Exploring the Data Structures and Algorithms behind Version Control Systems

Reece Donovan

Final Year Project - Extended Abstract
B.Sc. Computer Science

Supervisor: Prof. Ken Brown

Department of Computer Science
University College Cork

March 2023

Contents

1 Introduction

Version Control is a critical aspect of software development that helps developers keep track of changes made to a codebase. It enables software teams to work collaboratively, efficiently, and accurately on projects, reducing the likelihood of errors and conflicts. In essence, version control is the process of tracking and managing changes to files over time.

Version Control System (VCS) is a software tool that automates the version control process. It provides a centralised repository where developers can store their code, track changes, and collaborate with other team members. The Version Control System ensures that each team member has access to the latest version of the code and can work on it simultaneously.

The impact of **Version Control Systems** on software development has been immense. Before the advent of Version Control Systems, developers used to rely on manual processes to track changes, which was time-consuming and error-prone. With VCS, software teams can work together more efficiently, manage changes more effectively, and deliver better-quality software products.

Version Control Systems have also facilitated the rapid growth of **Continuous Integration** and **Continuous Delivery** (CI/CD), which have become essential systems in modern large scale software development. Overall, Version Control Systems have revolutionised how software is developed, making it easier, faster, and more reliable.

2 Motivation

Over the years, Version Control Systems have become essential for software developers, enabling them to collaborate and work more effectively. As the software development industry has evolved, so have Version Control Systems, leading to the creation of many different software solutions, each with unique features and strengths. However, these systems' core data structures and algorithms have remained relatively unchanged.

The performance of Version Control Systems can be crucial in determining the efficiency of software development projects. Slow or inefficient Version Control Systems can result in delays, errors, and even project failure. Therefore, it is important to explore alternative data structures and algorithms that could improve the performance of these systems.

3 Objective

Version Control Systems (VCS) have become essential for software developers to collaborate effectively and efficiently. However, many users may not fully comprehend the intricacies of the underlying concepts at the core of these systems.

This report aimed to address this knowledge gap by exploring the underlying data structures and algorithms used to power Version Control Systems and by evaluating potential trade-offs and benefits of alternative approaches. In order to provide insight into how these factors impact system performance, scalability, and overall effectiveness.

In addition, this report aimed to provide a comprehensive overview of the evolution of Version Control Systems, including an overview of the most popular version control solutions throughout the years, such as Source Code Control System (1972), Apache Subversion (2000), and Git (2005).

4 Approach

The following steps were taken to achieve the objectives of this report:

4.1 Background Research

This step involved a comprehensive review of Version Control Systems, including their history, different types, and existing systems. This review helped discover how the core principles of Version Control Systems have evolved over the years and establish which features are critical when evaluating different data structures and algorithms.

The aim was to understand better the historical and current state of Version Control Systems and identify the most relevant features and principles that influence the selection of data structures and algorithms.

4.2 Selection of Data Structures and Algorithms

Based on the background research conducted on Version Control Systems, a range of data structures were selected for evaluation. The chosen data structures were selected based on their relevance and suitability towards the core concepts of a Version Control System, taking into account various factors such as ease of implementation.

In addition to the data structures, a set of core functionalities were identified, including Searching, Diffing, and others. Then, two algorithms per feature were selected along with the data structures for further evaluation. The algorithms were chosen based on their suitability for the different data structures and their ability to provide efficient and effective solutions to the core functionalities required by a Version Control System.

4.3 Implementation of Data Structures

The selected data structures – linked lists, trees, hash tables, and directed acyclic graphs (DAGs) – were implemented in the Go programming language, the language chosen for this project. One of the design principles used was to implement the data structures to fit a common interface, allowing for one set of tests and benchmarks to be used for each data structure.

In the Go programming language, an interface is a collection of method signatures defining a set of behaviours that a type must implement to satisfy the interface. They make it possible to write generic functions and data structures that can work with various types without knowing anything about those types in advance.

4.4 Implementation of Algorithms

The algorithms selected for each core functionality, such as the Longest Common Subsequence (LCS) and the Myers' Diff algorithms selected for the core Diffing functionality, were implemented in the Go programming language. The implementations were checked against implementations found in well-regarded data structure and algorithms textbooks to ensure that the algorithms used in the project were accurate and representative of the algorithm itself, not a potentially flawed implementation.

The implementation of the algorithms followed the same design principles used for the data structures, which was to fit a common interface to enable uniform testing and benchmarking of each algorithm. In addition, this approach allowed for a fair comparison between the different algorithms and data structures evaluated in the report.

4.5 Design of Test Cases and Benchmarks

A set of test cases and benchmarks were designed to evaluate the performance of the different data structures and algorithms. The tests and benchmarks were designed to measure relevant metrics such as performance, memory usage, and scalability.

For the data structure benchmarks, tests were designed for evaluating data structure operations such as insert, remove, search, and more. To test the data structures' performance and memory usage under different conditions, mock data was generated to create data structures of varying sizes, from small to very large.

For the algorithm benchmarks, tests were designed similarly to the data structure benchmarks; mock data was generated to create data structures of varying sizes to test the algorithms' performance and memory usage under different conditions.

Benchmarking was conducted using the Go programming languages built-in testing and benchmarking features. The tests and benchmarks were designed to measure relevant metrics, and care was taken to ensure they were realistic and representative of real-world scenarios. In addition, the student had to learn how to use Go's built-in testing and benchmarking features, which were initially unfamiliar to them, to ensure the tests and benchmarks were executed correctly.

4.6 Execution of Benchmarks and Collection of Data

The student utilised the command line features of Go's benchmarking package to allow the execution of multiple benchmarks with minimal effort. Command flags were used when executing benchmarks through the command line to enable profiling of additional useful metrics such as CPU usage and memory allocation. These metrics provided further insight into the performance and efficiency of the different data structures and algorithms evaluated in the report.

The benchmark results were collected and recorded for each data structure and algorithm, then analysed to identify any patterns or trends in the performance of the different data structures and algorithms under different conditions. The collected data was then used to generate graphs and charts to visualise the performance of the different data structures and competing algorithms. These visualisations allow for easy comparison between the data

structures and algorithms evaluated in the report.

5 Challenges

Several challenges were faced during the course of this project, the main ones being:

Firstly, it was challenging to find information on the technical aspects of existing Version Control Systems, especially the earlier generations of Version Control Systems. The lack of information on these technical aspects made it difficult to understand how the systems were implemented and what underlying data structures and algorithms were used.

Secondly, it was challenging to identify which data structures and algorithms would provide meaningful points of comparison. The vast number of data structures and algorithms made it challenging to select a suitable set for evaluation. The selected data structures and algorithms had to be representative of those used in real-world Version Control Systems and provide a basis for meaningful comparison.

Thirdly, the student had to learn the Go programming language's built-in benchmarking package, which they were initially unfamiliar with. The learning curve for this package was steep, and it took time to understand how to use it effectively for the purposes of this research project.

Finally, it was challenging to design automated benchmarks using mock data that were still representative of real-world use. The mock data had to be generated in a way that was realistic and representative of the data used in real-world Version Control Systems. This required careful consideration of the different types of data used in Version Control Systems and how they could be represented in the mock data.

6 Future Work

One possible area for future work is implementing a fully functional Version Control System to obtain the most realistic data for benchmarking. The fully functional Version Control System would allow for more realistic testing and benchmarking of the data structures and algorithms and provide insight into their performance under real-world usage scenarios. Other areas for future work are evaluating other data structures and algorithms not considered in this research project and developing more advanced benchmarking techniques than those used in this project.