores

* Corresponding author.

C.-H. Hsu et al. (Eds.): NPC 2014, LNCS 8707, pp. 510–521, 2014.

the minimum amount of data as in Reed-Solomon codes. Another optimal point is the minimum bandwidth regenerating (MBR)codes, which allow each node to store more data to further minimize the repair bandwidth. The construction of MBR codes is found in [8], while that of MSR codes based on interference alignment is found in [9], [10]. We focus on MBR codes in this paper.

Therefore, using minimum bandwidth regenerating (MBR) codes can improve the performance of data recovery in case of node failure, while requiring less download bandwidth than traditional replication and erasure coding. However, recent work [12] has shown that regenerating codes takes too much computation overhead during data reconstruction which significantly increases the reconstruction time. Meanwhile, the time to rebuild a single disk has lengthened as the disk capacity far outpaces the disk bandwidth. Furthermore, the longer the period of single disk repair takes, the higher the possibility of a disk failure, which would probably lead to unrecoverable data loss. Hence, accelerating the data reconstruction process is becoming a pressing problem.

In this paper, we propose a data temperature-based reconstruction optimization algorithm in a network-coding-based file system, which uses data temperature to schedule reconstruction sequence in node recovery process. The frequently accessed data is called hot data, and the infrequently accessed data is determined as cold data. The hot data would be reconstructed prior to the cold data during the reconstruction process. Since the user request stream normally couple with the data rebuilding stream, we intend to reduce the disk seek time and disk head shuttling, so as to improve the rebuilding performance. The method is implemented atop network coding file system (NCFS) [11]. Extensive experiments indicate that our approach significantly outperforms the existing reconstruction schemes in terms of reconstruction time, throughput and average response time.

The rest of the paper proceeds as follows. Section 2 states background and motivation, and Section 3 describes the design and implementation issues. Section 4 presents our experimental results. Section 5 concludes this paper.

2 Background and Motivation

2.1 Definitions

Maximum-Distance Separable Codes(MDS Codes): An MDS code can be defined in the following way for storage: We can divide a file of size M into k blocks, each of size M/k, encode them into n $(n>k)$ encoded blocks and spread them to n nodes. Then, the original file can be reconstructed by any k coded blocks. This mechanism is optimal in terms of the redundancy–reliability tradeoff because k blocks, each of size M/k, provide the minimum data for reconstructing the file, which is of size M. The repair degree d is introduced for data repair, such that the repair for the lost blocks of one failed node are achieved by connecting to d nodes to recover the lost blocks. Both traditional storage codes RAID5 and RAID6 are MDS codes.

RAID5: In Fig. 1 (a), for special case $n = 4$, RAID5 is a (4,3) MDS code where $n = 4$, $k = d = 3$. RAID5 can tolerate at most a single node failure. In each segment, the sole code block is generated by the bitwise XOR-summing of the $k = n - 1$ native blocks. In reconstructing, the lost block can be rebuilt from the other $n - 1$ blocks in the same segment via bitwise XOR-summing.

RAID6: In Fig. 1 (b), RAID6 is a (4,2) MDS code where $n = 4$, $k = d = 2$. RAID-6 can tolerate at most two node failures with two code blocks known as the P and Q parities (corresponding to c1 and c2 in Fig.1). The P parity is generated by the bitwise XOR-summing of the $k = n - 2$ native blocks similar to RAID5, while the Q parity is generated by coefficient XOR-summing. In reconstruction, if single or double failures happen, then each lost block can be repaired from the blocks of the same segment in other surviving nodes.

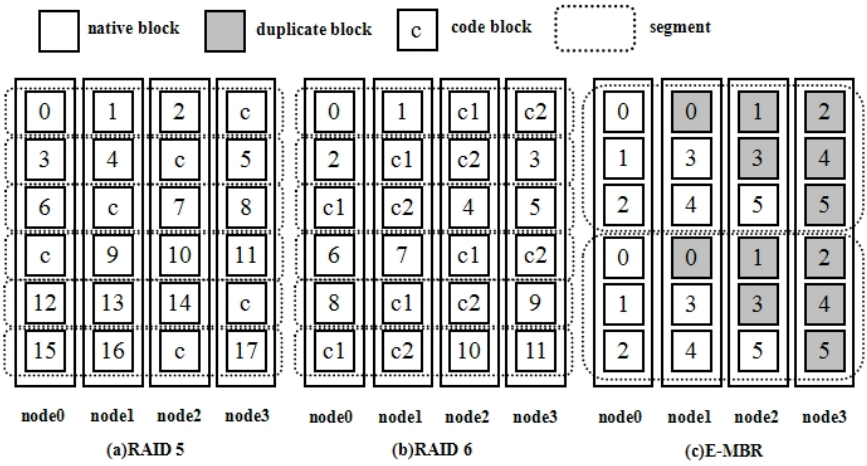


Fig. 1. The data layout of RAID5, RAID6, and E-MBR code for the case $n = 4$

2.2 MBR Codes

MBR is optimal repair bandwidth efficiency. It attains one of the two extreme points of the optimal Storage-Bandwidth Tradeoff curve [15] of regenerating codes. The tradeoff curve are given by [19], where $(\alpha_{MBR}, \beta_{MBR}) = (\frac{2Bd}{2kd-k^2+k}, \frac{2B}{2kd-k^2+k})$. When $\beta = 1$, we have $B = kd - \frac{k(k-1)}{2}$ and $\alpha = d$.

Table 1. Parameters of a regeneration code

Parameters	Descriptions
n	number of storage nodes
B	size of the source data to be stored, in terms of number of blocks
α	storage capacity of each node, in terms of number of blocks
k	the original file is recoverable from the data in any k nodes
d and β	on failure of a node, the replacement node connects to any d of the existing nodes, downloading at most β blocks from each of them
$d\beta$	reconstruct bandwidth, the total amount of data downloaded to reconstruct a failed node

2.3 E-MBR Codes

In regenerating codes, there are generally three data repair approaches [13]: (i) exact repair, which builds exactly the lost blocks in a new node, (ii) functional repair, simply reconstructs a new block that combined with the existing ones still forms an *(n,k)* MDS code, and (iii) a hybrid of both.

Dimakis[6] proposed regenerating codes. However, they gave only a theoretical description of the codes without discussing implementation issues or computational costs. Recently, some practical MSR codes [16] and exact MBR codes [14][15] are proposed. Specifically, NCFS[11] implemented exact MBR (E-MBR) code [15] into its system, along with RAID5, RAID6 coding schemes. In this paper, we focus on a particular case where $d = k = n-1$. For example, when n = 4 [Fig. 1 (c)2], specially, according to the formula $B = kd - \frac{k(k-1)}{2}$, the number of total native blocks is $B = kd - \frac{k(k-1)}{2} = 6$. For each native block, we create a duplicate copy, so the number of duplicate blocks in each segment is also *6* . According to formula $\alpha = d$, we have $\alpha = d = 3$, which means each node stores *3* blocks. When a node fails, each of living $d = 3$ contributes $\beta = 1$ block to reconstruct data on a new node.

Table 2. The Theoretical Overhead of RAID and E-MBR

Codes	Storage Cost	Reconstruction Traffic
RAID5	*B/(1-1/n)*	*B*
RAID6	*B/(1-2/n)*	*B*
E-MBR	*2B*	*2B/n*

Block allocation mechanism is shown as Fig. 1. We consider a segment of *B* native blocks M_0, M_1,…M_{B-1} and their duplicate blocks $\overline{M}_0$, $\overline{M}_1$,…$\overline{M}_{B-1}$. Thus, the total number of blocks in one segment is $2B = n(n-1)$, which means each node stores *(n-1)* blocks for each segment. There are *n(n-1)* fields for allocating if we regard it as a matrix. For each block M_i, we search for a free field from top to bottom in a column-by-column manner, starting from the leftmost column; for duplicate block $\overline{M}_i$, we search for a free field from left to right in a rowby-row manner, starting from the topmost row. Until all blocks have been distributed.

To reconstruct the data in failed node, we note that each native block has a duplicate copy, and the block and its copy are stored in two different nodes. Thus, for each lost block, we retrieve its duplicate copy from another survival node and write it to the new node. Note that based on the block allocation mechanism, each survival node contributes exactly one block for each segment. The theoretical comparison of storage cost and reconstruction traffic between RAID5, RAID6 and E-MBR is presented in Table 2.

2.4 Motivation

On the one hand, most of the distributed storage systems switch to a recovery mode after node failure to reconstruct data on a new node. On the other hand, systems continue to serve I/O requests. So we have reconstruction data stream and user request data stream contend the disk I/O simultaneously, which leads to frequent long seeks to and from the different separate data districts. Another problem is that regenerating codes take more computation than erasure codes [11][12], as a result, it needs more time to reconstruct data. Based on the test results in NCFS from our research, E-MBR suffers much more reconstruction time than either RAID5 or RAID6. They are 1.326s/MB, 0.133s/MB, 0.159s/MB respectively.

We believe that scheduling reconstruction sequence with user access patterns is a fundamental way to improve effectiveness of reconstruction process. The main idea is to reconstruct the hot data prior to the cold data so as to relieve disk I/O contentions. Tian et al. [17] proposed a popularity-based multi-threaded reconstruction optimization algorithm (PRO) to optimize the reconstruction process deployed in RAID-Structured Storage Systems by integrating the popularity and locality of workloads into the reconstruction process. To the best of our knowledge, our method is the first work that combines data temperature with the regenerating codes in reconstruction process.

3 Design and Implementation

3.1 System Architecture

The proposed idea has to track data temperature in the node so that reconstruct thread can conduct the repair sequence according to the temperature of each segment. In our architecture, there are four modules (Fig. 2.). Two of them, Scheduler and Taskexecuter, are inherent modules in NCFS. While the other two, Client and Tracker, are added by us for data temperature tracing. Data streams are represented by arrows with numbers in the Fig. 2. Streams 1,2 and 9,10 are user request data streams and reconstruction data streams respectively.

Client: It sends I/O requests to the NCFS with user access pattern, which implemented by using zipf distribution. Only read requests are considered in this paper. If the request falls on the failed segment on the failed node, reconstruct the data immediately and return it; otherwise, return the request data directly.

Tacker: Father process monitors I/O requests of Client to keeps track of access frequency of each segment, and maintain a segment number for next reconstruction. Child process communicates with Taskexecuter, conducts reconstruction sequence by using data temperature.

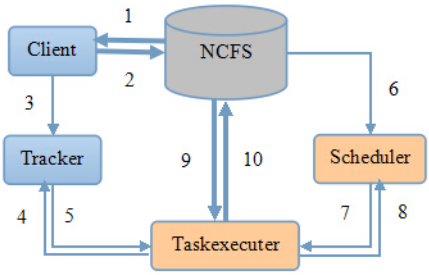


Fig. 2. System Architecture

Data Reconstruction Process: When node failure occurs, NCFS starts Scheduler and Taskexecuter. In the meantime, Client and Tracker begin to work. Scheduler receives reconstruction parameters and schedules reconstruction task, then sends task parameter struct to Taskexecuter telling it which task is going to be executed. Before task begins, Taskexecuter obtains data temperature characteristic from Tracker, then Taskexecuter launches reconstruction algorithm to rebuild data on the new node, until all the data have been reconstructed. The two major algorithms are detailed as below:

Algorithm of Client:

```
while (1) {
  initiate req_disk, recon[];
  offset = zipfDistribution() ;
  if  ( req_disk != fail_disk ) {
      open(req_disk);
      read(offset);}
  else {if  ( recon[offset] == 1 ) {
          open(req_disk);
          read(offset);
          good_req ++;}
        else if  ( recon[offset] == 0 ) {
          recover_mbr(fail_disk, new_disk);
          open(req_disk);
          read(offset);
          bad_req ++;}}
  if finish reconstruction, break; }
```

Algorithm of Tracker:

```
while (1) {
  initiate hot_seg;
  fpid = fork();
  if (fpid == 0) {    //Child process
      while (1) {hot_seg = max (temp[]);
          listen( taskexecuter_sd );
          send ( hot_seg );
          if finish reconstruction, break;}}
```

```
else if (fpid > 0) {    //father process
    recv (req_disk, offset);
    segment = offset / mbr_segment_size;
    temp [segment] ++;}
 if finish reconstruction, break;}
```

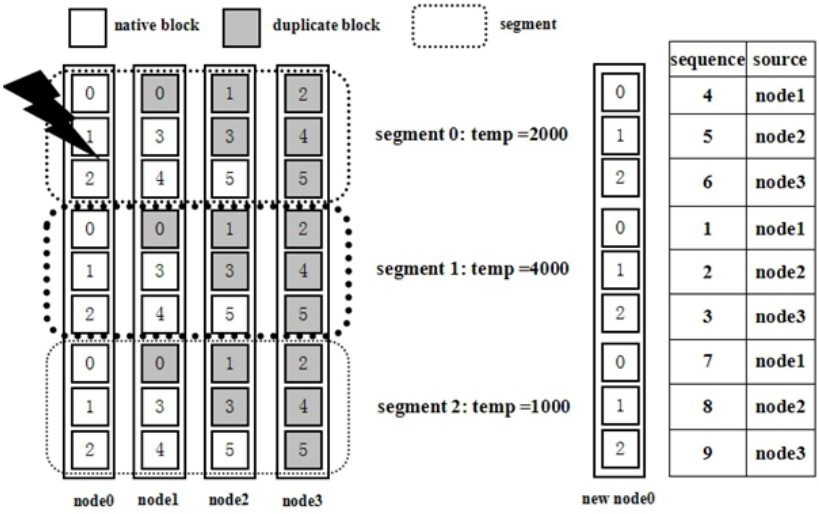


Fig. 3. An example of the reconstruction process of our method

Example: We describe our idea by Fig. 3, which shows a (4,3) E-MBR code and failures node 0. After node 0 failed, client continue to access data from node 0, the requests land up in different segments, which makes segment 1 the hottest zone, segment 0 and segment 2 in node 0 are relatively cold. When reconstruction thread starts, it selects segment 1 to rebuild data first. Inside segment 1, block rebuilding shall be at sequence order by finding corresponding duplicate blocks and copying them from other living nodes. In this case, reconstruction order should be segment 1:block 0 → segment 1:block 1 → segment 1:block 2 → segment 0:block 0 → segment 0:block 1 → segment 0:block 2 → segment 2:block 0 → segment 2:block 1 → segment 2:block 2.

3.2 Implementation Issues

Reconstruction Unit. Compare to the original approach reconstructing data block by block, our method use segment as a reconstruction unit. Considering that the data in the same segment are organized by the same allocation mechanism, it is more computational saving to recover all blocks in a segment once at a time than to recover each of them separately. On the other hand, spatial locality indicates that likelihood of referencing a resource is higher if a resource near it was just referenced. So blocks in the same segment would be very likely to be accessed by one request. Furthermore, preserving the inherent sequentiality inside segment is more conform to the disk drive

I/O pattern which is at many times the bandwidth of random accesses. This will lead to reduction of disk head rotations so as to save more reconstruction time.

4 System Evaluation

4.1 Experimental Settings

Our testbed is built on an open-source Network-Coding-Based Distributed File System (NCFS) [11], which supports a specific regenerating coding scheme called Exact Minimum Bandwidth Regenerating (E-MBR) codes [9]. We implement our approach DTemp on NCFS in the reconstruction process of E-MBR exploiting data temperature. The platform consists of 2.4GHz CPU, 2G DDR3 Memory, WDC WD5000AADS-00S9B0 disk, CentOS release 6.3 (Final), and NCFS 1.2.1.

4.2 Workloads

User access pattern is usually in accord with Pareto principle. Therefore, we use Zipf's law to imitate the distribution of workload characteristic from client. The Zipf-like distribution formula [18] depict this rule, given by $p_N(i) = \frac{\Omega}{i^{\alpha}}$, where $\Omega = (\sum_{i=1}^{N} \frac{1}{i^{\alpha}})^{-1}$, α is a constant, which is in the range $0 < \alpha \leq 1$. Let N be the total number of blocks in the node. So Ω is also a constant, while $P_N(i)$ should be the conditional probability of the ith block. For example, we have 10 blocks accessed by 10000 requests, then the Zipf-like distribution is shown in Figure 5. When $\alpha = 1$, block 0 and block 1 receive 7369 read requests while the other 8 blocks receive 2631 requests in total, which almost conform to the 80-20 rule.

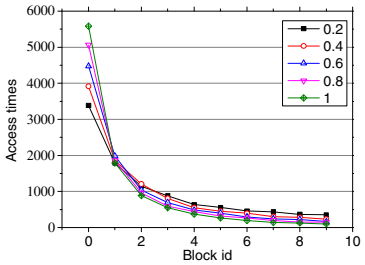


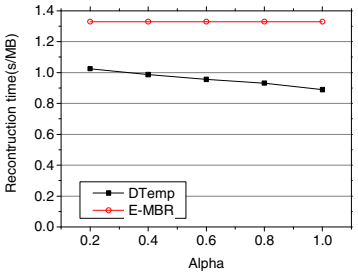
Fig. 4. Evaluate different alpha on Zipf-like access distribution

4.3 Performance Evaluation

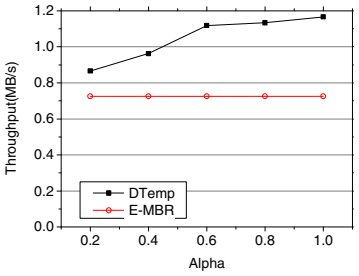
We are mainly interested in the metric of reconstruction performance including Reconstruction time, throughput and average response time. We obtain the experimental results as follows. We write 100MB of data into each node via NCFS using E-MBR. We disable one of the nodes in the array to make a single node failure. We then perform the recovery operation by starting reconstruction thread, that is,

reading data from surviving disks, reconstructing data, and writing data to a new node. In the meantime, client loads requests on the NCFS with the Zipf-like distribution workload mentioned above. Each result is the average value of at least three times experimental evaluation.

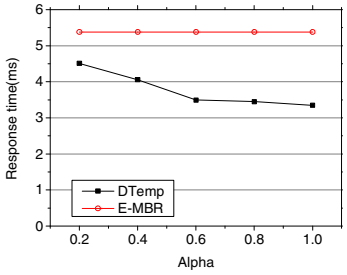
Impact of Alpha. The value of alpha indicates the dispersion degree of frequent access location. As we can see from Fig. 4. The lager the value of alpha is, the more centralized the hot data are. Apparently, the centralization of hot data is beneficial for data retrieval by reconstruction data stream and user access data stream simultaneously during the reconstruction process. The results show that DTemp perform over E-MBR at all alpha value in reconstruction time, throughput and average response time. As alpha value grows, the advantage expands.



(a) Impact of alpha on reconstruction time



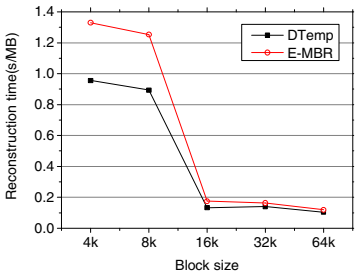
(b) Impact of alpha on throughput



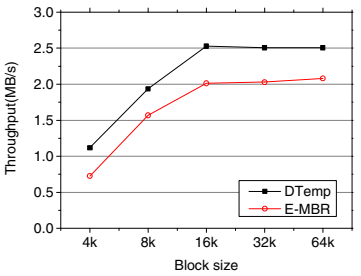
(c) Impact of alpha on average response time

Fig. 5. Impact of alpha. E-MBR is not affected by alpha. The E-MBR curve on the graph just for comparison.

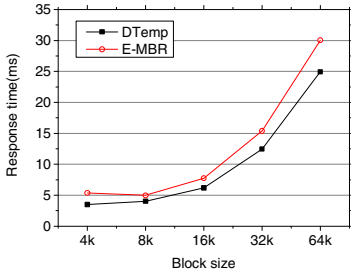
Impact of Block Size. We evaluate how different block sizes influence the reconstruction performance because block sizes directly affect I/O efficiency. Fig. 6 (a) shows the reconstruction time (s/MB) for different block sizes using conventional E-MBR and our method DTemp, respectively. We observe that as the block size increases, the reconstruction time decreases. The reason is that given the same amount of data, the block number decreases for a larger block size. In this way, the times of getting duplicate node and duplicate block are reduced greatly, which highly saves the computational cost. On the other side, average response time (Fig. 6 (c)) increase because reconstruction time of each block lengthened.



(a) Impact of block size on reconstruction time

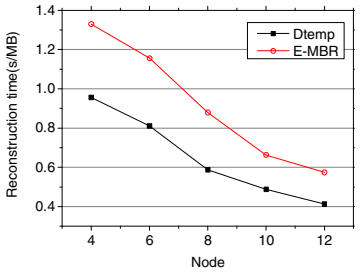


(b) Impact of block size on throughput

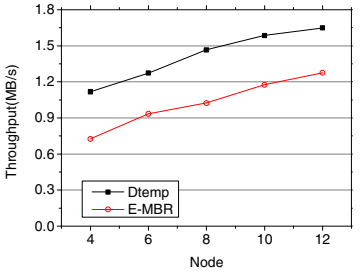


(c) Impact of block size on average response time

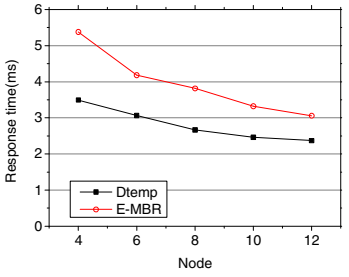
Fig. 6. Impact of block size



(a) Impact of node number on reconstruction time



(b) Impact of node number on throughput



(c) Impact of node number on average response time

Fig. 7. Impact of node number

Impact of Node Number. In this paper, we only discuss E-MBR the case of *(n,d)* where $d = n-1$. According to the coding scheme of E-MBR, each segment contains $n(n-1)$ blocks, and every nodes own $n-1$ blocks of one segment while each of these $n-1$ has a duplicate in other $n-1$ nodes respectively. When a node corrupts, all need to do is to find the duplicate blocks and copy them to the replacement node to accomplish reconstruction. As node number grows, segment size grows as well, but segment number in one node will reduce. As a result, it leads to access reduction of duplicate block read operation on one single node but gain more nodes contributing duplicate blocks together in a parallel way. That's why with node quantity grows, reconstruction performance improves.

5 Conclusions and Future Work

In this paper, we exploit data temperature into the reconstruction process of regenerating codes, which reconstruct the hot data prior to the cold data. We implement our method, call DTemp, alongside with NCFS reconstruction mechanism. Our experimental results prove that DTemp does better performance than existing E-MBR reconstruction with up to with up to 33.17%, 60.61%, 37.77% improvement in terms of reconstruction time, throughput and average response time. We believe that there are still many directions for future research on our work. One direction is to include more regenerating code schemes to evaluate reconstruction performance using data temperature for horizontal comparison. Another idea is to investigate the impacts of DTemp in distributed storage system with real workload.

Acknowledgments. This work is supported by the National Natural Science Foundation (NSF) of China under grant (No.61272073, No. 61073064), the key program of Natural Science Foundation of Guangdong Province (no. S2013020012865), the Scientific Research Foundation for the Returned Overseas Chinese Scholars (State Education Ministry), the Educational Commission of Guangdong Province (No. 2012KJCX0013). The corresponding author of this paper is Yuhui Deng.

References

1. Ghemawat, S., Gobioff, H., Leung, S.: The Google File System. In: Proc. of ACM SOSP (December 2003)
2. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: Amazon's Highly Available Key-Value Store. In: Proc. of ACM SOSP (2007)
3. Calder, B., Wang, J., Ogus, A., Nilakantan, N., Skjolsvold, A., McKelvie, S., Xu, Y., Srivastav, S., Wu, J., Simitci, H., et al.: Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In: Proc. of ACM SOSP (October 2011)
4. Rodrigues, R., Liskov, B.: High availability in DHTs: Erasure coding vs.replication. In: IPTPS (2005)

5. Weatherspoon, H., Kubiatowicz, J.D.: Erasure coding vs. replication: A quantitative comparison. In: IPTPS (2002)
6. Dimakis, A.G., Godfrey, P.B., Wu, Y., Wainwright, M., Ramchandran, K.: Network Coding for Distributed Storage Systems. IEEE Trans. on Information Theory 56(9), 4539–4551 (2010)
7. Ahlswede, R., Cai, N., Li, S.-Y.R., Yeung, R.W.: Network Information Flow. IEEE Trans. on Information Theory 46(4), 1204–1216 (2000)
8. Rashmi, K., Shah, N., Kumar, P.: Optimal Exact-Regenerating Codes for Distributed Storage at the MSR and MBR Points via a Product-Matrix Construction. IEEE Trans. on Information Theory 57(8), 5227–5239 (2011)
9. Rashmi, K.V., Shah, N.B., Kumar, P.V., Ramchandran, K.: Explicit Construction of Optimal Exact Regenerating Codes for Distributed Storage. In: Proc. of Allerton Conference (2009)
10. Suh, C., Ramchandran, K.: Exact-Repair MDS Code Construction using Interference Alignment. IEEE Trans. on Information Theory 57(3), 1425–1442 (2011)
11. Hu, Y., Yu, C.-M., Li, Y.-K., Lee, P.P.C., Lui, J.C.S.: NCFS: On the Practicality and Extensibility of a Network-Coding-Based Distributed File System. In: Proc. of NetCod (2011)
12. Duminuco, A., Biersack, E.: A Practical Study of Regenerating Codes for Peer-to-Peer Backup Systems. In: Proc. of IEEE ICDCS 2009 (2009)
13. Dimakis, A.G., Ramchandran, K., Wu, Y., Suh, C.: A survey on network codes for distributed storage. In: arXiv:1004.4438v1 [cs.IT] (2010)
14. Rashmi, K.V., Shah, N.B., Kumar, P.V.: Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a productmatrix construction. In: arXiv:1005.4178v1 [cs.IT] (2010)
15. Rashmi, K.V., Shah, N.B., Kumar, P.V., Ramchandran, K.: Explicit construction of optimal exact regenerating codes for distributed storage. In: Proc. of Allerton Conference (2009)
16. Suh, C., Ramchandran, K.: Exact-repair mds codes for distributed storage using interference alignment. In: Proc. of IEEE ISIT (2010)
17. Tian, L., Feng, D., Jiang, H., Zhou, K., Zeng, L., Chen, J., Wang, Z., Song, Z.: PRO: A Popularity-based Multi-threaded Reconstruction Optimization for RAID-Structured Storage Systems. In: FAST 2007, San Jose, CA (February 2007)
18. Breslau, L.: Pei Cao, Li Fan, G. Phillips, S Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In: Proc. of IEEE INFORCOM (March 1999)
19. Wu, Y., Dimakis, A.G., Ramchandran, K.: Deterministic Regenerating codes for distributed storage. In: Proc. Allerton Conference on Control, Computing and Communication, Urbana-Champaign, IL (September 2007)