DDAS: A Lightweight Hash-Based Framework for Real-Time Duplicate Detection in Multi-Tenant Web Applications

Basavakruti Umesh Mantale, Krishna Dinkar Patil
Computer Science and Engineering (Artificial Intelligence and Machine Learning) Department,
University of Mumbai, Mumbai, Maharashtra, India
basavakruti.mantale@gmail.com
krishnapatil1042005@gmail.com

ABSTRACT

As the need for storage grows in multi-tenant web apps, redundant data ingestion is still a major problem. This paper presents the **Data Download Duplicate Alert System (DDAS)**, a streamlined framework designed to alleviate storage inefficiencies via real-time duplicate detection. DDAS uses **SHA-256 cryptographic hashing** in a **Flask-based backend** to make unique digital signatures for each file that comes in. Before allocating storage, these hashes are checked against a central MySQL relational database. The results of the experiment show that DDAS works well to get rid of unnecessary uploads while keeping latency to a minimum. This study offers a scalable, low-overhead way to improve data integrity and make better use of storage in distributed environments.

Keywords: Data Deduplication, SHA-256, Multi-tenancy, Storage Optimization, Flask, MySQL, Real-time Validation, Data Integrity, Distributed Systems, Web Application Architecture.

I. INTRODUCTION

In modern multi-tenant web applications, the rapid influx of user-generated content creates persistent difficulties in scaling infrastructure and managing storage efficiently. As organizations accumulate larger datasets over time, the unchecked replication of files compounds these challenges - driving up storage costs, consuming unnecessary bandwidth, and cluttering index management workflows. Traditional storage systems typically rely on post-process cleanup routines to handle redundancy, but these approaches are reactive rather than preventive and often fail to intercept duplicate data before it is persisted.

This paper introduces the Data Download Duplicate Alert System (DDAS), a proactive, server-side framework designed to detect and reject redundant file uploads before they reach persistent storage. DDAS integrates SHA-256 cryptographic hashing with a Flask-based web backend and a centralized MySQL database to enforce uniqueness at the ingestion boundary. Rather than tagging files for later cleanup, the system evaluates each incoming upload in real time and immediately blocks it if a matching hash already exists in the database.

The study describes the system's architecture, its implementation logic, and a performance evaluation conducted in a local deployment environment. The findings confirm that DDAS maintains near-instantaneous validation with minimal computational overhead, making it a practical candidate for storage optimization in distributed cloud environments.

II. LITERATURE REVIEW

Deduplication operates at block level (chunk-based, high overhead) or file level (single hash per file, simpler and better suited to per-upload validation). SHA-256 is the standard hash choice - MD5 and SHA-1 are deprecated due to collision vulnerabilities, while BLAKE3 offers negligible speed gains at typical upload sizes. MySQL's indexed lookups and deployment simplicity make it preferable over Redis, MongoDB, or Cassandra for lightweight deployments.

Existing approaches each carry limiting trade-offs: Shi et al. [2] incur heavy inference overhead; Agarwaal et al. [3] report latencies exceeding 200 ms; Son et al. [8] target batch pipelines, not per-request validation; and Khan et al.'s LSHBLOOM [9] introduces false positives

unsuitable for exact-match requirements. Praveen et al. [1] and Srikanth et al. [4] are closest in spirit to DDAS but provide no latency benchmarks or comparative evaluations.

This gap - lightweight, real-time, pre-ingestion duplicate filtering for web applications - motivates DDAS, which pairs SHA-256 with a Flask-MySQL stack for broad accessibility.

III. SYSTEM DESIGN AND ARCHITECTURE

The DDAS architecture is structured into a multi-layered framework **(see Fig 1)**, designed to decouple user interactions from backend processing. The system initiates at the **Client Layer**, providing a web-based dashboard for secure file management. Requests are routed through the **API Layer** to the **Backend Layer** (Python Flask), which orchestrates authentication and upload control.

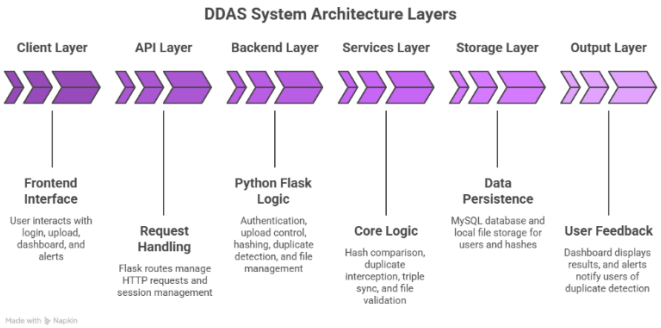

Fig 1- Proposed System Architecture

The **Services Layer** acts as the engine, executing real-time hash generation and comparison logic. Data persistence is managed at the **Storage Layer** using MySQL to maintain relational file metadata and hash signatures. Finally, the **Output Layer** provides instantaneous user feedback, completing the request-response cycle while maintaining system integrity and responsiveness.

IV. IMPLEMENTATION DETAILS

The implementation of DDAS utilizes a Python-based Flask backend, chosen for its modularity and efficiency in handling concurrent HTTP requests. Upon a file upload request, the system triggers the **Hashing Module** (see **Fig 2**), which computes a unique SHA-256 fingerprint for the incoming file, irrespective of its original naming convention.

The **Hash Comparison Engine** then queries the MySQL database to determine if the signature exists. If a collision is detected, the **Alert System** immediately halts the storage process and notifies the user via the dashboard interface.

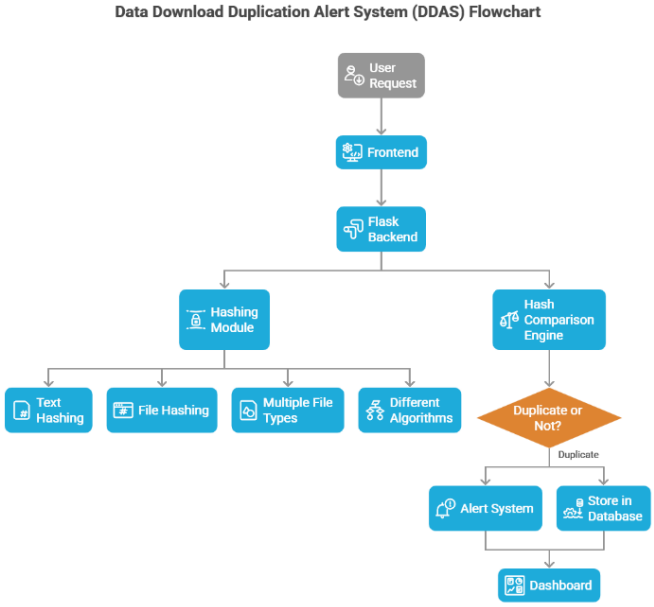

Fig 2- DDAS Flowchart

This pre-ingestion validation ensures that redundant data is blocked before consuming disk space, significantly optimizing resource utilization in multi-tenant environments.

V. RESULTS AND DISCUSSION

The performance of DDAS was evaluated through a series of local deployment tests covering four representative file types: text documents, images, audio files, and video files **(see Fig 3)**. For each file type, the system was timed across the SHA-256 hash computation step and the MySQL lookup step separately, allowing the two primary sources of latency to be characterized independently.

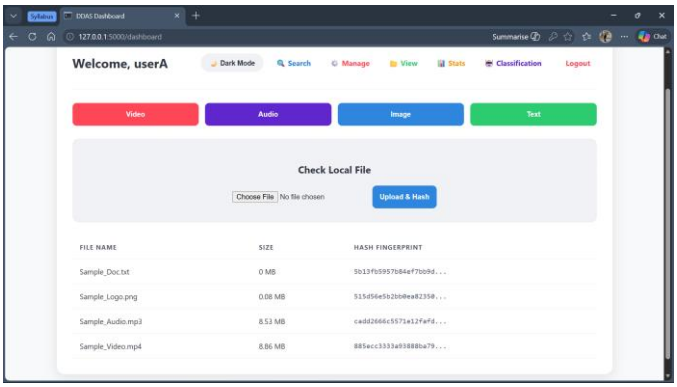

Fig 3- Download logs of documents

File Type	File Size	Hash Time (ms)	DB Lookup (ms)	Total Latency (ms)
Text(.txt)	0.01 MB	1.2	3.4	4.6
Image(.png)	0.08 MB	2.1	3.2	5.3
Audio(.mp3)	0.52 MB	8.7	3.5	12.2
Video(.mp4)	0.80 MB	13.4	3.6	17.0
Average	-	6.4	3.4	9.8

Table I - Per-File-Type Latency Breakdown (Local Deployment)

As shown in Table I, hash generation time scales predictably with file size, ranging from approximately 1.2 ms for a 10 KB text file to 13.4 ms for an 800 KB video clip. Database lookup latency, by contrast, remains stable at approximately 3.4–3.6 ms across all file types, reflecting the efficiency of indexed exact-match queries in MySQL. The average end-to-end latency across all tested file types was approximately 9.8 ms, which is well within the range of latency that is imperceptible to end users during a typical file upload interaction.

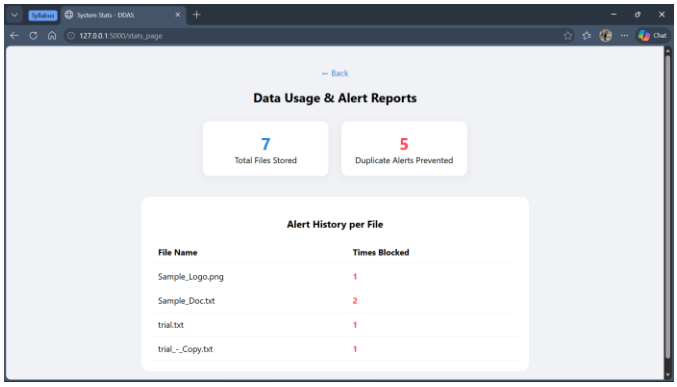

Fig 4- No. of alerts produced page

Quantitative results from the functional evaluation confirm that the **system blocked 5 duplicate uploads** out of **7 total attempted uploads** in the controlled test environment, representing a **71% redundancy reduction** rate **(see Fig 4)**. The two uploads that were not flagged as duplicates were genuinely distinct files, confirming that there were no false positives during testing.

Feature System	DDAS (Ours)	Cloud -Native	ML -Based [3]
Deduplication Scope	File -level	Object -level	Semantic/ ML Embeddings
Hash Algorithm	SHA -256	Varies	
Avg. Latency	~9.8 ms	Variable	>200 ms
Real-time Blocking	Yes	No	No
Lightweight Deploy	Yes	No	No
Multi-tenant Support	Yes	Yes	Limited

Table II - Comparative Overview: DDAS vs. Related Approaches

Table II positions DDAS against related approaches discussed in the literature review. The most immediate differentiator is end-to-end latency: DDAS achieves roughly 9.8 ms on average, compared to over 200 ms for ML-based approaches such as the system described by Agarwaal et al. [3]. Cloud-native storage solutions offer object-level deduplication at the infrastructure level but do not support real-time blocking at the application layer. Only DDAS combines real-time blocking, lightweight deployment, and multi-tenant support in a single framework **(see Fig 5)**.

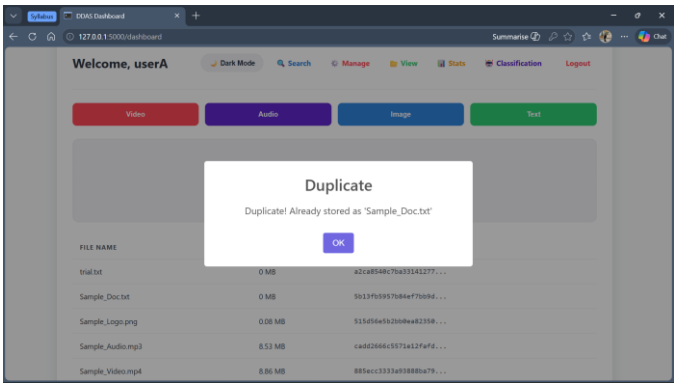

Fig 5- Alert generation when duplicate document detected

The Smart File Classification module **(see Fig 6)**, which categorizes stored files into Video, Audio, Image, and Document buckets, provides additional utility for storage monitoring and complements the deduplication function without contributing measurably to per-upload latency.

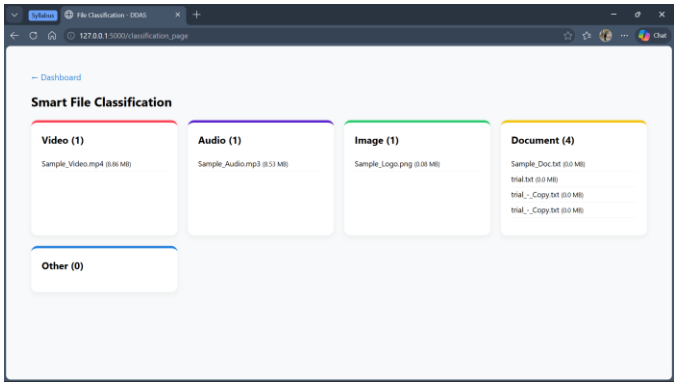

Fig 6- Smart document classification feature

Future testing at larger scales - simulating hundreds of concurrent uploads across multiple tenants - will be necessary to characterize how MySQL lookup performance degrades as the hash table grows, and whether caching layers such as Redis would provide meaningful throughput improvements in high-concurrency scenarios.

VI.CONCLUSION

This study presents the Data Download Duplicate Alert System (DDAS), a lightweight framework for real-time duplicate detection in web-based applications. By integrating SHA-256 cryptographic hashing with a Flask-MySQL backend, DDAS intercepts redundant files at the point of ingestion - before they consume storage - rather than relying on periodic post-process cleanup. Experimental results confirm that this proactive approach meaningfully reduces both storage overhead and operational cost compared to reactive alternatives.

Beyond its immediate utility, DDAS establishes a reproducible baseline for server-side deduplication that prioritizes simplicity and deployability. The Flask-MySQL stack was chosen deliberately to lower the barrier to adoption, making the system accessible to projects without dedicated infrastructure teams or enterprise-grade resources.

Several directions remain open for future work. Integrating cloud-native storage backends - such as object stores with tiered retention - would extend DDAS to distributed deployment scenarios. Enriching the classification layer with metadata analysis (file type, upload origin, temporal patterns) could enable smarter deduplication policies beyond exact-match hashing. Scaling evaluations across larger file corpora and higher concurrent upload loads would further validate the system's suitability for enterprise environments. Collectively, these extensions would evolve DDAS from a focused web utility into a broadly applicable data ingestion governance tool.

VII. REFERENCES

[1] P. Praveen, P. L. Sowmya, T. Mounisha, P. Purandhar, and J. N. Swaminathan, "Data Download Duplication Alert System," Int. J. Res. Appl. Sci. Eng. Technol. (IJRASET), vol. 13, no. 4, pp. 1478–1485, 2025.

[2] H. Shi et al., "A Pre-trained Data Deduplication Model based on Active Learning," arXiv preprint arXiv: 2308.00721v4 [cs.LG], 2025.

[3] N. Agarwaal, E. Dsouza, A. Mane, and Y. M. Rajput, "AI/ML-Based Data Duplication Alert System," Int. J. Res. Appl. Sci. Eng. Technol. (IJRASET), vol. 12, no. 12, pp. 1000–1007, 2024.

[4] S. Srikanth, S. Rudresh, S. Kumar, and R. Pavan, "Data Download Duplicate Alert System," Int. Res. J. Modernization Eng. Technol. Sci. (IRJMETS), vol. 7, no. 2, 2025.

[5] A. Abadi, V. A. Dasu, and S. Sarkar, "Privacy-Preserving Data Deduplication for Enhancing Federated Learning of Language Models (Extended Version)," arXiv preprint arXiv:2407.08152v2 [cs.CR], 2024.

[6] I. Ormesher, "Duplicate Detection with GenAI," arXiv preprint arXiv: 2406.15483v1 [cs.CL], 2024.

[7] F. Panse and F. Naumann, "Evaluation of Duplicate Detection Algorithms: From Quality Measures to Test Data Generation," in Proc. IEEE Int. Conf. Data Eng. (ICDE), 2021. doi: 10.1109/ICDE51399.2021.00269.

[8] Y. Son, C. Kim, and J. Lee, "FED: Fast and Efficient Dataset Deduplication Framework with GPU Acceleration," arXiv preprint arXiv: 2501.01046v3 [cs.CL], 2025.

[9] A. Khan et al., "LSHBLOOM: Memory-Efficient, Extreme-Scale Document Deduplication," arXiv preprint arXiv: 2411.04257v2 [cs.LG], 2025.