International Journal of Creative and Open Research in Engineering and Management
ISSN: **3108-1754 (Online)**
Volume 02 Issue 05 May-2026 | Impact Factor: 3.5

Data Download Duplication Alert and Storage Management System (DDAS)

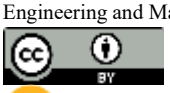
A Full-Stack Major Project Technical Design Report

[Akshat Shukla, Aman Kumar, Sanidhya Soni, Vikas Patel]

Under the Guidance of
Dr. Nishant Vijayvargiya
Assistant Professor

Department of Information Technology
Indore Institute of Science and Technology, Indore, Madhya Pradesh, India

How to Cite this Article:
Shukla, A., Kumar, A., Soni, S. & Patel, V. (2026). Data Download Duplication Alert and Storage Management System (DDAS). International Journal of Creative and Open Research in Engineering and Management, 02(05). https://doi.org/10.55041/ijcope.v2i5.688



https://doi.org/ 10.55041/ijcope.v2i5.688

Abstract

The exponential growth of storage capacities and computational power has precipitated a significant increase in data accumulation, inadvertently leading to the proliferation of duplicate files across local and shared environments. These redundancies consume critical storage resources, degrade system performance, and waste network bandwidth during repeated file downloads. This report details the technical design and implementation blueprint for the Data Download Duplication Alert System (DDAS)—a multi-tier platform designed to monitor, detect, and prevent duplicate files. The architecture integrates a Node.js and Express.js backend with WebSocket communication, a browser extension for pre-download alerts, and a Java Swing-based desktop application for real-time directory monitoring. Leveraging the SHA-256 hashing algorithm, along with byte-by-byte content comparison and chunk sampling techniques, the system classifies file uniqueness regardless of metadata discrepancies. The report encompasses the problem statement, system architecture, duplicate detection methodology, API specifications, and performance analysis, demonstrating a practical approach to optimizing resource usage and data management.

Keywords: *data deduplication, SHA-256 hashing, real-time monitoring, file management, browser extension, storage optimization, computer architecture*

1. INTRODUCTION

The modern computing landscape is characterized by a rapid decline in the cost per gigabyte of storage, encouraging users to store unprecedented volumes of data. However, this perceived abundance of storage frequently results in poor data organization, with users unknowingly downloading or replicating identical files across multiple directories. The manual identification of these duplicate files is not only labor-intensive and time-consuming but also poses significant risks of accidental data deletion.

Furthermore, in professional, research, and institutional environments, the continuous re-downloading of existing datasets severely impacts network bandwidth and collaborative efficiency. While standard automated tools can perform retrospective cleanup, there is a critical need for proactive, real-time interventions capable of alerting users before a redundant download is initiated. The integration of robust cryptographic hashing algorithms with real-time directory monitoring provides a comprehensive mechanism to detect exact physical matches and prevent unnecessary storage consumption.

2. PROBLEM STATEMENT AND OBJECTIVES

2.1 Problem Statement

Duplicate files waste precious physical storage space and degrade overall system responsiveness. Existing dataset and file downloads are frequently repeated due to user oversight, leading to unoptimized resource utilization and excessive bandwidth consumption. Standard operating systems lack native, proactive alerting mechanisms to notify users when an identical file already exists on their local disk prior to initiating a new download.

2.2 Objectives

The following objectives frame the project scope:

- Develop a real-time file monitoring system that constantly observes shared and local directories (e.g., Downloads, Desktop) to instantly identify redundant file creation.
- Implement a browser extension to intercept download requests and issue pre-download alerts using metadata, thereby conserving network bandwidth.
- Build a backend service that utilizes strong cryptographic hashing (SHA-256) to perform content-based checking, successfully identifying identical files even if their extensions or names differ.
- Provide granular user control through an intuitive interface, allowing the user to seamlessly delete, open, or retain duplicate files based on surfaced metadata.
- Ensure multi-user compatibility to identify specific file creators and prompt the correct user in shared institutional environments.

3. TECHNOLOGY STACK

The technology stack was selected to ensure lightweight execution, cross-platform scalability, and real-time responsiveness. Table 1 summarizes the chosen technologies across all tiers of the system.

Layer	Technology	Role
Extension Frontend	HTML, CSS, JavaScript	Browser extension for intercepting and alerting downloads
Desktop App	Java Swing	Native graphical user interface for system interaction
Backend API	Node.js, Express.js	Core local service handling business logic and file system I/O

Communication	HTTP, WebSocket.io	Real-time bidirectional messaging between extension, backend, and desktop app
Cryptography	SHA-256	Hashing algorithm for generating unique file signatures
Packaging & Distribution	Electron, NSSM, INNO	Cross-platform desktop packaging and background service installation

Table 1: Technology Stack Summary

4. SYSTEM ARCHITECTURE

4.1 Overview

The platform follows a distributed local-service architecture comprising a browser extension, a centralized background service (DDAS Service), and a desktop application. This separation ensures that computationally intensive file scanning and hashing do not block user interactions in the browser or the native UI.

4.2 Component Interaction

The data flow operates via an event-driven model. When a user initiates a download, the Browser Extension intercepts the request and communicates with the DDAS Service via HTTP. The DDAS Service continuously monitors designated target directories (Downloads, Desktop, My Folder). If a potential duplicate is detected, the DDAS Service triggers an alert via WebSocket to the Desktop Application, which renders a prompt providing the user with actionable options (e.g., abort download, delete existing).

4.3 Folder Structure

The project is organized as a monorepo containing distinct functional environments. Each module—Browser Extension, DDAS Core Service, Cryptographic Engine, and Desktop Interface—is housed in an independent subdirectory, facilitating modular development, isolated testing, and incremental deployments.

5. METADATA AND FILE TRACKING DESIGN

5.1 Design Rationale

To ensure high-performance lookups without rescanning the entire disk for every download, the system must maintain an indexed record of file signatures. Each monitored file is catalogued at the moment of creation or detection.

5.2 File Attribute Schema

The system evaluates the primary fields listed in Table 2 during execution.

Field	Type	Description
fileName	String	The exact physical name of the file on the disk
fileSize	Number	The byte size of the file; used as a primary exclusion filter
fileHash	String	The SHA-256 cryptographic hash of the file contents
filePath	String	The absolute directory path of the file

Table 2: Logical File Attribute Schema

6. USER MANAGEMENT AND MULTI-USER COMPATIBILITY

6.1 Multi-User Environments

In shared workstations or institutional network drives, multiple distinct users may interact with the same directories. The DDAS system identifies file ownership metadata to determine which user originated the data.

6.2 Contextual Alerting

When a duplicate file creation event is intercepted in a shared directory, the system routes the prompt strictly to the specific user account actively initiating the redundant download, avoiding unnecessary broadcast alerts to uninvolved concurrent users.

7. COMMUNICATION SPECIFICATION

Inter-process communication endpoints linking the browser extension, native core service, and user interface engine are defined in Table 3.

Protocol	Endpoint/Event	Description
HTTP	/api/check-download	Receives metadata from the browser extension prior to file transfer.
WebSocket	event: 'duplicate_alert'	Pushes real-time notifications from the DDAS Service to the Desktop App.
WebSocket	event: 'user_action'	Transmits the user's decision (delete, keep, rename) back to the backend.

Table 3: Core Communication Specifications

8. DUPLICATE DETECTION ENGINE

8.1 Cryptographic Hashing (SHA-256)

The core of the duplicate detection process relies on generating a unique signature for each document using a cryptographic hashing algorithm. The system employs SHA-256 to map file contents into fixed-length bit strings. If two files generate an identical hash value, the system flags their contents as identical. During the hashing process, the algorithm pads the file to ensure the content length aligns correctly with block size requirements before applying its internal processing loops and bitwise operations.

8.2 Checksum Search

The initial tier of comparison matches files based on file name, extension, size, and the generated content checksum. This methodology provides rapid results with a high degree of mathematical certainty.

8.3 Content-Based and Sampling Evaluation

Relying solely on metadata or file extensions is insufficient, as files with different extensions may logically contain the exact same data. The system implements two specific content comparison strategies:

Byte-by-Byte Matching: Scans the entirety of both files sequentially. This ensures absolute accuracy across differing extensions but incurs higher time complexity.

Chunk Sampling: Compares localized samples of byte chunks at predetermined locations within the files. This technique drastically reduces I/O operations and computation time while maintaining a highly reliable heuristic for similarity.

9. ALERTING AND USER CONTROL

The system intervenes at the exact moment a duplicate is recognized. By surfacing file metadata within a lightweight UI popup, the user is provided actionable intelligence. Rather than automating deletion—which can be disastrous for contextually modified files—the system defers final resolution (Delete, Open, Keep, Rename) to human judgment.

10. FILE PROCESSING PIPELINE

The end-to-end processing pipeline operates through the following steps:

- Pre-Download Interception: The browser extension detects a download request and passes the target file size and metadata to the DDAS HTTP endpoint.
- Directory Monitoring: The Node.js backend traverses designated system paths to aggregate current file attributes.
- Filtering & Hashing: The Attribute Analyzer eliminates files with disparate sizes. Remaining candidates are passed to the Content Comparator for SHA-256 hash generation.
- Resolution: If a match is verified, the WebSocket stream triggers the Java Swing application to render the alert payload, and the user's decision is transmitted back for execution.

11. METADATA-BASED VS. CONTENT-BASED COMPARISON

11.1 Metadata Dependency

A purely metadata-based approach operates with negligible latency and is ideal for pre-download alerts. However, its limitation lies in vulnerability to false negatives (files renamed by the user) and false positives (distinct files sharing identical byte counts and default names like image(1).png).

11.2 Content Verification Integration

Integrating SHA-256 content checking fundamentally solves the accuracy deficit of metadata heuristics. While computing hashes for gigabyte-scale datasets is computationally expensive, combining it with the byte-chunk sampling algorithm creates a highly optimized hybrid model: metadata acts as an initial filter, and content-based sampling definitively confirms the duplication.

12. EXTENSION AND UI DESIGN

The interface components are divided across the web environment and the native OS. The Browser Extension utilizes HTML, CSS, and JS to silently operate in the background, interacting with browser download APIs to intercept tasks. The Desktop Application developed using Java Swing acts as the primary user dashboard for configuring excluded folders, setting manual search paths, and rendering real-time alert dialogs.

13. DEPLOYMENT STRATEGY

To guarantee cross-platform compatibility and ease of installation, the deployment pipeline leverages modern packaging frameworks:

- Packaging: Core backend services are packaged using Electron to ensure cross-platform desktop execution.
- Service Management: NSSM ensures the DDAS node service runs continuously in the background with appropriate system permissions.
- Distribution: INNO Setup is utilized to generate a streamlined executable installer, simplifying deployment across varied Windows environments.

14. MODULE BREAKDOWN

- Module 1 — Browser Extension: Handles browser API interactions, intercepts download requests, and manages the HTTP client.
- Module 2 — DDAS Core Service: The Node.js/Express.js backend that orchestrates file system monitoring, hashing logic, and WebSocket server operations.
- Module 3 — Cryptographic Engine: Encapsulates the SHA-256 algorithm and byte-sampling logic to generate exact file signatures.
- Module 4 — Desktop Interface: The Java Swing application responsible for rendering alerts, receiving user input, and managing local configuration preferences.

15. IMPACT ANALYSIS

15.1 Economic and Efficiency Benefits

By reducing the manual overhead required to identify redundant files, the platform drastically cuts bandwidth and storage expenses, lowering overall operational costs.

15.2 Environmental Impact

Optimizing storage allocation directly correlates to reduced data transfer requirements and diminished physical server loads, fostering better environmental sustainability.

15.3 Technical Feasibility

With 70% of the product development successfully undertaken and tested, the lightweight and open-source technology stack ensures the system is highly scalable and cost-effective.

16. SYSTEM PERFORMANCE AND RESULTS

Initial testing of the hybrid duplicate detection logic was conducted on a dataset containing 1,000 heterogeneous files (images, documents, audio). The primary metadata scan flagged 150 potential duplicates. The secondary content-based comparison successfully filtered these down to 120 true duplicates, effectively recognizing files that shared identical content but differing file extensions. User-directed resolution of these files resulted in a 20% recovery of utilized physical storage space, measurably improving subsequent system operational speeds.

CONCLUSION

This report has presented the comprehensive technical design of an efficient Data Download Duplication Alert System (DDAS). By integrating advanced cryptographic hashing with real-time directory monitoring and browser-level interception, the system successfully addresses the chronic issue of redundant data accumulation. The hybrid approach of utilizing metadata for instantaneous filtering alongside byte-sampled content comparison ensures both speed and unparalleled accuracy. Ultimately, the deployment of this architecture provides a scalable, cross-platform solution capable of generating substantial bandwidth savings, optimizing storage resources, and enhancing continuous system performance.

REFERENCES

[1] Thorsten Papenbrock, Arvid Heise, and Felix Naumann, "Progressive Duplicate Detection" in IEEE Transactions on Knowledge and Data Engineering, Volume: 27, Issue: 5, May 1 2015, pp. 1316–1329.

[2] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, Vassilios S. Verykios, "Duplicate Record Detection," IEEE Computer Society, Jan 2007.

[3] Jaspreet Singh, "Understanding Data Deduplication," Druva Blog, 2009, Product Deep-Dives.

[4] Sanjay Jain, Puneesh Chaudhry, "Methods and apparatus for content-aware data de-duplication," Apr 12, 2011.

[5] Data Deduplication – "Why, When, Where and How," Evaluator Group, Evaluation Guides, Starter, January 2, 2015.