Continuous File Synchronization System

October 30, 2024

Chapter 1

Introduction

1.1 Motivation

In today's digital age, the need for seamless, real-time file synchronization across devices has become essential. With the growth of remote work, distributed teams, and multi-device use, individuals and organizations require reliable solutions that ensure data is accessible and up-to-date without manual intervention. While cloud storage providers offer some solutions, they often come with privacy concerns, limitations on customization, and dependency on third-party servers.

To address these challenges, our graduation project focuses on developing a robust file synchronization system inspired by platforms like Syncthing, aiming for a decentralized, peer-to-peer approach that priotrizes user privacy, security, and ease of use. our intention is to eliminate reliance on centralized servers, providing users with greater control over their data while ensuring conflict-free file synchronization across devices. Through this system, we seek to contribute to the field of distributed computing by offering an alternative that enhances data accessibility and resilience

1.2 Problem Statement

Background

In a world where digital collaboration, remote work, and data accessibility are becoming increasingly essential, users often need real-time access to the most up-to-date versions of files across multiple devices. Traditional file-sharing services can be limited by connectivity issues, centralized storage constraints, or restrictions in updating files consistently across distributed devices. Hence, developing an efficient, decentralized file synchronization system to ensure continuous file availability and consistency across multiple devices in real time is crucial.

Objective

This project aims to design, develop, and implement a Continuous File Synchronization System that enables users to automatically synchronize files across devices while maintaining data integrity, handling conflicts effectively, and ensuring minimal data transfer. The system will allow users to sync files locally and remotely, with features such as file versioning, conflict resolution, and support for multiple operating systems.

Scope

The system will:

1. **Support Real-time Synchronization**: Files updated on one device should be reflected on all connected devices in real-time, ensuring minimal delay.

2. **Enable Cross-platform Compatibility**: The system should work on Windows, macOS, and Linux, allowing users to sync files across different operating systems.

3. **Ensure Data Integrity and Security**: Files should be transferred securely, and the system should ensure data integrity with mechanisms to verify file consistency.

4. **Handle Conflict Resolution**: The system should include a mechanism to handle conflicts when files are modified simultaneously on multiple devices.

5. **Implement Scalability**: The solution should scale efficiently to accommodate an increasing number of files, devices, and users.

Functional Requirements:

1. **Automatic Synchronization**: Detect file changes automatically and sync them across connected devices.

2. **Version Control**: Maintain version history of files to allow users to revert to previous versions if needed.

3. **Conflict Management**: Identify and resolve conflicts between different versions of the same file across devices.

4. **User Interface**: Provide an intuitive GUI for easy user interactions and configuration of sync settings.

1.3 Goals

The primary goal of this project is to develop an efficient, user-friendly file synchronization system that addresses the limitations of existing solutions and provides an enhanced user experience for personal and professional use. Specifically, the objectives include:

- **Reliability and Data Integrity:** Ensure that files are synchronized accurately across devices with minimal risk of data corruption or loss. Implement robust mechanisms to verify the integrity of synchronized data.
- **Cross-Platform Compatibility:** Support multiple operating systems (e.g., Windows, macOS, Linux) and mobile platforms, allowing seamless synchronization across diverse devices.
- **Ease of Use:** Ensure the system is straightforward to install, configure, and operate, focusing on an intuitive user interface and clear instructions.

1.4 Related Work

1.4.1 Currently Available Solutions

Several file synchronization solutions currently exist in the market, each offering distinct features and trade-offs in terms of performance, ease of use, and security. Some of the most popular file sync systems include:

- **Syncthing**: Syncthing is an open-source, decentralized file synchronization application designed to replace proprietary sync and cloud services. It enables users to synchronize files directly between devices on a local network or over the internet without storing data on third-party servers. Syncthing emphasizes data privacy, ensuring that data is encrypted during transfer and only accessible by the devices involved in the sync process. However, due to its decentralized nature, managing complex sync setups may require manual configuration.
- **Dropbox**: Dropbox is a widely-used cloud storage service that includes file synchronization as a core feature. Dropbox offers seamless sync across multiple devices, enabling users to access files on desktop and mobile platforms. Its user-friendly interface and collaboration features make it ideal for everyday users and small teams. However, Dropbox relies on centralized storage, which can raise privacy concerns, and its file syncing speed can vary based on network and storage limitations.
- **Google Drive**: Google Drive combines cloud storage and file synchronization, allowing users to store and access files on any device

with internet connectivity. The service includes robust integration with other Google applications, such as Google Docs and Sheets, making it popular for collaborative work. However, similar to Dropbox, Google Drive uses a centralized storage model, and users must trust Google with their data privacy.

- **OneDrive**: Microsoft OneDrive is a cloud-based solution that integrates tightly with Windows and the Microsoft Office suite. It offers both cloud storage and file synchronization features, making it a strong option for users in the Microsoft ecosystem. OneDrive provides tools for sharing and collaboration, particularly for enterprise users. Like Google Drive and Dropbox, OneDrive is centralized and may not meet the privacy needs of all users.
- **rsync**: rsync is a command-line utility for Unix-based systems that offers efficient, incremental file synchronization. While not specifically a file sync service, rsync provides high-speed sync capabilities over networks and is popular for backups and transferring large amounts of data. However, its usage requires familiarity with command-line operations, making it less accessible to everyday users.
- **Nextcloud**: Nextcloud is an open-source, self-hosted cloud storage and file synchronization solution. It allows users to set up their own server, keeping full control over data. Nextcloud provides additional collaborative tools and security options tailored for businesses and privacy-focused users. However, it requires setup and maintenance on the part of the user, which may not be suitable for non-technical users.

1.4.2 Limitations of Current Solutions

While each of these solutions has its own strengths, they share certain limitations. Centralized systems, like Dropbox, Google Drive, and OneDrive, come with privacy and data ownership concerns. Decentralized and self-hosted solutions, like Syncthing and Nextcloud, address privacy but can be challenging to configure for less technical users. These tools may also face performance and reliability issues when syncing large datasets over fluctuating network connections.

1.4.3 Features Matrix

1.5 Software Development Methodology Used

Chapter 2

Requirements Analysis

2.1 Surveys

2.2 Results of (Part 2.1) represented as charts

2.3 Functional Requirements

2.4 Non-functional Requirements

2.5 Use Case Diagrams

Chapter 3

Design

3.1 Sequence Diagrams

3.2 ERD

3.3 Data Flow Diagrams (DFD-level 0 and DFD-level 1)

3.4 Algorithms

3.5 UML Class Diagrams

Chapter 4

Implementation Aspects

4.1 Overall System Architecture (Blocks diagram)

4.2 Tools, Technologies and/or Programming Languages

4.3 Prototype