|  |  |
| --- | --- |
| **Name** | Mayur Solankar, Manish Jadhav, Vishesh Savani, Shreyansh Salvi |
| **UID** | 2023301018, 2023301005, 2022300100, 2022300091 |
| **Subject** | Distributed Computing |
| **Experiment No.** | 4 |
| **Project title** | Social Media System |
| **Problem Statement** | To implement load balancing in social media platform. |
| **Objectives** | Make sure everyone gets to see and interact with social media posts quickly and without any problems by sharing the work among different computers, so none of them get too busy or slow down. |
| **Theory** | What is load balancing ? A load balancer is a device that acts as a reverse proxy and distributes network or application traffic across a number of servers. Load adjusting is the approach to conveying load units (i.e., occupations/assignments) across the organization which is associated with the distributed system. Load adjusting should be possible by the load balancer. The load balancer is a framework that can deal with the load and is utilized to disperse the assignments to the servers. The load balancers allocate the primary undertaking to the main server and the second assignment to the second server.    **Purpose of Load Balancing in Social media platform :**   * **Even Distribution of User Requests:** Social media platforms face heavy traffic with millions of users accessing content concurrently. Load balancing helps distribute this load evenly across servers, ensuring no single server gets overwhelmed, preventing slowdowns or crashes. * **Enhanced Performance and Responsiveness:** By spreading the load, servers can handle incoming requests more efficiently. This leads to faster response times, quicker content delivery, and an overall smoother user experience. * **Optimized Resource Utilization:**  Load balancing ensures that resources, such as server capacity and bandwidth, are used effectively. It helps avoid underutilization or overloading of specific servers, maximizing the platform's efficiency. * **Fault Tolerance and High Availability:** Load balancing can redirect traffic away from failed or underperforming servers to healthy ones. This helps in maintaining the platform's availability and minimizes disruptions in case of server failures or issues. * **Scalability:** As social media platforms grow, load balancing facilitates easy scaling by allowing new servers to be added to the system. It ensures that the increased traffic is efficiently distributed across the expanded infrastructure. * In essence, load balancing in social media platforms is vital for ensuring consistent, fast, and reliable access to content while efficiently managing the considerable traffic and user interactions these platforms experience. |
| **Code:** | Load balancing:-import http.serverimport socketserverfrom http import HTTPStatusimport urllib.parseimport requestsimport timefrom threading import Thread, Lockimport jsonclass DynamicLoadBalancer:def \_init\_(self, servers):self.servers = serversself.server\_health = {server: True for server in servers}self.server\_connections = {server: 0 for server in servers}self.lock = Lock()def get\_least\_busy\_server(self):with self.lock:healthy\_servers = [server for server in self.servers if self.server\_health[server]]if not healthy\_servers:return Nonereturn min(healthy\_servers, key=lambda s: self.server\_connections[s])def increment\_connections(self, server):with self.lock:self.server\_connections[server] += 1def decrement\_connections(self, server):with self.lock:self.server\_connections[server] = max(0, self.server\_connections[server] - 1)def check\_server\_health(self):while True:for server in self.servers:try:response = requests.get(f"{server}/health", timeout=5)with self.lock:self.server\_health[server] = response.status\_code == 200except:with self.lock:self.server\_health[server] = Falsetime.sleep(10) # Check health every 10 secondsclass LoadBalancerHandler(http.server.BaseHTTPRequestHandler):def do\_GET(self):if self.path == '/health':self.send\_response(200)self.end\_headers()returnif self.path == '/stats':self.send\_response(200)self.send\_header("Content-type", "application/json")self.end\_headers()stats = {"health": load\_balancer.server\_health,"connections": load\_balancer.server\_connections}self.wfile.write(json.dumps(stats).encode())returnserver\_url = load\_balancer.get\_least\_busy\_server()if server\_url:load\_balancer.increment\_connections(server\_url)try:response = requests.get(f"{server\_url}{self.path}", timeout=30)self.send\_response(response.status\_code)for header, value in response.headers.items():self.send\_header(header, value)self.end\_headers()self.wfile.write(response.content)except:self.send\_response(HTTPStatus.BAD\_GATEWAY)self.end\_headers()self.wfile.write(b"Error communicating with backend server")finally:load\_balancer.decrement\_connections(server\_url)else:self.send\_response(HTTPStatus.SERVICE\_UNAVAILABLE)self.end\_headers()self.wfile.write(b"No healthy servers available")servers = ["http://localhost:8001", "http://localhost:8002", "http://localhost:8003"]load\_balancer = DynamicLoadBalancer(servers)# Start health check threadhealth\_thread = Thread(target=load\_balancer.check\_server\_health)health\_thread.daemon = Truehealth\_thread.start()PORT = 8000with socketserver.TCPServer(("", PORT), LoadBalancerHandler) as httpd:print(f"Load balancer serving at port {PORT}")httpd.serve\_forever()Server 1:-import http.serverimport socketserverimport timeimport randomclass ServerHandler(http.server.BaseHTTPRequestHandler):def do\_GET(self):if self.path == '/health':self.send\_response(200)self.send\_header("Content-type", "text/plain")self.end\_headers()self.wfile.write(b"OK")else:# Simulate varying processing timeprocessing\_time = random.uniform(0.1, 2.0)time.sleep(processing\_time)self.send\_response(200)self.send\_header("Content-type", "text/html")self.end\_headers()response = f"Hello from Server on port {PORT}! (Processing time: {processing\_time:.2f}s)"self.wfile.write(response.encode())PORT = 8002 # Change this for each server instancewith socketserver.TCPServer(("", PORT), ServerHandler) as httpd:print(f"Server serving at port {PORT}")httpd.serve\_forever()Server 2:-import http.serverimport socketserverimport timeimport randomclass ServerHandler(http.server.BaseHTTPRequestHandler):def do\_GET(self):if self.path == '/health':self.send\_response(200)self.send\_header("Content-type", "text/plain")self.end\_headers()self.wfile.write(b"OK")else:# Simulate varying processing timeprocessing\_time = random.uniform(0.1, 2.0)time.sleep(processing\_time)self.send\_response(200)self.send\_header("Content-type", "text/html")self.end\_headers()response = f"Hello from Server on port {PORT}! (Processing time: {processing\_time:.2f}s)"self.wfile.write(response.encode())PORT = 8001 # Change this for each server instancewith socketserver.TCPServer(("", PORT), ServerHandler) as httpd:print(f"Server serving at port {PORT}")httpd.serve\_forever() |
| **Output:** |  |
| **Conclusion:** | Hence by completing we came to about implementation of Load Balancing in Distributed computing. |