Análise de Desempenho

MergeSort e QuickSort - gRPC e Go RPC

Centro de Informática UFPE

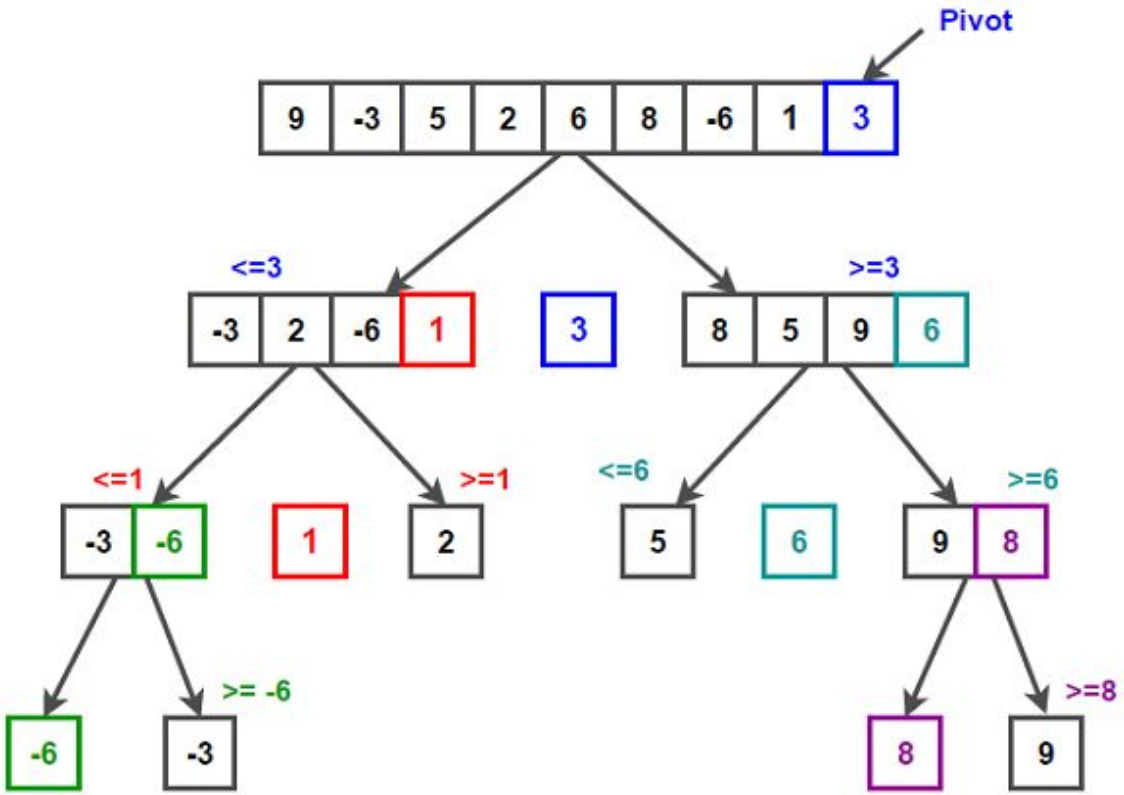
Universidade Federal de Pernambuco

Camila Xavier (cxm) e Vituriano Xisto (vox)

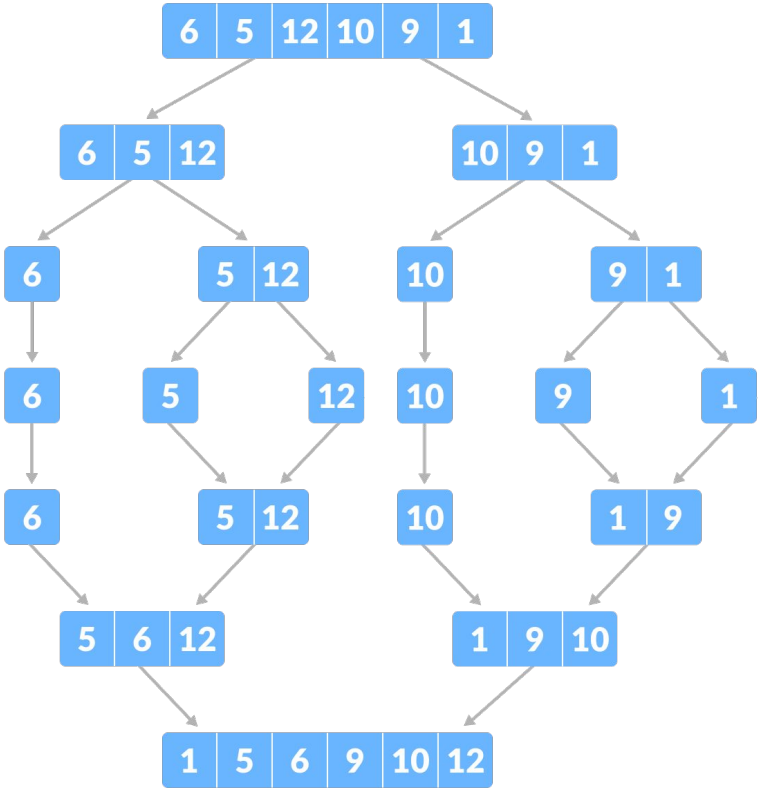
Objetivo

Comparar o desempenho de uma aplicação cliente/servidor usando dois mecanismos de comunicação diferentes (GoRPC e gRPC). Os algoritmos implementados foram os de ordenação *QuickSort* e *MergeSort*.

QuickSort



MergeSort


6 5 12 10 9 1
6 5 12
10 9 1
6
5 12
10
9 1
6
5
12
10
9
1
6
5 12
10
1 9
5 6 12
1 9 10
1 5 6 9 10 12

Serviços do sistema

- Servidor GoRPC:
  - Serviço de ordenação de array com MergeSort;
  - Serviço de ordenação de array com QuickSort;
- Servidor gRPC:
  - Serviço de ordenação de array com MergeSort;
  - Serviço de ordenação de array com QuickSort;
- Cliente GoRPC;
- Cliente gRPC.

Conexão GoRPC

```
client, err := rpc.Dial("tcp", "localhost:1234")
if err != nil {
        fmt.Println("Error dialing the server:", err)
        return
}
defer client.Close()

args := &Args{Array: []int{38, 27, 43, 3, 9, 82, 10}}
for i := 0; i < times; i++ {
        reply, duration, err := measureTime(client, "SortService.MergeSortRemote", args)
        if err != nil {
                fmt.Println("Error calling MergeSortRemote:", err)
                return
        }
}
```

Conexão GoRPC

```
func measureTime(client *rpc.Client, method string, args *Args) (SortResponse, time.Duration, error) {
        var reply SortResponse
        start := time.Now()
        err := client.Call(method, args, &reply)
        duration := time.Since(start)
        return reply, duration, err
}
```

Conexão GoRPC

```
sortService := new(SortService)
rpc.Register(sortService)

listener, err := net.Listen("tcp", ":1234")
if err != nil {
        fmt.Println("Error starting the server:", err)
        return
}

for {
        conn, err := listener.Accept()
        if err != nil {
                fmt.Println("Error accepting connection:", err)
                continue
        }
        go rpc.ServeConn(conn)
}
```

Conexão gRPC

```
conn, err := grpc.NewClient("localhost:50051", grpc.WithTransportCredentials(insecure.NewCredentials()))
if err != nil {
        fmt.Println("Error dialing the server:", err)
        return
}
defer conn.Close()

client := pb.NewSortServiceClient(conn)
req := &pb.SortRequest{Array: []int32{38, 27, 43, 3, 9, 82, 10}}

for i := 0; i < times; i++ {
        measureTime(client.MergeSort, req)
        if err != nil {
                log.Fatalf("could not call MergeSort: %v", err)
        }
}
```

Conexão gRPC

```
func measureTime(
        method func(context.Context, *pb.SortRequest, ...grpc.CallOption) (*pb.SortResponse, error),
        req *pb.SortRequest) (*pb.SortResponse, time.Duration, error
) {
        start := time.Now()
        resp, err := method(context.Background(), req)
        duration := time.Since(start)
        return resp, duration, err
}
```

Conexão gRPC

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

s := grpc.NewServer()
sort_grpc.RegisterSortServiceServer(s, &server{})
if err := s.Serve(lis); err != nil {
        log.Fatalf("failed to serve: %v", err)
}
```

Conexão gRPC

```
func convert[T, U any](arr []T, convertFunc func(T) U) []U {
        result := make([]U, len(arr))
        for i, v := range arr {
                result[i] = convertFunc(v)
        }
        return result
}
```

Métricas de Desempenho

- Tempo de execução de uma requisição ao serviço, medido no Cliente;

Parâmetros

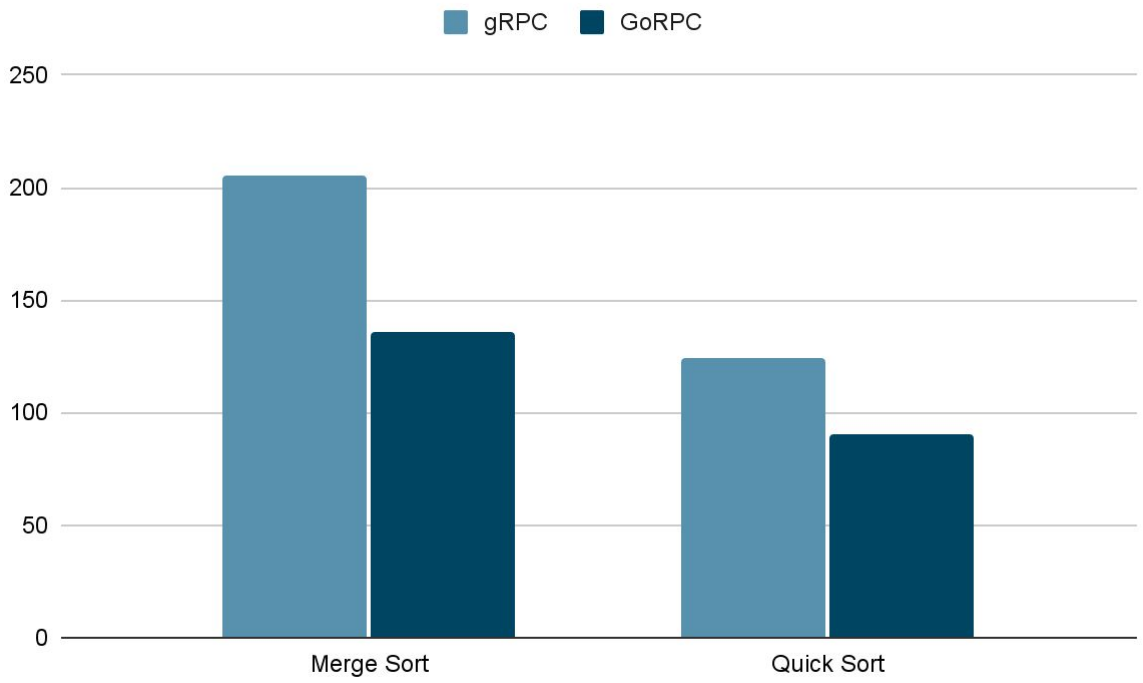
Parâmetro do Sistema	Valor
Hardware	MacBook M2 Pro, 16GB
Sistema operacional	macOS (Sonoma 14.1.1)
Linguagem de programação	Go
Interfaces de rede	Desligadas
Fonte de alimentação	Rede Elétrica
Processos em execução	Apenas os estritamente necessários à realização do experimento

Fatores

Fator	Nível
Algoritmo de Ordenação	MergeSort, QuickSort
Mecanismo de Comunicação	GoRPC, gRPC
N	(100, 1000, 10000)

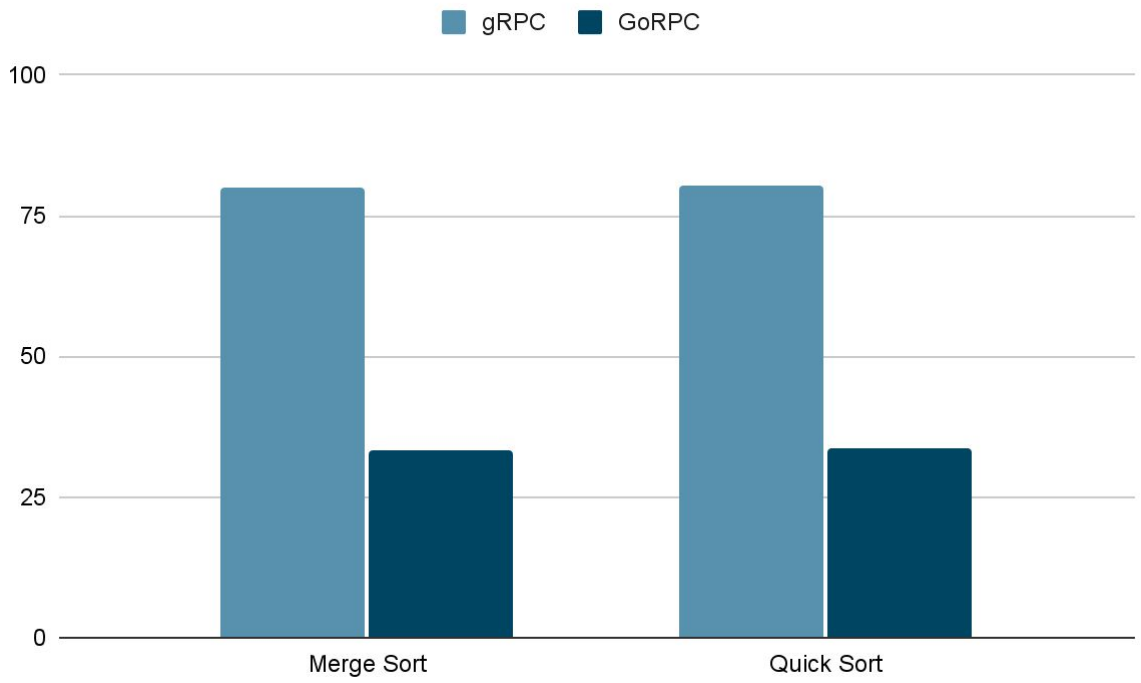
Resultados

- **N:** 100
- RTT Médio **gRPC**:
  - Merge Sort: 205.083µs
  - Quick Sort: 124.32µs
- RTT Médio **GoRPC**:
  - Merge Sort: 136.06µs
  - Quick Sort: 90.229µs



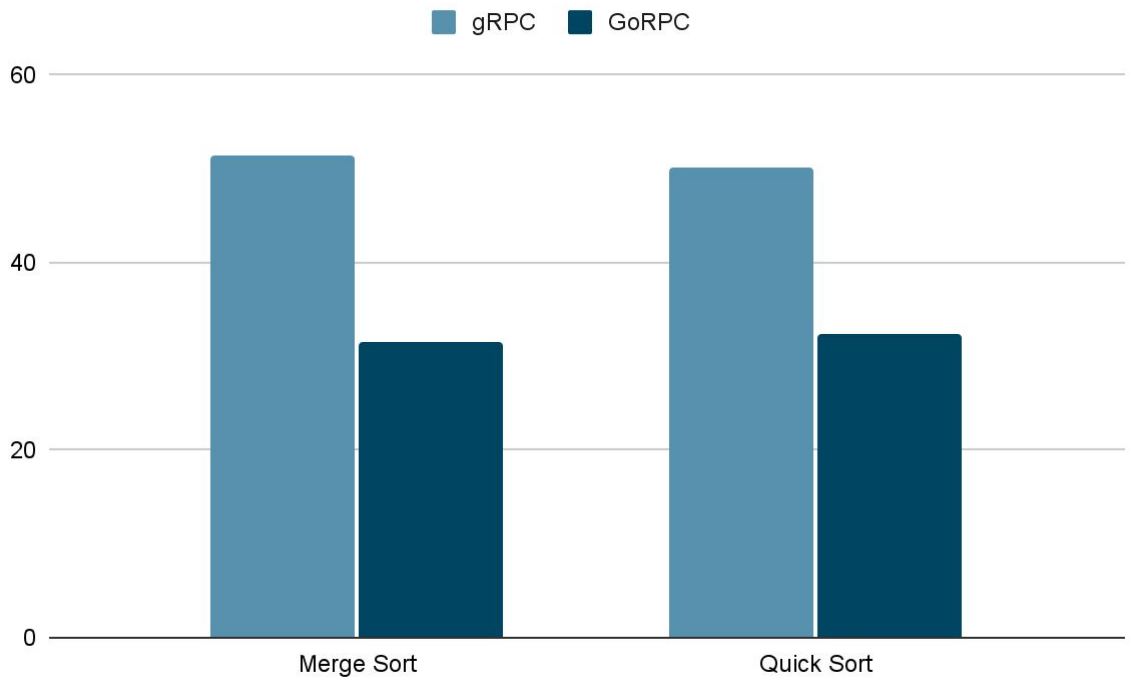
Resultados

- **N:** 1000
- RTT Médio **gRPC**:
  - Merge Sort: 79.98µs
  - Quick Sort: 80.25µs
- RTT Médio **GoRPC**:
  - Merge Sort: 33.208µs
  - Quick Sort: 33.5µs



Resultados

- **N:** 10000
- RTT Médio **gRPC**:
  - Merge Sort: 51.417µs
  - Quick Sort: 50.083µs
- RTT Médio **GoRPC**:
  - Merge Sort: 31.625µs
  - Quick Sort: 32.375µs



Interpretação de resultados

- GoRPC consistentemente supera gRPC em termos de latência para os algoritmos Merge Sort e Quick Sort, especialmente à medida que o tamanho da entrada aumenta;
- A diferença entre os algoritmos é menor do que a diferença entre as tecnologias.