# Cloud Based Object Storage (Cloud Computing Project)

**Cloud Computing**
**CS/SSG527**
*By*

**Debraj Dhar(2022H1120269P)**
**Ramananda Samantaray(2022H1120266P)**
**Srijon Mallick(2022H1120279P)**
**Sri Charan Ponnada (2022H1120274P)**

*Under the supervision of*

**Dr. Hari Babu**

**Assistant Professor**
**Department of Computer Science and Information Systems**

**Birla Institute of Technology and Science**

**Pilani, Pilani Campus, Rajasthan (India)**

# Contents

# Introduction

### Problem Statement

A cloud based object-based storage system is to be implemented.

## Requirements:

- Data (set of objects) is partitioned and stored using consistent hashing on a ring topology of 2^64 id space. Number of nodes (Containers/VMs) should be at least 3 and each representing 500 virtual nodes.
- Each object is replicated in N nodes (N is configurable). Consistency among replicas is achieved by using vector clocks, read-write quorums, and rea-repair and anti-entropy process.
- Every t seconds (t is configurable), the system should check for hotspots and migrate VM/container using volume-to-size ratio.
- REST API for users to be able to create/delete objects.
- The API should return the following in a JSON object: Success or failure, Vector clocks of all replicas, Node number where the write took place (in case of write).

# Overview

Object-based storage systems are a type of storage architecture used in cloud computing and other large-scale data storage scenarios. Unlike traditional file systems that organize data in a hierarchical structure with directories and files, object-based storage systems organize data as discrete objects. Each object contains data, metadata, and a unique identifier, allowing for efficient and scalable data management.

# Design Overview:

To implement object storage systems we have used 4 physical systems here having IP addresses: IP1,IP2,IP3,IP4. As of now, we are running "REST-WEB-SERVER" on one system (Node B-IP2) (Any node can be used as a REST server). We have used the "MD5" hashing algorithm. Now servers's IP addresses are kept on REST Server where a consistent hash is implemented as key-value pairs using MD5 hashing algorithm. Now as per consistent hashing concept, the object data will be distributed among the nodes based on their MD5 hash. For example, if the hashing algorithm is

"MOD-64", a list like the following will be created.

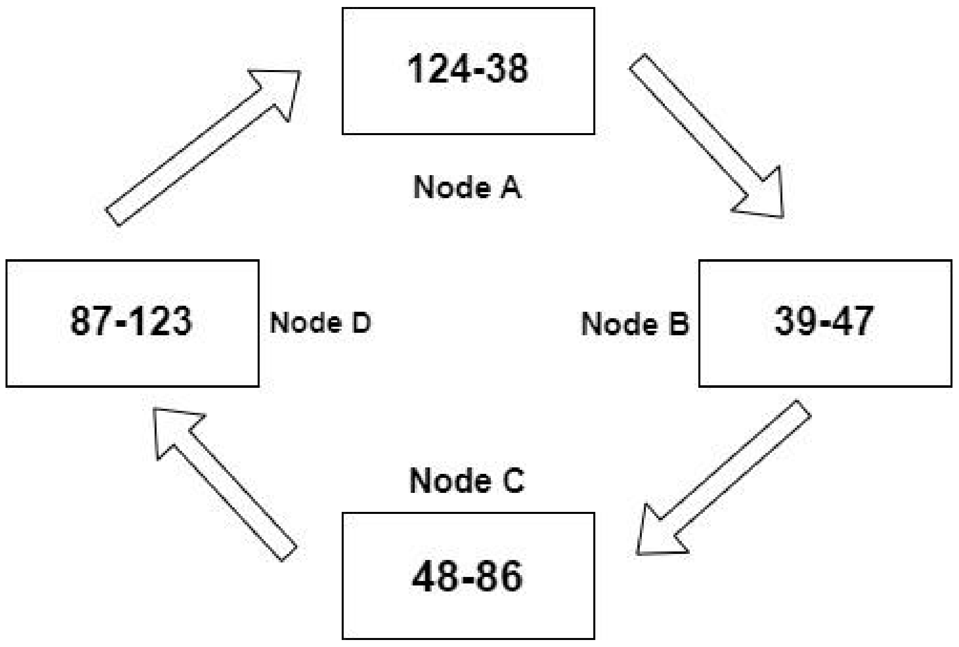
| Key | Node |
|---|---|
| 124-38 | IP1 |
| 39-47 | IP2 (REST Server) |
| 48-86 | IP3 |
| 87-123 | IP4 |

The list is sorted so that further process of creating/deleting any file will be easier.

## Design Considerations

| | |
|---|---|
| Number of nodes in the system | 4 |
| Replication factor | 3 |
| Number of virtual nodes | 500 |

## Architectural Design



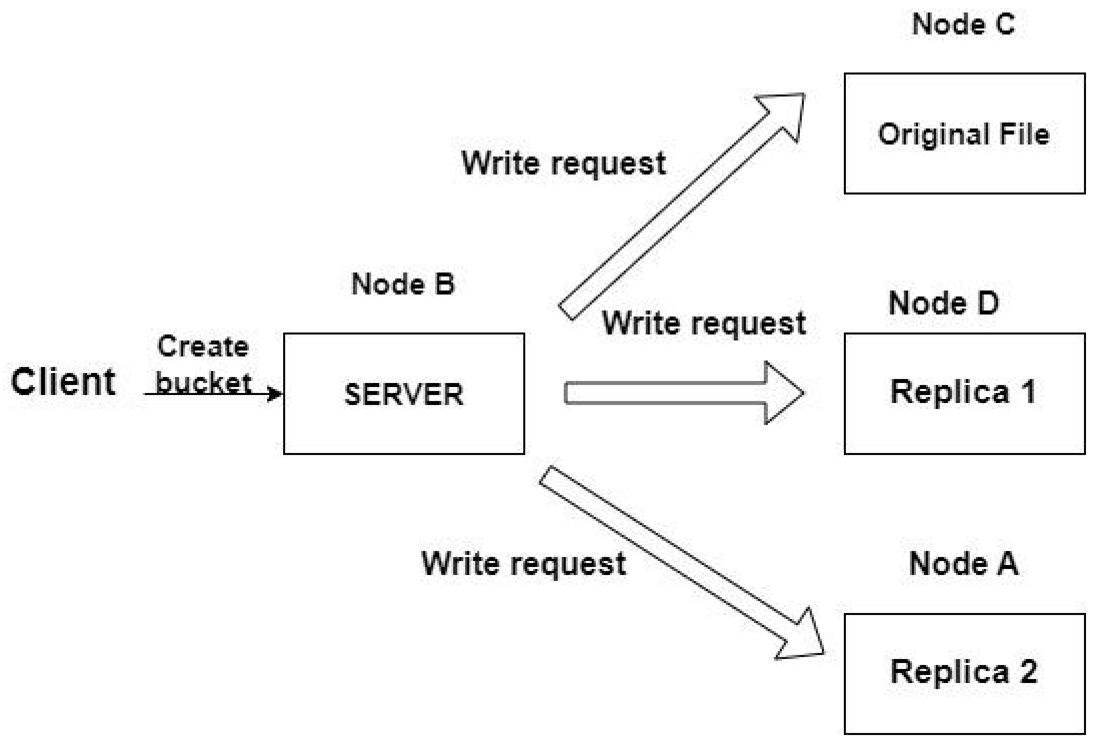
# Operations:

## Creating Bucket:

As of now we will assume that "REST-WEB-SERVER" is running on Node B. We have set the value of "N" as 3, meaning that when bucket/file is created on node A, its replicas will be maintained on node B and node C. Let us say, a request to create "bucket1" has come to "REST-WEB-SERVER". Now hash will be calculated as per the name of the bucket. Let us say this hash value comes out to be 50. Now a preference list will be fetched on the REST SERVER which will comprise of the following nodes:-

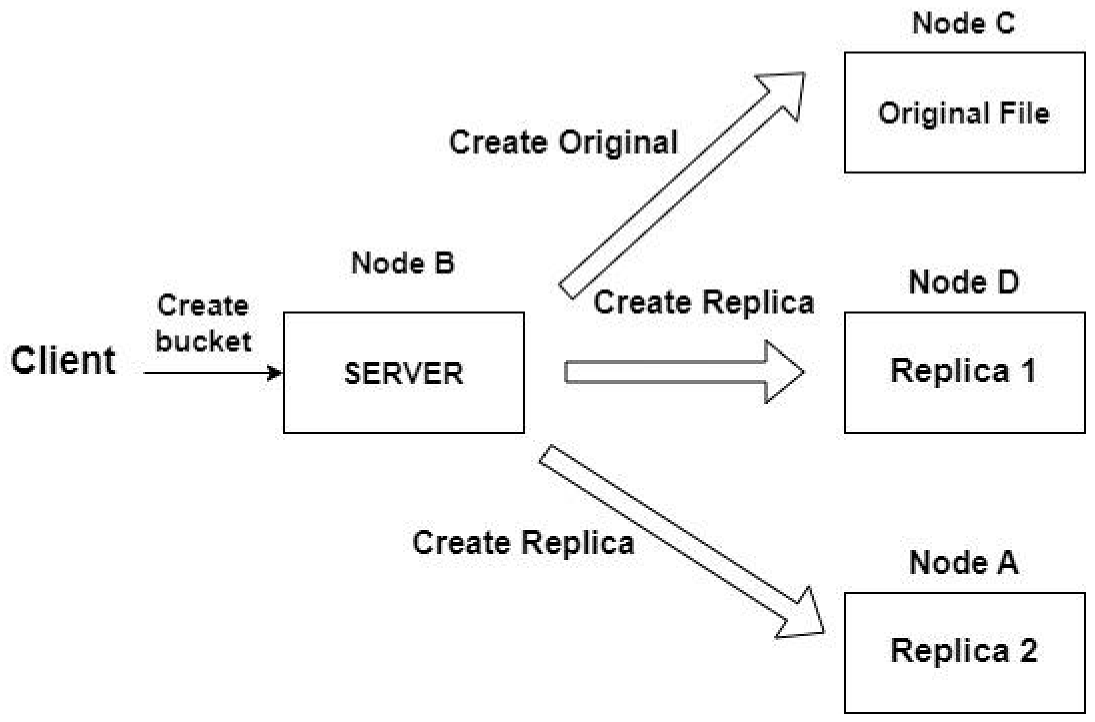
1. IP3 - Node C
2. IP4 - Node D
3. IP1 - Node A

Now a folder named "bucket1" will be created on Node C. And as N=3, it will be replicated to other nodes of preference list as well which are Node D and Node A.



## Creating/Updating the file:

Let us say, the user wants to upload the file named as “file1” in “bucket1”. Now this request comes to REST SERVER. It will create the file replica on all nodes in the preference list which are IP3(Node C), IP4(Node D) and IP1(Node A). We are handling communication between nodes using socket programming. As we have set the value of “W” as 2 , once any two nodes reply as positive, it will send positive acknowledgement to the client.



## Handling vector clock:

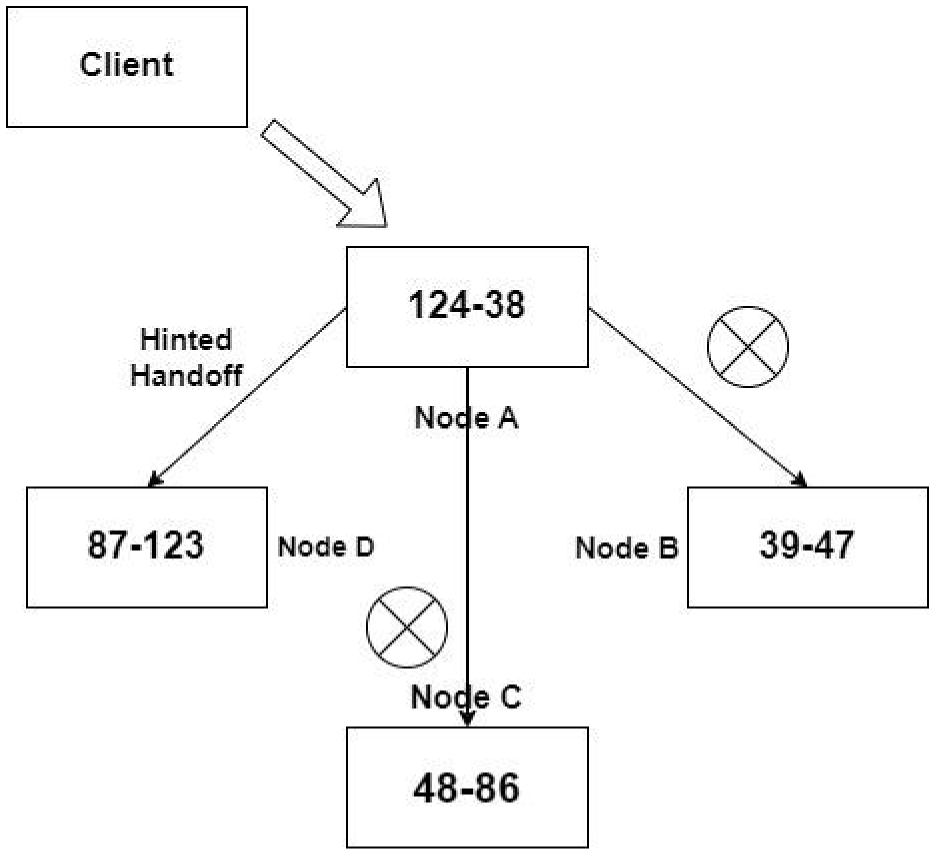
For every file we are keeping one metadata file. In this metadata file we are keeping the version number of the data which is nothing but an integer. Now when a client updates any file, we are returning a version number which is nothing but a vector clock. This is how we have implemented vector clock mechanism .

## Reconciliation / Read repair

Now if a client receives multiple version clocks with respect to a single update, the latest version clock will be considered at client’s end and will also be updated in the system. This is how we are implementing reconciliation mechanism and read repair mechanism .

## Hinted hand-off and sloppy quorum:

Let us say, a write request comes to write a file on node IP1 (Node A). Now as two replicas are to be maintained, files should also be saved on IP2 (Node B) and IP3 (Node C). Note that circular order according to consistent hashing is maintained. If those two nodes are unavailable, we are saving the file on the node IP4 (Node D) along with the hint. Now whenever the unavailable nodes become available, a hint is sent to those nodes by node D. Now the recovered nodes will save the file as per told. This is how we have implemented hinted hand-off mechanism.



Suppose Node B and C fail then Node D is the next after preference list. It needs to receive the replica of the data. HInted Handoff- replica at D points to Node B and C. When B and C node is available the Node D sends back the replicated data back to Node B and C. Now actually it was not the responsibility of node D to save that file, but still the file got saved on node D. So in a way D acted as a backup server for a while. However it is actually part of our system only. So in this way we have implemented the sloppy quorum mechanism.

*Submitted by Sri Charan, Debraj, Srijon and Ramananda*