Garbage Collector Impact

David Ferreira Quaresma (david.quaresma@ccc.ufcg.edu.br)

janeiro, 2019

Descrição

Os resultados analisados neste documento foram obtidos através da execução de múltiplas chamadas de uma mesma função de modo sequencial, isto é, não concorrente. Para tal, utilizamos o script **curl-workload** para gerar carga e observar o impacto do coletor de lixo no **listfiller-server**.

Experimento

Função e Ambiente

- Ambiente de execução: **servidor HTTP extraído do OpenFaaS**.
- Lógica de negócio da função: **Inserção de elementos em um ArrayList**.
- Número de inserções: 2 ^ 21, ou 2097152 (aproximadamente 16MB de consumo de memória por requisição).

Setup

- workload: 10.000 requisições enviadas sequencialmente.
- jvm: openjdk version "11"
- gc: Garbage First Garbage Collector (G1 GC)
- heap: 512mb
- taskset: 2 CPUs

Dados observados

- Quantidade de coletas de lixo durante execução da função.
- Duração das coletas de lixo durante execução da função.
- Tempo de execução da função.

Observações:

- Scavenge: coleta na Young Gen.
- MarkSweep: coleta na Old Gen.
- Warmup: removemos as 100 primeiras requisições

Resultados

```
results = read.csv("./results/listfiller-21p-512h-j11.csv", header=T, dec=".")
results = tail(results, -100)
nocollect <- filter(results, scavenge_count + marksweep_count == 0)
withcollect <- filter(results, scavenge_count + marksweep_count > 0)
```

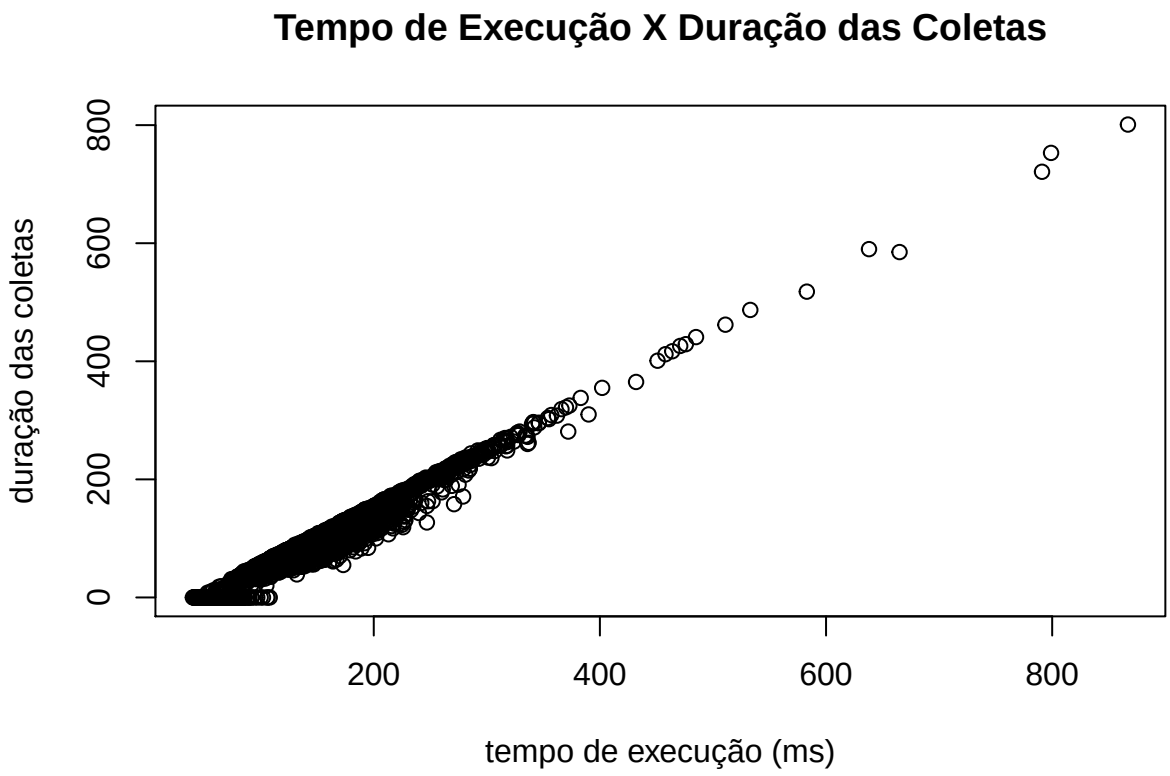
Número de coletas

```
coletas_dataframe(results)
```

```
##                   tipo coletas
## 1 scavenge (young gen)   10014
## 2  marksweep (old gen)       2
```

Scatterplot

```
time_collecting = results$scavenge_time + results$marksweep_time
plot(results$execution_time, time_collecting, xlab="tempo de execução (ms)",
     ylab="duração das coletas", main="Tempo de Execução X Duração das Coletas")
```



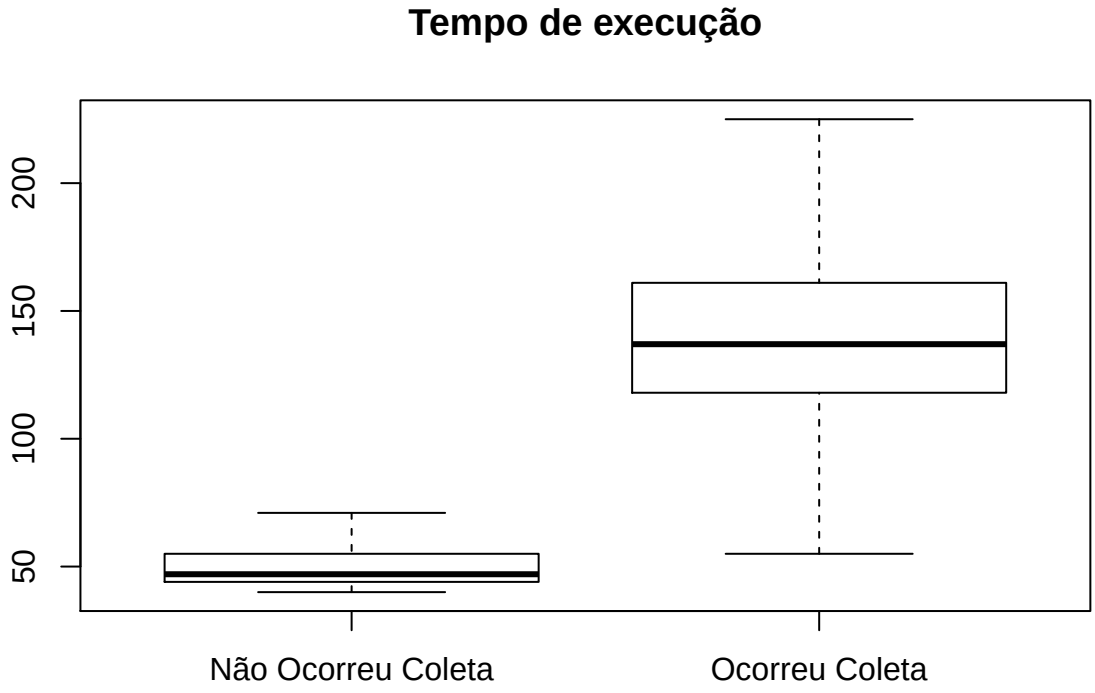
Correlação Linear

```
cor_dataframe(results)
```

```
##                                correlacao     valor
## 1 Tempo de serviÃ§o X nÃºmero de coletas 0.8367656
## 2    Tempo de serviÃ§o X tempo coletando 0.9850893
```

Boxplot

```
boxplot(nocollect$execution_time, withcollect$execution_time, outline=FALSE,
        names=c("Não Ocorreu Coleta", "Ocorreu Coleta"), main="Tempo de execução")
```



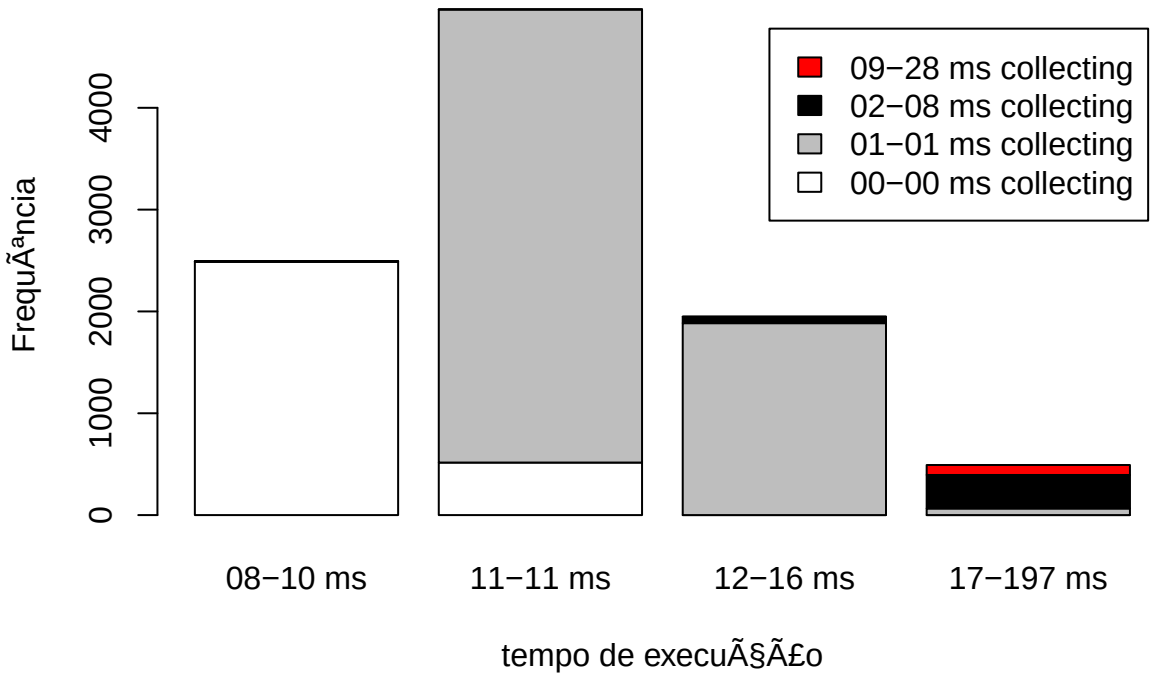
Média, Variância e Desvio Padrão

```
meanVarSd(nocollect, withcollect)
```

```
##   statistic noCollect withCollect comparison
## 1      mean 51.034609   145.08122   2.842801
## 2       var 83.956192  1964.98702  23.404909
## 3        sd  9.162761    44.32817   4.837862
```

Barplot

```
build_barplot(results, running_names = c("08-10 ms", "11-11 ms", "12-16 ms", "17-197 ms"),
              collecting_names = c("00-00 ms collecting", "01-01 ms collecting",
                "02-08 ms collecting", "09-28 ms collecting"))
```



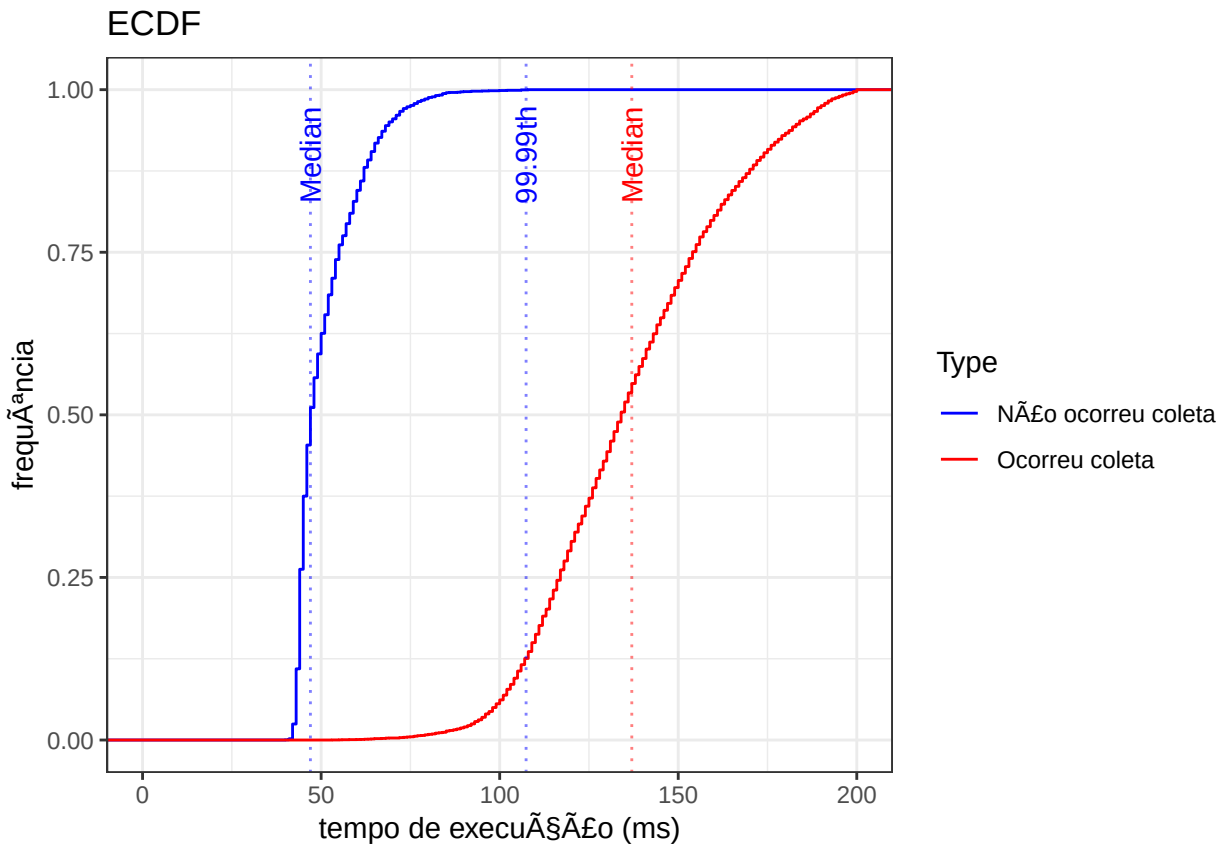
ECDF

```
graph_tail(nocollect$execution_time, withcollect$execution_time,
           title="ECDF", x_limit_inf=0, x_limit_sup=200, annotate_y=0.90)
```

```
## Warning: Removed 492 rows containing non-finite values (stat_ecdf).
```

```
## Warning: Removed 1 rows containing missing values (geom_text).
```

```
## Warning: Removed 1 rows containing missing values (geom_vline).
```



Quantiles

Tempo coletando

```
quantile_wrapped(time_collecting)
```

```
##       0%      25%      50%      75%      90%      95%      99%    99.9%
##   0.0000   0.0000  67.0000  91.0000 119.0000 139.0000 229.0000 426.3030
##   99.99%  99.999%     100%
## 753.4848 796.2485 801.0000
```

Tempo executando

Com e Sem Coleta

```
quantile_wrapped_for_execution_time(results)
```

```
##       0%      25%      50%      75%      90%      95%      99%    99.9%
##  40.0000  59.0000 120.0000 149.0000 178.0000 200.0000 281.0000 471.5050
##   99.99%  99.999%     100%
## 799.6868 860.2687 867.0000
```

Comparação: Sem coleta X Com coleta

```
quantiles_dataframe_comparison(nocollect, withcollect)
```

```
##         nocollect withcollect comparison
## 0%        40.0000     53.0000   1.325000
## 25%       44.0000    118.0000   2.681818
## 50%       47.0000    137.0000   2.914894
## 75%       55.0000    161.0000   2.927273
## 90%       64.0000    189.0000   2.953125
## 95%       69.0000    217.0000   3.144928
## 99%       82.0000    299.0600   3.647073
## 99.9%    101.9920    513.3320   5.033061
## 99.99%   107.3992    820.1208   7.636191
## 99.999%  107.9399    862.3121   7.988815
## 100%     108.0000    867.0000   8.027778
```

Comparação: Sem coleta X Com coleta

```
print_summary_table("Thumbnailator", nocollect, withcollect)
```

```
## Latency(ms)  Thumbnailator NC  avg: 51 51  | 50: 47 48  | 95: 68 71  | 99: 80 84  | 99.9: 94.33 109.5
## Latency(ms)  Thumbnailator C   avg: 140 150  | 50: 136 138  | 95: 212 223  | 99: 286.9 314.5  | 99.9
```