Exercício 06

Um serviço web que fornece a hora atual do servidor para os clientes.

- Carlos Veeck <chvmv>
- Rodrigo Sales <rdgs>
- Vinícius Seabra <vsll>

Desafio Proposto

Implementar um Serviço Web capaz de aceitar requisições HTTP, processá-las e enviar respostas.

Desenvolvemos 2 sistemas utilizando o gRPC:

- Um com funcionalidades concorrentes
- e outro não concorrente.

Código Proto:

```proto
syntax = "proto3";

package time;

option go_package = "exercicio-06/proto";

service TimeService {
  rpc Get(HttpRequest) returns (HttpResponse);
}

message HttpRequest {
  string method = 1;
  string url = 2;
  string host = 3;
}

message HttpResponse {
  string status = 1;
  map<string, string> headers = 2;
  string body = 3;
}

```

Código Servidor sem concorrência:

```go
func (s *server) Get(ctx context.Context, req *pb.HttpRequest) (*pb.HttpResponse, error) {
    serverAddress := req.Host + ":2233"
    conn, err := net.Dial("tcp", serverAddress)
    if err != nil {
        return nil, err
    }
    defer conn.Close()

    request := req.Method + " " + req.Url + " HTTP/1.1\r\n" +
        "Host: " + req.Host + "\r\n" +
        "Connection: close\r\n" +
        "\r\n"

    _, err = conn.Write([]byte(request))
    if err != nil {
        return nil, err
    }

    reader := bufio.NewReader(conn)
    status, err := reader.ReadString('\n')
    if err != nil {
        return nil, err
    }

    headers := make(map[string]string)
    for {
        line, err := reader.ReadString('\n')
        if err != nil {
            return nil, err
        }
        line = strings.TrimSpace(line)
        if line == "" {
            break
        }
```

```
    return &pb.HttpResponse{
        Status:  status,
        Headers: headers,
        Body:    body,
    }, nil
}

func timeHandler(w http.ResponseWriter, r *http.Request) {
    currentTime := time.Now().Format(time.RFC1123)
    fmt.Fprintln(w, currentTime)
}
```

```go
func main() {
    lis, err := net.Listen("tcp", ":50051")
    if err != nil {
        log.Fatalf("failed to listen: %v", err)
    }

    s := grpc.NewServer()
    pb.RegisterTimeServiceServer(s, &server{})

    http.HandleFunc("/time", timeHandler)

    go func() {
        log.Fatal(http.ListenAndServe(":2233", nil))
    }()

    fmt.Println("gRPC server running on port 50051")
    if err := s.Serve(lis); err != nil {
        log.Fatalf("failed to serve: %v", err)
    }
}
```

Código Servidor concorrente:

```go
// Método Get processa a solicitação HTTP manualmente e retorna a resposta
func (s *server) Get(ctx context.Context, req *pb.HttpRequest) (*pb.HttpResponse, error) {
    responseChan := make(chan *pb.HttpResponse)
    errorChan := make(chan error)

    // Inicia uma goroutine para processar a solicitação
    go func() {
        serverAddr := req.Host + ":2233"
        conn, err := net.Dial("tcp", serverAddr)
        if err != nil {
            errorChan <- err
            return
        }
        defer conn.Close()

        request := fmt.Sprintf("%s %s HTTP/1.1\r\nHost: %s\r\nConnection: close\r\n\r\n", req.Method, req.Url, req.Host)
        _, err = conn.Write([]byte(request))
        if err != nil {
            errorChan <- err
            return
        }

        reader := bufio.NewReader(conn)
        status, err := reader.ReadString('\n')
        if err != nil {
            errorChan <- err
            return
        }

        headers := make(map[string]string)
        for {
            line, err := reader.ReadString('\n')
            if err != nil {
```

```
69          body := ""
70          for {
71              line, err := reader.ReadString('\n')
72              if err != nil {
73                  break
74              }
75              body += line
76          }
77
78          responseChan <- &pb.HttpResponse{
79              Status:  status,
80              Headers: headers,
81              Body:    body,
82          }
83      }()
84
85      select {
86      case res := <-responseChan:
87          return res, nil
88      case err := <-errorChan:
89          return nil, err
90      }
91  }
```

Código Cliente:

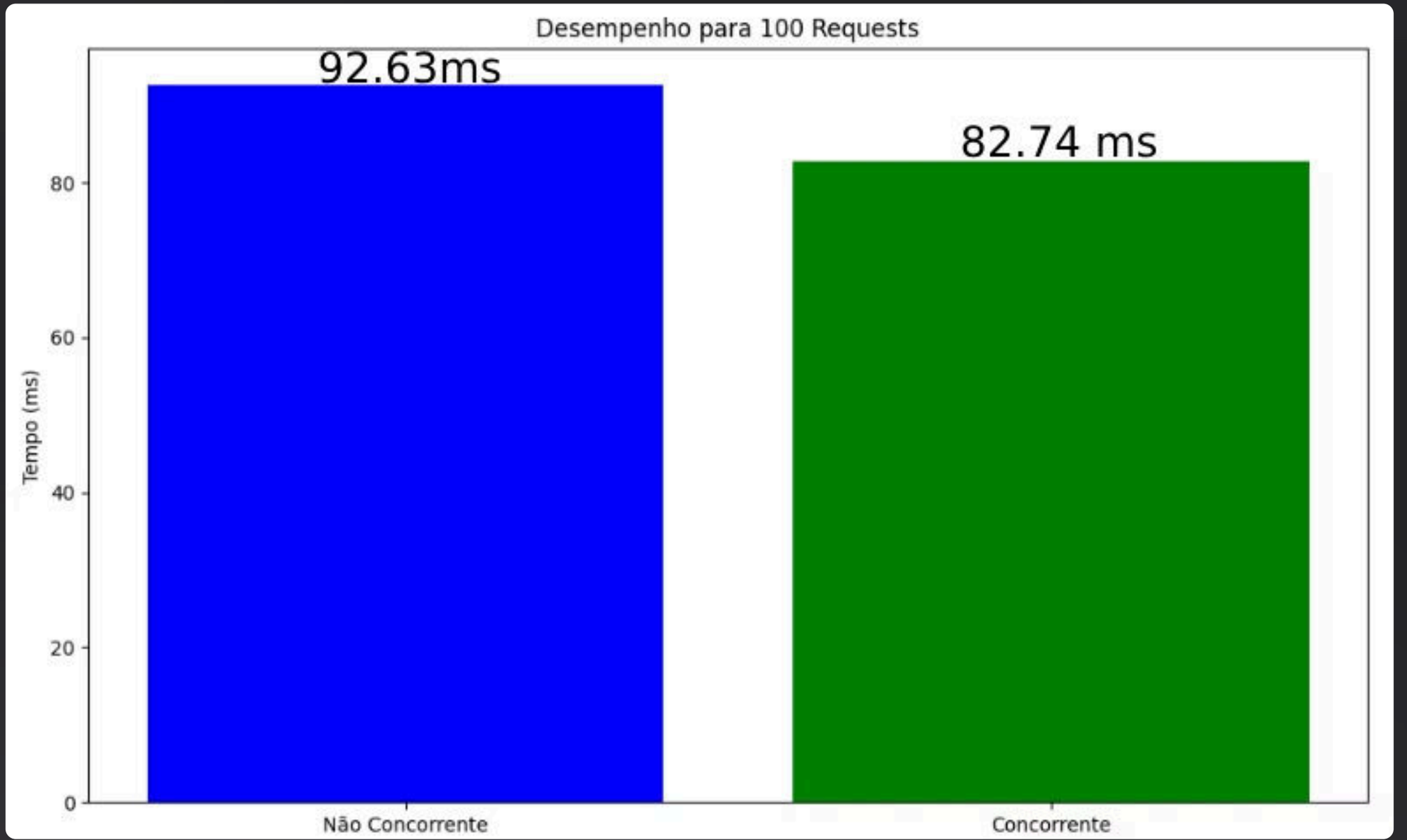
```go
func main() {
    conn, err := grpc.Dial("localhost:12345", grpc.WithInsecure(), grpc.WithBlock())
    if err != nil {
        log.Fatalf("did not connect: %v", err)
    }
    defer conn.Close()

    c := pb.NewTimeServiceClient(conn)

    req := &pb.HttpRequest{
        Method: "GET",
        Url:    "/time",
        Host:   "localhost",
    }

    numRequests := 1
    var totalElapsed time.Duration

    for i := 0; i < numRequests; i++ {
        startTime := time.Now()

        res, err := c.Get(context.Background(), req)
        if err != nil {
            fmt.Printf("Error on request %d: %v\n", i+1, err)
            continue
        }

        elapsed := time.Since(startTime)
        totalElapsed += elapsed

        fmt.Printf("Server Time %d: %s\n", i+1, res.Body)
        fmt.Printf("RPC request %d took %s\n", i+1, elapsed)
    }

    fmt.Printf("Total time taken for %d requests: %s\n", numRequests, totalElapsed)
    fmt.Printf("Average time per request: %s\n", totalElapsed/time.Duration(numRequests))
}
```

Parâmetros:

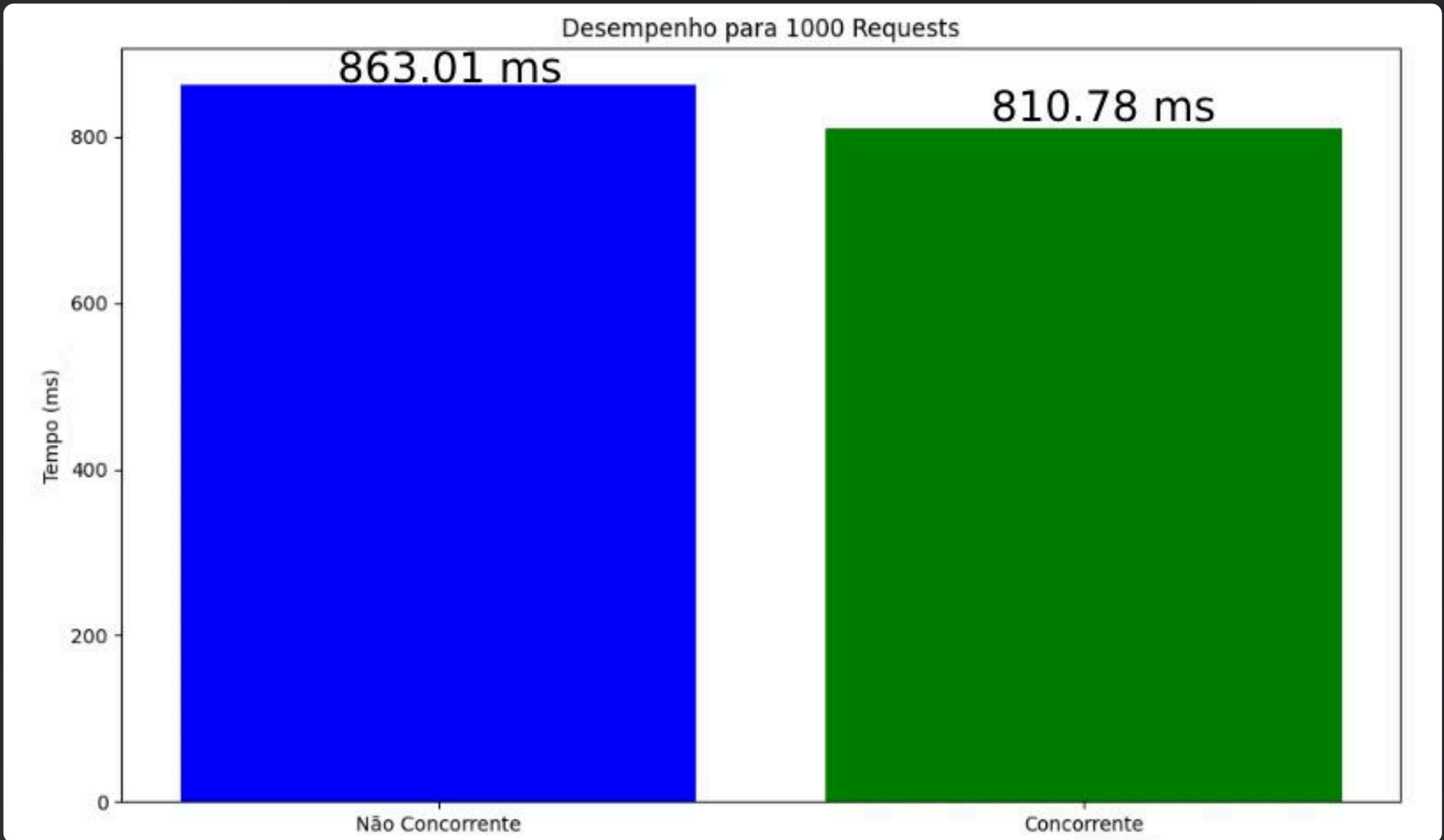
Parâmetros do sistema	Valores
Hardware	Processador: 11th Gen Intel(R) Core(TM) i5-11400 @ 2.60GHz 2.59 GHz SSD: KINGSTON SA400S37480G Memória RAM: 16GB
:Sistema Operacional	Microsoft Windows 10 Pro
Linguagem	Go
Fonte de alimentação	Rede elétrica
Processos em execução	Apenas processos necessários para a execução do experimento

Análise de desempenho:

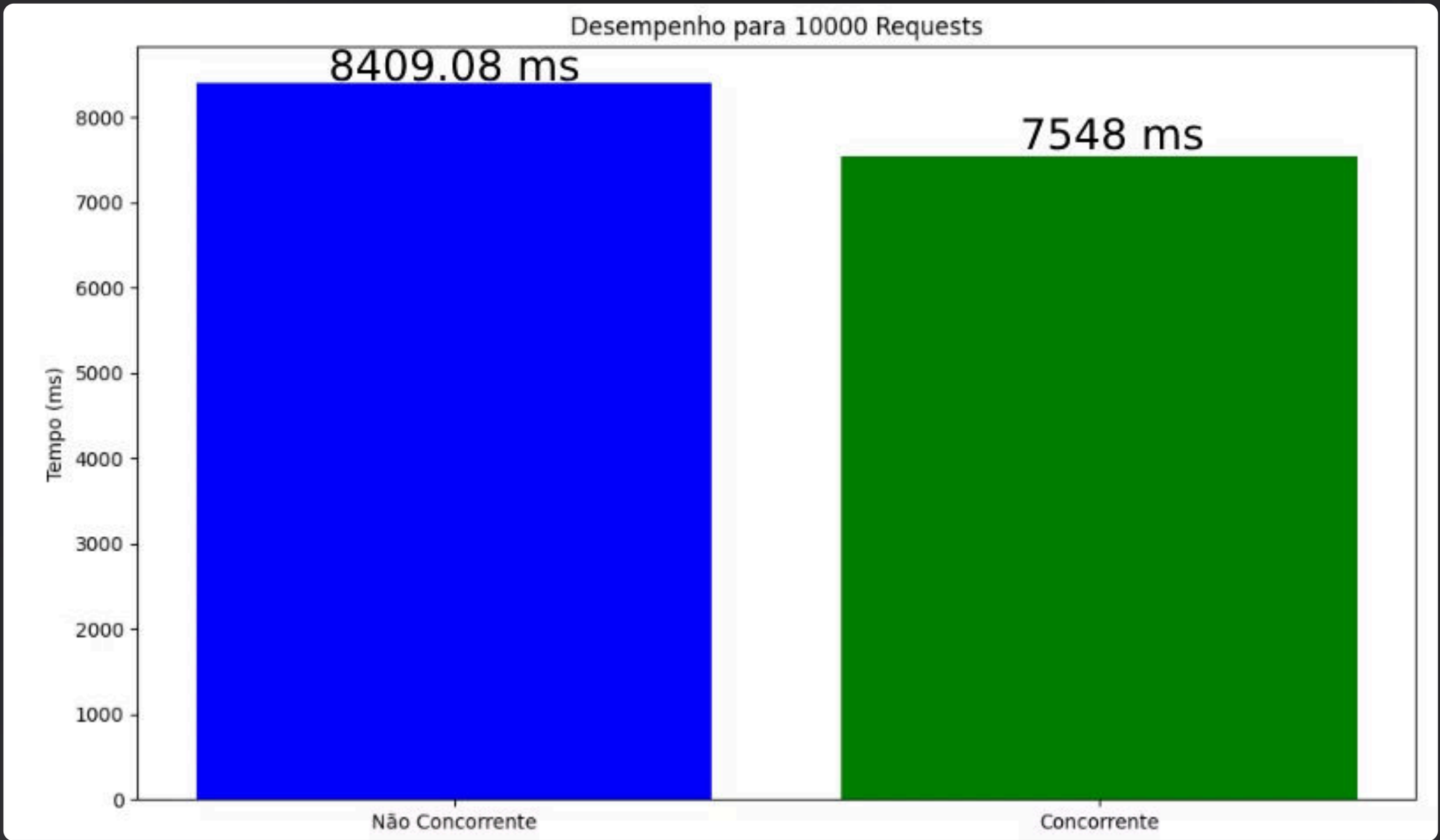
Com 100 requisições:



Com 1.000 requisições:



Com 10.000 requests:



Conclusões:

- O modelo concorrente apresentou uma melhora de eficiência de aproximadamente 12% na maioria dos casos.
- Acreditamos que isso se deve à inclusão da concorrência na hora da construção do pacote HTTP, o que ajuda o sistema a progredir o processamento das requisições de forma muito mais eficiente.
- A concorrência parece fazer mais efeito com números maiores de requisições.

Muito obrigado!