```go
package main

import (
	"bufio"
	"fmt"
	"log"
	"net/rpc"
	"os"
	"strconv"
	"time"
)

// ---------------------------------------------------------
// Data Structures and RPC Arg/Reply
// ---------------------------------------------------------

type SetKeyArgs struct {
	BucketName string
	Key        int
	Value      string
}

type GetKeyArgs struct {
	BucketName string
	Key        int
}

type DeleteKeyArgs struct {
	BucketName string
	Key        int
}

type EmptyRequest struct{}

type Response struct {
	Data    string
	Message string
}

// ---------------------------------------------------------
// RPC Helper Functions
// ---------------------------------------------------------

func getKey(client *rpc.Client, args GetKeyArgs) string {
	var reply Response
	err := client.Call("LeaderRPC.GetKey", args, &reply)
	if err != nil {
		return fmt.Sprintf("[ERROR getKey]: %v", err)
	}
	if reply.Message != "" {
		return fmt.Sprintf("[FAIL getKey]: %s", reply.Message)
	}
	return reply.Data
}

func setKey(client *rpc.Client, args SetKeyArgs) string {
	var reply Response
	err := client.Call("LeaderRPC.SetKey", args, &reply)
	if err != nil {
		return fmt.Sprintf("[ERROR setKey]: %v", err)
	}
	if reply.Message != "OK" {
		return fmt.Sprintf("[FAIL setKey]: %s", reply.Message)
	}
	return "OK"
}

func deleteKey(client *rpc.Client, args DeleteKeyArgs) string {
	var reply Response
	err := client.Call("LeaderRPC.DeleteKey", args, &reply)
	if err != nil {
		return fmt.Sprintf("[ERROR deleteKey]: %v", err)
	}
	return reply.Message
}

// For the new test calls
func getStoreInfo(client *rpc.Client) string {
	var reply Response
	err := client.Call("LeaderRPC.GetStoreInfo", EmptyRequest{}, &reply)
	if err != nil {
		return fmt.Sprintf("[ERROR getStoreInfo]: %v", err)
	}
	if reply.Message != "" {
		return fmt.Sprintf("[FAIL getStoreInfo]: %s", reply.Message)
	}
	return reply.Data
}

// ---------------------------------------------------------
// Test Routines
// ---------------------------------------------------------

func testSetGetDelete(client *rpc.Client) {
	fmt.Println("== TestSetGetDelete on Leader ==")
	// 1) Set
	setRes := setKey(client, SetKeyArgs{"User", 1, "User A"})
	fmt.Println(" SetKey(User,1) =>", setRes)

	// 2) Get
	getRes := getKey(client, GetKeyArgs{"User", 1})
	fmt.Println(" GetKey(User,1) =>", getRes)

	// 3) Delete
	delRes := deleteKey(client, DeleteKeyArgs{"User", 1})
	fmt.Println(" DeleteKey(User,1) =>", delRes)

	// 4) Verify it’s really deleted
	postDel := getKey(client, GetKeyArgs{"User", 1})
	fmt.Println(" GetKey(User,1) after Delete =>", postDel)
	fmt.Println("== Done TestSetGetDelete ==\n")
}

func testMassSetAndGet(client *rpc.Client, bucketName string, count int) {
	fmt.Printf("== Mass set/get test for %d items in bucket %q ==\n",
count, bucketName)
	startTime := time.Now()

	// Bulk set
	for i := 0; i < count; i++ {
		setRes := setKey(client, SetKeyArgs{
			BucketName: bucketName,
			Key:        i,
			Value:      "Item " + strconv.Itoa(i),
		})
		if setRes != "OK" {
			fmt.Printf(" Failed setKey at i=%d => %s\n", i,
setRes)
```

```go
		}
	}

	// Bulk get
	missing := 0
	for i := 0; i < count; i++ {
		val := getKey(client, GetKeyArgs{
			BucketName: bucketName,
			Key:        i,
		})
		// If val is [FAIL or [ERROR, it indicates an issue
		if len(val) >= 5 && (val[:5] == "[FAIL" || val[:6] == "[ERROR") {
			missing++
		}
	}

	elapsed := time.Since(startTime)
	fmt.Printf(" Mass set/get done in %v. Missing count=%d\n", elapsed, missing)
	fmt.Println("== Done Mass set/get ==\n")
}

// ---------------------------------------------------------
// main: orchestrates the test
// ---------------------------------------------------------

func main() {
	// Hard-coded addresses:
	// - Leader is on :8000
	// - Backup is on :8001
	leaderAddr := "localhost:8000"
	backupAddr := "localhost:8001"

	// 1) Connect to the leader
	leaderClient, err := rpc.Dial("tcp", leaderAddr)
	if err != nil {
		log.Fatalf("Failed to connect leader @ %s: %v", leaderAddr, err)
	}
	fmt.Println("[Connected to leader @", leaderAddr, "]")

	// 2) Basic Set/Get/Delete on Leader
	testSetGetDelete(leaderClient)

	// 3) Mass test
	testMassSetAndGet(leaderClient, "BulkBucket", 20)

	// ---------- NEW TEST CASES FOR GetStoreInfo ----------
	fmt.Println("== Test #1: GetStoreInfo from the leader (should succeed) ==")
	storeLeader := getStoreInfo(leaderClient)
	fmt.Println(" Leader's store info =>", storeLeader)

	fmt.Println("\n== Test #2: GetStoreInfo from backup (should fail or show 'Not the leader') ==")
	backupClient, err := rpc.Dial("tcp", backupAddr)
	if err != nil {
		fmt.Printf("[WARN] Could not connect to backup @ %s: %v\n", backupAddr, err)
		fmt.Println("Skipping backup storeInfo test.")
	} else {
		storeBackup := getStoreInfo(backupClient)
		fmt.Println(" Backup's store info =>", storeBackup)
		backupClient.Close()
```

```go
        }
        // ------------------------------------------------------

        // 4) Ask user to kill the leader
        fmt.Println("\n[ACTION REQUIRED] Please kill/stop the leader node. Then press ENTER.")
        bufio.NewReader(os.Stdin).ReadBytes('\n')

        // 5) Wait for failover
        fmt.Println("Waiting 5s to let cluster elect new leader on :8000...")
        time.Sleep(5 * time.Second)

        // 6) Connect to the new leader (still :8000)
        newLeaderClient, err := rpc.Dial("tcp", leaderAddr)
        if err != nil {
                log.Fatalf("Failed to connect new leader @ %s: %v", leaderAddr, err)
        }
        fmt.Println("[Connected to new leader @", leaderAddr, "]\n")

        // 7) Verify data is consistent
        fmt.Println("== Checking data from BulkBucket on new leader ==\n")
        lastVal := getKey(newLeaderClient, GetKeyArgs{"BulkBucket", 20})
        fmt.Println(" GetKey(BulkBucket,20) =>", lastVal)

        // 8) Check store info again on new leader
        fmt.Println("\n== Test #3: GetStoreInfo on new leader (post-failover) ==\n")
        storeNewLeader := getStoreInfo(newLeaderClient)
        fmt.Println(" NewLeader store info =>", storeNewLeader)

        // 9) Additional sets/deletes on new leader
        fmt.Println("\n== Testing another SET/DELETE on new leader ==\n")
        setRes := setKey(newLeaderClient, SetKeyArgs{"FailoverBucket", 1, "DataAfterFailover"})
        fmt.Println(" SetKey(FailoverBucket,1) =>", setRes)

        delRes := deleteKey(newLeaderClient, DeleteKeyArgs{"FailoverBucket", 1})
        fmt.Println(" DeleteKey(FailoverBucket,1) =>", delRes)

        postDel := getKey(newLeaderClient, GetKeyArgs{"FailoverBucket", 1})
        fmt.Println(" GetKey(FailoverBucket,1) =>", postDel)

        newLeaderClient.Close()
        leaderClient.Close()

        fmt.Println("\n=== End of All Tests ===")
}
```