DMV: Distributed Media Versioning across devices

Michael J. Murphy Otto J. Anshus John Markus Bjørndalen

November 2017

Abstract

A typical computer user has multiple devices holding an increasing amount of data. Most users will have at least a computer and a mobile phone. Many will also have a work computer, tablet, or other devices. These devices have varying resources, including processing, memory, and storage. They may also be in different locations, on different networks, or turned off at any time. The user's data will be in files of varying sizes and media types, from kilobyte text documents to multi-gigabyte videos and beyond. The volume of data is also always increasing as data is authored, collected from the internet, or gathered from mobile sensors. This data is strewn across these devices in an ad-hoc fashion, according to where it is produced and consumed. When the user needs a particular file, they must either remember where it is or perform a frustrating, manual, multi-device search. Also, copies of data on different devices will diverge if updates are made separately and not reconciled.

Cloud computing eases these problems by centralizing storage, searching, and update reconciliation. However, the user's access to their data depends on the reliability of their network connection and the reliability and longevity of the cloud service. Handing data over to a third party also raises concerns about privacy. The cloud service may also charge a recurring subscription fee. The user might prefer to use the devices they already own, provided there is an easier way to manage the data.

This paper explores distributed version control systems as an alternative approach to managing data across a spectrum of devices. A DVCS keeps writable copies of a data set at multiple locations, tracks update history, and allows diverging versions to be merged at a later date. However, version control systems are designed for the small text files of source code and are not suited to larger binary files.

We describe the architecture, design, and implementation of a new system we call Distributed Media Versioning (DMV) that resembles version control but is more flexible. DMV will allow the user to shard and replicate data across many devices with fine-grained control. It will keep a unified view of the data set as subsets of the data are copied or moved between devices by user request. It will allow data to be updated on any device, and it will track history so that diverging versions can be merged later.

We perform experiments to explore the scalability limits of selected version control systems. We find that the maximum file size is limited by

This paper was presented at the NIK-2017 conference; see http://www.nik.no/.

available RAM, and that commit times increase sharply as the number of files increases into the millions. We also perform the same experiments against a DMV prototype for comparison. DMV avoids the file-size limitations by using a rolling hash algorithm to break larger files into smaller chunks. Unfortunately, our early DMV prototype suffers the same problems with numerous files because it uses the underlying filesystem in a similar way. We conclude that the key to processing large files is to break them into many smaller chunks, and the key to storing many small files is to aggregate them into larger packs. We propose corrective changes for future work on DMV.

OUTLINE

```
- Problem: many devices, more data, difficult to follow what is where

- Cloud not solution. Relies on third party. Connection, privacy, etc.

- DVCS: An alternate approach

    - Extreme availability
    - Version history gives chain of causality for later reconciliation

- Interesting properties of DAG

    - DAG gives de-duplication
    - Content addressing gives tampering/bitrot protection
    - DAG also gives convenient ways to shard data

- Problem 1: dealing with larger files: chunking

    - Bup has chunking but locked into backup workflow

- Problem 2: dealing with many files: packing

    - Git has packing but in separate step that fails for large files

- Problem 3: increasing data: sharding

- In-between: de-duplication

- DMV prototype

- Experiments

    - File size and number of files
    - Random writes

- Results
```

```
- Conclusion

    - chunk, content-address, re-pack
    - DMV not yet viable, but it's a start
```

1 Introduction

Write: A typical computer user has multiple devices holding an increasing amount of data.

Write: Most users will have at least a computer and a mobile phone.

Write: Many will also have a work computer, tablet, or other devices.

Write: These devices have varying resources, including processing, memory, and storage.

Write: They may also be in different locations, on different networks, or turned off at any time.

Write: The user's data will be in files of varying sizes and media types, from kilobyte text documents to multi-gigabyte videos and beyond.

Write: The volume of data is also always increasing as data is authored, collected from the internet, or gathered from mobile sensors.

Write: This data is strewn across these devices in an ad-hoc fashion, according to where it is produced and consumed.

Write: When the user needs a particular file, they must either remember where it is or perform a frustrating, manual, multi-device search.

Write: Also, copies of data on different devices will diverge if updates are made separately and not reconciled.

Shortcomings of Cloud-Based Solutions

Write: Cloud computing eases these problems by centralizing storage, searching, and update reconciliation.

Write: However, the user's access to their data depends on the reliability of their network connection and the reliability and longevity of the cloud service.

Write: Handing data over to a third party also raises concerns about privacy.

Write: The cloud service may also charge a recurring subscription fee.

Write: The user might prefer to use the devices they already own, provided there is an easier way to manage the data.

Potential of Version Control

Write: This paper explores distributed version control systems as an alternative approach to managing data across a spectrum of devices.

Write: A DVCS keeps writable copies of a data set at multiple locations, tracks update history, and allows diverging versions to be merged at a later date.

Write: However, version control systems are designed for the small text files of source code and are not suited to larger binary files.

2 DMV Architecture and Design

Write: We describe the architecture, design, and implementation of a new system we call Distributed Media Versioning (DMV) that resembles version control but is more flexible.

Write: DMV will allow the user to shard and replicate data across many devices with fine-grained control.

Write: It will keep a unified view of the data set as subsets of the data are copied or moved between devices by user request.

Write: It will allow data to be updated on any device, and it will track history so that

Write: diverging versions can be merged later.

[Figure 1 about here.]

[Figure 2 about here.]

[Figure 3 about here.]

3 Evaluation

Write: We perform experiments to explore the scalability limits of selected version control systems.

Write: We find that the maximum file size is limited by available RAM, and that commit times increase sharply as the number of files increases into the

Write: millions.

Write: We also perform the same experiments against a DMV prototype for comparison.

Write: DMV avoids the file-size limitations by using a rolling hash algorithm to break larger files into smaller chunks.

Write: Unfortunately, our early DMV prototype suffers the same problems with numerous files because it uses the underlying filesystem in a similar way.

4 Conclusion

Write: We conclude that the key to processing large files is to break them into many smaller chunks, and the key to storing many small files is to aggregate them into larger packs.

Write: We propose corrective changes for future work on DMV.

Todo list

List of Figures

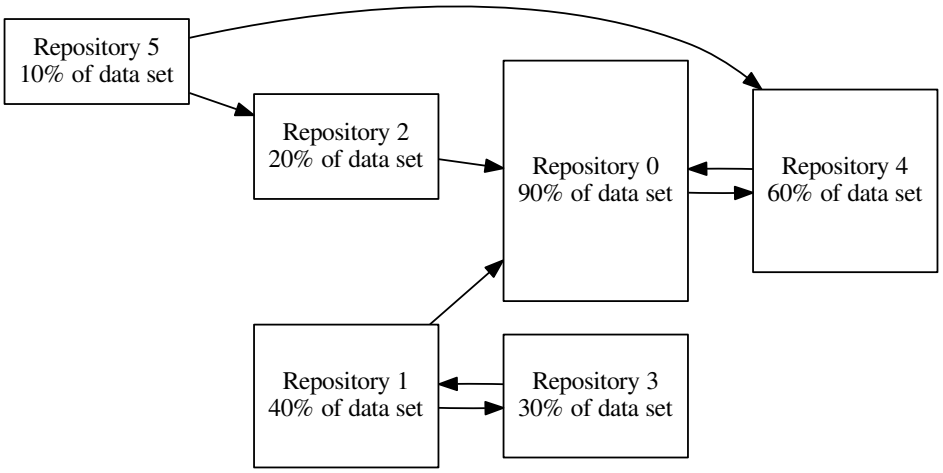


Figure 1: Repositories in an ad-hoc network

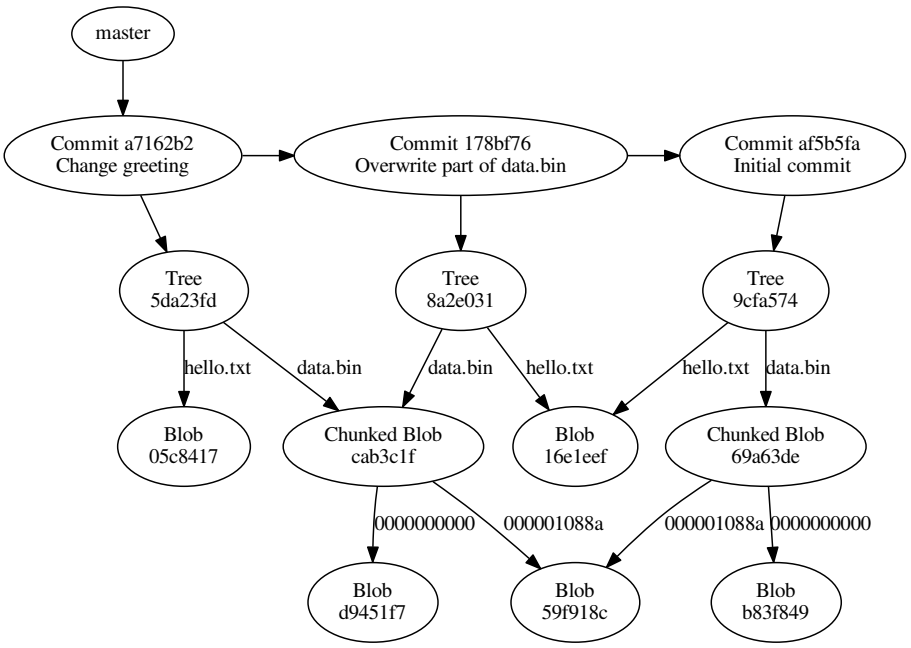
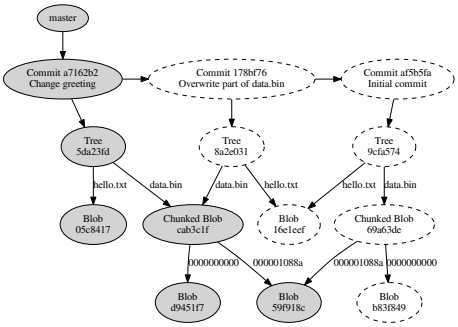
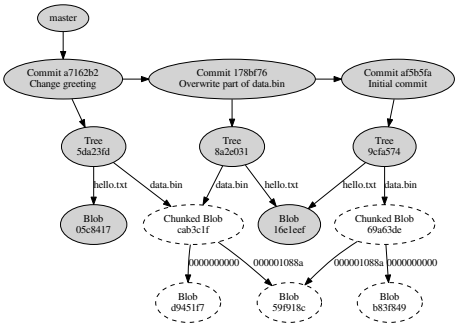


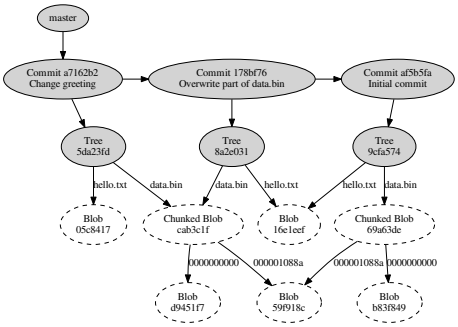
Figure 2: A simple DMV DAG with three commits



(a) Partial history of full data set



(b) Full history of part of data set



(c) Full history of metadata

Figure 3: A DMV DAG, sliced in different dimensions