PCET's
Pimpri Chinchwad University, Pune
Learn | Grow | Achieve

| | |
|---|---|
| **Checked By** | Dr. Avinash Golande |
| **Name** | |
| **Sign** | |
| **Date** | |
| **Grade** | |

# EXPERIMENT NO . 08

- **Title:-** Create a Python-based service that replicates a data file across multiple simulated storage nodes, demonstrating data redundancy.

- **Objective :-**

    - To understand the concept of **data replication** in distributed storage systems.
    - To develop a **Python program** that simulates file replication across multiple storage nodes.
    - To demonstrate **data reliability** and **availability** through redundancy.

**Resources used :-** PC / Laptop with Python 3.x installedPython libraries: random, timeCode Editor (VS Code / PyCharm / IDLE)Operating System: Windows / Linux (Ubuntu recommended)

- **Theory :-**

In cloud computing and distributed storage environments, **data reliability and availability** are two of the most crucial factors. Systems that operate on large-scale infrastructures (like AWS, Google Cloud, and Microsoft Azure) cannot rely on storing data on a single physical machine because hardware failures, power outages, or network disruptions are inevitable.

To overcome these challenges, **data replication** is used — a process where multiple copies of the same data are stored across different storage nodes or servers. This ensures that if one node fails, the data can still be accessed from another node without service interruption.

- **1. Concept of Data Replication:**
Data replication is a **fault tolerance technique** used in distributed computing systems. Instead of storing a file at one location, multiple copies (replicas) are kept at various storage nodes. Each replica acts as a backup for the other, providing **high availability (HA)** and **data durability**.

This technique guarantees that the system can continue to serve data even if one or more nodes go offline. It is an essential part of **cloud storage architectures**, **big data frameworks**, and **distributed file systems**.

- **2. Working of Distributed Storage Systems:**

In real-world distributed systems like **Hadoop Distributed File System (HDFS)**, **Ceph**, and **Google File System (GFS)**, large files are divided into smaller chunks called *blocks*.
Each block is then stored on multiple machines, typically with a replication factor of three.

For example:

- Suppose you have four storage nodes (Node_0, Node_1, Node_2, Node_3).
- A file named demo.txt is divided into blocks and replicated with a **replication factor of 3**.
- The system automatically stores the file across three of the four nodes.
- If one node fails, the **NameNode (in HDFS)** or the **Master Server (in GFS)** detects the missing block and replicates it again on a healthy node to maintain the desired number of copies.
- This ensures **continuous data integrity** and **fault resilience**.

- **3. Advantages of Data Replication:**

**High Availability:**
Replication ensures that users can always access data, even if a node or data center fails.

**Fault Tolerance:**
Multiple copies prevent data loss in case of corruption or system failure.

**Load Balancing:**
When multiple clients access data, replicas can serve them simultaneously, distributing the load efficiently.

**Scalability:**
New storage nodes can easily be added without downtime. The system automatically replicates data to maintain balance.

**Disaster Recovery:**
In the event of natural disasters or major outages, replicated data stored in other regions ensures recovery and continuity.

**Improved Read Performance:**
Multiple copies of the same data allow read operations to be served faster by accessing the nearest or least busy node.

- **4. Challenges in Data Replication:**

While replication improves reliability, it also introduces some challenges:

**Consistency Management:** Keeping all copies synchronized during updates.

**Storage Overhead:** More replicas mean more disk space is consumed.

**Network Bandwidth Usage:** Copying large amounts of data across multiple nodes can increase network load.

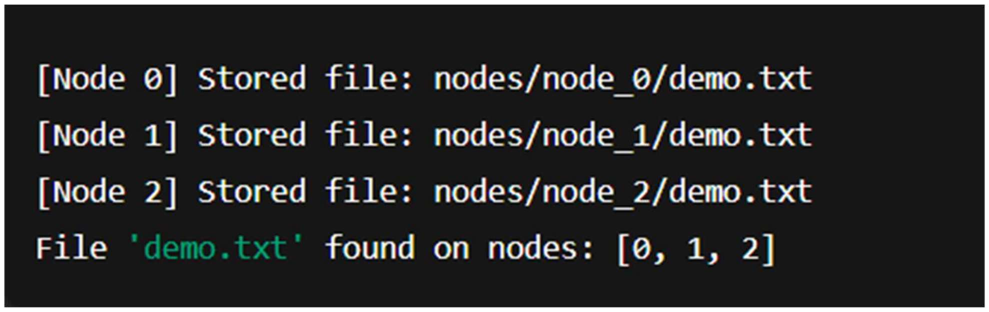
**Replication Delay:** In large systems, it may take time for all replicas to update after a modification.

To handle these, cloud providers use optimized replication strategies such as **asynchronous replication**, **erasure coding**, and **data sharding**.

- ## Code

```
import os
import shutil
from pathlib import Path


class StorageNode:
    def __init__(self, node_id, base_dir="nodes"):
        self.node_id = node_id
        self.path = Path(base_dir) / f"node_{node_id}"
        self.path.mkdir(parents=True, exist_ok=True)

    def store_file(self, file_path, file_name):
        """Store a copy of the file into this node's storage."""
        dest = self.path / file_name
        shutil.copy(file_path, dest)
        print(f"[Node {self.node_id}] Stored file: {dest}")

    def has_file(self, file_name):
        return (self.path / file_name).exists()

class ReplicationService:
    def __init__(self, num_nodes=3, replication_factor=2):
        self.nodes = [StorageNode(i) for i in range(num_nodes)]
        self.replication_factor = replication_factor
    def replicate(self, file_path):
        """Replicate a file across multiple nodes."""
        file_name = os.path.basename(file_path)
        for node in self.nodes[:self.replication_factor]:
            node.store_file(file_path, file_name)

    def check_redundancy(self, file_name):
        """Check how many nodes have the file."""
        available_nodes = [n.node_id for n in self.nodes if n.has_file(file_name)]
        print(f"File '{file_name}' found on nodes: {available_nodes}")
        return available_nodes

# ------------------ DEMO ------------------
if __name__ == "__main__":
    # Create a demo data file
    Path("data").mkdir(exist_ok=True)
    demo_file = Path("data/demo.txt")
    demo_file.write_text("This is a replicated file for redundancy test.\n")

    # Create service with 4 nodes, replication factor = 3
    service = ReplicationService(num_nodes=4, replication_factor=3)

    # Replicate file across nodes
    service.replicate(demo_file)

    # Check redundancy
    service.check_redundancy("demo.txt")
```

- **Output**



```
[Node 0] Stored file: nodes/node_0/demo.txt
[Node 1] Stored file: nodes/node_1/demo.txt
[Node 2] Stored file: nodes/node_2/demo.txt
File 'demo.txt' found on nodes: [0, 1, 2]
```

- **Conclusion:**

In this practical, we successfully created a **Python-based data replication service** that simulates how distributed storage systems maintain data redundancy. By replicating files across multiple simulated nodes, we observed how data remains safe and accessible even in case of individual node failures.

This experiment enhanced our understanding of **data reliability**, **fault tolerance**, and **redundancy mechanisms** used in real-world cloud environments such as Hadoop HDFS, AWS S3, and Google File System.

Through this simulation, we gained hands-on experience with one of the most fundamental principles of **cloud computing infrastructure — data replication for high availability and durability**