

# Processamento Paralelo e Distribuído

## Relatório

## Trabalho 2

## Ataque de Dicionário - JMS

**Nome:** Wanderson Ralph Silva Vita

**Matrícula:** 2013101887


**Professor:** João Paulo

# Sumário

<b>Processamento Paralelo e Distribuído</b>	<b>1</b>
<b>Relatório</b>	<b>1</b>
<b>Trabalho 2</b>	<b>1</b>
<b>Ataque de Dicionário - JMS</b>	<b>1</b>
Sumário	2
Configurações	3
Executar	3
Compilar	3
Executar Metres	3
Executar Escravo	3
Executar Cliente	3
Para rodar a aplicação siga essa ordem	3
Classes principais	4
Arquitetura	4
SubAttack	4
Guess	5
Análises e Resultados	5
Tempo de Resposta	5
Comparando Trabalhos	7
Adendo	7
Observações	8

## Configurações

As fontes possuem o arquivo de configuração `config.properties` onde pode ser configurado o ip do mestre e do servidor glassfish, nome do mestre e granularidade `m`.

```
trab2 > properties >  config.properties
1  master.name = mestre
2  run.parallel = true
3  m = 1000
4  glassfish.hostname = 172.18.0.3
5  server.hostname = localhost
```

No Makefile, deve-se configurar a variável:

- **JAVA\_PARANS**=-Djava.rmi.server.hostname=localhost

## Executar

O código fonte possui um arquivo `makefile` que permite compilar, e executar o mestre, escravo e cliente.

O Makefile já está configurado com os pacotes `.jar` de dependência.

Compilar

*make*

Executar Metres

*make master*

Executar Escravo

*make slave*

Executar Cliente

*make client*

Para rodar a aplicação siga essa ordem

1. *make*
2. **Terminal 1**
  - 2.1. *cd bin*
  - 2.2. *rmiregistry*
3. **Terminal 2**
  - 3.1. *make master*
4. **Terminal 3 à n**
  - 4.1. *make slave*
5. **Terminal n+1**
  - 5.1. *make client*
    - 5.1.1. Aqui gera um texto de 1Kb criptografado, para teste.

## Classes principais

- Mestre
  - `br.inf.ufes.ppd.Crack.Server.MestreServer`
- Escravo
  - `br.inf.ufes.ppd.Crack.Client.EscravoClient`
- Cliente
  - `br.inf.ufes.ppd.Crack.Client.Cliente`

## Arquitetura

Foi utilizado o Protocol Buffers, para transmitir as mensagens. E a implementação desses, está no diretório “proto”.

O Mestre gera um ID, e filtra o cabeçalho das mensagens de Guess com esse ID. Ao ser enviado uma mensagem Subattack, o mestre envia esse ID, que retorna como cabeçalho das mensagens de Guess. Assim, fica permitido ter vários servidores, e garantido que o mestre não ler mensagens que fiquem de lixo no servidor.

O classe Guess, mostrada abaixo, é enviada com o parâmetro `done = true`, quando o escravo termina um sub ataque. O Mestre que ao iniciar um ataque carrega uma variável com a quantidade de sub ataques, vai decrementando sempre que recebe um `done`. Quando essa variável zera, é feito o retorno para o cliente.

Um problema aqui que não foi tratado, é se um escravo parar de funcionar no meio de um ataque. O mestre não vai receber o `done` deste sub ataque, e como não vai decrementar a variável de count nunca vai zerar. Deixando esse cliente sem resposta.

## SubAttack

```
trab2 > proto > ≡ SubAttack.proto
1  syntax = "proto2";
2
3  package br.inf.ufes.ppd.Crack;
4
5  option java_package = "br.inf.ufes.ppd.Crack";
6
7  message SubAttack {
8      optional bytes ciphertext = 1;
9      optional bytes knowntext = 2;
10     optional int32 initialwordindex = 3;
11     optional int32 finalwordindex = 4;
12     optional int32 attackNumbe = 5;
13     optional int32 serverID = 6;
14 }
```

## Guess

```
trab2 > proto > Guess.proto
1  syntax = "proto2";
2
3  package br.inf.ufes.ppd.Crack;
4
5  option java_package = "br.inf.ufes.ppd.Crack";
6
7
8  message Guess {
9      optional int32 attackNumber = 1;
10     optional int32 currentIndex = 2;
11     optional string key = 3;
12     optional bytes message = 4;
13     optional bool done = 5;
14     optional string nameSlave = 7;
15 }
16
```

## Análises e Resultados

### Tempo de Resposta

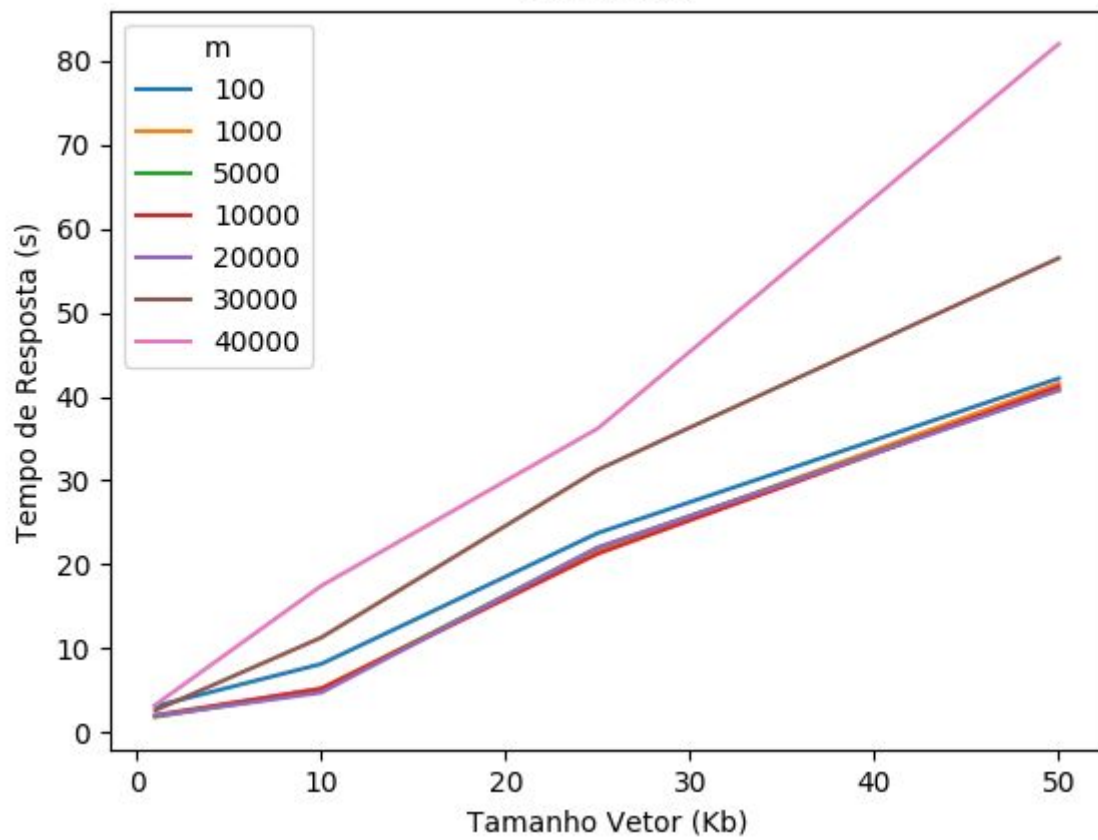
Rodei testes no labgrad com a seguinte configuração:

- Máquinas
  - 1 -> Mestre
  - 2 -> Servidor Glassfish
  - 5, 6, 7 e 8 -> Escravos

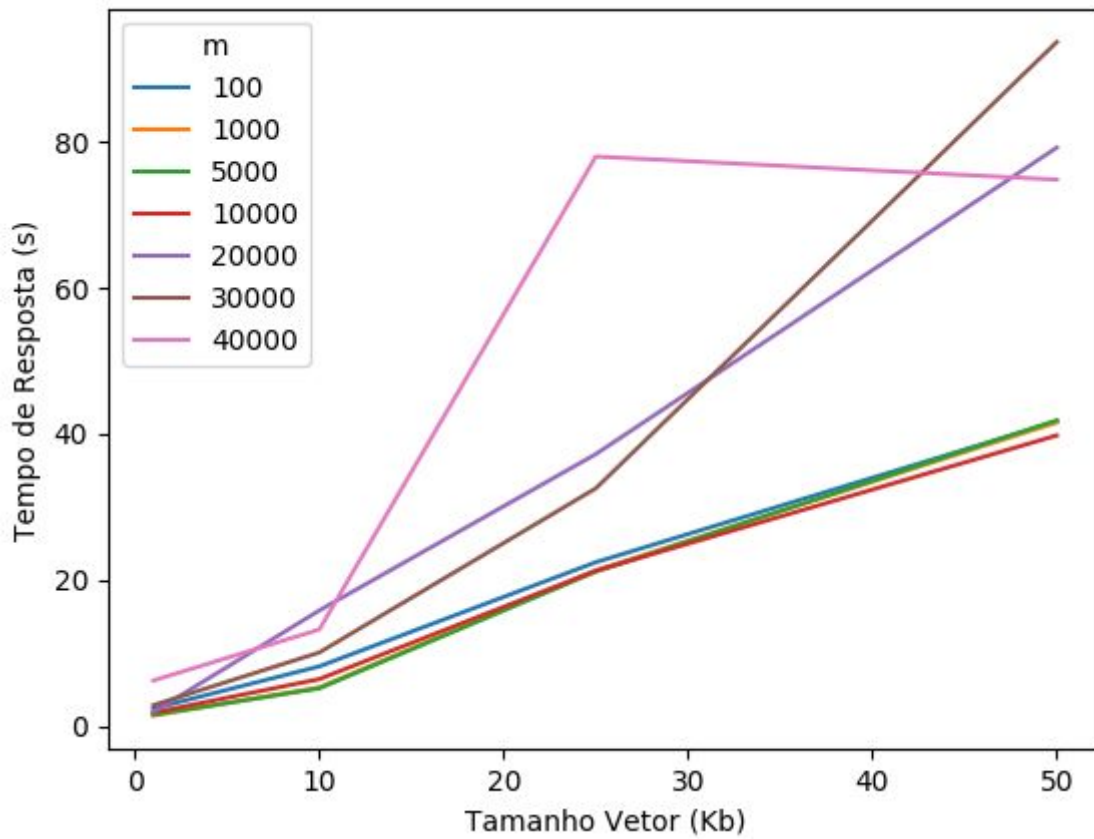
No cenário B, adicionei mais um escravo nas máquinas 5 e 6.

- Tamanho dos arquivos de teste:
  - 1 Kb
  - 10 Kb
  - 25 Kb
  - 50 Kb
- m
  - 100
  - 1000
  - 5000
  - 10000
  - 20000
  - 30000
  - 40000

Cenário A



Cenário B



Vemos que para o cenário A, o  $m$  ideal está entre 1000 e 20000, já para o cenário B,  $m$  ideal fica entre 1000 e 10000.

Para  $m$  pequeno o tempo é levemente maior, por causa do overhead de requisições. Já quando o  $m$  é muito grande, não é feita uma boa distribuição da carga de trabalho. Uma máquina pode ficar ociosa enquanto outra tem muita carga de trabalho, ou uma máquina mais lenta pode acabar pegando um serviço enquanto a mais rápida fica ocioso.

No cenário B, para  $m = 2000$ , o tempo de resposta cresce bastante. Isso pode ser explicado pelo fato de que neste cenário, uma máquina que tem 2 escravos, pode pegar 2 sub ataques de 20000, e segura o processamento fazendo chaveamento de processo.

Já essa reta sem tendência do  $m = 40000$  no caso B, acontece pois no teste de 25Kb, pode ter distribuído mal os escravos, e uma só máquina com 2 escravos, ter pegado os dois sub ataques, enquanto quando foi rodar o teste do 50Kb, a distribuição foi para duas máquinas diferente.

## Comparando Trabalhos

### Adendo

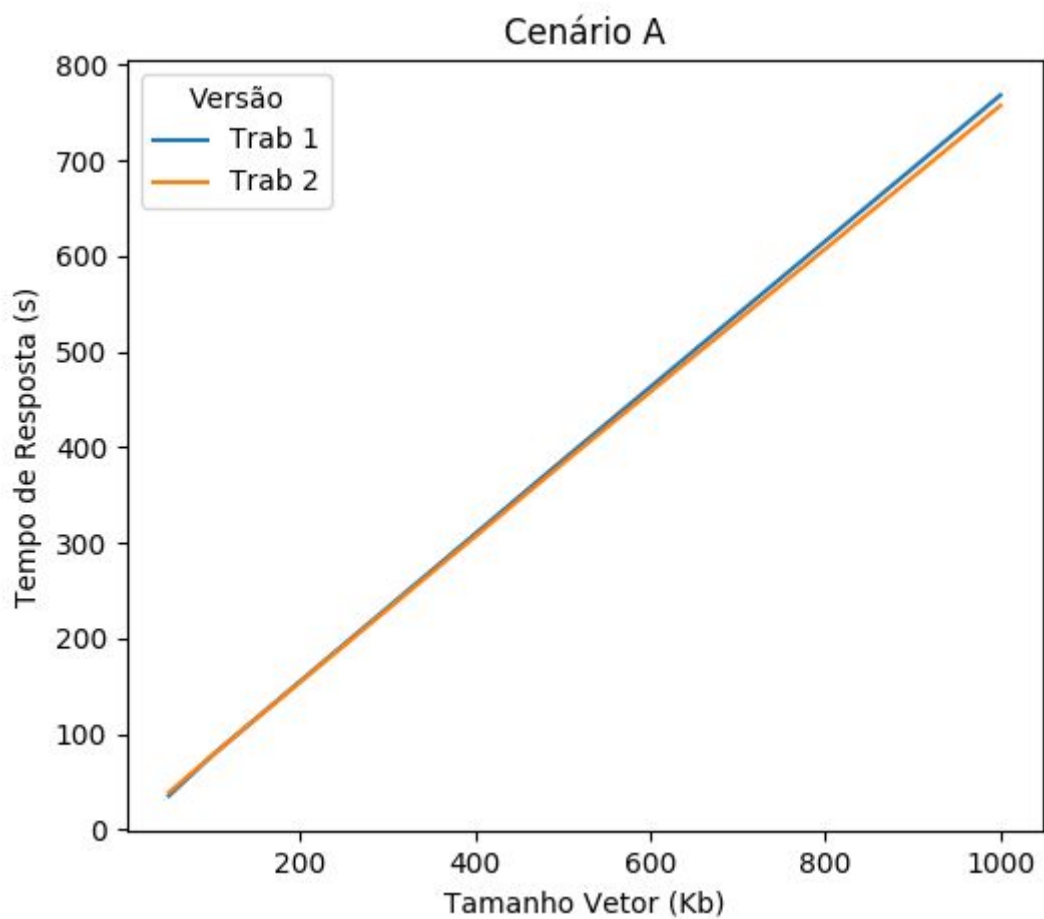
Não consegui rodar o trabalho 1 no labgrad dessa vez. Tentei executar o mesmo código que já estava lá, e que tinha usado na época, para análise do trabalho 1. E ainda tentei subir de novo o código lá, mas sem sucesso. O escravo encontra o mestre e tudo, mas na hora que começa o ataque fica tudo parado. Testei localmente e funcionou perfeitamente, só lá no labgrad que não.

Então resolvi rodar os mesmo testes que rodei no trab1, no trab2 para compará-los. Com isso só consegui compará-los pelo cenário A.

A única coisa que muda aqui, é o tamanho dos arquivos:

- 50 Kb
- 100 Kb
- 500 Kb
- 1 Mb

E a granularidade escolhida foi  $m = 1000$ .



Como no trabalho 1, eu já usava estratégia de dividir em faixas de 1000 itens por sub ataque, o resultado dos dois trabalhos ficou bem parecido.

## Observações

Todos os dados de testes gerados, estão em dois arquivos .csv salvos no diretório 'análise'.