CENG 465 Distributed Computing Project Proposal

Project Title: OmniCloud – Erasure Coded Multi-Cloud Storage API

Type: RESTful Web Service

Version: 1.0

Date: December 06, 2025

1. Project Summary

OmniCloud is a next-generation storage orchestration layer and RESTful Web Service designed to fundamentally rethink how enterprises interact with public cloud infrastructure. In the current digital landscape, the centralization of data storage with hyperscale providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) has introduced critical systemic vulnerabilities.

Organizations today face the "Vendor Lock-in" paradox: while uploading data to the cloud is cost-effective, migrating it elsewhere or retrieving large volumes incurs prohibitive egress fees, effectively holding organizational data hostage [4]. Furthermore, reliance on a single provider is no longer a gamble; it is a guaranteed failure. The October 20, 2025 AWS US-East-1 outage crippled global banking and aviation for over 10 hours solely due to a DNS failure. Similarly, the Cloudflare blackout on November 18, 2025, took down 28% of global HTTP traffic due to a single configuration error. OmniCloud helps enterprises survive exactly these scenarios where providers fail.

OmniCloud addresses these challenges by acting as a "Meta-Cloud" gateway or “Cloud-of-Clouds” [4]. It abstracts the underlying storage infrastructure, providing developers with a unified, S3-compatible API interface while managing complex, multi-cloud distribution logic in the background.

The primary objective of this project is to democratize Erasure Coding technology—typically reserved for internal use by storage giants like Google [3] and Microsoft Azure [5]—and make it accessible via a simple REST API. Unlike traditional backup solutions that rely on "Replication" (creating expensive 1:1 copies of files, which increases storage costs by 200–300%) [5], OmniCloud utilizes the Reed-Solomon Erasure Coding algorithm, mathematically derived from Rabin’s Information Dispersal Algorithm (IDA) [2].

When a client uploads a file, the system processes it through a secure stream:

1. **Encryption:** The data is first encrypted using AES-256, ensuring that the content is unreadable before it leaves the client's control.
2. **Fragmentation:** The encrypted stream is mathematically divided into n pieces such that any m pieces suffice to reconstruct the file ($m < n$) [2].
3. **Distribution:** These shards are distributed across a heterogeneous network of providers. For example, a file might be split so that fragments 1 & 2 are stored on AWS S3, fragments 3 & 4 on Azure Blob Storage, and parity blocks on a local server.

This architecture provides mathematical guarantees of data durability. The original file can be fully reconstructed using any subset of surviving fragments ($k < n$). Consequently, even if an entire cloud provider suffers a catastrophic outage or permanently ceases operations, the data remains retrievable using the fragments stored on the remaining providers.

Beyond durability, OmniCloud introduces a new paradigm of "Plausible Deniability" and privacy. Since no single cloud provider possesses the complete file, only meaningless and encrypted binary shards the privacy of the data is cryptographically guaranteed against vendor-level snooping or data leaks [4]. Additionally, the system includes an intelligent Cost Arbitrage engine that routes read requests to the provider with the lowest current egress fees, and a Geo-Fencing policy manager that ensures data fragments are strictly stored within or excluded from specific national borders to comply with data sovereignty laws (e.g., GDPR, KVKK).

Ultimately, OmniCloud delivers a self-healing, cost-efficient, and provider-agnostic storage solution. It empowers developers to build resilient applications that are immune to the failure of any single infrastructure provider, moving the industry from "Cloud-Native" to "Cloud-Agnostic" resilience.

2. Project Team

Name	Student Number	Contribution
Ahmet Kaya	**21118080003**	**Distributed Systems Engineer:** API Architecture, Erasure Coding Implementation (Reed-Solomon), Multi-Cloud Adaptors (AWS/Azure/GCP integration).
Şevval Çakmak	**22118080010**	**Security Specialist:** Database Design (PostgreSQL/CockroachDB), AES-256 Encryption Implementation, Key Management, Metadata Service.

3. Problem Definition

Why this project?

As organizations migrate critical infrastructure to the cloud, they face three significant risks that current solutions fail to address adequately:

1. **Service Outages (Availability Risk):** Reliance on a single provider is a gamble. As noted in the analysis of the Google File System (GFS), component failures in distributed systems are the norm rather than the exception [3]. Large clusters inevitably experience node or rack failures; relying on a single cloud provider exposes the client to that provider's specific infrastructure failures.
2. **Vendor Lock-in (Economic Risk):** While uploading data to the cloud is cheap, retrieving it (Egress Fees) is prohibitively expensive. This creates a barrier to entry for switching providers. The "DepSky" model highlights that leveraging a diversity of clouds allows clients to switch providers or read from the cheapest provider to avoid these costs [4].
3. **Data Sovereignty (Legal Risk):** Strict regulations (such as GDPR or KVKK) often require data to remain within specific national borders. Global providers cannot always guarantee the granular control needed to exclude specific server locations.

Why existing systems fail:

Current object storage solutions like MinIO or Ceph are infrastructure-level tools that require managing clusters. Rclone acts as a synchronization tool but lacks a programmable API for real-time application integration. While GFS solved distributed storage within a single datacenter using replication [3], it does not inherently bridge multiple public clouds. OmniCloud fills this gap by acting as a gateway that implements the "Cloud-of-Clouds" paradigm [4].

4. State of the Art & Current Situation

Existing Competitors & Alternatives:

1. **Rclone:**
    - *Description:* A command-line program to manage files on cloud storage.
    - *Gap:* It is a CLI tool, not a REST API service. It relies on full-file replication rather than erasure coding, leading to higher storage costs.
2. **IPFS / Filecoin:**
    - *Description:* Decentralized storage networks.

- *Gap:* They struggle to meet enterprise Service Level Agreements (SLAs) and strict Access Control List (ACL) standards.

3. **Google File System (GFS) & Hadoop (HDFS):**
    - *Description:* GFS uses a single master to manage metadata and chunkservers for data storage [3].
    - *Gap:* GFS relies on 3x replication for fault tolerance [3]. While effective for performance, this triples storage overhead. OmniCloud aims to reduce this overhead to ~1.33x–1.5x using Erasure Coding, similar to modern Azure implementations [5].
4. **Windows Azure Storage (LRC):**
    - *Description:* Azure has moved from replication to "Local Reconstruction Codes" (LRC) to reduce overhead [5].
    - *Gap:* This is a proprietary internal optimization within Azure. OmniCloud democratizes this technology, applying it *across* providers rather than *within* one.

OmniCloud's Novelty:

OmniCloud differentiates itself through Erasure Coding rather than Replication. Standard replication (copying files 1:1) increases storage costs by 200–300% (storing 3 copies) [3][5]. By utilizing the math defined by Reed & Solomon [1] and Rabin [2], OmniCloud achieves higher durability with only ~50% overhead while ensuring that no single provider sees the raw data.

5. Proposed Solution

System Overview

- **Framework: Spring Boot (Java)** is selected for its robust ecosystem and multi-threading capabilities required for parallel cloud uploads.
- **Metadata Store: CockroachDB** (PostgreSQL compatible) to store file maps, fragment locations, and encryption keys. This mirrors the "Master" node concept in GFS but uses a distributed database to avoid the single point of failure inherent in the original GFS design [3].
- **Storage Engine:** A "Pass-through" streaming architecture that processes incoming file streams in memory, encrypts/splits them on the fly, and flushes them to providers.

Mathematical Implementation (Erasure Coding)

We will implement the Information Dispersal Algorithm (IDA) as proposed by Michael O. Rabin [2], which generalizes the polynomial interpolation work of Reed and Solomon [1].

The Dispersal Logic:

Let a file *F* be a sequence of bytes. We split *F* into sequences of length *m*. Let's treat a segment of the file as a vector $S = (b_1, b_2, \ldots, b_m)$ where each b_i is a byte (an element in a finite field, typically $GF(2^8)$) [2].

We wish to disperse this into *n* pieces (where $n > m$) such that any *m* pieces can reconstruct *S*. We choose *n* vectors $a_i = (a_{\{i1\}}, \ldots, a_{\{im\}})$ for $1 \leq i \leq n$ such that any subset of *m* vectors are linearly independent [2].

The fragments c_i (to be stored on different clouds) are calculated via matrix multiplication:

$$c_i = a_i \cdot S = \sum_{j=1}^{m} a_{ij} \cdot b_j$$

In matrix form, we select an $n \times m$ matrix *A* (Vandermonde matrix or Cauchy matrix) to generate the *n* fragments.

The Reconstruction Logic:

To reconstruct the file, we retrieve any *m* fragments. Let the retrieved fragments be vector *C′*. We construct the $m \times m$ matrix A' consisting of the rows of *A* corresponding to the retrieved fragments.

$$A' \cdot S = C'$$

Since the rows of *A* were chosen to be linearly independent, A' is invertible. We reconstruct the original data *S* by:

$$S = (A')^{\{-1\}} \cdot C'$$

Implementation Detail:

All arithmetic will be performed in the Galois Field $GF(2^8)$ (or $GF(2^{\{16\}})$ for larger blocks) to ensure the result of the addition/multiplication remains within the byte size limit (0-255) [1][2]. This ensures space efficiency, where the total storage used is $\frac{n}{m} \times |File|$. For example, with $m = 4$ and $n = 6$, we store 1.5x the data but can survive the loss of any 2 clouds.

Key Features

- **Self-Healing:** A background daemon monitors fragment health. If a fragment on Provider A is corrupted (detected via checksums, similar to GFS [3]), it is automatically reconstructed using parity data ($S = (A')^{\{-1\}} \cdot C'$) and re-uploaded to a new provider.
- **Geo-Fencing:** Users can define policies, and the placement algorithm will select the target *n* clouds based on allowed regions.
- **Cost Arbitrage:** The system intelligently directs read requests to the provider with the lowest current egress fees.

6. API Definitions and Endpoints

The following endpoints define the RESTful interface for the OmniCloud service:

#	Method	Endpoint	Description
1	**POST**	/api/v1/auth/register	Register a new tenant organization.
2	**POST**	/api/v1/auth/login	Authenticate and obtain JWT Bearer token.
3	**GET**	/api/v1/providers	List connected cloud providers (AWS, Azure, etc.).
4	**POST**	/api/v1/providers	Add a new cloud provider configuration (Access Keys).
5	**DELETE**	/api/v1/providers/{id}	Remove a provider from the pool.
6	**POST**	/api/v1/files/upload	Stream upload a file (Encryption + Splitting).
7	**GET**	/api/v1/files/{id}/download	Stream download (Reconstruction + Decryption).
8	**GET**	/api/v1/files	List all files in the virtual bucket.
9	**GET**	/api/v1/files/{id}/metadata	View shard distribution and health status.
10	**DELETE**	/api/v1/files/{id}	Permanently delete a file and all remote shards.
11	**PUT**	/api/v1/policies/geo-fence	Update allowed/blocked regions for data placement.
12	**POST**	/api/v1/maintenance/repair/{id}	Manually trigger a reconstruction job for a file.

13	**GET**	/api/v1/analytics/storage	Get current storage usage and cost savings report.
14	**GET**	/api/v1/admin/health	System-wide health check of all connected clouds.
15	**GET**	/api/v1/audit/logs	Retrieve security access logs for files.

7. Market Potential & Monetization Model

Target Users:

- Financial Institutions (Banks, Fintech) requiring zero data loss and compliance with DepSky-like multi-cloud resilience models [4].
- Government Archives requiring high data privacy.
- Media Companies needing cost-effective long-term storage.

Revenue Model:

- **Managed License:** Monthly fee per Terabyte of data managed.
- **Insurance Model:** Premium membership offering a "Data Loss Guarantee" backed by the mathematical certainty of Erasure Coding.

Market Size: With the global cloud storage market approaching $100 Billion, the "Multi-Cloud Management" sector is the fastest-growing segment as enterprises seek to avoid vendor lock-in.**8. Sustainability & Scalability Plan**

Technical Sustainability:

- **Microservice Architecture:** The application is containerized (Docker). The API Gateway scales independently of the Worker Nodes (Sharding Engine). This separation of control (metadata) and data flow is a lesson drawn from the GFS architecture to prevent the master node from becoming a bottleneck [3].
- **Open Standards:** Uses standard REST JSON and S3-compatible logic, ensuring long-term client compatibility.

Operational Sustainability:

- **CI/CD:** Automated testing pipelines via GitHub Actions ensure code reliability before deployment.
- **Documentation:** Full OpenAPI (Swagger) documentation provided for ease of integration.

Economic Sustainability:

- **Cost Optimization:** By using Erasure Coding (LRC concept), we reduce the storage overhead from the industry standard 3x (Replication) to approximately 1.33x [5]. This allows us to utilize cheaper "Cold Storage" (e.g., AWS Glacier) for parity shards.
- **Low Infrastructure Cost:** Designed to run on minimal hardware (<$100/mo) during the early stage, as heavy storage is offloaded to public clouds.

Social Sustainability:

- **Democratization:** Enables local data centers to compete with global giants by integrating them into the OmniCloud grid.
- **Transparency:** Public endpoints for verifying data integrity and placement.

9. Development Roadmap

The project will be completed within the semester using an Agile methodology:

- **Week 1: Core Logic Implementation**
  - Implement Galois Field Arithmetic (GF(2^8)) tables for performance [5].
  - Implement Reed-Solomon/IDA matrix multiplication and inversion in Java [1][2].
  - Implement AES-256 encryption/decryption utilities.
  - Setup CockroachDB/PostgreSQL schemas.
- **Week 2: Database & Authentication & Cloud Adapters**
  - Integrate AWS S3 SDK and Azure Blob Storage SDK.
  - Implement Spring Security with JWT.
  - Create the "Provider Interface" for abstraction.
- **Week 3: Streaming & Integration**
  - Develop the pass-through streaming service (Upload/Download).
  - Connect the Core Logic to the REST API endpoints.
- **Week 4: Reliability & Testing**
  - Implement the "Self-Healing" background worker.

- Perform failure simulation tests (manually deleting shards).
- Finalize documentation and presentation.

10. References

[1] Reed, I. S., & Solomon, G. (1960). Polynomial Codes over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2), 300–304.

[2] Rabin, M. O. (1989). Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2), 335-348.

[3] Ghemawat, S., Gobioff, H., & Leung, S. T. (2003). The Google file system. *ACM SOSP*, 29-43.

[4] Bessani, A., Correia, M., Quaresma, B., André, F., & Sousa, P. (2011). DepSky: dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage (TOS)*, 9(4), 1-33.

[5] Huang, C., Simitci, H., Xu, Y., Ogus, A., Calder, B., Gopalan, P., Li, J., & Yekhanin, S. (2012). Erasure Coding in Windows Azure Storage. *USENIX Annual Technical Conference*, 15-26.

[6] Amazon Web Services. (2024). "Amazon S3 REST API Documentation".

[7] Gartner Research. (2023). "Magic Quadrant for Distributed File Systems and Object Storage".