```go
package main

import (
        "fmt"
        "log"
        "net"
        "net/http"
        "net/rpc"
        "sync"
)

// ==== Structs Request & Reply ====
type RegisterServerRequest struct {
        Address string
}
type RegisterServerReply struct {
        Success bool
        Message string
}

type RemoveServerRequest struct {
        Address string
}

type RemoveServerReply struct {
        Success bool
        Message string
}

type GetRequest struct {
        Bucket string
        Key    int
}
type GetReply struct {
        Success bool
        Err     error
        Data    []byte
}

type SetRequest struct {
        Bucket string
        Key    int
        Data   []byte
}
type SetReply struct {
        Success bool
        Err     error
}

type DeleteRequest struct {
        Bucket string
        Key    int
}
```

```
54  type DeleteReply struct {
55          Success bool
56          Err     error
57  }
58
59  type GetAllRequest struct {
60          Bucket string
61  }
62  type GetAllReply struct {
63          Success bool
64          Data    map[int][]byte
65          Err     error
66  }
67
68  type GetInfoRequest struct{}
69  type GetInfoReply struct {
70          Success bool
71          Info    string
72          Err     error
73  }
74
75  // ==== LoadBalancer ====
76  type LoadBalancer struct {
77          hashRing *Map
78          servers  map[string]*rpc.Client
79          //nodes    []string
80          mutex sync.RWMutex
81  }
82
83  // ==== Khởi tạo LoadBalancer ====
84  func NewLoadBalancer() *LoadBalancer {
85          return &LoadBalancer{
86                  hashRing: NewConsistentHash(10, nil),
87                  servers:  make(map[string]*rpc.Client),
88                  //nodes:    []string{},
89          }
90  }
91
92  // ==== Đăng ký Server mới ====
93  func (lb *LoadBalancer) RegisterServer(req *RegisterServerRequest, reply
    *RegisterServerReply) error {
94          lb.mutex.Lock()
95          defer lb.mutex.Unlock()
96
97          if _, exists := lb.servers[req.Address]; exists {
98                  reply.Success = false
99                  reply.Message = fmt.Sprintf("Server %s is already
    registered", req.Address)
100                 log.Println(reply.Message)
101                 return nil
102         }
103
104         client, err := rpc.DialHTTP("tcp", req.Address)
105         if err ≠ nil {
106                 reply.Success = false
```

```
107                 reply.Message = fmt.Sprintf("Failed to connect to server %s:
    %v", req.Address, err)
108                 log.Println(reply.Message)
109                 return err
110         }
111
112         // Thêm vào hash ring
113         lb.hashRing.Add(req.Address)
114         lb.servers[req.Address] = client
115
116         reply.Success = true
117         reply.Message = fmt.Sprintf("Added server: %s", req.Address)
118
119         log.Printf("NewConsistentHash server registered: %s", req.Address)
120         log.Println("Current servers in LoadBalancer:")
121         for addr, _ := range lb.servers {
122                 log.Printf("   - %s", addr)
123         }
124
125         // Thực hiện rebalance dữ liệu
126         lb.rebalanceData(req.Address)
127         log.Println("————————————————————————————————")
128         return nil
129 }
130
131 // ═══ Rebalance Data ═══
132 func (lb *LoadBalancer) rebalanceData(newServerAddr string) {
133         log.Printf("Rebalancing data for new server: %s", newServerAddr)
134
135         // Nếu chỉ có 1 node, không cần rebalance
136         if len(lb.servers) < 2 {
137                 log.Printf("Not enough servers for rebalancing")
138                 return
139         }
140
141         // Tìm node đứng liền sau trong vòng băm
142         nextServerAddr := lb.hashRing.Get(newServerAddr)
143
144         log.Printf("Data will be moved from next server: %s", nextServerAddr)
145
146         // Kiểm tra kết nối
147         nextServer, exists := lb.servers[nextServerAddr]
148         if !exists {
149                 log.Printf("next server %s is not connected, skipping
    migration", nextServerAddr)
150                 return
151         }
152
153         // Lấy toàn bộ dữ liệu từ node sau
154         getAllReq := &GetAllRequest{Bucket: "user"}
155         getAllReply := &GetAllReply{}
156         err := nextServer.Call("Service.GetAll", getAllReq, getAllReply)
157         if err ≠ nil {
158                 log.Printf("Failed to get data from %s: %v", nextServerAddr,
    err)
```

```
159                 return
160         }
161
162         // Di chuyển dữ liệu sang node mới
163         for key, value := range getAllReply.Data {
164                 newServer, _ := lb.GetServerForKey(fmt.Sprintf("user_%d",
    key))
165                 if newServer == newServerAddr {
166                         lb.migrateData(newServerAddr, nextServerAddr, key,
    value)
167                 }
168         }
169 }
170
171 func (lb *LoadBalancer) migrateData(newServerAddr string, oldServerAddr
    string, key int, data []byte) {
172         log.Printf("Migrating key %d to new server %s", key, newServerAddr)
173
174         // Gửi Set request tới server mới
175         newServer := lb.servers[newServerAddr]
176         setReq := &SetRequest{Bucket: "user", Key: key, Data: data}
177         setReply := &SetReply{}
178
179         err := newServer.Call("Service.Set", setReq, setReply)
180         if err ≠ nil || !setReply.Success {
181                 log.Printf("Failed to migrate key %d to %s: %v", key,
    newServerAddr, err)
182                 return
183         }
184         log.Printf("Successfully migrated key %d to server %s", key,
    newServerAddr)
185
186         // Xóa dữ liệu trên server cũ sau khi di chuyển thành công
187         if oldServerAddr ≠ "" {
188                 log.Printf("Deleting key %d from old server %s", key,
    oldServerAddr)
189
190                 oldServer := lb.servers[oldServerAddr]
191                 delReq := &DeleteRequest{Bucket: "user", Key: key}
192                 delReply := &DeleteReply{}
193
194                 err = oldServer.Call("Service.Delete", delReq, delReply)
195                 if err ≠ nil || !delReply.Success {
196                         log.Printf("Failed to delete key %d from old server
    %s: %v", key, oldServerAddr, err)
197                 } else {
198                         log.Printf("Successfully deleted key %d from old
    server %s", key, oldServerAddr)
199                 }
200         }
201
202 }
203
204 // ═══ Xóa Server ═══
```

```
205 func (lb *LoadBalancer) RemoveServer(req *RemoveServerRequest, reply
    *RemoveServerReply) error {
206         lb.mutex.Lock()
207         defer lb.mutex.Unlock()
208 
209         address := req.Address
210 
211         // Kiểm tra server có tồn tại trong danh sách không
212         if _, exists := lb.servers[address]; !exists {
213                 reply.Success = false
214                 reply.Message = fmt.Sprintf("Server %s not found, skipping
    removal", address)
215                 log.Printf(reply.Message)
216                 return nil
217         }
218 
219         // Lấy toàn bộ dữ liệu từ server cần xóa
220         removeThisServer := lb.servers[address]
221         getAllReq := &GetAllRequest{Bucket: "user"}
222         getAllReply := &GetAllReply{}
223         err := removeThisServer.Call("Service.GetAll", getAllReq,
    getAllReply)
224         if err ≠ nil {
225                 reply.Success = false
226                 reply.Message = fmt.Sprintf("Failed to get data from %s: %v",
    address, err)
227                 log.Printf(reply.Message)
228                 return err
229         }
230 
231         // Đóng kết nối RPC với server
232         err = lb.servers[address].Close()
233         if err ≠ nil {
234                 log.Printf("Failed to close connection with server %s: %v",
    address, err)
235         }
236 
237         // Xóa server khỏi danh sách servers
238         delete(lb.servers, address)
239 
240         // Cập nhật lại vòng băm (consistent hash)
241         lb.hashRing = NewConsistentHash(10, nil)
242 
243         // Thêm lại các node còn lại vào vòng băm
244         for nodeAddress := range lb.servers {
245                 lb.hashRing.Add(nodeAddress)
246         }
247 
248         // Di chuyển dữ liệu sang server mới
249         log.Printf("Rebalancing data after removal of server %s", address)
250         for key, value := range getAllReply.Data {
251                 newServer, _ := lb.GetServerForKey(fmt.Sprintf("user_%d",
    key))
252                 lb.migrateData(newServer, "", key, value)
253         }
```

```
254
255         reply.Success = true
256         reply.Message = fmt.Sprintf("Removed server: %s", address)
257         log.Printf("Removed server: %s", address)
258
259         return nil
260 }
261
262 // ==== Lấy Server theo key ====
263 func (lb *LoadBalancer) GetServerForKey(key string) (string, error) {
264         //lb.mutex.RLock()
265         //defer lb.mutex.RUnlock()
266
267         server := lb.hashRing.Get(key)
268         if server == "" {
269                 return "", fmt.Errorf("no available servers")
270         }
271         log.Printf("%s will be moved/stayed at %s", key, server)
272         return server, nil
273 }
274
275 // ==== Set Request ====
276 func (lb *LoadBalancer) Set(req *SetRequest, reply *SetReply) error {
277         log.Printf("LoadBalancer received Set request for key: %d, bucket:
    %s", req.Key, req.Bucket)
278
279         server, err := lb.GetServerForKey(fmt.Sprintf("%s_%d", req.Bucket,
    req.Key))
280         if err != nil {
281                 log.Printf("No server found for key: %d, error: %v", req.Key,
    err)
282                 return err
283         }
284         log.Printf("LoadBalancer selected server %s for key %d", server,
    req.Key)
285
286         client, exists := lb.servers[server]
287         if !exists {
288                 log.Printf("Server %s not found in lb.servers map", server)
289                 return fmt.Errorf("server %s not found", server)
290         }
291
292         err = client.Call("Service.Set", req, reply)
293         if err != nil {
294                 log.Printf("Failed to forward Set request to server %s: %v",
    server, err)
295                 return err
296         }
297
298         log.Printf("LoadBalancer received response from server %s: Success =
    %v", server, reply.Success)
299         return nil
300 }
301
302 // ==== Get Request ====
```

```
303 func (lb *LoadBalancer) Get(req *GetRequest, reply *GetReply) error {
304         log.Printf("LoadBalancer received Get request for key: %d, bucket:
    %s", req.Key, req.Bucket)
305 
306         // Tìm server thích hợp bằng consistent hashing
307         server, err := lb.GetServerForKey(fmt.Sprintf("%s_%d", req.Bucket,
    req.Key))
308         if err != nil {
309                 log.Printf("No server found for key: %d, error: %v", req.Key,
    err)
310                 return err
311         }
312         log.Printf("LoadBalancer selected server %s for key %d", server,
    req.Key)
313 
314         // Kiểm tra server có tồn tại không
315         client, exists := lb.servers[server]
316         if !exists {
317                 log.Printf("Server %s not found in lb.servers map", server)
318                 return fmt.Errorf("server %s not found", server)
319         }
320 
321         // Gửi request Get tới server node
322         err = client.Call("Service.Get", req, reply)
323         if err != nil {
324                 log.Printf("Failed to forward Get request to server %s: %v",
    server, err)
325                 return err
326         }
327 
328         // Log kết quả từ server
329         if reply.Success {
330                 log.Printf("LoadBalancer received response from server %s:
    Key %d found", server, req.Key)
331         } else {
332                 log.Printf("LoadBalancer received response from server %s:
    Key %d not found", server, req.Key)
333         }
334 
335         return nil
336 }
337 
338 // ==== Chạy LoadBalancer ====
339 func main() {
340         lb := NewLoadBalancer()
341         rpc.Register(lb)
342         rpc.HandleHTTP()
343         listener, _ := net.Listen("tcp", ":9000")
344         log.Println("LoadBalancer listening on port 9000...")
345         http.Serve(listener, nil)
346 }
347 
```