UFPE

Entrega 4 socket

Programação concorrente e distribuida

Data Reader

```go
package datareader

import (
    "encoding/csv"
    "fmt"
    "log"
    "os"
)

type DataReader struct{}

func (DataReader) GetData(quantLinhas int) [][]string {
    var linhas = make([][]string, quantLinhas)
    fmt.Println()
    f, err := os.Open("../datareader/data.csv")
    if err != nil {
        log.Fatal(err)
    }

    csvReader := csv.NewReader(f)

    for i := 0; i <= quantLinhas; i++ {
        // read csv values using csv.Reader
        data, err := csvReader.Read()
        if err != nil {
            log.Fatal(err)
        }
        if i > 0 {
            linhas[i-1] = append(linhas[i-1], data...)
        }
    }
    f.Close()

    return linhas[:]
}
```

SERVER TCP

```go
package main

import (
    "encoding/json"
    "fmt"
    "net"
    "os"
    "tcp_module_lib/datareader"
)

func ServerTCP() {
    r, err := net.ResolveTCPAddr("tcp", "localhost:1313")
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }

    ln, err := net.ListenTCP("tcp", r)
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }

    fmt.Println("Server listening on:", r)

    for {
        conn, err := ln.Accept()
        if err != nil {
            fmt.Printf(err.Error())
            os.Exit(0)
        }
        go HandleTCPConnection(conn)
    }
}
```

SERVER TCP

```go
func HandleTCPConnection(conn net.Conn) {
    var msgFromClient  int

    //Close connection
    defer func(conn net.Conn) {
        err := conn.Close()
        if err != nil {
            fmt.Println(err)
            os.Exit(0)
        }
    }(conn)

    // Create coder/decoder
    jsonDecoder := json.NewDecoder(conn)
    jsonEncoder := json.NewEncoder(conn)


    for{

        err := jsonDecoder.Decode(&msgFromClient)
        if err != nil && err.Error() == "EOF" {
            //conn.Close()
            // no further requests
            break
        }

        if err != nil {
            fmt.Println(err)
            os.Exit(0)
        }

        // Process request
        r := datareader.DataReader{}.GetData(msgFromClient)

        // Create response
        msgToClient := r

        // Serialise and send response to client
        err = jsonEncoder.Encode(msgToClient)
        if err != nil {
            os.Exit(0)
            break
        }
    }
    // conn.Close()
}
```

CLIENT TCP

```go
func client_TCP() {
    file, err := openLogFile("../tests/logs/results_tcp.log")
    if err != nil {
        log.Fatal(err)
    }
    log.SetOutput(file)
    log.SetFlags(log.LstdFlags | log.Lshortfile | log.Lmicroseconds)

    var requestTime time.Duration
    r, err := net.ResolveTCPAddr("tcp", "localhost:1313")
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }
    conn, err := net.DialTCP("tcp", nil, r)
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }
    // fecha conexão
    defer func(conn *net.TCPConn) {
        err := conn.Close()
        if err != nil {
        }
    }(conn)
```

CLIENT TCP

```
    var response [][]string
    // create coder/decoder
    decoder := json.NewDecoder(conn)
    encoder := json.NewEncoder(conn)
    // Create request
    request := 2
    for i := 0; i < 10000; i++ {
        // prepara request & start time
        t1 := time.Now()
        // Serialise and send request
        err = encoder.Encode(&request)
        if err != nil {
            fmt.Println(err)
            os.Exit(0)
        }
        // Receive response from servidor
        err = decoder.Decode(&response)
        if err != nil {
            fmt.Println(err)
            os.Exit(0)
        }

        requestTime = time.Now().Sub(t1)
        var arg = ""
        if(len(os.Args) > 1){
            arg = os.Args[1]
        }

        log.Println(arg + strconv.Itoa(int(requestTime.Nanoseconds())))
    }
}
```

SERVER UDP

```go
func ServerUDP() {
    msgFromClient := make([]byte, 1024)
    // resolve server address
    addr, err := net.ResolveUDPAddr("udp", "localhost:1313")
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }
    // listen on udp port
    conn, err := net.ListenUDP("udp", addr)
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }
    // close conn
    defer func(conn *net.UDPConn) {
        err := conn.Close()
        if err != nil {
        }
    }(conn)
    fmt.Println("Server UDP is ready to accept requests at port 1313...")
    for {
        // receive request
        n, addr, err := conn.ReadFromUDP(msgFromClient)
        if err != nil {
            fmt.Println(err)
            os.Exit(0)
        }
        // handle request
        HandleUDPRequest(conn, msgFromClient, n, addr)
    }
}
```

SERVER UDP

```go
func HandleUDPRequest(conn *net.UDPConn, msgFromClient []byte,
    n int, addr *net.UDPAddr) {
    var msgToClient []byte
    var request int

    //unmarshall request
    err := json.Unmarshal(msgFromClient[:n], &request)
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }

    // process request
    r := datareader.DataReader{}.GetData(request)

    // serialise response
    msgToClient, err = json.Marshal(r)
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }

    // send response
    _, err = conn.WriteTo(msgToClient, addr)
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }
}
```

CLIENT UDP

```go
func ClientUDP() {
    file, err := openLogFile("../tests/logs/results_udp.log")
    if err != nil {
        log.Fatal(err)
    }
    log.SetOutput(file)
    log.SetFlags(log.LstdFlags | log.Lshortfile | log.Lmicroseconds)

    var requestTime time.Duration
    var response [][]string
    // resolve server address
    addr, err := net.ResolveUDPAddr("udp", "localhost:1313")
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }
    // connect to server -- does not create a connection
    conn, err := net.DialUDP("udp", nil, addr)
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }
    // create coder/decoder
    decoder := json.NewDecoder(conn)
    encoder := json.NewEncoder(conn)
    // Close connection
    defer func(conn *net.UDPConn) {
        err := conn.Close()
        if err != nil {
        }
    }(conn)
```

CLIENT UDP

```
// Create request
request := 2

for i := 0; i < 10000; i++ {

    // prepara request & start time
    t1 := time.Now()
    // Serialise and send request
    err = encoder.Encode(request)
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }

    // Receive response from servidor
    err = decoder.Decode(&response)
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }

    requestTime = time.Now().Sub(t1)
    var arg = ""
    if(len(os.Args) > 1){
        arg = os.Args[1]
    }

    log.Println(arg + strconv.Itoa(int(requestTime.Nanoseconds())))
}
```

Clientes concorrentes

```
#!/bin/bash
#go run ../../TCP/server.go

go run  client.go &
go run  client.go &
go run  client.go &
go run  client.go &
go run  client.go &
go run  client.go &
go run  client.go "teste " &
go run  client.go &
go run  client.go &
go run  client.go &
go run  client.go &
go run  client.go &
go run  client.go &
go run  client.go &
go run  client.go &
```

RTT

```go
// Create request
request := 2

for i := 0; i < 10000; i++ {

    // prepara request & start time
    t1 := time.Now()
    // Serialise and send request
    err = encoder.Encode(request)
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }

    // Receive response from servidor
    err = decoder.Decode(&response)
    if err != nil {
        fmt.Println(err)
        os.Exit(0)
    }

    requestTime = time.Now().Sub(t1)
    var arg = ""
    if(len(os.Args) > 1){
        arg = os.Args[1]
    }

    log.Println(arg + strconv.Itoa(int(requestTime.Nanoseconds())))
}
```

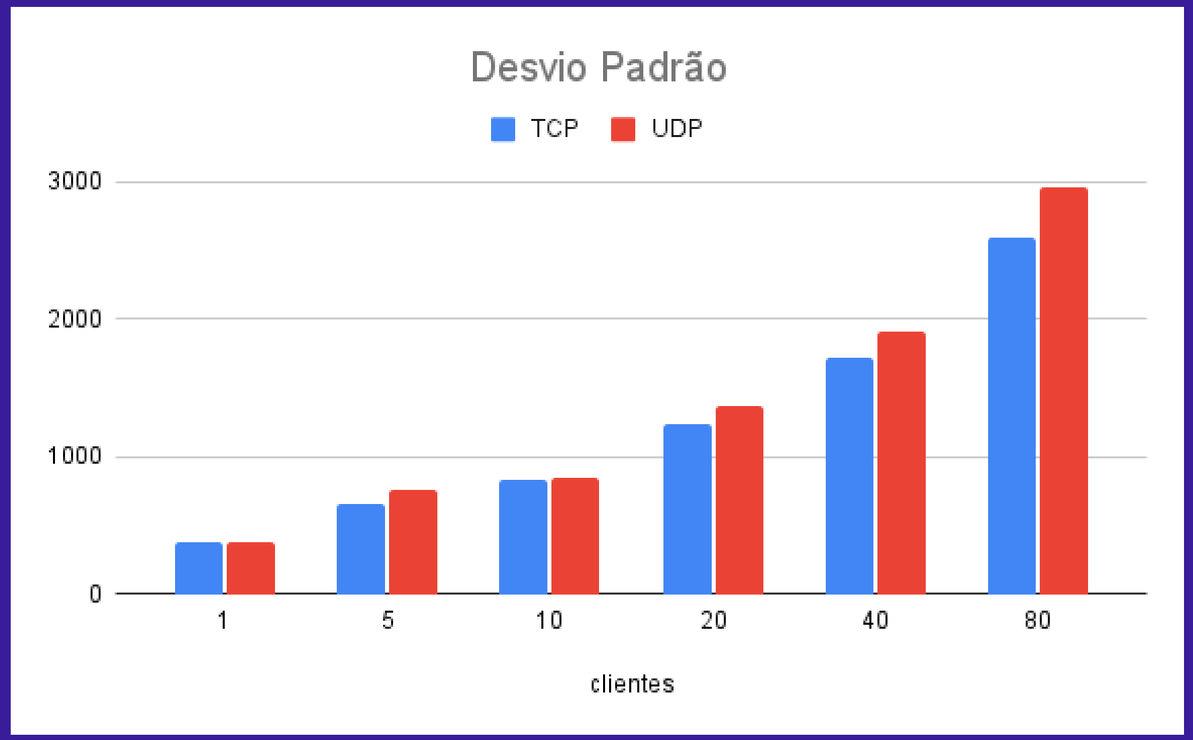
LOGS

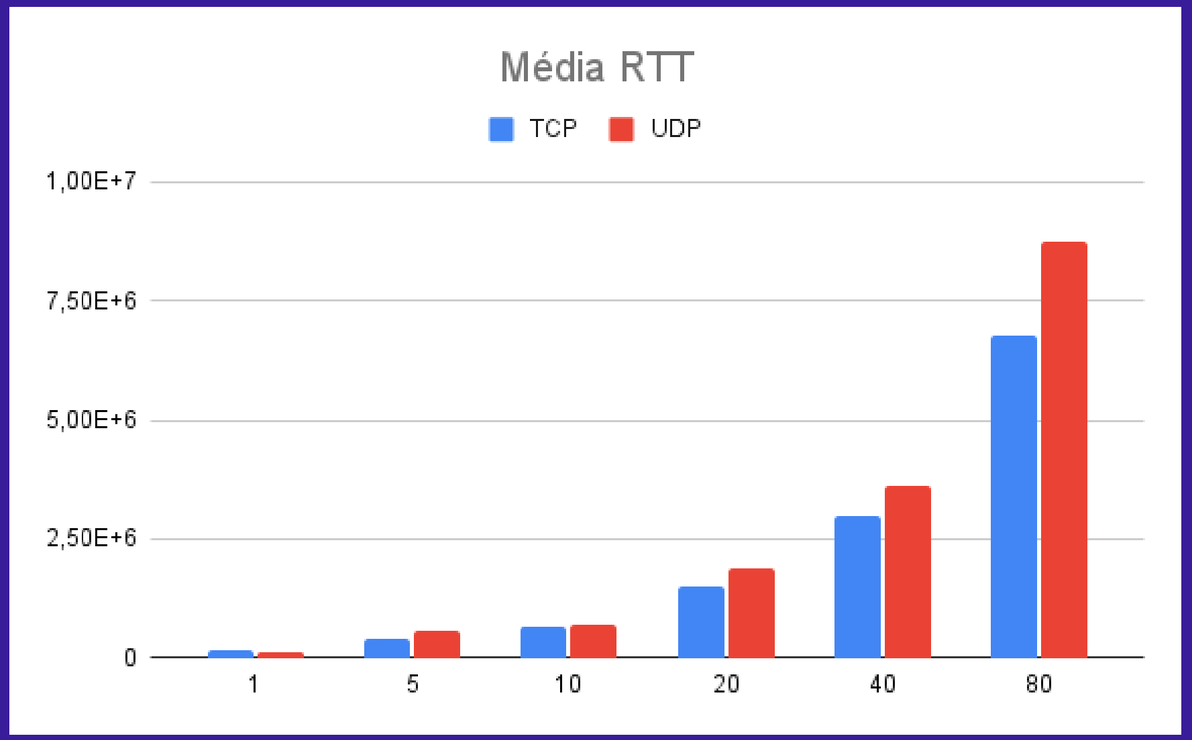
```
2023/07/17 00:51:05.390690 client.go:86: 783000
2023/07/17 00:51:05.391170 client.go:86: 1723843
2023/07/17 00:51:05.391467 client.go:86: 1405223
2023/07/17 00:51:05.391492 client.go:86: teste 2311006
2023/07/17 00:51:05.391713 client.go:86: 5468708
2023/07/17 00:51:05.392117 client.go:86: 1348512
2023/07/17 00:51:05.392706 client.go:86: 417725
2023/07/17 00:51:05.393193 client.go:86: 440913
2023/07/17 00:51:05.393431 client.go:86: 1850188
2023/07/17 00:51:05.393734 client.go:86: 232015
2023/07/17 00:51:05.393824 client.go:86: 4093237
2023/07/17 00:51:05.394145 client.go:86: 910531
2023/07/17 00:51:05.394454 client.go:86: 4128787
2023/07/17 00:51:05.394933 client.go:86: 1097848
2023/07/17 00:51:05.395256 client.go:86: 224263
2023/07/17 00:51:05.395549 client.go:86: 187666
2023/07/17 00:51:05.395715 client.go:86: 3951527
2023/07/17 00:51:05.396840 client.go:86: 2618939
2023/07/17 00:51:05.397144 client.go:86: 709177
2023/07/17 00:51:05.397545 client.go:86: 632840
2023/07/17 00:51:05.397991 client.go:86: 721469
2023/07/17 00:51:05.398234 client.go:86: 207920
2023/07/17 00:51:05.398452 client.go:86: 192415
2023/07/17 00:51:05.398805 client.go:86: 327490
2023/07/17 00:51:05.399097 client.go:86: 7870438
2023/07/17 00:51:05.399108 client.go:86: 258137
2023/07/17 00:51:05.399320 client.go:86: 186967
2023/07/17 00:51:05.399405 client.go:86: 5512917
2023/07/17 00:51:05.399721 client.go:86: 227196
2023/07/17 00:51:05.400030 client.go:86: 275667
2023/07/17 00:51:05.400287 client.go:86: 208897
2023/07/17 00:51:05.400685 client.go:86: teste 9078428
2023/07/17 00:51:05.401080 client.go:86: 761698
2023/07/17 00:51:05.401062 client.go:86: 644852
2023/07/17 00:51:05.401366 client.go:86: 2223075
2023/07/17 00:51:05.401491 client.go:86: 332100
2023/07/17 00:51:05.401611 client.go:86: 201355
2023/07/17 00:51:05.401713 client.go:86: 575360
2023/07/17 00:51:05.401686 client.go:86: teste 943078
2023/07/17 00:51:05.401778 client.go:86: 2308003
2023/07/17 00:51:05.401956 client.go:86: 7454109
2023/07/17 00:51:05.402227 client.go:86: 399147
2023/07/17 00:51:05.402272 client.go:86: 486311
2023/07/17 00:51:05.402444 client.go:86: 799343
2023/07/17 00:51:05.402571 client.go:86: 1048330
2023/07/17 00:51:05.402787 client.go:86: 302137
```

Cálculo RTT

```go
func main() {
    var times = make([]float64, 10000)
    var row = make([]string, 3)
    var somatorio float64 = 0
    var media float64 = 0
    f, err := os.Open("../tests/logs/results_udp.log")

    if err != nil {
        log.Fatal(err)
    }

    defer f.Close()

    scanner := bufio.NewScanner(f)
    i := 0
    for scanner.Scan() {
        row = strings.Split(scanner.Text(), " ")
        if row[3] == "teste" {
            times[i], err = strconv.ParseFloat(row[4], 64)
            somatorio += times[i]
            fmt.Println(somatorio)
            i++
        }
    }
    //calculo da media
    media = somatorio / 10000
    fmt.Println("Média: ", media)
    var somatorioDP float64 = 0
    var dp float64 = 0

    //calculo do desvio padrão
    for i := 0; i < 10000; i++ {
        somatorioDP += math.Pow(math.Abs(times[i]-media), 2)
    }
    dp = math.Sqrt(somatorio / 10000)

    fmt.Println("Desvio padrão:", dp)
    if err := scanner.Err(); err != nil {
        log.Fatal(err)
    }
}
```

Avaliações comparativas de desempenho





TCP	1	5	10	20	40	80
Media	141443,4188	423115,0838	685524,9368	1528222,254	2966257,862	6758032,765
DP	376,0896419	650,4729693	827,964333	1236,212868	1722,282747	2599,621658
UDP	1 cliente	5 clientes	10 clientes	20 clientes	40 clientes	80 clientes
Media	140316,6628	577590,2792	719247,7876	1870830,021	3630825,488	8739784,01
DP	374,5886581	759,9936047	848,0487762	1367,782885	1905,47251	2956,31257