Effective Replica Management in Car Navigation System using Edge Cloud Environment

Lavish Gulati **Vakul Gupta**
Indian Institute of Technology Guwahati
{gulat170123030, vakul170101076}@iitg.ac.in

Abstract

Car navigation system is nowadays based on centralized cloud architectures. However, edge cloud computing provides robust computational and storage resources improving response times, system scalability and data reliability. The multi-replica strategy used in edge cloud computing architecture can create multiple data replicas and store them in different edge nodes, which improves data availability and data service quality. Due to time-varying user demand, the number of data replicas need to be dynamically adjusted. Load balancing is also required while placing the newly added replicas. We also consider the problem of data replica synchronization and data recovery in case of failures. We propose an optimization problem based on the performance of the edge cloud computing architecture in car navigation system, and improve upon the existing replication algorithms.

1 Introduction

A car navigation system (1) uses a satellite navigation device to collect various information such as origin-destination stations, occupancy of vehicles, pedestrian activity trends, and more. The on fly traffic information collected from various such navigation systems can be used to plan the optimal path to some destination and reduce traffic congestion.

Edge computing is a distributed computing paradigm that brings computation and data storage closer to the location where it is needed (in our case the car users), to improve response times and save bandwidth.

The architecture of the car navigation system is divided into three layers: centralized cloud layer, edge cloud layer and client layer. The client layer includes the cars equipped with navigation devices and an application interface. The edge cloud layer includes the computing nodes and data centers located near the clients. The centralized cloud layer includes the central cloud computing server and the nodes monitor.

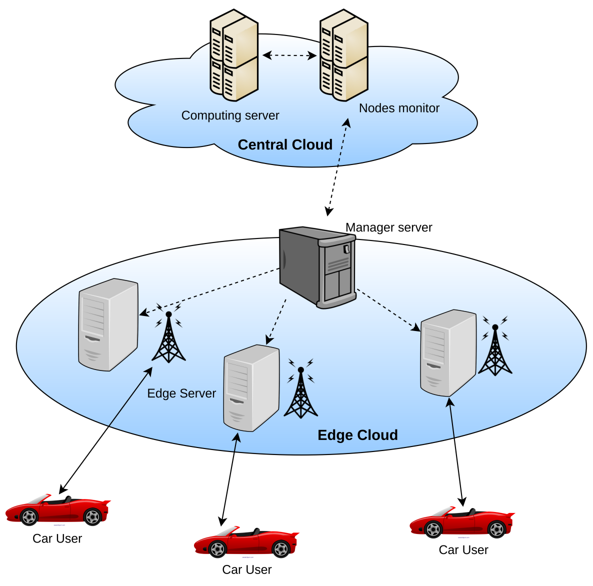

Figure 1: Navigation system in edge cloud architecture

Edge computing devices provide computational and storage resources to the clients. The data is first transmitted to the edge node for local processing, which reduces network traffic and response times (2; 3). Since the computing nodes are located over a large geographical region, failures or outages might interrupt the service to the clients which can be solved by data replication approach (4). Data replication is the process in which the data is copied at multiple locations (different edge servers) to improve data availability and data service quality.

The number of data replicas needs to be dynamically adjusted because of the time-varying data and computation requests. The continuously changing number of replicas leads to the requirement of placing the newly added replicas on edge nodes. Replica placement (5) involves identifying the best possible node to duplicate data based on network latency and user request.

We also need to ensure data synchronization between two or more edge nodes and update changes automatically between them to maintain consistency. This assures congruence between each source of data and its different endpoints. So, in order to ensure accurate, secure, compliant data and successful end-user experience, data synchronization is required among different edge nodes.

To protect volume data from unrecoverable failure, we need to replicate a volume to a replica node. Data loss occurs due to crashing, correlated failure, logical failure, power outages and security threats. We can recover the data from the replica node once the issue has been rectified. In such scenario, it is important to resume data availability as soon as possible to prevent or limit application downtime.

The rest of the paper is organized as follows: The motivation to use edge cloud computing is discussed in Section 2. The related work is discussed in Section 3 which comprises of a dynamic replica creation strategy (3.1), a replica placement strategy (3.2), a replica synchronization strategy based on replica delayed update (3.3) and a replica recovery strategy based on load-balancing (3.4). The objective and problem formulation are discussed in Sections 4 and 5 respectively. The proposed implementation is discussed in Section 6.

2 Motivation

Traditional cloud-based architectures impose several limitations which edge cloud architecture is able to resolve. The advantages offered by edge computing over traditional forms of cloud architectures are summarized below.

1. **Speed** Since edge computing devices collect and process data locally in nearby edge nodes, there is no need for the data to be routed to distant centralized servers, thus reducing network latency and response times, and increasing performance of the network.

2. **Security** Traditional cloud architectures are vulnerable to distributed denial of service (DDoS) attacks and power failures because they are inherently centralized. On the other hand, edge computing distributes applications, storage, and processing over a large geographical region and wide range of computing nodes, which makes it secure against such attacks and failures.

3. **Scalability** The development of edge computing based architecture have made it easier for organisations to scale their operations. Small edge computing nodes are distributed and located nearer to the clients as compared to large centralized data centers of traditional cloud architectures, and hence the same computational and storage capabilities can be scaled more effectively. By offering a low cost route to scalability, it further makes it easy for the organisations to expand their computing capacities.

4. **Reliability** Since edge computing nodes are located closer to the clients, a network failure in a distant location will not affect the performance of the architecture in other areas. Even in case of a data center failure, edge nodes will continue to operate effectively because they process vital functions natively.

3 Related Work

3.1 Dynamic replica creation strategy

The proposed replica creation strategy takes into account two factors: dynamic and static. The dynamic factor is calculated based on data block heat and data access response time, and the static factor is calculated based on data block size and file size. We consider three time periods for the access frequency of data blocks: past, present and future while calculating the data block heat. Given past and present information, we need to predict the access frequency of a data block in the future which can be modeled by a Markov chain (6). Finally, we use the dynamic and static factors to predict the replica creation factor, which basically gives the optimal number of replicas of a data block.

Hence, the proposed strategy calculates the dynamic replica creation factor by predicting the access frequency of a data block in the near future. However, this might create a computational bottleneck for the node manager if we consider a large number of samples for the prediction which also leads to increase in the complexity of the transaction log and finally affecting the availability of the file system. So we need to take only a limited number of data samples for the prediction.

3.1.1 Dynamic replica creation strategy model

Calculation of dynamic factor

The dynamic factor is calculated based on data block heat and data access response time. Let us suppose that at the beginning of the time period t, the access frequency of the data block d is $f_{d,t}$ and the number of access for the ith replica of data block d is $nc^i_{d,t}$. Let the number of replicas of data block d be n, then, the access frequency of data block d can be expressed as

$$f_{d,t} = \sum_{i=1}^{n} nc^i_{d,t} \tag{1}$$

Now, we need to model the original data sequence of access frequency values of data block d over n time intervals during time period t, for the prediction of access frequency in the future. This is given as follows

$$f_{d,t} = (f_{d,t}(1), f_{d,t}(2), \ldots, f_{d,t}(n)) \tag{2}$$

Let the average access frequency of data block d at time period t be indicated by $h_{d,t}$, which is given as

$$h_{d,t} = \frac{\sum_{k=1}^{n} f_{d,t}(k)}{n} \tag{3}$$

where $f_{d,t}(k)$ denotes the access frequency value of data block d of the kth time interval at time period t. The heat of data block d in time period t is given as

$$heat_{d,t} = h_{d,t} + h_{d,t-1} + ph_{d,t+1} \tag{4}$$

where $h_{d,t-1}$ denotes the average access frequency of data block d at time period $t-1$, and $ph_{d,t+1}$ denotes the predicted value of the access frequency of data block d at time period $t+1$. Let $RT_{d,t}(k)$ indicate the access response time of data block d of the kth time interval at time period t. Then, the average response time of data block d at the time period t is given as

$$ART_{d,t} = \frac{\sum_{k=1}^{n} RT_{d,t}(k)}{n} \tag{5}$$

We define the dynamic factor $DF_{d,t}$ of data block d at time period t as

$$DF_{d,t} = heat_{d,t} \times ART_{d,t} \tag{6}$$

This is because the heat and response time of data blocks change continuously with time.

Calculation of static factor
The static factor is calculated based on data block size and file size, because the file size and the size of the data block fragment remain constant over time. Let the file size of the data block d be F_d and the file system fragment data block size be S_0.

$$Q_d = \begin{cases} \lceil F_d/S_0 \rceil & , Q_d \% S_0 = 0 \\ \lceil F_d/S_0 \rceil + 1 & , Q_d \% S_0 \neq 0 \end{cases} \tag{7}$$

Calculation of dynamic replica creation factor
Therefore, the dynamic replica creation factor of data block d is given as follows

$$\begin{aligned} &H_{d,0} = 0 \\ &H_{d,t} = [\alpha \times H_{d,t-1}(f_j)] + \\ &\quad [(1-\alpha) \times DF_{d,t} \times Q_d^{-1}] \end{aligned} \tag{8}$$

where α denotes the impact factor for data block heat, and is given by

$$\alpha = \Delta T_t / (\Delta T_t + \Delta T_{t-1}) \tag{9}$$

Here, ΔT_t is the time difference from the starting time to the time period t. The dynamic replica creation factor $H_{d,t}$ gives the optimal number of replicas of data block d at the time period t.

3.2 Replica placement strategy

The replica placement strategy is used when the number of replicas of the data block needs to be increased. In that scenario, while ensuring the load balancing of the file system, we need to place the newly added replicas on appropriate edge nodes. To judge the performance, a multi-objective replica placement problem is formulated which is affected by the following 3 parameters, i.e., the file system cluster storage load, the edge node's performance, and the network distance.

3.2.1 Load balancing related sub-objective function

Edge node load performance
The edge node performance mainly depends on the following 4 parameters, i.e., the CPU processing capacity, memory load capacity, disk load capacity and disk read/write performance. The CPU processing capacity of node j can be expressed as

$$C_{j,cpu} = fr_j \times nc_j \times (1 - uf_j) \tag{10}$$

where fr_j denotes the main frequency of CPU of the node j, nc_j denotes the number of CPU cores, and uf_j denotes the usage of CPU. Also, during the period of data intensive operations, the disk read/write performance considerably affects the execution time of the task. Hence, the disk read/write performance is an essential performance factor, which can be expressed as

$$C_{j,w/r} = S_{j,r} \times a + S_{j,w} \times (1 - a) \tag{11}$$

where $S_{j,r}$ denotes the maximum number of bytes that the hard disk can read per second (read speed). $S_{j,w}$ denotes the write speed of the disk, and $\alpha \in (0, 1)$ denotes the weight parameter. The load capacity of the memory can be expressed as

$$C_{j,mem} = ms_j \times (1 - mc_j) \tag{12}$$

where ms_j denotes the size of the memory, and mc_j denotes the memory usage. The load capacity of the disk space can be expressed as

$$C_{j,ds} = D_{j,\text{size}} - D_{j,usage} \tag{13}$$

where $D_{j,size}$ denotes the size of the disk, and $D_{j,usage}$ denotes the used capacity of the disk. Taking the 4 performance parameters i.e., the CPU processing capacity, read and write speed of the disk, load capacity of the memory and load capacity of disk space, the sub-objective function of edge node j can be expressed as

$$\begin{aligned} \max \quad s_1 = C_{j,cpu} + C_{j,w/r} \\ + C_{j,mem} + C_{j,ds} \end{aligned} \tag{14}$$

File system cluster storage load

Let SS_j denote the total size of storage space for edge node j. Assume that the size of each data block is equal, the usage of storage space of edge node j can be expressed as

$$\beta_j = \frac{F_j \times B_{\text{size}}}{SS_j} \tag{15}$$

where B_{size} denotes the uniform size of the data block in the file system and F_j denotes the number of data blocks stored in edge node j. The average storage usage of the file system cluster can be expressed as

$$\bar{\beta} = \frac{\sum_{j=1}^{N^{\text{node}}} SS_j + \sum_{i=1}^{N^{\text{block}}} k_i}{\sum_{j=1}^{N^{\text{node}}} SS_j} \tag{16}$$

where N^{block} denotes the number of newly added replicas of all data blocks, N^{node} denotes the number of edge nodes. k_i denotes the number of data block i that already exists in the cluster. The sub-objective function of file system cluster storage load can be expressed as

$$\max \quad s_2 = \left(\frac{\sum_{j=1}^{N^{\text{node}}} (\beta_j - \bar{\beta})}{N^{\text{node}}} \right)^{-1} \tag{17}$$

Network distance

We use a tree topology to describe the network distance in the edge cloud architecture, by labelling edge nodes as leaf nodes, and network devices as internal nodes. In the network topology, the distance between any child node and its corresponding parent node is 1, and the distance between any two edge nodes can be calculated by summing their corresponding distances to their lowest common ancestor (LCA).

Let the maximum distance between a pair of nodes be nd_{max} and the distance between the source node and the target node j is nd_j. Then the network distance coefficient can be expressed as

$$D_j = \frac{nd_j}{nd_{\max}} \tag{18}$$

The network distance coefficient is directly proportional to the network distance between the edge nodes. Because as we decrease the distance between the edge nodes, the efficiency of the data transmission increases. Therefore, the network distance sub-objective function between edge nodes can be expressed as

$$\max \quad s_3 = (D_j \times S_0 \times C_{tr})^{-1} \tag{19}$$

where S_0 denotes data block size, C_{tr} denotes transmission cost per bit of data per unit time.

3.2.2 Multi-objective optimized replica placement problem

Taking into consideration all the 3 parameters, i.e., the edge node performance, the file system cluster storage load and the network distance, the multi-objective optimization problem of the replica placement problem can be expressed as

$$\begin{aligned} \max S(d) = [s_1(d), s_2(d), s_3(d)] \\ \text{s.t. } d \in X \end{aligned} \tag{20}$$

where, $s_1(d)$, $s_2(d)$ and $s_3(d)$ are the sub-objective functions corresponding to 14, 17 and 19 respectively. Here, X represents the set of solutions that satisfy the constraints, and d denotes a feasible solution that satisfies the constraints.

3.3 Replica synchronization strategy based on replica delayed update

In Hadoop Distributed File System (HDFS), written data can only be accessed after all replicas of the data are synchronized (7). This synchronization methodology leads to good read performance, strong consistency, and user friendliness, but at the same time has disadvantages such as low write throughput and high data write latency because all replicas need to be synchronized before a new write.

However, in an edge cloud architecture, the edge nodes may be spread over a large geographical region such as a province, state or even a country. In such scenarios, network congestion and insufficient bandwidth may cause synchronization delays, thus rendering the HDFS replica synchronization methodology ineffective. This will eventually cause failure in synchronization of the data replicas, resulting in low write throughput and higher write latency.

Therefore, a Replica Synchronization strategy which is adaptive to delays is illustrated, as explained in the following sections.

3.3.1 Selective data replica synchronization

Suppose there occurs a data block write request from a client and the number of redundant replicas of the requested data block d is N. W replicas are synchronized before the success indicator is returned to the client. Here, W is expressed as follows

$$W = (N/2) + 1 \tag{21}$$

This strategy selects W edge nodes holding the replica of the requested data block which are closest to the client in terms of network distance. These W edge nodes are synchronized, thus forming a data transmission channel for replica synchronization. The distance between the edge node and the client can be calculated by summing their corresponding distances to their lowest common ancestor in their network topology. The W edge nodes are then sorted in order of increasing distance from the client to the node, and form a transmission channel to synchronize the replicas of the written data.

3.3.2 Replica status table

Following the aforementioned strategy, we find that $N - W$ replicas are still not synchronized since we formed a transmission channel comprising of only W replicas. For these unsynchronized replicas, we use a data structure called Replica Status Table (RST). The RST stores the synchronization status of the data block replicas, which helps in identifying unsynchronized replicas and eventually facilitating the synchronization of such replicas. Each data block replica maintains an RST, and each

RST of data block A	
Location	Old
Edge Node 1	0
Edge Node 2	0
Edge Node 3	1

Table 1: Sample Replica Status Table

RST has two entries: RST.Location and RST.Old. RST.Location depicts the edge node that stores the data block replica, and RST.Old depicts whether the replica has been updated. RST.Old = 1 means that the replica has not been synchronized, and RST.Old = 0 means that the replica has been synchronized. The RST of the replicas of data block A is shown in Table 1.

3.3.3 Delay-adaptive synchronization

For the $N - W$ unsynchronized replicas, the synchronization update is not triggered immediately but delayed to subsequent time periods. The synchronization updates of such replicas will be triggered in the following two cases:

1. The data block that has not been synchronized becomes the hot data.

2. The load performance of the edge node where the replica has not been synchronized is strong.

Case 1

To check whether a data block becomes hot, we impose the condition that the access frequency of the data block should be greater than the average access frequency of all data blocks, indicating that the data block has higher demand than the remaining data blocks.

Let us take a replica a of data block A. Based on the proposed strategy, the replica a is not synchronized, but only the RST.Old is set to 1.

Hence, we calculate the access frequency $h_{A,t}$ of data block A at time period t according to equation 3. Assuming there are N data blocks present, the average access frequency at time period t is expressed as

$$hah = \frac{\sum_{k=1}^{N} h_{k,t}}{N} \tag{22}$$

If the access frequency of data block A at time period t is greater than the average access frequency, i.e., $h_{A,t} > hah$, then data block A is defined as hot, and its replica a will be synchronized, and the RST.Old of replica a is set to 0.

Case 2

Similarly, to check whether an edge node has strong load performance, we impose the condition that the load capacity of the edge node should be greater than the average load capacity of all edge nodes.

Let us take edge node i, which stores the replica of data that is not synchronized. We define the load capacity $C_{i,t}$ of the edge node i at time period t based on equation 13. Assuming that the total number of edge nodes is J, then the average load capacity of all edge nodes at time period t can be expressed as

$$alc = \frac{\sum_{i=1}^{J} C_{i,t}}{J} \tag{23}$$

If $C_{i,t} > alc$, then the load capacity of edge node i is strong, and thus, the unsynchronized replicas on this edge node are synchronized, and RST.Old of these replicas is set to 0.

3.4 Replica recovery strategy based on load-balancing

In the edge cloud computing architecture, node failure in the system might occur due to a number of reasons. In that scenario, the data stored on the failed node will be lost. But since, multiple replicas of the same data are distributed across the file system, this gives the opportunity to the other nodes to continue providing data access to the failed edge node. The failure of edge nodes contributes to an increase in the access of other non-failed edge nodes, which will increase their load, resulting in the load imbalance of the entire system. In order to avoid this, the data replicas on the failed edge node should be restored to the non-failed edge nodes with a strong load capacity in the system. To solve the above scenario, we propose a Load-Balancing based Replica Recovery strategy.

3.4.1 Failure edge node detection

We use HDFS as the edge cloud file system. In HDFS, an edge node periodically generates heartbeat packets according to its node status and storage status, and sends the heartbeat packets to a specialized edge node denoted by CollectorNode at a certain interval, so as to let CollectorNode know its running state. In case of node failure, it would not be able to generate the heartbeat packet. As a result, the CollectorNode would not receive the heartbeat packet from the edge node within a certain period of time. The CollectorNode would thus establish the failure of the corresponding edge node.

3.4.2 Selection of data blocks to be recovered

Assume data block d is a data block stored on a failed edge node. Let $ap_{d,t}$ denote the access probability of data block d at time period t, then $ap_{d,t}$ can be expressed as

$$ap_{d,t} = \frac{co_{d,t}}{sco_t} \tag{24}$$

where $co_{d,t}$ denotes the count of access of data block d at time period t; sco_t denotes the total count of accesses for all data blocks at time period t.

Considering the block size, block replica location time, and bandwidth, the block recovery costs of data block d can be expressed as

$$\text{cost}_d = \frac{\text{size}_d + \text{Tseek}_{\ d}}{BW_d} \tag{25}$$

where $size_d$ depicts data block size, and $Tseek_d$ depicts the time required to locate other replicas of the data block d. BW_d represents the maximum available network bandwidth between the target node and other edge node that store the replica of the data block d. Based on this, the replica selection factor for the data block d can be expressed as

$$\text{Sel}_d = \frac{ap_{d,t}}{\text{cost}_d} \tag{26}$$

If the number of data replicas on the failed edge node is N, then the average value of replica selection factor for all data block in the failed edge node can be expressed as

$$ASel = \frac{\sum_{n=1}^{N} Sel_n}{N} \tag{27}$$

If $Sel_d > ASel$, then data block d is chosen as the data block that needs to be recovered.

3.4.3 Selection of the target node

In HDFS systems, load balancing between nodes is mainly affected by 3 parameters, i.e., the disk space of the edge node, the CPU capacity, and the cache capacity. Hence, to ensure load balancing, we consider these factors while selecting the target edge node for replica recovery.

The greater the remaining amount of node disk space, the stronger the node disk space load capacity. The disk space availability of edge node can be expressed as

$$P_{\text{disk}} = 1 - U_{\text{disk}} \tag{28}$$

where U_{disk} denotes the using rate for the node disk. The CPU load capacity of the edge node can be expressed as

$$P_{CPU} = f_{CPU} \times C_{cpu} \times (1 - U_{cpu}) \times V_{cpu} \tag{29}$$

where f_{CPU} denotes the frequency of the CPU of the edge node, C_{cpu} denotes the number of cores of the CPU, $(1 - U_{cpu})$ denotes the remaining usage rate of the CPU, V_{cpu} denotes the bus speed of the CPU. The frequently accessed data blocks can be put into memory to increase data access speed. Hence, the capacity of the cache also plays a major role in load balancing, which can be expressed as

$$P_{\text{cache}} = \frac{L_{\text{cache}}}{S_{\text{cache}}} \tag{30}$$

where L_{cache} denotes cache remaining capacity, S_{cache} denotes the time required to access data.

Considering all the 3 parameters mentioned above which affects load balancing, the overall load capacity of the edge node can be expressed as

$$\begin{aligned} CP = (A_1 \times P_{disk}) + (A_2 \times P_{CPU}) \\ + (A_3 \times P_{\text{cache}}) \end{aligned} \tag{31}$$

where the A_1, A_2 and A_3 are the weight coefficients for disk space availability, CPU load capacity, cache capacity, and they satisfy $\sum A_i = 1$.

For the recovery of data block d, we need to find an optimal target edge node which does not contain a replica of data block d. The edge node with the largest CP value satisfying the above condition is chosen to be the optimal target edge node.

4 Objective

To study the performance of edge cloud computing architecture using effective replica management in car navigation system based on different parameters like response time and load balancing capabilities, and in the process maintaining data consistency using synchronization and the capability of the system to recover from failures.

5 Problem Formulation

Suppose we have a edge cloud architecture A with m central nodes given by $\{C_1, C_2, \ldots, C_m\}$ and n edge nodes given by $\{E_1, E_2, \ldots, E_n\}$. The car navigation system data contains information about road network and traffic intensity of different localities. The network is represented as a directed graph where each edge denotes a unidirectional road and weight of the edge denotes the traffic intensity on that road.

The data of each locality is considered as a single data block. Thus, we get a set of d data blocks containing the entire information for all the localities denoted by $(D_1, D_2, \ldots, D_d)$, where D_i represents the data block of the ith locality. The car user queries for the optimal path between the current location and the desired destination in a specific locality via a client-end application interface. The query for the specific locality is received by the nearest load balancer, which is then re-routed to one of the edge nodes containing data block of that locality. The edge node solves the shortest path query using Dijkstra's algorithm (10).

The dynamic replica creation strategy finds the optimal number of replicas $H_{i,t}$ of the ith data block at time period t.

For replica placement, we assign a binary encoding B_j (Fig. 2) to each edge node E_j whose length is same as the number of data blocks d and the ith bit in the encoding takes a value of 1 or 0, which means whether a replica of ith data block is stored in the edge node or not. The set of n such encodings for all the edge nodes determine where the replicas are placed in the edge cloud system.

The encodings should optimize the multi-objective replica placement performance indicator function $S(d)$ as proposed in 20 under the constraint that the optimal number of replicas of ith data block is $H_{i,t}$ which can be modelled as

$$\begin{gathered} \sum_{j=1}^{n} B_{j,1} = H_{1,t} \\ \vdots \\ \sum_{j=1}^{n} B_{j,i} = H_{i,t} \\ \vdots \\ \sum_{j=1}^{n} B_{j,d} = H_{d,t} \end{gathered} \tag{32}$$

where $B_{j,i}$ represents the ith bit of the binary encoding B_j.

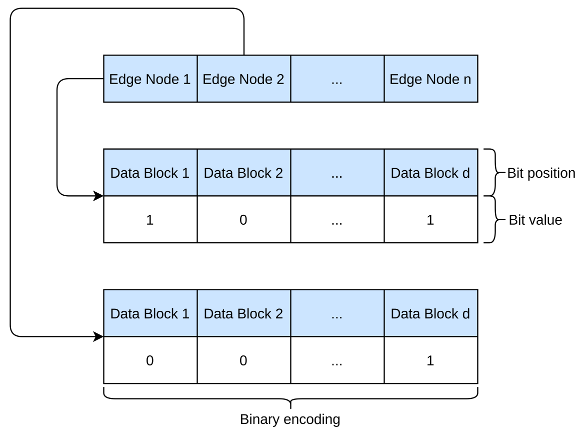

Figure 2: Binary Encoding Scheme

6 Proposed Direction

We will use Docker containers (8) to simulate each application interface or computing node. In total, we will have five different types of Docker containers namely car user application interface, load balancer node, edge server node, manager server node, and centralized server node. We will use socket programming (11) in Python for inter-container communication.

The car users send optimal path queries to the load balancer which re-routes them to edge node containers with information about the specified locality. The edge node containers solve the query and return the optimal path to the car users.

The manager node will collect information about the processing load, file system load and response time from all the edge node containers and send it to the centralized server. The centralized server algorithmically adjusts the placement of replicas in the architecture to improve performance.

We will synchronize only W replicas of ith data block as proposed in 21 before returning the result to the client. The remaining $n - W$ replicas are synchronized with time once the conditions in 3.3.3 are achieved. The performance of synchronization is measured by (i) the time that it takes to write the data on W synchronized replicas and (ii) the time that it takes to fetch the updated data from $n - W$ unsynchronized replicas.

In case of node failure consisting of ith data block meeting condition 27, we will select the edge node with the highest CP value (31) for replacement out of all the edge nodes which do not store a replica of the ith data block. For measuring the performance of data recovery, we need to introduce failures in the architecture. For this, we model a probability distribution for each edge node over a certain period of time. The probability $p_{j,t}$ denotes the probability of failure of edge node j at time period t. We will measure the performance by the average time taken for a failed edge node to recover, i.e. to copy all the data blocks in the edge node to the target node.

References

[1] B. Qi, L. Kang, S. Banerjee, A vehicle-based edge computing platform for transit and human mobility analytics, in: ACM/IEEE Symposium, ACM, 2017, pp. 1–14.

[2] Nasir Abbas, Yan Zhang, et al., Mobile edge computing: A survey, IEEE Internet Things J. 5 (1) (2018) 450–465.

[3] Rodrigo Romana, Javier Lopez, et al., Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges, Future Gener. Comp. Syst. 78 (2) (2018) 680–698.

[4] W.C. Chang, P.C. Wang, An adaptable replication scheme in mobile online system for mobile-edge cloud computing, in: IEEE, 2017, pp. 109–114.

[5] R. Kingsy Grace, R. Manimegalai, et al., Dynamic replica placement and selection strategies in data grids — A comprehensive survey

[6] Charles J. Geyer, Practical Markov chain Monte Carlo, Statist. Sci. 7 (4) (1992) 473–483.

[7] R. Chandakanna, Veerabhadra, REHDFS: A random read/write enhanced HDFS, J. Netw. Comput. Appl. 103 (2018) 85–100.

[8] Bashari Rad, Babak Bhatti, Harrison Ahmadi, Mohammad. (2017). An Introduction to Docker and Analysis of its Performance. IJCSNS International Journal of Computer Science and Network Security. 173. 8.

[9] Chunlin Li, Mingyang Songa, Min Zhang, Youlong Luo: Effective replica management for improving reliability and availability in edge-cloud computing environment, Journal of Parallel and Distributed Computing, Volume 143, September 2020, Pages 107-128

[10] Javaid, Adeel. (2013). Understanding Dijkstra Algorithm. SSRN Electronic Journal. 10.2139/ssrn.2340905.

[11] Maata, Rolou Lyn Cordova, Ronald Sudramurthy, Balaji Halibas, Alrence. (2017). Design and Implementation of Client-Server Based Application Using Socket Programming in a Distributed Computing Environment. 10.1109/ICCIC.2017.8524573.