PUSH: A Pipelined Reconstruction I/O for Erasure-Coded Storage Clusters

Jianzhong Huang, Xianhai Liang, Xiao Qin, *Senior Member, IEEE*, Qiang Cao, *Member, IEEE*, and Changsheng Xie, *Member, IEEE*

Abstract—A key design goal of erasure-coded storage clusters is to minimize reconstruction time, which in turn leads to high reliability by reducing vulnerability window size. PULL-Rep and PULL-Sur are two existing reconstruction schemes based on PULL-type transmission, where a rebuilding node initiates reconstruction by sending a set of read requests to surviving nodes to retrieve surviving blocks. To eliminate the transmission bottleneck of replacement nodes in PULL-Rep and mitigate the extra overhead caused by non-contiguous disk access in PULL-Sur, we incorporate PUSH-type transmissions to node reconstruction, where the reconstruction procedure is divided into multiple tasks accomplished by surviving nodes in a pipelining manner. We also propose two PUSH-based reconstruction schemes (i.e., PUSH-Rep and PUSH-Sur), which can not only exploit the I/O parallelism of PULL-Sur, but also maintain sequential I/O accesses inherited from PULL-Rep. We build four reconstruction-time models to study the reconstruction process and estimate the reconstruction time of the four schemes in large-scale storage clusters. We implement a proof-of-concept prototype where the four reconstruction schemes are deployed and quantitatively evaluated. Experimental results show that the PUSH-based reconstruction schemes outperform the PULL-based counterparts. In a real-world (9,6)RS-coded storage cluster, PUSH-Rep speeds up the reconstruction time by a factor of 5.76 compared with PULL-Rep; PUSH-Sur accelerates the reconstruction by a factor of 1.85 relative to PULL-Sur.

Index Terms—Erasure-coded storage cluster, reconstruction, PULL-type transmission, PUSH-type transmission, TCP Incast

1 Introduction

1.1 Motivations

Traditional reconstruction techniques in storage clusters advocate the pull model, where a master node initiates reconstruction by sending requests to worker nodes dedicated to the reconstruction process. The passive pull model inevitably encounters a transmission bottleneck problem that lies in rebuilding nodes. In this paper, we propose two PUSH-based reconstruction schemes—PUSH-Rep and PUSH-Sur—to improve reconstruction performance in a distributed storage cluster. At the heart of this study is the proactive PUSH technique that evenly distributes network and I/O loads among surviving nodes to shorten reconstruction times.

The following three factors motivate us to propose the PUSH-based reconstruction technique for erasure-coded clustered storage.

- the high cost-effectiveness of erasure-coded storage,
- the severe impact of recovery time on reliability, and
- the deficiency of PULL-based reconstruction I/Os.

Motivation 1. Erasure-coded storage clusters have increasingly become a cost-effective and fault-tolerant solution for archive storage [1], [2], data centers [3], [4], cloud storage [5], [6], and the like. Especially, Reed-Solomon (RS) codes [7], [8] are widely used in storage clusters to provide high data reliability. For example, Windows Azure Storage (WAS) adopts a variant of RS codes to implement a four-fault-tolerant cluster system [9]. A detailed review on the RS-coded distributed storage is provided in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2014.2311808.

Motivation 2. Ideally, erasure-coded storage clusters should protect against data loss caused by node failures, because high reliability is an indispensable requirement for building large-scale storage systems. The mean-time-to-data-loss or MTTDL of a r-fault-tolerant storage system is inversely proportional to the rth power of recovery time of a storage node [10]. Therefore, it is extremely important to speed up the reconstruction process, which in turn can improve system reliability by shrinking vulnerability window size [11], [12].

Motivation 3. The existing reconstruction schemes adopt a PULL-transmission mode, where a rebuilding node initiates the reconstruction by sending read requests to fetch/pull surviving blocks. Such a PULL mode not only raises the TCP *Incast* problem due to its synchronized many-to-one traffic pattern [13], but also yields poor reconstruction performance. When it comes to a reconstruction which relies on replacement nodes [2], the network traffic of replacement nodes contributes to an excessively long reconstruction

- *J. Huang is with the Department of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: hjzh@hust.edu.cn.*
- *X. Liang, Q. Cao, and C. Xie are with Wuhan National Lab. for Optoelectronics, Huazhong University of Science and Technology, Wuhan, China 430074. E-mail: {caoqiang, cs_xie}@hust.edu.cn, hustlxh@gmail.com.*
- *X. Qin is with the Department of Computer Science and Software Engineering, Shelby Center for Engineering Technology, Samuel Ginn College of Engineering, Auburn University, AL 36849-5347. E-mail: xqin@auburn.edu.*

Manuscript received 14 May 2013; revised 27 Feb. 2014; accepted 4 Mar. 2014. Date of publication 12 Mar. 2014; date of current version 9 Jan. 2015. Recommended for acceptance by A.R. Butt.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2311808

time. The problem with the reconstruction among surviving nodes [14] is that each surviving node bears extra seek time owing to the non-contiguous disk access. This problem makes the low write bandwidth become a major reconstruction performance bottleneck.

In this paper, we introduce a PUSH-type transmission to speed up node-reconstruction performance. Our PUSH enables surviving nodes to accomplish reconstruction tasks in a pipelining manner—Each surviving node combines its local block with an intermediate block from another surviving node to partially generate an intermediate block forwarded to a subsequent node. Thus, PUSH can speed up the reconstruction process by maximizing the utilization of both network and I/O bandwidth of all the surviving nodes.

1.2 Contributions

The contributions of this study are summarized as follows:

- We introduce a PUSH-type transmission (PUSH for short) in the field of node reconstruction. We propose two PUSH-based reconstruction schemes, which exploit high I/O parallelism and sequential I/ O access pattern. PUSH supports the one-to-one traffic pattern, which naturally solve the *Incast* problem.
- We develop four reconstruction-time models for the proposed schemes. The models, which are validated using a real storage system, are used to pinpoint performance bottlenecks in the reconstruction process.
- We implement PUSH in a real-world erasure-coded storage cluster, on which reconstruction processes are systematically evaluated. Experimental results show that the PUSH-based reconstruction schemes outperform the PULL-based counterparts. Under the (9, 6)RS-coded storage cluster, PUSH-Rep speeds up the reconstruction time by a factor of 5.76 compared with PULL-Rep; PUSH-Sur accelerates the reconstruction by a factor of 1.85 relative to PULL-Sur.

1.3 Roadmap

The rest of the paper is organized as follows. Section 2 surveys the related work of data reconstruction techniques. The design of PUSH is detailed in Section 3. Section 4 presents the four reconstruction-time models. Section 5 describes the experimental settings and results. Section 6 discusses important applicability issues. Finally, a conclusion is made in Section 7.

2 RELATED WORK

A vast majority of existing reconstruction techniques are optimized for disk arrays or Redundant Array of Inexpensive Disks (RAID). These reconstruction approaches can be classified into four categories:

i) Maximizing recovery parallelism. SOR creates a number of reconstruction processes associated with strips [15]; DOR makes every surviving disks busy with reconstruction reads at all time [16];
ii) Reducing interference between reconstruction and user I/Os. WorkOut speeds up the recovery process by outsourcing all write requests and popular read requests to a surrogate RAID set [17];
iii) Optimizing decoding operations. Cassidy and Hafner proposed a code-specific hybrid reconstruction algorithm to reduce XOR operations and improve decoding performance during recovery [18];
iv) Minimizing the number of reads on surviving disks. RDOR recovers a failed disk in a RDP-coded RAID with the decreased number of disk reads [19]; MICRO utilizes storage cache and RAID controller cache to diminish the number of reconstruction I/Os [20].

Increasing attention has been paid to repairing solutions in the arena of erasure-coded storage clusters. Representative reconstruction approaches tailored for clusters can be divided into three groups:

i) Improving reconstruction I/O parallelism. HydraStor makes all remaining nodes contribute to data rebuilding (i.e., *bulk rebuilding*), which maximizes I/ O utilization [14]. The Per-file RAID offered by Panasas allows a metadata manager to rebuild files in parallel [21].
ii) Reducing parity-group size. WAS divides data fragments into multiple equal-sized groups and computes one local parity for each group, thereby reducing the number of surviving nodes needed to reconstruct a failed data fragment [9];
iii) Minimizing the number of reconstruction I/Os. A handful of *Regenerating Codes* designed to optimize reconstruction I/O bandwidth include MSR/MBR [22], MCR [23], and MBCR [24], to name just a few.

In summary, *Regenerating Codes* achieve repair optimization by designing a new linear coding scheme. Both HydraStor and WAS adopt the pull-type transmission pattern—a certain node (e.g., a replacement node, or a surviving node acting as a rebuilding node) simultaneously issues multiple reconstruction read requests to surviving nodes. In particular, *k* surviving blocks are fetched to a different Extent Node (EN) performing as a replacement node in WAS; *k-1* surviving blocks are delivered to one surviving node in HydraStor. Different from the existing PULL-based reconstruction schemes, our PUSH technique aims to fully exploit both network and I/O bandwidth to significantly speed up the recovery of failed storage nodes.

3 PUSH RECONSTRUCTIONS

3.1 Preliminaries

In contrast to replication, erasure codes can provide equivalent fault tolerance with significantly low storage overhead, hence saving storage bandwidth [22]. We show how to build a (k+r,k) RS-coded storage cluster. Since RS codes are systematic in nature, source data $\{D_1, D_2, \ldots, D_k\}$ are embedded in encoded data $\{D_1, D_2, \ldots, D_k, P_1, P_2, \ldots, P_r\}$. Fig. 1 illustrates that parity blocks $\{P_1, P_2, \ldots, P_r\}$ are generated by multiplying *k* data blocks with the $k \times r$ redundancy matrix. For simplicity, both data and parity blocks are exclusively stored on the $k + r$ storage nodes, which

$$\underbrace{\begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,k} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,k} \\ \vdots & \vdots & & \vdots \\ \alpha_{r,1} & \alpha_{r,2} & \cdots & \alpha_{r,k} \end{bmatrix}}_{\text{Transposed Redundancy Matrix}} \times \underbrace{\begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_k \end{bmatrix}}_{\text{Data Block}} = \underbrace{\begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_r \end{bmatrix}}_{\text{Parity Block}}$$

Fig. 1. Generating parity blocks in (k+r,k) Reed-Solomon codes.

are designated data nodes $\{DN_1, DN_2, \ldots, DN_k\}$ and parity nodes $\{PN_1, PN_2, \ldots, PN_r\}$.

Let us take a single-node failure in a (6, 4)RS-coded storage cluster as an example. Without losing generality (i.e., thanks to the symmetry of RS coding), we assume data node DN_1 is faulty. According to the encoding algorithm of RS codes, the parity blocks $P_{row,1}$ and $P_{row,2}$ at the rowth row can be created from data blocks as

$$P_{row,1} = \alpha_{1,1}D_{row,1} + \alpha_{1,2}D_{row,2} + \alpha_{1,3}D_{row,3} + \alpha_{1,4}D_{row,4} \quad (1)$$

and,

$$P_{row,2} = \alpha_{2,1}D_{row,1} + \alpha_{2,2}D_{row,2} + \alpha_{2,3}D_{row,3} + \alpha_{2,4}D_{row,4}. \quad (2)$$

The coefficient $\alpha_{1,i}$ (i $\in$ [1,2,. . .,k]) is 1 in the Vandermonde Reed-Solomon codes [8], [25]. Thus, we can derive block $P_{row,1}$ from blocks $D_{row,1}, \ldots, D_{row,4}$ as:

$$P_{row,1} = D_{row,1} + D_{row,2} + D_{row,3} + D_{row,4}. \quad (3)$$

The MDS property of RS codes suggests that a failed block can be recovered from any k surviving blocks. Thus, we calculate blocks $\{D_{row,1}, D_{row+1,1}, \ldots, D_{row+4,1}\}$ using Eqs. {4, 5, . . ., 8}, respectively. For example, Eq. (4) indicates that a replacement node can fetch four surviving blocks $\{P_{row,1}, D_{row,2}, D_{row,3}, D_{row,4}\}$ to regenerate the failed block $D_{row,1}$. Note that Eq. (4) can be decomposed as follows:

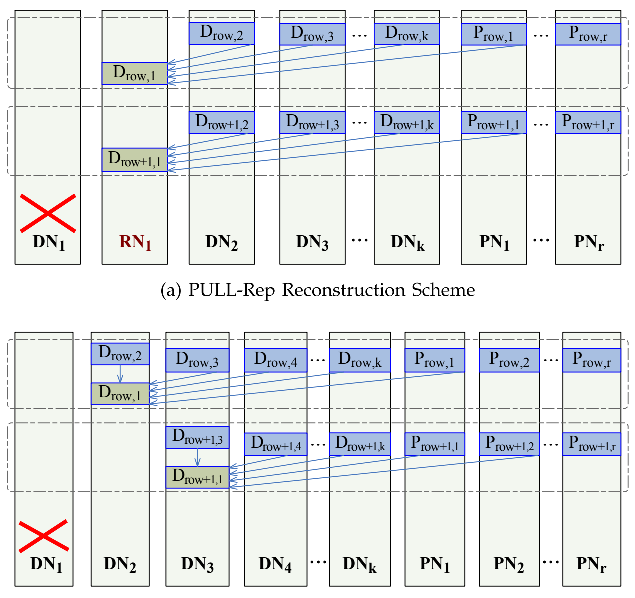
$$I_1 = I_0 + (1/\alpha_{1,1})P_{row,1}, \text{ where } I_0 = 0, \quad (4.1)$$

$$I_2 = I_1 + (-\alpha_{1,2}/\alpha_{1,1})D_{row,2}, \quad (4.2)$$

$$I_3 = I_2 + (-\alpha_{1,3}/\alpha_{1,1})D_{row,3}, \quad (4.3)$$

$$D_{row,1} = I_3 + (-\alpha_{1,4}/\alpha_{1,1})D_{row,4}. \quad (4.4)$$

Since storage nodes offer sufficient computing capability in addition to I/O services [2], [26], [27], nodes $\{PN_1, DN_2, DN_3\}$ can compute the linear combinations using Eqs. {4.1, 4.2, 4.3} and then forwards intermediate blocks $\{I_1, I_2, I_3\}$ to nodes $\{DN_2, DN_3, DN_4\}$, respectively. Thus, each surviving node accomplishes a portion of the entire reconstruction process, making it possible to evenly distribute network and computing load caused by node reconstruction among the surviving nodes.



(a) PULL-Rep Reconstruction Scheme

(b) PULL-Sur Reconstruction Scheme

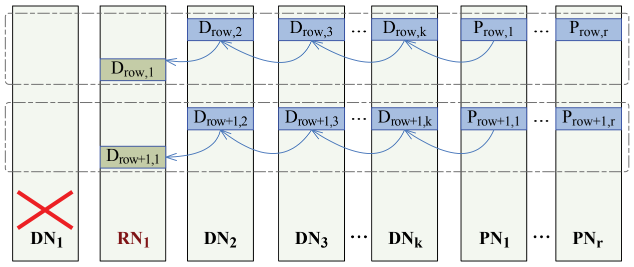
Fig. 2. Two PULL-mode-based Reconstruction Schemes. (a) PULL-Rep: A replacement node fetches *k* surviving blocks and rebuilds the corresponding failed block. (b) PULL-Sur: each surviving node fetches $k-1$ surviving blocks and rebuilds the corresponding failed blocks. After the entire reconstruction process is completed, all rebuilt blocks are migrated to an new node.

3.2 Two PULL-Based Reconstruction Approaches

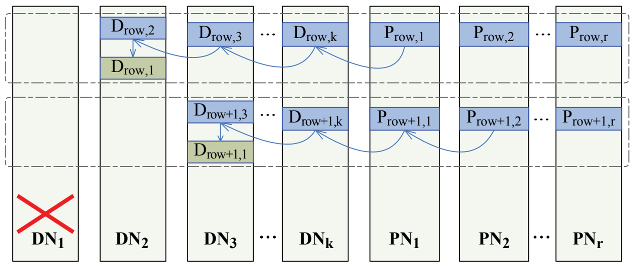
Let us consider two existing reconstruction techniques that rely on the pull mode, where a rebuilding node first *issues* read requests to surviving nodes and then reconstructs a failed block using the requested blocks.

The PULL-based reconstruction can be envisioned as a master-worker computing model [21], in which a *master* triggers a reconstruction procedure by sending a set of read requests, then waits for the requests to be completed by *workers*. There are two classical reconstruction approaches in real-world erasure-coded storage clusters: i) a designated master (e.g., a replacement node) fetches *k* surviving blocks and reconstructs a failed block [2], [9]; and ii) each surviving node plays the role of a master (i. e., acting as a rebuilding node) and all surviving nodes perform as workers, where write I/Os of rebuilt blocks are spread out over all the surviving nodes [14], [21]. From the angle of message communication, this 'Master-Worker' pattern belongs to the category of PULL-type transmission. Throughout this paper, we refer to the reconstruction scheme using replacement nodes as *PULL-Rep*; we term the solution of distributing reconstruction load among surviving nodes as *PULL-Sur*.

In the case of PULL-Rep, all reconstruction reads are sequential requests that minimize disk seeking times; rebuilt blocks are sequentially written to disks of replacement nodes. Fig. 2a shows that *k* surviving blocks should be delivered to a replacement node (e.g., RN_1), which becomes a network bottleneck that slows down the entire reconstruction process. Furthermore, such a many-to-one ('M:1') communication pattern may cause the severe *Incast* problem (see Section 3.4 and Appendix B, available online, for details).



(a) PUSH-Rep Reconstruction Scheme



(b) PUSH-Sur Reconstruction Scheme

Fig. 3. Two PUSH-mode-based Reconstruction Schemes. (a) PUSH-Rep: A surviving node creates an intermediate block using a linear combination based on what it owns and receives, sends the resulting intermediate block to a subsequent node; (b) PUSH-Sur: This scheme is similar to PUSH-Rep except that all the surviving nodes are concurrently reconstructing blocks.

As shown in Fig. 2b, PULL-Sur allows each surviving node to rebuild a subset of failed data. As a result, all the surviving nodes accomplish the reconstruction in parallel. The downside of PULL-Sur is that apart from serving one read and one write, a surviving node must simultaneously respond to $k-1$ read requests from other surviving nodes. The write ratio of PULL-Sur is $1/(\mathrm{k}+1)$; I/O requests in PULL-Sur are not sequential, which result in a low write throughput. For example, our evidence (see Fig. 6) confirms that the write throughput is less than 5.0 MBps under 14 percent write ratio and 60 percent random distribution.

3.3 Two PUSH-Based Reconstruction Schemes

The goal of the PUSH technique for node reconstruction is two-fold. First, PUSH aims to alleviate the reconstruction performance bottleneck caused by a replacement node's network bandwidth in PULL-Rep. Second, PUSH also aims to mitigate extra seeking times induced by the non-sequential disk accesses in PULL-Sur. In comparison to surviving nodes that passively respond to reconstruction reads in PULL, the surviving nodes in PUSH proactively participate in the entire reconstruction process:

$$\mathrm{D}_{\mathrm{row},1} = (\mathrm{P}_{\mathrm{row},1} - \alpha_{1,2}\mathrm{D}_{\mathrm{row},2} - \alpha_{1,3}\mathrm{D}_{\mathrm{row},3} - \alpha_{1,4}\mathrm{D}_{\mathrm{row},4})/\alpha_{1,1}, \tag{4}$$

$$\begin{aligned}\mathrm{D}_{\mathrm{row}+1,1} = (&\alpha_{2,2}\mathrm{P}_{\mathrm{row}+1,1} - \mathrm{P}_{\mathrm{row}+1,2} - (\alpha_{2,2} - \alpha_{2,3})\\ &\mathrm{D}_{\mathrm{row}+1,3} - (\alpha_{2,2} - \alpha_{2,4})\mathrm{D}_{\mathrm{row}+1,4})/(\alpha_{2,2} - \alpha_{2,1}),\end{aligned} \tag{5}$$

$$\begin{aligned}\mathrm{D}_{\mathrm{row}+2,1} = (&\alpha_{2,3}\mathrm{P}_{\mathrm{row}+2,1} - \mathrm{P}_{\mathrm{row}+2,2} - (\alpha_{2,3} - \alpha_{2,2})\\ &\mathrm{D}_{\mathrm{row}+2,2} - (\alpha_{2,3} - \alpha_{2,4})\mathrm{D}_{\mathrm{row}+2,4})/(\alpha_{2,3} - \alpha_{2,1}),\end{aligned} \tag{6}$$

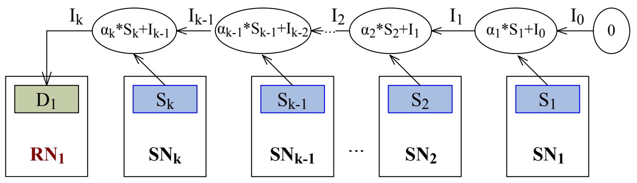


Fig. 4. The I/O procedure of PUSH-Rep. All the nodes involved in the reconstruction form a reconstruction chain.

$$\begin{aligned}\mathrm{D}_{\mathrm{row}+3,1} = (&\alpha_{2,4}\mathrm{P}_{\mathrm{row}+3,1} - \mathrm{P}_{\mathrm{row}+3,2} - (\alpha_{2,4} - \alpha_{2,2})\\ &\mathrm{D}_{\mathrm{row}+3,2} - (\alpha_{2,4} - \alpha_{2,3})\mathrm{D}_{\mathrm{row}+3,3})/(\alpha_{2,4} - \alpha_{2,1}),\end{aligned} \tag{7}$$

$$\begin{aligned}\mathrm{D}_{\mathrm{row}+4,1} = (&\mathrm{P}_{\mathrm{row}+4,2} - \alpha_{2,2}\mathrm{D}_{\mathrm{row}+4,2} - \alpha_{2,3}\\ &\mathrm{D}_{\mathrm{row}+4,3} - \alpha_{2,4}\mathrm{D}_{\mathrm{row}+4,4})/\alpha_{2,1}.\end{aligned} \tag{8}$$

We refer to the PUSH-based reconstruction using replacement nodes as *PUSH-Rep*; we call the PUSH-based scheme that distributes reconstruction load among surviving nodes as *PUSH-Sur*. In PUSH-Rep and PUSH-Sur, each surviving node first combines a locally stored block with a block received from another node to produce part of a final block, and then delivers the resulting intermediate block to a subsequent node (see Fig. 3). In doing so, all the surviving node can devote all their resources, including CPU time, I/O capacity and network bandwidth, to the reconstruction process. Conceptually, a surviving node is an object-based storage device that can semi-independently manipulate its stored data [28].

Fig. 4 illustrates the I/O processing of PUSH-Rep, where all the nodes involved in the reconstruction process form a *reconstruction chain*, (e.g., $\{\mathrm{SN}_1 \rightarrow \mathrm{SN}_2 \rightarrow \cdots \rightarrow \mathrm{SN}_k \rightarrow \mathrm{RN}_1\}$). The 'PUSH' step in surviving node SN_i includes the following operations: (i) to read a surviving block S_i from a local disk; (ii) to receive an intermediate block $\mathrm{I}_{\mathrm{i}-1}$ from another node; (iii) to compute a linear combination of the multiple of S_i with $\mathrm{I}_{\mathrm{i}-1}$, and (iv) to deliver a resulting block (i.e., $\alpha_\mathrm{i} \times \mathrm{S}_\mathrm{i} + \mathrm{I}_{\mathrm{i}-1}$) to the subsequent node in the reconstruction chain.

PUSH involves multiple storage nodes (e.g., k surviving nodes and a replacement node in PUSH-Rep). Only after each node pushes local intermediate blocks to the node's corresponding destination can failed blocks be successfully reconstructed. So the operations of reading a local block or receiving an intermediate block over network may stall the reconstruction process. To address this performance issue, we pre-allocates a memory region in each surviving node to cache both local and intermediate blocks (see Fig. 5a). Each block is associated with a tag indicating the block's status. If two blocks in a block pair (e.g., $\mathrm{S}_{\mathrm{row,i}}$ and $\mathrm{I}_{\mathrm{row,i}-1}$) are tagged with '1', then the node will be triggered to carry out a linear-combination computation. Furthermore, each pre-allocated memory region should be sufficiently large to maintain affluent reconstruction processes and to achieve high I/O throughput (see Appendix E, available online).

Each surviving node in PUSH-Sur should store final rebuilt blocks to its local disk. In practice, a rebuilt block

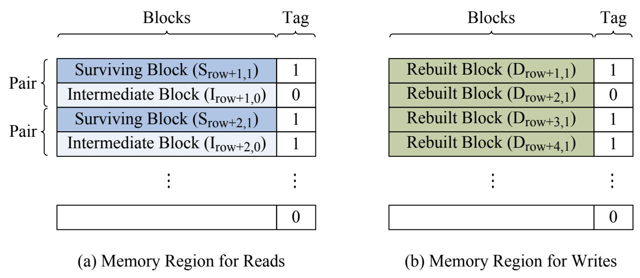

Blocks
Tag
Pair
Surviving Block ($S_{row+1,1}$) 1
Intermediate Block ($I_{row+1,0}$) 0
Pair
Surviving Block ($S_{row+2,1}$) 1
Intermediate Block ($I_{row+2,0}$) 1
0
Blocks
Tag
Rebuilt Block ($D_{row+1,1}$) 1
Rebuilt Block ($D_{row+2,1}$) 0
Rebuilt Block ($D_{row+3,1}$) 1
Rebuilt Block ($D_{row+4,1}$) 1
0
(a) Memory Region for Reads
(b) Memory Region for Writes

Fig. 5. PUSH's RAM Regions, where a tag indicates the status of a block. (a) A memory region for reads is used to cache both local surviving blocks and intermediate blocks; (b) A memory region for writes is applied to collect rebuilt blocks that are then stored to a disk.

is written to a separate spare space in the disk, leading to small and non-sequential I/Os. To reduce seek times, we also pre-allocate a memory region for writes (see Fig. 5b). The RAM region is dedicated to collecting reconstruction writes to form a large sequential write transferred onto the disk when several contiguous rebuilt blocks are ready.

3.4 Incast in Reconstruction

Phanishayee et al. observed that the TCP-*Incast* problem is caused by packet loss due to the 'M:1' communication and insufficient buffer space allocated at an Ethernet switch [13]. The switch buffers usually are overloaded by large influx of messages delivered in the 'M:1' communication; then dropped packets in turn trigger the TCP/IP retry mechanism and the *multiplicative decrease* algorithm in TCP will halve TCP/IP windows; finally, observed throughput (a.k.a. Goodput) is substantially reduced.

Recall that the existing PULL-based reconstruction schemes have the 'M:1' communication pattern. For example, k blocks are simultaneously fetched from k surviving nodes to a replacement node in PULL-Rep (see Fig. 2a). Such an 'M:1' communication pattern can induce the TCP *Incast* problem [13]. We carry out a set of PULL-Rep tests where parameter k is varied from 1 to 18 (see Appendix B, available online). The experimental results clearly reveal that the *Incast* problem exists in the PULL-based reconstruction schemes, in which a serious breakdown in read bandwidth is emerging when the coding parameter k increases.

Vasudevan et al. use high-resolution timers (e.g., to set TCP Retransmission Timeout (RTO) to 200 μs) to mitigate the adverse impact of the *Incast* problem in Ethernet-based clusters in data centers [29], [30]. These fine-granularity timeout approaches can alleviate the *Incast* effect, however, they are incapable of eliminating the root cause of the *Incast* problem. More importantly, thanks to the '1:1' communication, our proposed PUSH-based reconstruction schemes can obviate the occurrence of *Incast*.

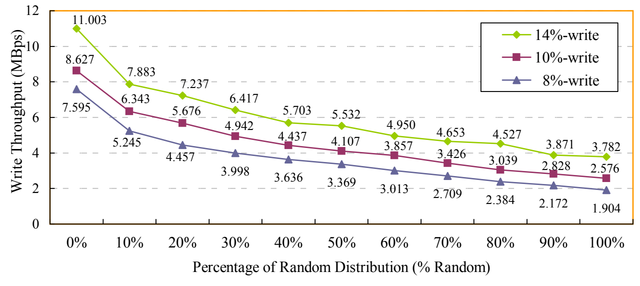


Fig. 6. Impact of random distribution percent on write throughput with respect to different write percentages (14, 10 and 8 percent).

TABLE 1
Symbol and Notation

Symbol	Notation
$D_{row,col}$	col^th data block at row^th row, with $1 \leq col \leq k$
$S_{row,i}$	i^th surviving block at row^th row
$I_{row,i}$	an intermediate block generated from $S_{row,i}$
S_{node}, S_{block}	Node capacity and block size in MB, respectively
$t_{read\_S_{row,i}}$	Time spent in reading block $S_{row,i}$
$t_{read\_send\_S_{row,i}}$	Time spent in reading and sending block $S_{row,i}$
$t_{recv\_<blk>}$	Receiving time, $<blk> \in [S_{row,i}, I_{row,i}, D_{row,col}]$
$t_{comp\_<blk>}$	Computing time, $<blk> \in [D_{row,col}, I_{row,i}]$
$t_{write\_D_{row,col}}$	Write time of data block $D_{row,col}$
$t_{push\_S_{row,i}}$	Pushing time spent by i^th surviving node (read $S_{row,i}$; receive $I_{row,i-1}$; comp and send $I_{row,i}$)
$T_{PULL\text{-}Rep\_D_{row,col}}$	Reconstruction time of $D_{row,col}$ in PULL-Rep
$T_{PULL\text{-}Rep}$	Total reconstruction time in PULL-Rep
$T_{PULL\text{-}Sur\_D_{row,col}}$	Reconstruction time of $D_{row,col}$ in PULL-Sur
$T_{PULL\text{-}Sur}$	Total reconstruction time in PULL-Sur
$T_{PUSH\text{-}Rep\_D_{row,col}}$	Reconstruction time of $D_{row,col}$ in PUSH-Rep
$T_{PUSH\text{-}Rep}$	Total reconstruction time in PUSH-Rep
$T_{PUSH\text{-}Sur\_D_{row,col}}$	Reconstruction time of $D_{row,col}$ in PUSH-Sur
$T_{PUSH\text{-}Sur}$	Total reconstruction time in PUSH-Sur

4 Reconstruction Models

This section presents analytical reconstruction models to predict performance of PUSH (i.e., PUSH-Rep, and PUSH-Sur) as well as the existing counterparts (i.e., PULL-Rep, PULL-Sur). The accuracy of all the four performance models is validated against the results collected on a real-world storage cluster.

Table 1 lists the notation used in the models.

4.1 Reconstruction Time

There are clear distinctions between recovery and reconstruction from the standpoint of recovery stage (see Appendix C, available online). In this study, we investigate reconstruction schemes and evaluate their performance in terms of reconstruction time. Due to the space limit, here we only present four equations of reconstruction time for the four schemes, and the derivation of reconstruction-time equations is detailed in Appendix D, available online.

Reconstruction times $T_{PULL-Rep}$, $T_{PULL-Sur}$, $T_{PUSH-Rep}$ and $T_{PUSH-Sur}$ can be expressed as Eqs. (9), (10), (11) and (12), respectively,

$$T_{PULL-Rep} = \left(\sum_{i=1}^{k} t_{recv\_S_{row,i}}\right) \times \frac{S_{node}}{S_{block}}, \tag{9}$$

$$T_{PULL-Sur} = \max_{i=0}^{k+r-2} \left\{t_{write\_D_{row+i,col}}\right\} \times \frac{S_{node}}{(k+r-1)S_{block}}, \tag{10}$$

$$T_{PUSH-Rep} = \max_{i=1}^{k} \left\{t_{recv\_I_{row,i}}, t_{read\_S_{row,i}}\right\} \times \frac{S_{node}}{S_{block}}, \tag{11}$$

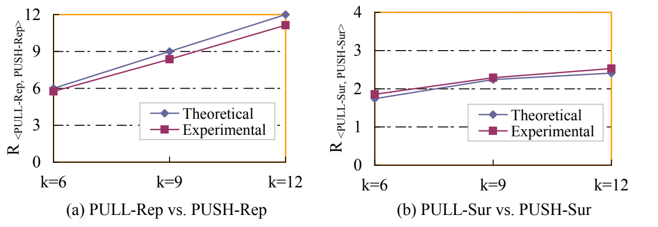


Fig. 7. Comparisons of reconstruction time ratio obtained from experimental results and theoretical formulas.

$$T_{\text{PUSH-Sur}} = \max\left\{\sum_{i=1}^{k-1} t_{\text{recv\_I}_{\text{row},i}}, t_{\text{write\_D}_{\text{row},\text{col}}}\right\} \times \frac{S_{\text{node}}}{(k+r-1)S_{\text{block}}}. \tag{12}$$

4.2 Model Validation

We validate the above reconstruction models by comparing reconstruction times obtained from the models with experimental data collected on a real-world storage cluster.

4.2.1 Write Throughput

To evaluate the impact of percent of random distribution on the write throughput, we run IOMeter [31] on HDD disks deployed in a tested storage cluster (see details of the experimental environment in Section 5.1). Recall that the write ratio is 1/(k+1) in PULL-Sur, we set the write ratio to be 14, 10, and 8 percent using parameter *k* of 6, 9, and 12, respectively. We draw two observations from Fig. 6. First, the write throughput is very low in the case of non-sequential accesses; therefore, the dominating factor of PULL-Sur's reconstruction time is write throughput rather than the surviving-block-receiving bandwidth. Note that, the network bandwidth available to receive *k−1* surviving blocks ranges from 700 to 900 Mbps. Second, the percent of random distribution has a significant impact on the write throughput, indicating that improving PULL-Sur's write throughput can be achieved by batching small non-sequential writes into large sequential ones.

4.2.2 Reconstruction Time

Let $R_{<\text{PULL-Rep,PUSH-Rep}>}$ be a ratio between the reconstruction times of PULL-Rep and PUSH-Rep. We can derive $R_{<\text{PULL-Rep,PUSH-Rep}>}$ from Eqs. (9) and (11) as:

$$R_{<\text{PULL-Rep,PUSH-Rep}>} = \frac{\sum_{i=1}^{k} t_{\text{recv\_S}_{\text{row},i}}}{\max_{i=1}^{k}\{t_{\text{recv\_I}_{\text{row},i}}, t_{\text{read\_S}_{\text{row},i}}\}}. \tag{13}$$

With the pre-allocated RAM region in place, each surviving node enjoys a sequential read pattern to fetch surviving blocks. The read throughput is larger than the receiving bandwidth (120 versus 800 Mbps), so the ratio $R_{<\text{PULL-Rep,PUSH-Rep}>}$ is approximated as *k*.

Each surviving node simultaneously responds to $k-1$ reconstruction reads, one local read and one local write in PULL-Sur, thereby resulting in a random I/O access. When *k* is set to 6, a large number of (e.g., 80 percent) I/O accesses in a surviving node are random; when *k* is large (e.g., 9 and 12), the random percent increases (e.g., 90 percent). Although PUSH-Sur reduces the percentage of random

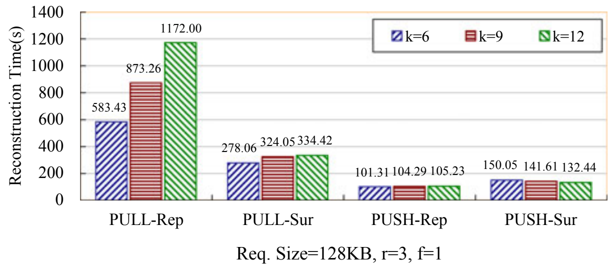


Fig. 8. Reconstruction times of the four schemes with respect to different numbers of data nodes (k = 6, 9, 12). SRU = 128 KB, r = 3, f = 1.

accesses, a small number of (e.g., 10 percent) I/Os are not sequential. This is because rebuilt blocks should be written to a separate free disk space. Therefore, reconstruction-time ratio $R_{<\text{PULL-Sur,PUSH-Sur}>}$ between PULL-Sur and PUSH-Sur is approximately 1.74, 2.24, and 2.41 when k is 6, 9, and 12 using the experimental data in Fig. 6, respectively.

Fig. 7 shows the reconstruction time derived from our models (i.e., Eqs. (9), (10), (11), (12)) and the experimental results obtained from the implemented prototypes on a storage cluster (see Fig. 8). It is observed that the difference between the theoretical ratios and experimental counterparts is very small (≤8 percent). The model validation confirms that the proposed performance models can accurately predict the reconstruction times spent in rebuilding failed nodes in erasure-coded storage clusters.

4.3 Model Applicability

The models described and validated in this section can be applied in the following four scenarios. First, the models allow us to quantify the reconstruction times of the PULL-Rep, PULL-Sur, PUSH-Rep, and PUSH-Sur schemes, respectively. It is usually intractable to leverage prototypes to evaluate multiple reconstruction schemes on large-scale storage clusters. The models can be applied to estimate reconstruction times of erasure-coded clusters where fault-tolerance parameters 'k' and 'r' are large.

Second, the models offer insightful design guidelines for reconstruction schemes. Let us consider PUSH-Sur as an example. The model suggests that a large number *r* of parity nodes shortens the reconstruction time (see Eq. (12)). When the parameter *r* goes up, an increasing number of available nodes can participate in the reconstruction process to collaboratively reduce reconstruction time. On the other hand, increasing number of parity nodes leads to high building cost of a cluster without increasing storage spaces. Therefore, there exists a tradeoff between performance gain and cost of storage clusters.

Third, the models are used to pinpoint potential performance problems. Let us take the heterogeneity issue as an example (see also Section 6—Further Discussions). Among a group of heterogeneous surviving nodes, a node with poor I/O performance may become the bottleneck in PULL-Sur due to increased writing time $t_{\text{write\_D}_{\text{row,col}}}$.

Last, the models confirm that a large value of parameter *k* has a negative impact on reconstruction times. The reconstruction bandwidth of PULL-Rep usually is a function of network bandwidth and parameter *k*. A large number k of data nodes can lead to a long reconstruction time. Therefore, some existing erasure-coded storage (e.g., WAS [9]) are inclined to reduce the parity group size.

5 Performance Evaluation

We implement the proposed PUSH-based reconstruction schemes along with the two alternatives (i.e., PULL-Rep and PULL-Sur) in a real-world storage cluster. We conduct a wide range of experiments to quantitatively compare the reconstruction performance of the four solutions.

5.1 Experimental Setup

Our testbed is an RS-coded storage cluster that consists of 18 commodity-based storage nodes and a master node (i.e., server). All the nodes are connected through a Cisco GibE switch. Each storage node contains an Intel(R) E5800 @ 3.2 GHz CPU, 2,GB DDR3 memory, and an integrated Gigabit Ethernet interface. All the disks installed in the nodes are West Digital's Enterprise WD1003FBYX SATA2.0 disks. The operating systems running in the storage nodes is Ubuntu 10.04 X86 64 (Kernel 2.6.32); the operation system installed in the storage server is Fedora 12 X86_64 (Kernel 2.6.32). The master node is equipped with two Xeon(R) X5650 @2.80 GHz (four cores) CPUs, 12 GB DDR3 memory, and the Intel X58 Chipset Mainboard, and serves as a replacement node in the case of TCP *Incast* test (see Appendix B, available online). All the storage nodes play the role of rebuilding nodes and surviving nodes in the tests of PULL-Rep, PULL-Sur, PUSH-Rep, and PUSH-Sur.

5.2 Evaluation Methodology

Evidence shows that a configuration of '$r = 3$' achieves a sufficiently large MTTDL for archival storage systems [2]. Moreover, code parameter '$r \geq 3$' is a common setting in production storage systems, e.g., '$r = 3$' in the Google's new GFS [32] and '$r = 4$' in Microsoft's WAS [9]. We adopt '$r = 3$' and '$r = 4$' in our experiments to resemble real-world storage cluster systems. Since a single-node failure is the most common case, we mainly focus on the scenario of single-node reconstruction.

The amount of data stored on each storage node is set to 10 GBytes, which is sufficiently large to evaluate the reconstruction times of the tested solutions. We set the size of pre-allocated memory region for reads to 256 MBytes. The reconstruction performance is measured in terms of the completion time spent in reconstructing 10 GB data. Each experiment is repeatedly conducted five times; then the average reconstruction time is calculated.

In both PULL-Sur and PUSH-Sur, each surviving node performs as a rebuilding node that writes rebuilt blocks to its local disk. For the sake of simplicity, we let each surviving node equally rebuild 10 GB/($k+r-1$) data. The longest reconstruction time among the $k+r-1$ nodes is a metric of the reconstruction performance. To make a fair comparison between PULL-Sur and PUSH-Sur, we pre-allocate two memory regions (i.e., one for reads and another for writes) in each surviving node for PULL-Sur.

Both PUSH-Rep and PUSH-Sur compute a linear combination of a set of data blocks using the finite field arithmetic from Jerasure [33].

To investigate PUSH-type transmissions, we compare the PUSH-based reconstruction schemes with the PULL-based reconstruction schemes that eliminate the adverse impact of *Incast*. Hence, we set parameter RTO to be 200 μs in both the PULL-Rep and PULL-Sur schemes.

5.3 Experimental Results

The reconstruction performance is affected by several important factors, including the number k of data nodes, the redundancy r of erasure codes, the number f of failed nodes, and the request unit size (*SRU*).

5.3.1 k—*Number of Data Nodes*

In (k+r,k)RS-coded storage, k surviving blocks are loaded to reconstruct a failed block. We evaluate the impacts of the number k of surviving blocks on reconstruction performance by setting the value of k to 6, 9, and 12, respectively. Fig. 8 plots the reconstruction times of the four schemes when parameters SRU, r, and f are set to 128 KB, 3, and 1, respectively.

It is observed that with the increasing value of k, PULL-Rep achieves poorer reconstruction performance (see the first three bars in Fig. 8). The reason lies in the fact that, in PULL-Rep, a large k value makes an excessive number of surviving blocks loaded to rebuild a failed block, which in turn causes a long receiving time experienced by the replacement node. PUSH-Rep is sightly sensitive to parameter k, because k surviving nodes constitute a reconstruction pipeline and all the surviving nodes receive equal amount of data regardless of the k value.

PULL-Sur exhibits long reconstruction time when parameter k is large, because the percent of random distribution increases along with the increasing value of k. On the contrary, the reconstruction time of PUSH-Sur decreases when the data node number k increases, thanks to the fact that the number of blocks rebuilt by one surviving node reduces (see the second item in Eq. 10).

Fig. 8 shows that PUSH-Rep can speed up the reconstruction process of PULL-Rep by a factor of 5.76, 8.37, and 11.14 when k is set to 6, 9, and 12, respectively. Compared with PULL-Sur, PUSH-Sur accelerates the reconstruction by a factor of 1.85, 2.29, and 2.53 when k equals to 6, 9, and 12, respectively. The reason why the reconstruction time of PULL-Sur is larger than that of PUSH-Sur is that a surviving node has a slim chance of performing sequential I/Os in PULL-Sur. Although a surviving node can prefetch multiple contiguous blocks to its RAM region to boost read performance, such a prefetching mechanism is unable to guarantee sequential reads in PULL-Sur due to the fact that this surviving node must passively serve read requests from k-1 other surviving nodes while the requested surviving blocks are residing on different disk regions. However, in PUSH-Sur, each surviving node can sequentially read blocks to its local RAM region for reads, thereby minimizing random read I/O load.

5.3.2 r—*Redundancy of Erasure Codes*

In this group of experiments, we examine the sensitivity of the four reconstruction schemes to the redundancy r of erasure codes. We conduct the experiments on the storage

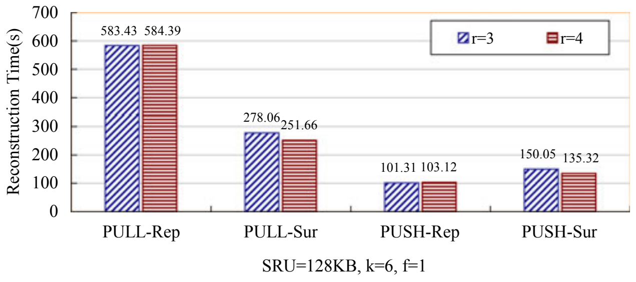


Fig. 9. Comparisons of reconstruction time with respect to different redundancy levels (r = 3, 4), with settings of ‘SRU = 128 KB, k = 6, f = 1’.

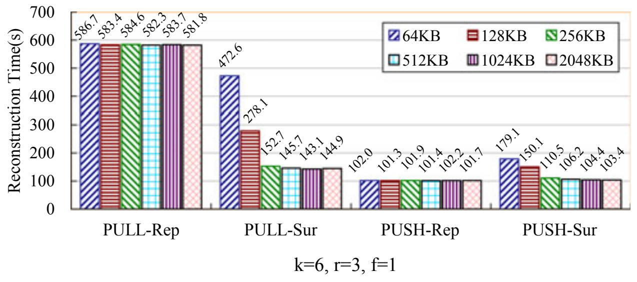


Fig. 11. A comparison of reconstruction times with respect to different size of request unit, which is set to 64, 128, and 256 KB. ‘k = 6, r = 3, f = 1’.

cluster where the request size is 128 KB and the number of data nodes is 6.

Fig. 9 reveals that the parameter r almost has no impact on the reconstruction time of PULL-Rep, because exact k of $k+r-1$ surviving nodes are involved in the reconstruction.

Not surprisingly, the reconstruction time of PULL-Sur and PUSH-Sur decreases when the redundancy r goes up. This is because a larger r value indicates more surviving nodes contributing to the reconstruction process, which enjoys high parallelism in reconstruction I/Os. For example, in the PULL-Sur case, the theoretical reconstruction-time ratio $\mathrm{R}_{<r1,r2>}$ between ‘$\mathrm{r}=\mathrm{r}_1$’ and ‘$\mathrm{r}=\mathrm{r}_2$’ approximates $(\mathrm{k}+\mathrm{r}_2-1)/(\mathrm{k}+\mathrm{r}_1-1)$ according to Eq. (10). $\mathrm{R}_{<r1,r2>}$ equals to 1.125 when $k=6, r_1=3$, and $r_2=4$, which is close to the ratio obtained from the experimental results (i.e., 1.105 for PULL-Sur, and 1.109 for PUSH-Sur). The above analysis confirms the correctness of Eqs. (10) and (12).

5.3.3 f – *Number of Failed Nodes*

In the f-node failure case, there are $k+r$ nodes involved in the reconstruction for both PULL-Rep and PUSH-Rep. These nodes include f replacement nodes and $k+r-f$ surviving nodes. As to both PULL-Sur and PUSH-Sur, $k+r-f$ nodes are involved in the reconstruction.

To examine the impact of number of node failures on the reconstruction performance, we carry out two-node reconstruction experiments on the (9, 6) RS-coded storage cluster when the request size is 128 KB. Note that the results of single-node reconstruction are reported in Section 5.3.1.

We draw three observations from Fig. 10. First, the reconstruction time of PULL-Rep is not very sensitive to the number f of failed nodes (see the first two bars on the left of Fig. 10). The reason is two-fold: i) the replacement node’s reception bandwidth dominates the reconstruction times, and ii) each replacement node receives the same amount of surviving data, i.e., $\mathrm{k}\times 10\,\mathrm{GB}$.

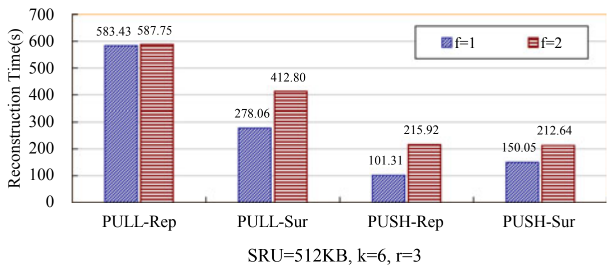


Fig. 10. Comparisons of reconstruction time with respect to different node failures (f = 1, 2), with settings of ‘SRU = 128 KB, k = 6, r = 3’.

Second, compared with the single-node reconstruction, the reconstruction times of PULL-Sur and PUSH-Sur increase by a factor of around 1.4 in the case of double-node reconstruction. Now we explain why the factor is 1.4× rather than 2.0×.

The reconstruction time $\mathrm{T}_{\text{PULL-Sur\_f}}$ for PULL-Sur under f-node failure can be derived from Eq. (10) as below:

$$\mathrm{T}_{\text{PULL-Sur\_f}} = \max_{i=0}^{k+r-f-1}\left\{\mathrm{t}_{\mathrm{write\_D}_{\mathrm{row+i,col}}}\right\} \times \frac{\mathrm{f}\times \mathrm{S}_{\mathrm{node}}}{(\mathrm{k}+\mathrm{r}-\mathrm{f})\mathrm{S}_{\mathrm{block}}}. \tag{14}$$

Parameter $\mathrm{t}_{\mathrm{write\_D}_{\mathrm{row+i,col}}}$ varies under different number of node failures, because the write ratio of each surviving node is $\mathrm{f}/(\mathrm{k}+\mathrm{f})$. Note that in addition to one read and f writes, each surviving node should simultaneously respond to $k-1$ reconstruction reads from other surviving nodes. A large write ratio implies high write throughput thus small write latency. Eq. (14) shows that the reconstruction time $\mathrm{T}_{\text{PULL-Sur\_f}}$ is not directly proportional to the number f of failures.

Third, the reconstruction time of PUSH-Rep in the two-node-reconstruction case doubles that of the one-node-reconstruction case. This reconstruction trend is reasonable, because i) the receiving phase of each surviving node dominates the reconstruction time, and ii) each surviving node should receive two intermediate blocks during two-node reconstruction process.

5.3.4 *SRU—Size of Request Unit*

To examine the impact of the request unit size or SRU, we conduct experiments on the (9, 6) erasure-coded storage cluster where SRU is set to 64, 128, 256, 512, 1,024, and 2,048 KB, respectively.

Fig. 11 shows that both PULL-Rep and PUSH-Rep are not sensitive to SRU, because it is the receiving phase rather than disk I/Os dominates the overhead of reconstruction process.

We observe that PULL-Sur and PUSH-Sur perform poorly when the size of request unit is small. The performance improvement of PUSH-Sur over PULL-Sur becomes more pronounced when SRU is small. For example, the reconstruction-time ratio $\mathrm{R}_{<\text{PULL-Sur,PUSH-Sur}>}$ between PULL-Sur and PUSH-Sur is 2.64, 1.85, and 1.38 when SRU is 64, 128, and 256 KB, respectively. Such a performance trend is attributed to the fact that the room for sequential bandwidth improvement becomes small when SRU is large, and the chance of improving performance through an increased SRU in PUSH-Sur is smaller than that in PULL-Sur. On the other hand, when the size of SRU is larger than 256 KB, the reconstruction times of PULL-Sur and PUSH-Sur are insensitive to SRU, because both PULL-Sur and PUSH-Sur are unable to gain further benefit from saturated disk I/O bandwidth when the SRU size reaches a specific value. Furthermore, regardless of the SRU size, PUSH-Sur is constantly superior to PULL-Sur because each surviving node in PULL-Sur passively responds to $k-1$ reads from other surviving nodes, thereby suffering from non-sequential reads.

5.4 A Summary of Observations

Important observations drawn from Figs. 8, 9, 10 and 11 are summarized as follows.

- Among the four performance factors (i.e., k, r, f, and *SRU*), only the number k of data nodes and the number f of failed nodes make significant impacts on the reconstruction performance of PULL-Rep and PUSH-Rep, respectively;
- both PULL-Sur and PUSH-Sur are substantially affected by the size of request unit, which agrees with the fact that disk writes dominate the overhead of reconstruction for the reconstruction among surviving nodes; and
- the two PUSH-based schemes outperform both PULL-based counterparts in terms of reconstruction time regardless of the parameters k, r, f, and *SRU*.

6 Further Discussions

Eqs. (4) $\sim$ (8) show the way of regenerating failed blocks when the number k of data nodes and the number r of parity nodes is four and two, respectively. In fact, each failed block can be reconstructed by computing a certain linear combination when parameters k, r are arbitrary.

This paper evaluates both PUSH-Rep and PUSH-Sur schemes under both single- and double-node reconstruction scenarios. We show evidence that PUSH can be employed to address the issue of any multiple node failure. For example, let us consider a triple-node reconstruction handled by PUSH-Sur. Each surviving node generates three intermediate blocks using its local block and the received block with three different coefficients, and forwards the three resulting blocks to a subsequent surviving node.

Our reconstruction performance models suggest that there is a demand to address the heterogeneity issue (see Section 4.3), where surviving nodes may have various I/O performance. In a heterogeneous storage cluster, a surviving node with low I/O throughput inevitably slows down the entire reconstruction process. To remedy such a deficiency, we intend to incorporate a load balancing strategy into PUSH, where the slow surviving node joins a limited number of reconstruction chains. A challenge that awaits us is to determine the appropriate number of reconstruction chains for each surviving node to join according to its available I/O bandwidth.

Our proposed PUSH-based schemes outperform the PULL-based ones under the *non-Incast* environments. It is not a surprise that in the TCP *Incast* case (e.g., RTO = 200 ms) both PUSH-Rep and PUSH-Sur achieve higher reconstruction speedup over the PULL-based solutions because of the '1:1' traffic pattern.

7 Conclusion and Future Work

Existing PULL-based reconstruction techniques (e.g., PULL-Rep and PULL-Sur) have two major drawbacks—the network capacity of replacement nodes and disk writes of rebuilding nodes dominate the reconstruction time in PULL-Rep and PULL-Sur, respectively. To address these issues, we proposed the PUSH approach, in which a PUSH-type transmission is incorporated into node reconstruction. We developed two PUSH-based reconstruction schemes (i.e., PUSH-Rep and PUSH-Sur). Compared to the PULL-based counterparts where surviving blocks are transferred in a synchronized 'M:1' traffic pattern, our PUSH-based reconstruction solutions support the '1:1' pattern, which naturally solves the *Incast* problem. We built performance models to investigate the reconstruction times of our PUSH-based schemes applied in large-scale storage clusters. We extensively evaluated the four schemes on a real-world cluster. Our empirical results show that the PUSH-based reconstruction schemes outperform the PULL-based ones. On the (9, 6)RS-coded storage cluster, PUSH-Rep speeds up the reconstruction time by a factor of 5.76 over PULL-Rep; PUSH-Sur accelerates the reconstruction of PULL-Sur by a factor of 1.85.

Nowadays a grand challenge for storage clusters is efficiently migrating data replicas to create an erasure-coded archive [34]. To take this challenge, we are going to integrate the PUSH-type transmission into the archival migration in erasure-coded storage clusters. Moreover, since PUSH-based reconstruction schemes are sensitive to slow nodes, we plan to extend the PUSH-based reconstruction schemes for heterogeneous erasure-coded storage clusters by taking into account both load and heterogeneity of surviving nodes.

Acknowledgments

The authors would like to thank Fenghao Zhang and Siti Huang for their constructive discussions. This work was supported in part by the National Basic Research Program of China under Grant No.2011CB302303 and the Development Program under Grant No. 2013AA013203. Xiao Qin's work is supported by the US National Science Foundation under Grants CCF-0845257(CAREER), CNS-0917137(CSR), and CCF-0742187(CPA). Qiang Cao is the corresponding author of this paper.

References

[1] S. Frolund, A. Merchant, Y. Saito, S. Spence, and A. Veitch, "A decentralized algorithm for erasure-coded virtual disks," in *Proc. Int. Conf. Dependable Systems Networks*, 2004, pp. 125–134.

[2] M. Storer, K. Greenan, E. Miller, and K. Voruganti, "Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage," in *Proc. 6th USENIX Conf. File Storage Technol.*, 2008, p. 1.

[3] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sarma, R. Murthy, and H. Liu, "Data warehousing and analytics infrastructure at facebook," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 1013–1020.

[4] Z. Zhang, A. Deshpande, X. Ma, and E. Thereska, "Does erasure coding have a role to play in my data center?" Microsoft research MSR-TR-2010, vol. 52, 2010.

[5] B. Calder et al., "Windows azure storage: A highly available cloud storage service with strong consistency," in *Proc. 23rd ACM Symp. Operating Syst. Principles*, 2011, pp. 143–157.

[6] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads," in *Proc. 10th USENIX Conf. File Storage Technol.*, 2012, pp. 251–264.

[7] J. Plank et al., "A tutorial on reed-solomon coding for fault-tolerance in raid-like systems," *Softw. Practice Experience*, vol. 27, no. 9, pp. 995–1012, 1997.

[8] M. Manasse, C. Thekkath, and A. Silverberg, "A reed-solomon code for disk storage, and efficient recovery computations for erasure-coded disk storage," *Proc. Inf.*, pp. 1–11, 2009.

[9] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Proc. USENIX Annu. Tech. Conf.*, 2012, p. 2.

[10] K. Rao, J. Hafner, and R. Golding, "Reliability for networked storage nodes," *IEEE Trans. Dependable Secure Comput.*, vol. 8, no. 3, pp. 404–418, May 2011.

[11] Q. Xin, E. Miller, T. Schwarz, D. Long, S. Brandt, and W. Litwin, "Reliability mechanisms for very large storage systems," in *Proc. 20th IEEE/11th NASA Goddard Conf. Mass Storage Syst. Technol.*, 2003, pp. 146–156.

[12] Q. Xin, E. Miller, and S. Schwarz, "Evaluation of distributed recovery in large-scale storage systems," in *Proc. 13th IEEE Int. Symp. High Performance Distrib. Comput.*, 2004, pp. 172–181.

[13] A. Phanishayee, E. Krevat, V. Vasudevan, D. Andersen, G. Ganger, G. Gibson, and S. Seshan, "Measurement and analysis of TCP throughput collapse in cluster-based storage systems," in *Proc. 6th USENIX Conf. File Storage Technol.*, 2008, p. 12.

[14] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki, "Hydrastor: A scalable secondary storage," in *Proc. 7th Conf. File Storage Technol.*, 2009, pp. 197–210.

[15] M. Holland, G. Gibson, and D. Siewiorek, "Fast, on-line failure recovery in redundant disk arrays," in *Proc. 23rd Int. Symp. Fault-Tolerant Comput.*, 1993, pp. 422–431.

[16] M. Holland, G. Gibson, and D. Siewiorek, "Architectures and algorithms for on-line failure recovery in redundant disk arrays," *Distrib. Parallel Databases*, vol. 2, pp. 295–335, 1994.

[17] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao, "Workout: I/O workload outsourcing for boosting raid reconstruction performance," in *Proc. 7th Conf. File Storage Technol.*, 2009, pp. 239–252.

[18] B. Cassidy, J. Hafner, Space efficient matrix methods for lost data reconstruction in erasure codes," IBM Res., Armonk, NY, USA, Tech. Rep. RJ10415, 2007.

[19] L. Xiang, Y. Xu, J. Lui, and Q. Chang, "Optimal recovery of single disk failure in rdp code storage systems," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 1, pp. 119–130, 2010.

[20] T. Xie and H. Wang, "Micro: A multilevel caching-based reconstruction optimization for mobile storage systems," *IEEE Trans. Comput.*, vol. 57, no. 10, pp. 1386–1398, Oct. 2008.

[21] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable performance of the panasas parallel file system," in *Proc. 6th USENIX Conf. File Storage Technol.*, vol. 2, 2008, pp. 1–2.

[22] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inform. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.

[23] Y. Hu, Y. Xu, X. Wang, C. Zhan, and P. Li, "Cooperative recovery of distributed storage systems from multiple losses with network coding," *IEEE J. Select. Areas Commun.*, vol. 28, no. 2, pp. 268–276, Feb. 2010.

[24] A. Kermarrec, N. Le Scouarnec, and G. Straub, "Repairing multiple failures with coordinated and adaptive regenerating codes," in *Proc. Int. Symp. Netw. Coding*, 2011, pp. 1–6.

[25] I. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, 1960.

[26] M. Aguilera, R. Janakiraman, and L. Xu, "Using erasure codes efficiently for storage in a distributed system," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2005, pp. 336–345.

[27] J. Plank, J. Luo, C. Schuman, L. Xu, and Z. Wilcox-O'Hearn, "A performance evaluation and examination of open-source erasure coding libraries for storage," in *Proc. 7th Conf. File Storage Technol.*, 2009, pp. 253–265.

[28] F. Wang, S.A. Brandt, E.L. Miller, and D.D. Long, "Obfs: A file system for object-based storage devices," in *Proc. 21st IEEE/12th NASA Goddard Conf. Mass Storage Syst. Technol.*, 2004, pp. 283–300.

[29] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. Andersen, G. Ganger, G. Gibson, and B. Mueller, "Safe and effective fine-grained TCP retransmissions for datacenter communication," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 303–314, 2009.

[30] H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, V. Vasudevan, and A. Phanishayee, "A (In)cast of thousands: scaling datacenter tcp to kiloservers and gigabits," Carnegie Mellon Univ. Parallel Data Lab, Pittsburgh, PA, USA, Tech. Rep. CMU-PDL-09-101, 2009.

[31] T. IOMETER, "Iometer: I/o subsystem measurement and characterization tool," Open source code distribution: http://www.iometer.org, 1997.

[32] D. Ford, F. Labelle, F. Popovici, M. Stokely, V. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Proc. 9th USENIX Symp. Operating Syst. Design Implementation*, 2010.

[33] J. S. Plank, S. Simmerman, and C. D. Schuman, "Jerasure: A library in c/c++ facilitating erasure coding for storage applications-version 1.2," Univ. Tennessee, Knoxville, TN, USA, Tech. Rep. CS-08-627, 2008.

[34] L. Pamies-Juarez, F. Oggier, and A. Datta, "Data insertion and archiving in erasure-coding based large-scale storage systems," in *Proc. Distrib. Comput. Internet Technol.*, 2013, pp. 47–68.

[35] TAHO-LAFS. (2010). *Tahoe: The least-authority filesystem* [Online]. Available: Open source code distribution: http://tahoe-lafs.org/trac/tahoe-lafs

[36] C. Ungureanu, B. Atkin, A. Aranya, S. Gokhale, S. Rago, G. Calkowski, C. Dubnicki, and A. Bohra, "Hydrafs: A high-throughput file system for the hydrastor content-addressable storage system," in *Proc. 8th USENIX Conf. File Storage Technol.*, 2010, pp. 225–238.

[37] I. CLEVERSAFE. (2008). *Cleversafe dispersed storage*, [Online]. Available: Open source code distribution: http://www. cleversafe. org/downloads.

[38] L. Rizzo, "On the feasibility of software fec," Univ. di Pisa, Italy, Tech. Rep. LR-970131, 1997.

[39] F. F. J. MacWilliams and N.N.J.A. Sloane, *The Theory of Error-Correcting Codes: Part 2*, vol. 16., Amsterdam, The Netherlands: Elsevier, 1977.

[40] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," *ACM SIGCOMM Computer Commun. Rev.*, vol. 39, no. 4, pp. 51–62, 2009.

[41] F. Zhang, J. Huang, and C. Xie, "Two efficient partial-updating schemes for erasure-coded storage clusters," in *Proc. IEEE 7th Int. Conf. Netw., Archit. Storage*, 2012, pp. 21–30.

Jianzhong Huang received the PhD degree in computer architecture in 2005 and completed the post doctoral research in information engineering in 2007 from the Huazhong University of Science and Technology (HUST), Wuhan, China. He is currently an associate professor in the Wuhan National Laboratory for Optoelectronics at HUST. His research interests include computer architecture and dependable storage systems. He received the National Science Foundation of China in Storage System Research Award in 2007.

Xianhai Liang received the BS in computer science and technology in 2012 from the Wuhan University of Technology (WHUT), China. He is currently working toward the MS degree at HUST. His research interests include networked storage systems and file system.

Xiao Qin (S'00-M'04-SM'09) received the BS and MS degrees in computer science from HUST, China, and the PhD degree in computer science from the University of Nebraska-Lincoln, in 1992, 1999, and 2004, respectively. He is currently an associate professor with the Department of Computer Science and Software Engineering, Auburn University. His research interests include parallel and distributed systems, storage systems, fault tolerance, real-time systems, and performance evaluation. He received the US National Science Foundation Computing Processes and Artifacts Award and the NSF Computer System Research Award in 2007 and the NSF CAREER Award in 2009. He is a senior member of the IEEE.

Qiang Cao received the BS degree in applied physics, the MS degree in computer technology, and the PhD degree in computer architecture in 1997, 2000 and 2003, respectively. He is currently an associate professor at the Huazhong University of Science and Technology. His research interests include computer architecture, large scale storage systems, and performance evaluation. He is a senior member of China Computer Federation (CCF) and a member of the IEEE.

Changsheng Xie received the BS and MS degrees in computer science both from HUST, Wuhan, China, in 1982 and 1988, respectively. He is currently a professor in the Department of Computer Engineering at HUST. He is also the director of the Data Storage Systems Laboratory of HUST and the deputy director of the Wuhan National Laboratory for Optoelectronics. His research interests include computer architecture, I/O system, and networked storage system. He is the vice chair of the expert committee of Storage Networking Industry Association (SNIA), China. He is a member of the IEEE.