|  |  |
| --- | --- |
| **Name** | Mayur Solankar, Manish Jadhav, Vishesh Savani, Shreyansh Salvi |
| **UID** | 2023301018, 2023301005, 2022300100, 2022300091 |
| **Subject** | Distributed Computing |
| **Experiment No.** | 5 |
| **Project title** | Social Media System |
| **Problem Statement** | To implement data consistency and replication in social media platform. |
| **Objectives** | To ensure seamless user experience, maintain real-time interactions and data accuracy across the social media platform using balanced data consistency models and efficient replication strategies. |
| **Theory** | **What is replication?**  The first thing is that it makes our system more stable because of node replication. It is good to have replicas of a node in a network due to following reasons:   * If a node stops working, the distributed network will still work fine due to its replicas which will be there. Thus it increases the fault tolerance of the system. * It also helps in load sharing where loads on a server are shared among different replicas. * It enhances the availability of the data. If the replicas are created and data is stored near to the consumers, it would be easier and faster to fetch data.   **Types of Replication**   * Active Replication * Passive Replication   **Active Replication:**  The request of the client goes to all the replicas.  It is to be made sure that every replica receives the client request in the same order else the system will get inconsistent.  There is no need for coordination because each copy processes the same request in the same sequence.  All replicas respond to the client’s request.  ***Advantages:***   1. It is really simple. The codes in active replication are the same throughout. 2. It is transparent. 3. Even if a node fails, it will be easily handled by replicas of that node.   ***Disadvantages:***   1. It increases resource consumption. The greater the number of replicas, the greater the memory needed. 2. It increases the time complexity. If some change is done on one replica it should also be done in all others.   **Passive Replication:**  The client request goes to the primary replica, also called the main replica.  There are more replicas that act as backup for the primary replica. Primary replica informs all other backup replicas about any modification done.  The response is returned to the client by a primary replica.  Periodically primary replica sends some signal to backup replicas to let them know that it is working perfectly fine.  In case of failure of a primary replica, a backup replica becomes the primary replica.  ***Advantages:***   * The resource consumption is less as backup servers only come into play when the primary server fails. * The time complexity of this is also less as there’s no need for updating in all the nodes replicas, unlike active replication.   ***Disadvantages:***   * If some failure occurs, the response time is delayed.     **Consistency in Social Media Platform**  The choice of a consistency model for a social media platform depends on various factors, including the nature of interactions, user expectations, and trade-offs between consistency, availability, and partition tolerance.   1. **Eventual Consistency:** Social media platforms often prioritize availability and responsiveness. Eventual consistency allows for faster updates and responsiveness to users. It might result in temporary inconsistencies, like seeing different likes or comments on different devices for a short period. However, it ensures that eventually, all replicas converge to the same state. 2. **Causal Consistency:** This model maintains a causal relationship between related events. In a social media context, it ensures that if a user comments on a post, that comment will appear before any subsequent actions related to it. It strikes a balance between strong consistency and eventual consistency. 3. **Session Consistency:** This model guarantees consistency for a particular user session. Within a session, a user's interactions are consistent and ordered, providing a better user experience without sacrificing too much on performance.   Ultimately, social media platforms often prioritize user experience and responsiveness over strict consistency. They might choose a consistency model that allows for quick updates and interactions while ensuring that eventual consistency or a similar model converges all data to a consistent state over time. Balancing these factors is crucial to creating an engaging and reliable platform for users. |
| **Code:** | import threading  import socket  import pickle  import random  from datetime import datetime  class DistributedStorage:  def \_init\_(self, num\_nodes):  self.num\_nodes = num\_nodes  self.user\_profiles = {}  self.lock = threading.Lock()  self.node\_sockets = {}  self.quorum\_size = num\_nodes // 2 + 1  self.initialize\_sockets()  def initialize\_sockets(self):  for node\_id in range(1, self.num\_nodes + 1):  self.node\_sockets[node\_id] = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  self.node\_sockets[node\_id].bind(('127.0.0.1', 9000 + node\_id))  self.node\_sockets[node\_id].listen()  def put\_user\_profile(self, user\_id, profile):  with self.lock:  profile['created\_at'] = datetime.now().strftime("%Y-%m-%d %H:%M:%S")  self.user\_profiles[user\_id] = profile  print(f"User profile replicated: {user\_id} -> {profile}\n")  def get\_user\_profile(self, user\_id):  with self.lock:  return self.user\_profiles.get(user\_id, None)  def perform\_replication(self, user\_id, profile):  replication\_nodes = random.sample(range(1, self.num\_nodes + 1), self.quorum\_size)  successful\_replications = 0    for node in replication\_nodes:  print(f"Replicating user profile to Node {node}-----------")  if self.replicate\_user\_profile\_to\_node(node, user\_id, profile):  successful\_replications += 1  if successful\_replications >= self.quorum\_size:  print(".....QUORUM ACHIEVED!!!!!! User profile replicated successfully.......")  else:  print("Quorum not achieved. Rolling back the update.")  def replicate\_user\_profile\_to\_node(self, target\_node, user\_id, profile):  try:  with socket.create\_connection(('127.0.0.1', 9000 + target\_node)) as s:  data\_to\_send = {'user\_id': user\_id, 'profile': profile}  s.sendall(pickle.dumps(data\_to\_send))  print(f"Replicated user profile to Node {target\_node}: {user\_id} -> {profile}\n")  return True  except Exception as e:  print(f"Failed to replicate user profile to Node {target\_node}: {e}")  return False  def node\_listener(storage, node\_id):  while True:  connection, \_ = storage.node\_sockets[node\_id].accept()  data = connection.recv(1024)  if data:  received\_data = pickle.loads(data)  storage.put\_user\_profile(received\_data['user\_id'], received\_data['profile'])  num\_nodes = 3  distributed\_storage = DistributedStorage(num\_nodes)  for node\_id in range(1, num\_nodes + 1):  threading.Thread(target=node\_listener, args=(distributed\_storage, node\_id), daemon=True).start()  user\_id = "user" + str(random.randint(10000000, 99999999))  username = input("Enter username: ")  bio = input("Enter bio: ")  profile\_pic = input("Enter profile picture URL: ")  user\_profile = {"username": username, "bio": bio, "profile\_pic": profile\_pic}  distributed\_storage.put\_user\_profile(user\_id, user\_profile)  distributed\_storage.perform\_replication(user\_id, user\_profile)  random\_node = random.randint(1, distributed\_storage.num\_nodes)  retrieved\_profile = distributed\_storage.get\_user\_profile(user\_id)  print(f"\nRetrieved user profile from Node {random\_node}: {user\_id} -> {retrieved\_profile}")  print("=" \* 30) |
| **Output:** |  |
| **Conclusion:** | Hence by completing we came to about implementation of data consistency and replication in social media platform. |