Future Generation Computer Systems 69 (2017) 24–40

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Efficient in-place update with grouped and pipelined data transmission in erasure-coded storage systems

Xiaoqiang Pei, Yijie Wang*, Xingkong Ma, Fangliang Xu

National Key Laboratory for Parallel and Distributed Processing, College of Computer, National University of Defense Technology, Changsha, Hunan, 410073, PR China

HIGHLIGHTS

- We propose a three-layer framework to support both single and multiple updates.
- We propose a workload-aware group technique to dynamically adjust the group size.
- We propose a distributed pipeline technique to distribute the computation.
- We propose a hybrid update technique to be compatible with the node failure.
- We conduct extensive experiments to confirm the advantages of our approach.

ARTICLE INFO

Article history:
Received 26 May 2015
Received in revised form
27 September 2016
Accepted 9 October 2016
Available online 6 December 2016

Keywords:
In-place
Update
Grouping
In-time
Lazy-update

ABSTRACT

Distributed storage systems usually adopt erasure coding to achieve better tradeoff between the space efficiency and the data reliability. In-place updates are often used to overwrite the existing data rather than append the new data so as to ensure the data access efficiency. However, existing in-place update approaches either introduce significant I/O overhead or cause low update efficiency in erasure-coded storage systems due to the consistent update of parity blocks. In this paper, we propose a grouped and pipelined update scheme based on erasure codes, called Group-U, which comprises four key design features. (1) It groups the data nodes to complete the data transmission and dynamically adjusts the group size according to the update workload. (2) It pipelines the data transmission and distributes the update computation to all the participating nodes to improve the update efficiency. (3) It adopts the in-time update for data nodes and lazy-update for parity nodes to further reduce the update overhead. (4) It adjusts the occasion triggering the update to be compatible with the node failure. We design and implement Group-U on our Raid Distributed Storage System (RDFS) and conduct testbed experiments on different update schemes under various parameter settings. The analysis and experimental results show that Group-U consumes 22% increase of update overhead compared with PUM and achieves 46% reduction of update overhead compared with PDP-P and PUS. Furthermore, Group-U achieves 69%, 34% and 21% reduction of update time on average compared with PUM, PDP-P and PUS respectively.

1. Introduction

Distributed storage systems usually adopt erasure codes as the redundancy scheme to ensure the data reliability, either in the context of data centers [1,2] or in the context of peer-assisted online storage systems [3–7]. The original data object is divided into multiple blocks and these blocks are encoded into parity blocks, such that a subset of blocks is sufficient to construct the original data or reconstruct the failed block. It is known that erasure codes save much storage space with the same fault tolerance or significantly improve the data reliability with the same storage space compared with replication [4,8,9]. Erasure codes have been deployed by many storage systems [10–14] to balance the storage cost and data reliability. Existing researches of erasure codes mainly focus on the optimization of encoding/decoding algorithms [15–19], pipelined encoding process [20–25], data repair [26–31], or the degraded read [32,33], leaving a few works for optimizing the performance of data update for erasure codes [34–37].

In fact, data updates are prevalent for many real-world workloads in enterprise servers and network file systems [38,39].

* Corresponding author.
E-mail addresses: xiaoqiangpei@nudt.edu.cn (X. Pei), wangyijie@nudt.edu.cn (Y. Wang), maxingkong@nudt.edu.cn (X. Ma), xufl89@nudt.edu.cn (F. Xu).

http://dx.doi.org/10.1016/j.future.2016.10.016
0167-739X/© 2016 Elsevier B.V. All rights reserved.

The trace analysis in [35] tells us three observations. Firstly, the updates are common in both the analyzed storage traces. Then, the updates are small, where all the updates are generally small in size. Finally, the update coverage varies, where the updates may cover large percentage of the existing blocks. Furthermore, the Read–Write scenarios are gradually adopted in erasure-coded storage systems, where there exist the update issues [37]. Compared with the updates of replication which only need to transmit the modified information and overwrite the existing data, the updates of erasure coding involve more operations and steps to complete the update process. Specifically, the updates for data nodes and parity nodes are different with each other, where the former is completed by overwriting the existing data, and the later involves the data transmission and data computation. Thus, the updates for erasure coding consume more network traffic and cause longer time to complete the update, which poses the new challenge: how to maximize the update efficiency with the least extra network traffic cost of the erasure codes.

There are two ways of performing updates [35]: *in-place updates* and *log-based updates*. In-place update schemes modify the data and parity range according to the updated information. It first reads the range of the data block to be modified and computes the delta, which is the change between old and new data at the modified offset of the data block. It then forwards the modified data range and the recomputed parity delta to the data node and all the parity nodes respectively. In-place updates ensure the data consistency and show good support for the read request. However, in-place updates need to update both the data blocks and the parity blocks simultaneously, which may prolong the update process and degrade the update efficiency. Log-based updates complete the updates by appending the new data to the original data [40]. The appended data is combined with the original data after a given threshold, such as the append in GFS [1], Azure [13], Self-tuning LFS [41], and Gecko [42]. Log-based updates do not need to modify the original data when updating, which show higher update efficiency compared with in-place updates. However, log-based updates show degraded support for data read, since it needs to combine the original data and the appended data before reading. This in particular hurts the repair performance, since repair needs to access large volume of data in the surviving nodes to repair the lost data [35]. Besides the pure in-place updates or the pure log-based updates, researchers propose to combine them to get a more balanced performance, where the data nodes are updated with in-place pattern and parity nodes are updated with log-based pattern. For example, The parity logging with reserved space (PLR) [35] keeps parity updates in a reserved space next to the parity chunk to mitigate disk seeks and improve update efficiency. However, it takes more resource and makes it more complex to maintain two kinds of update patterns and the log-based updates for parity nodes degrade the repair performance as that in pure log-based updates.

In fact, the network traffic cost and the update efficiency are two important metrics of the system, where too much network traffic cost may lead to the degraded performance for the applications, and too long update time may lead to the higher possibility of the successive failures. However, it is contrary to optimize both of them, where the optimization of update efficiency may lead to the extra network traffic cost and vice versa. In this paper, we aim to minimize the update time with the least extra network traffic cost.

However, existing in-place data update schemes are inadequate to satisfy the challenges of high efficiency with the least network traffic cost. This mainly stems from the following two reasons. Firstly, existing in-place data update schemes [35] or the pipelined approaches proposed in [20,26] consume much network traffic and cause much disk I/O overhead as they treat the data updates for multiple data nodes as independent with each other. In fact, the multiple updates within a stripe may share the data transmission or data computation to reduce the update cost and improve the update efficiency. Secondly, existing update schemes mentioned above adopt the star structure [34,37] to complete the update process, where all the data nodes to be updated connect to the parity nodes and come into a star structure with one data node or parity node as the core. This may cause computation or network bottleneck when updating. Update efficiency may be further exacerbated when there are multiple updates. Furthermore, most of the data update schemes are propitious to either single update [35] or multiple updates [37], rather than giving consideration to both sides. In contrast, as the updates are common and the coverage of updates varies, it is common to encounter the situations of single data node update or multiple data node updates.

To this end, this paper presents a grouped and pipelined update scheme based on erasure codes, called Group-U, for improving the update efficiency with the least network traffic cost. Specifically, we mainly focus on the two questions: (1) how to construct a general update model for both single and multiple updates? (2) how to improve the update efficiency with the least extra overhead, including the network traffic cost and the disk I/O? We provide the following contributions in this paper:

First, we propose a general three-layer update framework to support both single and multiple updates. To reduce the update overhead, we propose a workload-aware group technique to exploit the data locality within each group and dynamically adjust the group size according to the update workload. To improve the update efficiency, we propose a distributed pipeline technique to pipeline the data transmission between nodes and distribute the update computation to all the participating nodes. To further reduce the update overhead, we propose a hybrid update technique to organize the in-time update for data nodes and lazy-update for parity nodes, which ensures the data reliability by combining the trigger metrics and handles the node failures by caching the data within each node.

Then, we implement Group-U in our Raid Distributed Storage System (RDFS), which supports different erasure coding and updates schemes, and is available from public-domain at GitHub [43].

Finally, we conduct extensive experiments on both physical and virtual machines under different parameter settings to confirm the high update efficiency of our approach.

The rest of the paper proceeds as follows. Section 2 introduces the background of erasure coding update. Section 3 discusses the related work of data update with erasure codes. Section 4 describes the architecture of framework and how Group-U achieves the efficient updates. Section 5 details the hybrid update technique for both data and parity blocks. Section 6 describes the implementation of Group-U and presents the testbed experimental results. Section 7 concludes the paper.

2. Background of erasure coding update

In an erasure code, the data object (denoted as its size M) is divided into multiple data blocks, which are separated into multiple stripes, with k data blocks within each stripe. In each stripe, the k data blocks are encoded into r parity blocks, and the k data blocks and the r parity blocks are distributed among $n = k + r$ storage nodes to maximize the data reliability for a (n, k)-code. We consider the Maximum Distance Separable (MDS) [44] erasure coding, where the original data or any failed block can be reconstructed by accessing any k out of n blocks. In this way, the (n, k)-code could tolerate $r = n - k$ block failure at most. For a linear (n, k)-code, each parity block $P_i, 0 \leq i < n - k$ could be represented by a linear combination of the k data blocks, as illustrated in Eq. (1), where P_i denotes the ith parity block, D_j