Future Generation Computer Systems 59 (2016) 47–62

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

DiVers: An erasure code based storage architecture for versioning exploiting sparsity

J. Harshan [a,*], Anwitaman Datta [b], Frédérique Oggier [a]

[a] Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
[b] School of Computer Engineering, Nanyang Technological University, Singapore

HIGHLIGHTS

- A networked storage architecture called DiVers is proposed to store versioned data.
- Sparsity exploiting erasure coding is used to reduce storage overhead in DiVers.
- Reliability aspect for DiVers is addressed to achieve best fault tolerance.
- System level issues such as metadata management and network protocol are discussed.

ARTICLE INFO

Article history:
Received 17 July 2015
Received in revised form
23 December 2015
Accepted 13 January 2016
Available online 29 January 2016

Keywords:
Datacenter networking
Version management
Fault tolerance
Erasure coding

ABSTRACT

We propose a differential versioning based data storage (DiVers) architecture for distributed storage systems, which relies on a novel erasure coding technique that exploits sparsity across versions. The emphasis of this work is to demonstrate how sparsity exploiting codes (SEC), originally designed for I/O optimization, can be extended to significantly reduce storage overhead in a repository of versioned data. In addition to facilitating reduced storage, we address some key reliability aspects for DiVers such as (i) mechanisms to deploy the coding technique with arbitrarily varying size of data across versions, and (ii) investigating the right allocation strategy for the encoded blocks over a network of distributed nodes across different versions so as to achieve the best fault tolerance. We also discuss system issues related to the management of data structures for accessing and manipulating the files over the differential versions.

1. Introduction

After years of hesitation, compelled both by cost implications borne from the scale of data volumes, and by advances in hardware and implementation optimization, erasure codes have in the recent years been enthusiastically embraced by major industry players dealing with big volume of data, e.g. Microsoft's Azure [1] and Facebook's F4 [2]. This has piqued a renewed research interest in designing erasure codes that are particularly suited to the needs of networked storage systems. The focus has primarily been on efficient repairs and recovery from failures [3,4]. Most existing erasure coding techniques have no inherent support or optimization for storing multiple versions as and when data mutates.

The need to store multiple versions of data arises in many scenarios. For instance, when editing and updating files, users may want to explicitly create a version repository using a framework like SVN [5] or Git [6]. Cloud based document editing or storage services also often provide the users access to older versions of the documents. In system level back-up, where whole file systems or databases are archived, versions would refer to the different snapshots over time. In either of the two file-centric settings, irrespective of whether a working copy used during editing is stored locally or on the cloud, or in a system level back up, say using copy-on-write [7], the back-end storage system needs to preserve the different versions reliably, and can leverage on erasure coding for reducing the storage overheads.

A naive approach will be to apply coding to each version independently. In a recent work [8], we proposed *Sparsity Exploiting Codes (SEC)* to optimize the I/O performance of storage systems storing and manipulating multiple versions of data, by leveraging techniques from sparse sampling [9]. Motivated by the practical issues arising from SEC's differential coding approach, we address the design of an erasure coding based storage architecture supporting versioning. We demonstrate how Sparsity Exploiting Codes (SEC) [8], originally designed for I/O optimization, can be

* Corresponding author.
E-mail address: jharshan@ntu.edu.sg (J. Harshan).

http://dx.doi.org/10.1016/j.future.2016.01.005
0167-739X/

extended to significantly reduce storage overhead in a repository of versioned data. We show that sparsity in the difference between successive versions of an object can be exploited to efficiently store the difference objects (typically referred to as *deltas*) rather than the whole objects. It is evident that this approach results in deltas whose sizes are variables (depending on the update pattern) rather than fixed numbers. As a result, contrary to the conventional usage of erasure coding which assumes fixed sized data objects, the design and deployment of our proposed codes need to handle variable sized deltas across different versions. The central focus of this work is thus to explore new techniques that take into account the peculiarities of deltas across versions, while addressing the age old two-fold problem of storing data *efficiently* and *reliably*.

1.1. Contributions

(i) We first tackle the aspect of efficiently storing versioned data. In Section 2, we demonstrate how SEC can be extended to reduce storage overhead in a repository of versioned data. We then propose the overall architectural design of DiVers (see Section 3), where we address a practical way to encode and store mutable content. Indeed, the design of erasure codes usually assumes fixed sized data objects, to be divided into fixed sized blocks of data. Furthermore, even if a single bit in one of the blocks changes, the coding semantic treats it as a distinct block. Consequently, even a minor change near the start of a file (during the update process) may ripple changes across all the blocks at the coding granularity. This would in particular render the sparse sampling techniques of SEC useless, and obliterate the consequent benefits. We incorporate zero padding schemes into DiVers to ameliorate the aforementioned problems, taking into account insertions, deletions and in-place alterations in the update process.

(ii) We then explore the reliability aspect of DiVers' design. As a consequence of employing sparsity based compression technique, we note that the deltas associated with different versions are potentially of different sizes. One straightforward option to encode such variable sized deltas is to fix the symbol size of erasure coding and then choose different dimensional erasure codes based on the size of deltas (depending on the sparsity level). However, to cater to different sizes, the architecture would need to manage multiple erasure codes, which is undesirable in practice. In Section 4, we propose an erasure coding framework wherein a single erasure code can be employed to cater to variable sized deltas, thus making it more practical than using multiple erasure codes. Further, we address the problem of allocating encoded blocks from different versions across a pool of servers to realize best fault tolerance. In Section 5, we specifically investigate placement strategies exploring whether to store the encoded blocks across different versions in a scattered manner across different clusters of servers, or in a co-located manner within a single group.

(iii) Subsequently, two critical aspects of the DiVers design, which form further important contributions of this paper, are discussed in Sections 6 and 7. Access of relevant data has to be supported by appropriate meta-information since the differences among consecutive versions, rather than full copy of a new version of data are encoded and stored in a dispersed manner, and additionally, because of the introduction of zero-padding. In Section 6, we elaborate the data structure to encapsulate the necessary meta-information. Then, in Section 7 we elaborate a protocol that explores the interactions among storage nodes for an application client to be able to manipulate versioned data in DiVers.

The modules and algorithms presented in the paper have been implemented individually, but an actual working versioned file system integrating everything has not been implemented. Storage gains offered by the DiVers architecture are evaluated through simulations of the algorithm in Section 8, where we explore the effect of quantum and placement of zero pads, and demonstrate the storage efficiency against a wide range of realistic workloads.

1.2. Related work

Although erasure coded systems have received great interest in the recent past [10–14] to store large volumes of data, it is worth highlighting that system implementations of erasure codes were addressed in the past in the context of Peer-to-Peer systems [15–17]. In terms of versioned data, [18] had proposed a family of consistency protocols that exploits data versioning within storage nodes. In this work, the primary objective was to ensure strong consistency of erasure coded data across a network of storage devices. In general, reliability and storage aspects of versioned data fall under the broad area of data deduplication, which addresses algorithms to clean up redundant and unnecessary data chunks from a network of storage nodes (see [19] and the references within). Recently, [20] has identified the fact that most storage systems such as Microsoft Azure or EMC Atmos support appending operations in the file storage systems, thereby, leading to increased amount of redundant storage. To fix that, the authors of [20] integrate the deduplication capability with erasure coding techniques. Wang and Cadambe [21] have addressed multi-version coding for distributed data, where the underlying problem is to encode different versions so that certain subsets of storage nodes can be accessed to retrieve the most common version among them. Their strategy has been shown applicable when the updates for the latest version do not reach all the nodes, possibly due to network problems. More recently, in [22], the authors consider the problem of synchronizing data in storage networks under an edit model that includes deletions and insertions, where the authors propose several variants of erasure codes that allow updates on the parity check values with low-bit rates and small storage overhead. Unlike the above listed related works, we propose an erasure-code based system architecture to store versioned data by exploiting the underlying sparsity in the difference between subsequent versions of data objects.

We next present a formal model for our sparsity exploiting erasure coding technique, summarizing and then extending [8].

2. Background: sparsity exploiting erasure coding

We first summarize how ideas from compressed sensing can be used to encode differences between versions of some data differentially, optimizing I/O operations for simultaneously reading multiple versions, a technique that we refer to as sparsity exploiting coding (SEC) [8]. We then build upon the existing work on SEC, and show how to furthermore achieve storage savings, while storing versioned data. The rest of the paper would then explore the DiVers architecture for storing versioned data using this enhanced storage saving differential coding technique. To demonstrate its practicality and compatibility, we will anchor our exposition of DiVers as tweaks/adaptation to existing architecture like GFS [23].

2.1. Preliminaries

Consider a fixed sized data object, denoted by $\mathbf{x} \in \mathbb{F}_q^k$ to be stored over a network, so that the object is seen as a vector of k blocks taking value in the alphabet $\mathbb{F}_q$, with $\mathbb{F}_q$ the finite field with q elements, q a power of 2 typically. Encoding for archival of an