```go
package main

import (
	"fmt"
	"github.com/marcelloh/fastdb"
	"log"
	"net"
	"net/http"
	"net/rpc"
	"os"
	"strconv"
	"sync"
	"time"
)

// Service quản lý database của server
type Service struct {
	db    *fastdb.DB
	mutex *sync.Mutex
}

// Khởi tạo database riêng cho từng server
func NewService(dbFile string) (*Service, error) {
	db, err := fastdb.Open(dbFile, 100)
	if err != nil {
		log.Fatal(err)
	}

	return &Service{
		db:    db,
		mutex: new(sync.Mutex),
	}, nil
}

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

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

```

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

type DeleteRequest struct {
	Bucket string
	Key    int
}
type DeleteReply struct {
	Success bool
	Err     error
}

type GetAllRequest struct {
	Bucket string
}
type GetAllReply struct {
	Success bool
	Data    map[int][]byte
	Err     error
}

type GetInfoRequest struct{}
type GetInfoReply struct {
	Success bool
	Info    string
	Err     error
}

// ==== Các phương thức RPC ====
func (s *Service) Get(req *GetRequest, reply *GetReply) error {
	log.Printf("Server received Get request - Bucket: %s, Key: %d", req.Bucket, req.Key)

	data, ok := s.db.Get(req.Bucket, req.Key)
	if ok {
		reply.Success = true
		reply.Data = data
		log.Printf("Server found data for Key %d: %s", req.Key, string(data))
	} else {
		reply.Success = false
		reply.Err = fmt.Errorf("key not found")
		log.Printf("Server could not find Key %d", req.Key)
	}
	return nil
}

func (s *Service) Set(req *SetRequest, reply *SetReply) error {
	log.Printf("Server received Set request - Bucket: %s, Key: %d",
req.Bucket, req.Key)

	s.mutex.Lock()
	defer s.mutex.Unlock()
	err := s.db.Set(req.Bucket, req.Key, req.Data)
	if err != nil {
		reply.Success = false
		reply.Err = err
		log.Printf("Server failed to store data - Key: %d", req.Key)
		return err
	}

	reply.Success = true
	log.Printf("Server successfully stored data - Key: %d", req.Key)
	return nil
}

func (s *Service) Delete(req *DeleteRequest, reply *DeleteReply) error {
	s.mutex.Lock()
	defer s.mutex.Unlock()
	_, err := s.db.Del(req.Bucket, req.Key)
	if err != nil {
		reply.Success = false
		reply.Err = err
	} else {
		reply.Success = true
	}
	return nil
}

func (s *Service) GetAll(req *GetAllRequest, reply *GetAllReply) error {
	data, err := s.db.GetAll(req.Bucket)
	if err != nil {
		reply.Success = false
	} else {
		reply.Success = true
	}
	reply.Data = data
	return nil
}

func (s *Service) GetInfo(req *GetInfoRequest, reply *GetInfoReply) error {
	info := s.db.Info()
	if info != "" {
		reply.Success = true
		reply.Info = info
	} else {
		reply.Success = false
		reply.Err = fmt.Errorf("no info available")
	}
	return nil
}

// === Server tự động kết nối LoadBalancer ===
func registerWithLoadBalancer(serverAddress string) {
	// Đợi 1 giây trước khi gửi request để đảm bảo server đã mở cổng
	time.Sleep(1 * time.Second)

	client, err := rpc.DialHTTP("tcp", "localhost:9000") // LoadBalancer
chạy trên port 9000
	if err ≠ nil {
		log.Printf("⚠️ Failed to connect to LoadBalancer: %v", err)
		return
	}
	defer client.Close()

	req := &RegisterServerRequest{Address: serverAddress}
	reply := &RegisterServerReply{}

	err = client.Call("LoadBalancer.RegisterServer", req, reply)
	if err ≠ nil {
		log.Printf("Failed to register with LoadBalancer: %v", err)
		return
	}

	log.Printf("Server registered with LoadBalancer: %s", reply.Message)
}

// ═══ Chạy Server ═══
func main() {
	// Nhận port từ dòng lệnh
	if len(os.Args) < 2 {
		log.Fatal("Usage: go run server_main.go <port>")
	}
	port := os.Args[1]
	_, err := strconv.Atoi(port)
	if err ≠ nil {
		log.Fatal("Invalid port number:", port)
	}

	// Mỗi server có database riêng theo port
	dbFile := fmt.Sprintf("server_%s.db", port)
	service, err := NewService(dbFile)
	if err ≠ nil {
		log.Fatal("Error creating service:", err)
	}

	// Đăng ký RPC server
	err = rpc.Register(service)
	if err ≠ nil {
		log.Fatal("Error registering RPC:", err)
	}
	rpc.HandleHTTP()

	// Lắng nghe trên port được chỉ định
	address := ":" + port
	listener, err := net.Listen("tcp", address)
	if err ≠ nil {
		log.Fatal("Listen error:", err)
```

```go
	}

	log.Printf("Server started on port %s with database: %s", port, dbFile)

	// Chạy server trong goroutine để không block chương trình
	go func() {
		err = http.Serve(listener, nil)
		if err ≠ nil {
			log.Fatal("HTTP serve error:", err)
		}
	}()

	// Đăng ký với LoadBalancer
	registerWithLoadBalancer("localhost:" + port)

	// Giữ chương trình chạy
	select {}
}

```