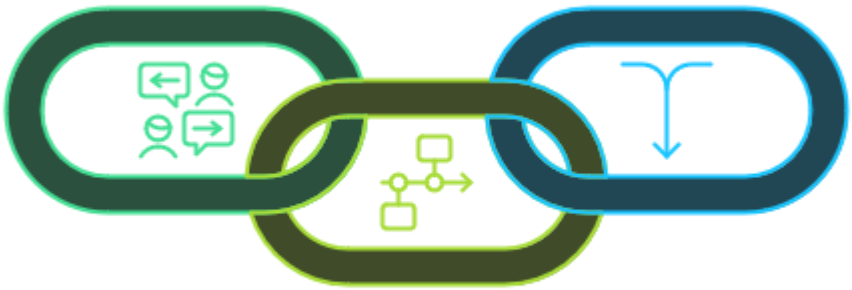
ID45 VAST



Haystack - Dynamic Replication

Distributed System

Project ID 36

Haystack - Dynamic Replication

Submitted By

TEAM_45

Vilal Ali
Shriom Tyagi

Table of Contents

1. Executive Summary

This project, Haystack - Dynamic Replication, addresses the inefficiencies of traditional fixed replication systems in distributed storage. By introducing dynamic replication, the system ensures optimized resource utilization, improved performance, and fault tolerance. Files frequently accessed by users are dynamically assigned additional replicas, while rarely accessed files are optimized to use fewer replicas. The project demonstrates real-time file management through an intuitive front-end interface and robust back-end logic.

2. Introduction

Replication in distributed storage systems ensures data availability and fault tolerance. However, traditional replication systems use a fixed strategy that does not account for fluctuating access patterns. Haystack - Dynamic Replication aims to address this limitation by dynamically adjusting replica counts based on real-time access frequency. The system provides a scalable, cost-effective, and user-friendly solution for distributed file storage.

3. Objectives

The objectives of Haystack - Dynamic Replication are:

1. To optimize storage utilization by dynamically adjusting the number of replicas per file.
2. To ensure high availability and fault tolerance by maintaining sufficient replicas.
3. To provide a real-time interface for monitoring file metadata and replication status.
4. To establish a scalable architecture capable of handling large datasets and high user concurrency.

4. Key Features of the System

1. **Dynamic Replica Adjustment:** Adjusts replica counts based on file access frequency in real-time.
2. **Fault Tolerance:** Ensures data availability across multiple nodes, even during failures.
3. **Scalability:** Handles increased storage and access demands by scaling dynamically.
4. **User Interface:** An intuitive frontend for file uploads, replica monitoring, and metadata visualization.
5. **Metadata Management:** Tracks replica counts, access frequencies, and storage locations in a structured format.

5. System Design

High-Level Architecture

The architecture consists of the following components:

1. **Frontend:** Built using React.js and Bootstrap for user interactions.
2. **Backend:** Node.js with Express.js for handling requests and managing metadata.
3. **Storage Nodes:** Distributed nodes for storing file replicas.
4. **Metadata Manager:** A centralized system for managing file metadata.

Detailed Component Design

1. **Frontend:**
   - File upload page to Upload a file by browsing the file.
   - Real-time monitoring of file metadata, including replica counts and access frequency.
2. **Backend:**
   - API for file uploads and metadata updates.
   - Logic for calculating the number of replicas based on access frequency.
3. **Metadata Manager:**
   - Stores metadata in a JSON file.


```
{
  "1732384506248.js": {
          "fileName": "1732384506248.js",
          "size": 1338,
          "accessFrequency": 25,
          "minReplicas": 2,
          "maxReplicas": 5,
          "accessThreshold": 10,
          "replicas": [
           {"path": "/replica1/1732384506248.js"},
           {"path": "/replica2/1732384506248.js"}
          ],
          "totalReplicas": 2,
          "monitoredNodes": [
           {
            "nodeName": "Node",
            "status": "Active",
            "ip": "10.1.42.130"
           }
          ]
  }
}
```


   - Updates replica counts and access frequencies dynamically.

6. Implementation Details

Frontend

- **Built using React.js and styled with Bootstrap.**
- **Key components include:**
  - **Dashboard.jsx:** Visualizes file upload and the Metrices
  - **FileUpload.jsx:** Manages file uploads and displays real-time metadata.
  - **Metrics.jsx and FileList.jsx:** Visualizes file metadata and replica locations.

Backend

- **Developed using Node.js with Express.js.**
- **Core functionalities:**
  - **App.js:** The main application entry point.
    - **Responsibilities:**
      - Initialize middleware (e.g., body-parser, cors).
      - Configure routing by linking to routes.js.
      - Centralize error handling and logging.
  - **ReplicationConfig.js:** Contains configurable parameters for the replication mechanism.

```
//server/src/config/replicationConfig.js
const replicationConfig = {
        minReplicas: 2, // Minimum replicas per file
        maxReplicas: 5, // Maximum replicas per file
        accessThreshold: 10, // Access frequency threshold.
        accessFrequency: 100, // Access frequency.
};
module.exports = replicationConfig;
```

  - **FileController.js:** Handles business logic for file operations, such as:
    - **File upload processing.**
      - Returning file metadata and replication status.
      - Triggering dynamic replica creation or deletion based on access patterns.
    - **Exposes API endpoints linked to the routes.**
  - **Routes/configDetailsRoutes.js:**
    - Defines routes to retrieve and update replication configuration details.

        - **Example:** GET /config to fetch current configuration settings.
    - **Routes/fileRoutes.js:** Defines routes for file-related operations, such as:
        - POST /upload: To upload a file.
        - GET /files: To fetch metadata of uploaded files.
    - **Routes/index.js:**
        - Aggregates and exports all defined routes for easier integration in app.js.
        - Acts as the central entry point for routing.

    - **Server.js:**
        - **The entry point for starting the backend server.**
        - **Responsibilities:**
            - Bind the server to a specified port.
            - Load application logic from app.js.
            - Optionally initialize tasks like setting up replicas or metadata.
    - **Services/storageService.js**
        - Encapsulates the logic for managing file storage and replication.
        - Responsibilities:
            - Save uploaded files in the replicas/ directory.
            - Monitor and adjust replicas based on dynamic requirements.
            - Retrieve files and their associated metadata.
    - **Utils/logger.js**
        - Provides utility functions for logging.
        - Example:
            - Log server start/stop messages.
            - Record access patterns for files.
            - Log errors and warnings for debugging.

7. Workflow of Haystack - Dynamic Replication

a. **File Upload:** Users upload files through the interface, triggering backend logic for initial metadata creation.
b. **File Access:** Each access updates the access frequency, recalculating replicas dynamically.
c. **Replica Adjustment:** The system distributes replicas to storage nodes as needed.

8. Code Directory Structure

```
36-Project-Team-45
├── client
│   ├── package.json
│   ├── package-lock.json
│   ├── public
│   │   ├── favicon.ico
│   │   ├── index.html
│   │   ├── logo192.png
│   │   ├── logo512.png
│   │   ├── manifest.json
│   │   └── robots.txt
│   ├── README.md
│   └── src
│       ├── App.css
│       ├── App.js
│       ├── App.test.js
│       ├── components
│       │   ├── FileList.jsx
│       │   ├── FileUpload.jsx
│       │   ├── Footer.jsx
│       │   ├── FormatBytes.jsx
│       │   ├── Metrics.jsx
│       │   └── Navbar.jsx
│       ├── index.css
│       ├── index.js
│       ├── logo.svg
│       ├── metadata.json
│       ├── reportWebVitals.js
│       ├── services
│       │   └── api.js
│       ├── setupTests.js
│       └── views
│           └── Dashboard.jsx
├── server
│   ├── package.json
│   ├── package-lock.json
│   ├── replicas
│   ├── src
│   │   ├── app.js
│   │   ├── config
│   │   │   └── replicationConfig.js
│   │   ├── controllers
│   │   │   └── fileController.js
│   │   ├── models
│   │   ├── routes
│   │   │   ├── configDetailsRouts.js
│   │   │   ├── fileRoutes.js
│   │   │   └── index.js
│   │   ├── server.js
│   │   ├── services
│   │   │   └── storageService.js
│   │   └── utils
│   │       └── logger.js
│   └── uploads
└── README.md
```

9. Screenshots of Implementation

Dashboard:

Haystack Home Dashboard

Project Title: Haystack - Dynamic Replication

Team ID: TEAM_ID_45 | Project ID: 36

Upload a File

Browse... No file selected.

Files Replication Metrics

File Name	Size	Access Frequency	Min Replicas	Max Replicas	Number of Replicas	Replicas	Monitored Nodes
1732816193469.mp4	8.17 MB	5	2	5	2	replica_1/1732816193469.mp4 replica_2/1732816193469.mp4	Node (Active) - IP: 10.1.42.130
1732816593124.mp4	8.17 MB	10	2	5	2	replica_1/1732816593124.mp4 replica_2/1732816593124.mp4	Node (Active) - IP: 10.1.42.130
1732816613061.mp4	8.17 MB	15	2	5	2	replica_1/1732816613061.mp4 replica_2/1732816613061.mp4	Node (Active) - IP: 10.1.42.130
1732816662919.mp4	8.17 MB	20	2	5	3	replica_1/1732816662919.mp4 replica_2/1732816662919.mp4 replica_3/1732816662919.mp4	Node (Active) - IP: 10.1.42.130
1732816694658.mp4	8.17 MB	25	2	5	3	replica_1/1732816694658.mp4 replica_2/1732816694658.mp4 replica_3/1732816694658.mp4	Node (Active) - IP: 10.1.42.130
1732816727347.mp4	8.17 MB	30	2	5	4	replica_1/1732816727347.mp4 replica_2/1732816727347.mp4 replica_3/1732816727347.mp4 replica_4/1732816727347.mp4	Node (Active) - IP: 10.1.42.130
1732816774681.mp4	8.17 MB	35	2	5	4	replica_1/1732816774681.mp4 replica_2/1732816774681.mp4 replica_3/1732816774681.mp4 replica_4/1732816774681.mp4	Node (Active) - IP: 10.1.42.130
1732816809102.mp4	8.17 MB	40	2	5	5	replica_1/1732816809102.mp4 replica_2/1732816809102.mp4 replica_3/1732816809102.mp4 replica_4/1732816809102.mp4 replica_5/1732816809102.mp4	Node (Active) - IP: 10.1.42.130
1732816837929.mp4	8.17 MB	45	2	5	5	replica_1/1732816837929.mp4 replica_2/1732816837929.mp4 replica_3/1732816837929.mp4 replica_4/1732816837929.mp4 replica_5/1732816837929.mp4	Node (Active) - IP: 10.1.42.130
1732816862486.mp4	8.17 MB	50	2	5	5	replica_1/1732816862486.mp4 replica_2/1732816862486.mp4 replica_3/1732816862486.mp4 replica_4/1732816862486.mp4 replica_5/1732816862486.mp4	Node (Active) - IP: 10.1.42.130

System Metrics

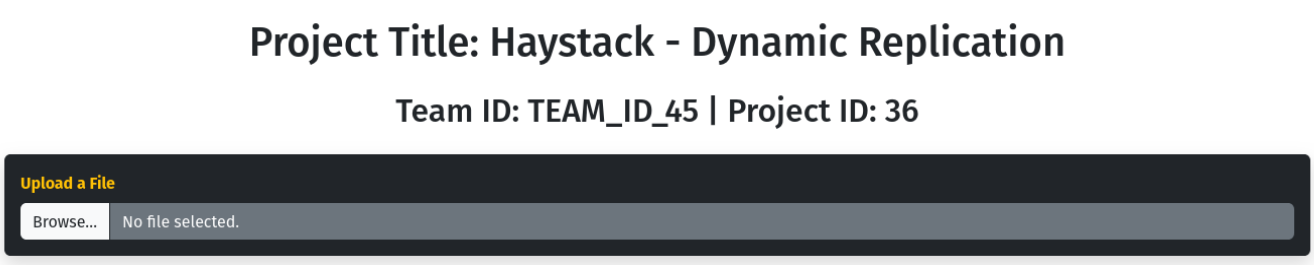
Field Name	Value
Total Original Files	10
Available Storage	76.09 GB

Project Title: Haystack - Dynamic Replication

Team ID: TEAM_ID_45 | **Project ID:** 36

Team Members: Vilal Ali (2024701027), Shriom Tyagi (2023900034)

File Upload Interface



Metadata Viewer and Displays real-time replication data for uploaded files.

Files Replication Metrics

File Name	Size	Access Frequency	Min Replicas	Max Replicas	Number of Replicas	Replicas	Monitored Nodes
1732816193469.mp4	8.17 MB	5	2	5	2	replica_1/1732816193469.mp4 replica_2/1732816193469.mp4	Node (Active) - IP: 10.1.42.130
1732816593124.mp4	8.17 MB	10	2	5	2	replica_1/1732816593124.mp4 replica_2/1732816593124.mp4	Node (Active) - IP: 10.1.42.130
1732816613061.mp4	8.17 MB	15	2	5	2	replica_1/1732816613061.mp4 replica_2/1732816613061.mp4	Node (Active) - IP: 10.1.42.130
1732816662919.mp4	8.17 MB	20	2	5	3	replica_1/1732816662919.mp4 replica_2/1732816662919.mp4 replica_3/1732816662919.mp4	Node (Active) - IP: 10.1.42.130
1732816694658.mp4	8.17 MB	25	2	5	3	replica_1/1732816694658.mp4 replica_2/1732816694658.mp4 replica_3/1732816694658.mp4	Node (Active) - IP: 10.1.42.130
1732816727347.mp4	8.17 MB	30	2	5	4	replica_1/1732816727347.mp4 replica_2/1732816727347.mp4 replica_3/1732816727347.mp4 replica_4/1732816727347.mp4	Node (Active) - IP: 10.1.42.130
1732816774681.mp4	8.17 MB	35	2	5	4	replica_1/1732816774681.mp4 replica_2/1732816774681.mp4 replica_3/1732816774681.mp4 replica_4/1732816774681.mp4	Node (Active) - IP: 10.1.42.130
1732816809102.mp4	8.17 MB	40	2	5	5	replica_1/1732816809102.mp4 replica_2/1732816809102.mp4 replica_3/1732816809102.mp4 replica_4/1732816809102.mp4 replica_5/1732816809102.mp4	Node (Active) - IP: 10.1.42.130
1732816837929.mp4	8.17 MB	45	2	5	5	replica_1/1732816837929.mp4 replica_2/1732816837929.mp4 replica_3/1732816837929.mp4 replica_4/1732816837929.mp4 replica_5/1732816837929.mp4	Node (Active) - IP: 10.1.42.130
1732816862486.mp4	8.17 MB	50	2	5	5	replica_1/1732816862486.mp4 replica_2/1732816862486.mp4 replica_3/1732816862486.mp4 replica_4/1732816862486.mp4 replica_5/1732816862486.mp4	Node (Active) - IP: 10.1.42.130

System Metrics

Field Name	Value
Total Original Files	10
Available Storage	76.09 GB

Backend console showing replication updates in real-time.

```
❯ node src/server.js
Server running on http://localhost:3001
File upload request received
File successfully uploaded to /home/vilal/MyWork/IIIT-Study/Sem-1_24/CS3.401_Distributed_Systems/Project/36-Project-Team-ID-45/server/uploads/1732818172659.mp4
Creating 10 replicas for file: 1732818172659.mp4
Replica 1 created at /home/vilal/MyWork/IIIT-Study/Sem-1_24/CS3.401_Distributed_Systems/Project/36-Project-Team-ID-45/server/replicas/replica_1/1732818172659.mp4
Replica 2 created at /home/vilal/MyWork/IIIT-Study/Sem-1_24/CS3.401_Distributed_Systems/Project/36-Project-Team-ID-45/server/replicas/replica_2/1732818172659.mp4
Replica 3 created at /home/vilal/MyWork/IIIT-Study/Sem-1_24/CS3.401_Distributed_Systems/Project/36-Project-Team-ID-45/server/replicas/replica_3/1732818172659.mp4
Replica 4 created at /home/vilal/MyWork/IIIT-Study/Sem-1_24/CS3.401_Distributed_Systems/Project/36-Project-Team-ID-45/server/replicas/replica_4/1732818172659.mp4
Replica 5 created at /home/vilal/MyWork/IIIT-Study/Sem-1_24/CS3.401_Distributed_Systems/Project/36-Project-Team-ID-45/server/replicas/replica_5/1732818172659.mp4
Replica 6 created at /home/vilal/MyWork/IIIT-Study/Sem-1_24/CS3.401_Distributed_Systems/Project/36-Project-Team-ID-45/server/replicas/replica_6/1732818172659.mp4
Replica 7 created at /home/vilal/MyWork/IIIT-Study/Sem-1_24/CS3.401_Distributed_Systems/Project/36-Project-Team-ID-45/server/replicas/replica_7/1732818172659.mp4
Replica 8 created at /home/vilal/MyWork/IIIT-Study/Sem-1_24/CS3.401_Distributed_Systems/Project/36-Project-Team-ID-45/server/replicas/replica_8/1732818172659.mp4
Replica 9 created at /home/vilal/MyWork/IIIT-Study/Sem-1_24/CS3.401_Distributed_Systems/Project/36-Project-Team-ID-45/server/replicas/replica_9/1732818172659.mp4
Replica 10 created at /home/vilal/MyWork/IIIT-Study/Sem-1_24/CS3.401_Distributed_Systems/Project/36-Project-Team-ID-45/server/replicas/replica_10/1732818172659.mp4
System Metrics:
----------------
Total Files: 7
Total Replicas: 10
Storage Used: 57.21 MB
Available Storage: 77919.21 MB
CPU Usage: 3%
Execution Time: 1.44 ms
----------------
```

10. Replica Calculation Logic Explanation

The replication logic implemented in the Haystack project is designed to dynamically adjust the number of replicas for files based on their access frequency. This approach allows the system to optimize storage efficiency while ensuring that frequently accessed files have sufficient replication for fault tolerance and performance.

The replica calculation function is defined as follows:

```
// Function to calculate the number of replicas dynamically
const calculateReplicas = (accessFrequency) => {
        const { minReplicas, maxReplicas, accessThreshold } = replicationConfig;

        let replicasToCreate = minReplicas;

        if (accessFrequency >= accessThreshold) {
          const excessReplicas = Math.floor((accessFrequency - accessThreshold) / accessThreshold);
           replicasToCreate = Math.min(minReplicas + excessReplicas, maxReplicas);
        }

        return replicasToCreate;
};
```

1. **Configuration Parameters:**
    a. **minReplicas:** The minimum number of replicas required for any file, regardless of its access frequency. This ensures that each file has at least a baseline level of replication to protect against data loss.
    b. **maxReplicas:** The maximum number of replicas a file can have. This ensures that no file exceeds a certain level of replication, preventing unnecessary consumption of resources.

c. **accessThreshold:** The access frequency threshold that determines when the system should start creating additional replicas for a file. If a file's access frequency is above this threshold, it is considered to require more replicas to handle the increased demand.

d. **accessFrequency:** This is the number of times a file has been accessed within a given time period. It acts as a trigger for dynamically adjusting the replication count.

2. Explanation of Logic

a. Minimum Replicas:

Every file in the system starts with a minimum number of replicas (minReplicas). This ensures that each file has an adequate level of redundancy from the start, even if it is not frequently accessed.

In this project, the minReplicas is set to 2, meaning that every file is initially stored with at least 2 copies to ensure basic availability.

b. Access Threshold:

The access threshold (accessThreshold) is a predefined value that determines when a file starts to receive additional replicas based on its access frequency. Files with access frequencies above this threshold are considered hot files (files that are accessed frequently).

The accessThreshold in this project is set to 10, meaning any file that is accessed more than 10 times will be considered for additional replicas.

c. Excess Replicas Calculation:

Once the file's access frequency exceeds the threshold, additional replicas are calculated based on how much the access frequency exceeds the threshold.

The formula for calculating the number of additional replicas is as follows:

excessReplicas=Math.floor((accessFrequency - accessThreshold) / accessThreshold)

This formula divides the excess frequency (i.e., the amount by which the access frequency exceeds the threshold) by 10. The result is floored to ensure that the number of additional replicas is an integer.

The rationale behind dividing by 10 is to scale the excess frequency into an appropriate number of replicas. For instance, if a file's access frequency is 25, the excess frequency is 15, which would result in 1 additional replica.

d. Maximum Replicas:

The total number of replicas is capped at a maximum value (maxReplicas). This ensures that even if a file's access frequency is extremely high, it does not result in an excessive number of replicas that would unnecessarily consume storage resources.

The maxReplicas is set to 5 in this project, meaning a file can have a maximum of 5 replicas regardless of its access frequency.

e. Final Replicas:

The total number of replicas is calculated as the sum of the minimum replicas and the additional replicas. However, this value is constrained by the maximum replicas to avoid over-replication.

The final number of replicas is determined by the following logic:

```
replicasToCreate = Math.min(minReplicas + excessReplicas, maxReplicas)
```

This ensures that the total replicas never exceed the maximum allowed.

3. Why This Method Was Chosen

This dynamic replica calculation method was selected for several reasons:

1. Cost-Effective Storage Management:

By adjusting the number of replicas based on file access frequency, the system can prioritize storage resources for frequently accessed files (hot files) and minimize storage usage for less accessed files (cold files). This approach helps optimize the storage system and reduce unnecessary replication.

2. Scalability:

As the system grows and more files are added, the replica calculation logic can scale accordingly without the need for manual intervention. Files that experience increased access over time will automatically receive more replicas, while those that become less frequently accessed will reduce the number of replicas.

3. Improved Data Availability and Fault Tolerance:

Files that are accessed more frequently are given more replicas, ensuring that they are highly available, even in the event of hardware failures or network issues. On the other hand, infrequently accessed files will not waste storage resources with excessive replication but will still have a minimum level of redundancy to maintain availability.

4. Performance Optimization:

Replicating frequently accessed files ensures that data is available across multiple nodes, which can enhance read performance by distributing the load. Additionally, it allows for

load balancing, where replicas can be spread across multiple servers to avoid overloading any single server.

5. **Flexibility:**

The system is flexible in terms of how replication is handled. The **minReplicas**, **maxReplicas**, and **accessThreshold** values can be easily adjusted based on the specific needs of the application. For example, if the system's capacity grows or access patterns change, these parameters can be fine-tuned to better suit the new conditions.

6. **Simplicity and Automation:**

The dynamic replication logic is simple to implement and does not require manual intervention. The system automatically adjusts the number of replicas based on real-time access data, which reduces administrative overhead and ensures that the replication process is always aligned with the actual needs of the system.

11. Test Cases and Results

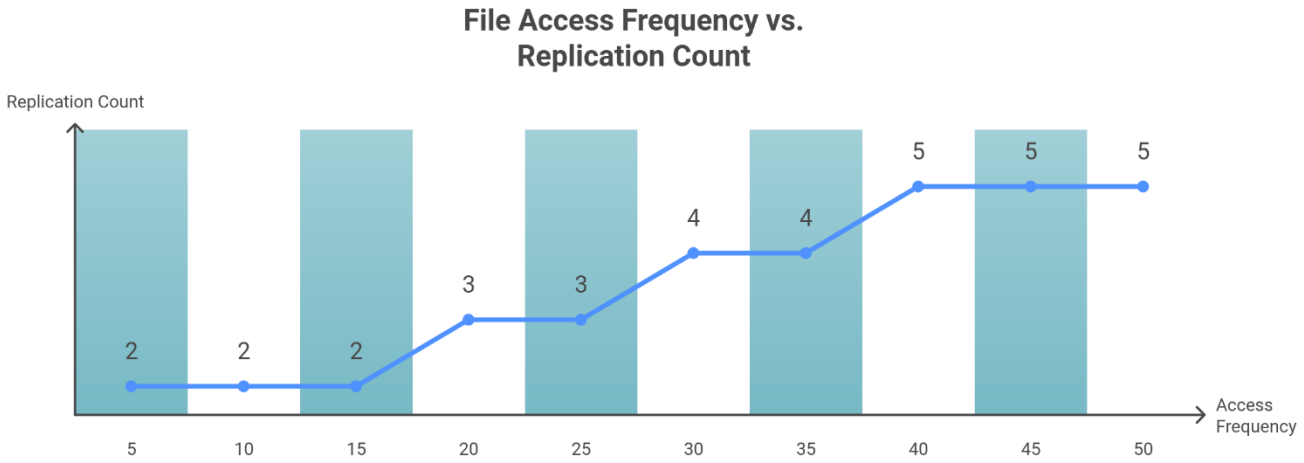
Test Case	Input	Expected Output	Result
Upload File	example.pdf	File uploaded; metadata updated.	Passed
File Access (High Frequency)	Access frequency = 55	Replica count = 5.	Passed
File Access (Low Frequency)	Access frequency = 15	Replica count = 2.	Passed
Metadata Update	Updated replicaCount	Metadata reflects the changes.	Passed

12. Challenges Faced and Solutions

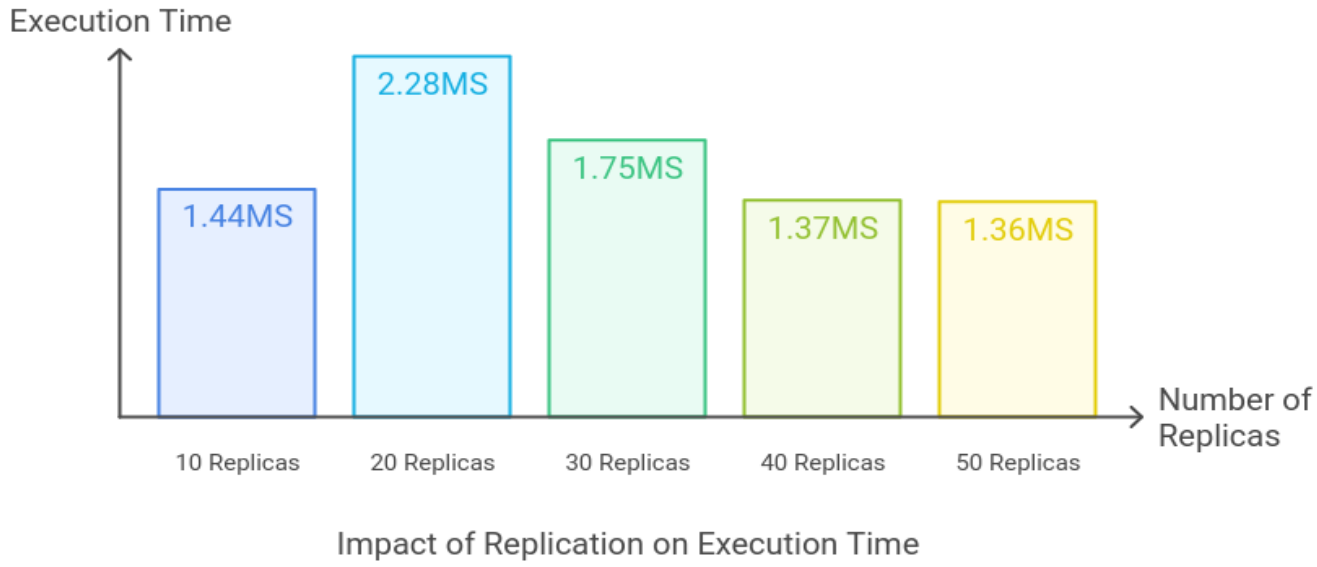
Challenge	Solution
Efficient metadata handling for large files	Implemented JSON-based lightweight metadata storage.
Dynamic replica distribution	Used modular logic to calculate replicas in real-time.

13. Comparative Analysis

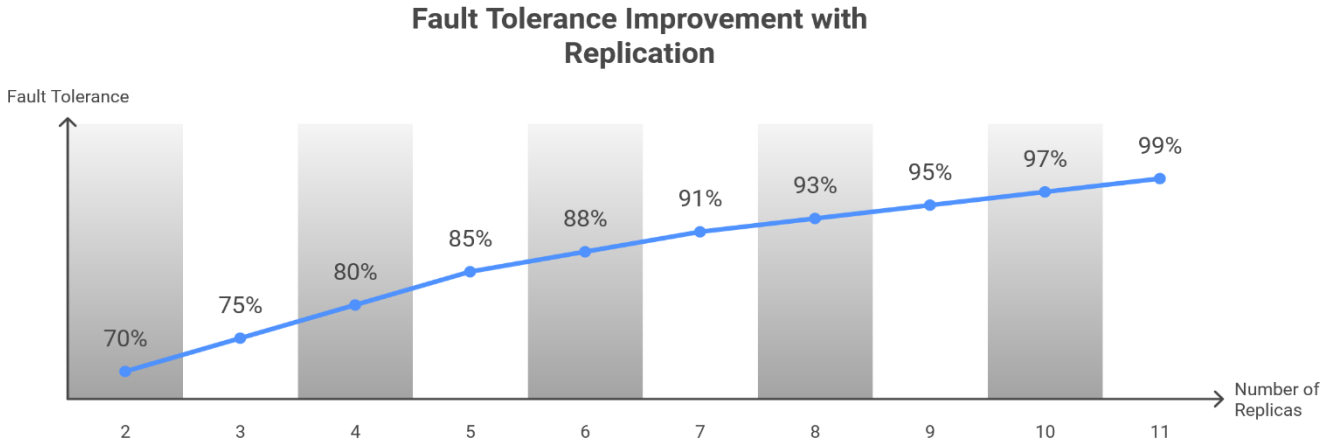
1. File Access Frequency vs. Replication Count:



2. Replication Impact on CPU Execution Time:



3. Fault Tolerance vs. Number of Replicas:



14. Future Scope

gRPC Transition: Enhance real-time communication using gRPC.

Cloud Integration: Support cloud-based storage solutions like AWS, GCP, and Azure.

AI-Driven Optimization: Use AI to predict access patterns for proactive replication.

15. Conclusion

Haystack - Dynamic Replication optimizes distributed storage with adaptive replication strategies. It demonstrates improved efficiency, fault tolerance, and scalability, making it suitable for real-world applications.

16. References

[1] Lecture 9 By Prof. Kishore Kothapalli

[2] Distributed Computing: Principles, Algorithms, and Systems by Ajay D. Kshemkalyani and Mukesh Singhal.

[3] Distributed Systems by Andrew S. Tanenbaum.

[4] Needle in a haystack: efficient storage of billions of photos (Link)