A

PROJECT PROPOSAL REPORT

On

DISTRIBUTED FILE SYSTEM (HAYSTACK)

DISTRIBUTED SYSTEMS

By

Rishi Nayak 2023201004
Abhinav Bahuguna 2023201040
Anmol Vashishtha 2023201079

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY, HYDERABAD

1. ABSTRACT

This project aims to build a fast, scalable distributed file storage system inspired by Facebook's Haystack. The system will focus on minimizing metadata overhead while ensuring fast lookups. We will use a **flat namespace**, **volume-based storage**, and **caching** to improve performance. The system will also support **replication** and **fault tolerance** to ensure data durability.

2. PROBLEM STATEMENT

Managing millions of small files efficiently is a common challenge in large scale systems. Traditional file systems struggle with excessive metadata overhead and slow lookup times. Our system will address these issues by **reducing metadata bottlenecks**, leveraging **caching for faster reads**, and distributing storage across multiple nodes to enhance scalability and resilience.

3. SOLUTION

1. **Flat Namespace:** Files will be identified by unique IDs for fast lookups.
2. **Volume-Based Storage:** Grouping files into volumes to reduce metadata operations.
3. **Caching Layer:** A Memcached layer to reduce disk reads and improve response times.
4. **Distributed Storage Nodes:** Multiple nodes deployed for efficient load distribution.
5. **Replication & Fault Tolerance:** Ensures data availability and protection.

4. TECHNICAL IMPLEMENTATION

1. **Programming Language: Go/Python,** for its performance, and concurrency support.
2. **Framework: gRPC** for fast, lightweight communication between system components.

3. **Caching System: Memcached** to improve read performance.
4. **Storage Management:** Organising files into volume structures for better efficiency.
5. **Replication and Fault Tolerance:** Data will be replicated to ensure high availability.

5. DELIVERABLES

1. **System Architecture Document** explaining design choices and data flow.
2. **gRPC API Implementation** for file uploads, retrievals, and deletions.
3. **Storage Node Implementation** supporting efficient file management.
4. **Caching Integration** using Memcached.
5. **Replication and Fault Tolerance Module** to ensure resilience.
6. **User Documentation** for setup, deployment, and usage instructions.

6. TIMELINE

1. **Week 1:** Design architecture and API endpoints.
2. **Week 2-3:** Develop core API functionality and storage node logic.
3. **Week 4:** Integrate caching and replication mechanisms.
4. **Week 5:** Testing, debugging, and performance tuning.
5. **Week 6:** Documentation, deployment preparation, and final review.