Usando teoria dos grafos para aumentar a performance de um back-end em 300%

Leonardo Brito

Full Stack @ SensorTower

SensorTower

We're Hiring:

Come Work With Us!

Open Positions:

Full Stack Engineer
(Remote Opportunities Available)

DevOps Engineer
(Remote Opportunities Available)

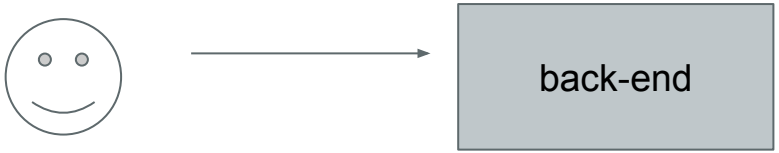
email: recruiting@sensortower.com

Roteiro

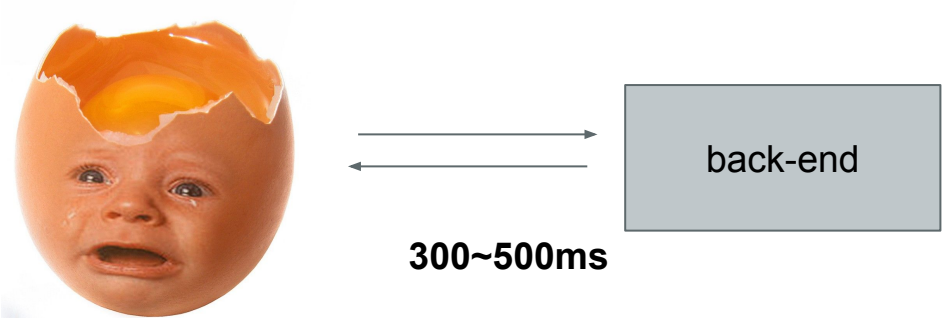
- Motivação
- Otimizações mais simples
- Entenda o problema
- Primeira otimização
- Segunda otimização
- Resultados finais
- Conclusão
- Perguntas

Motivação

Motivação



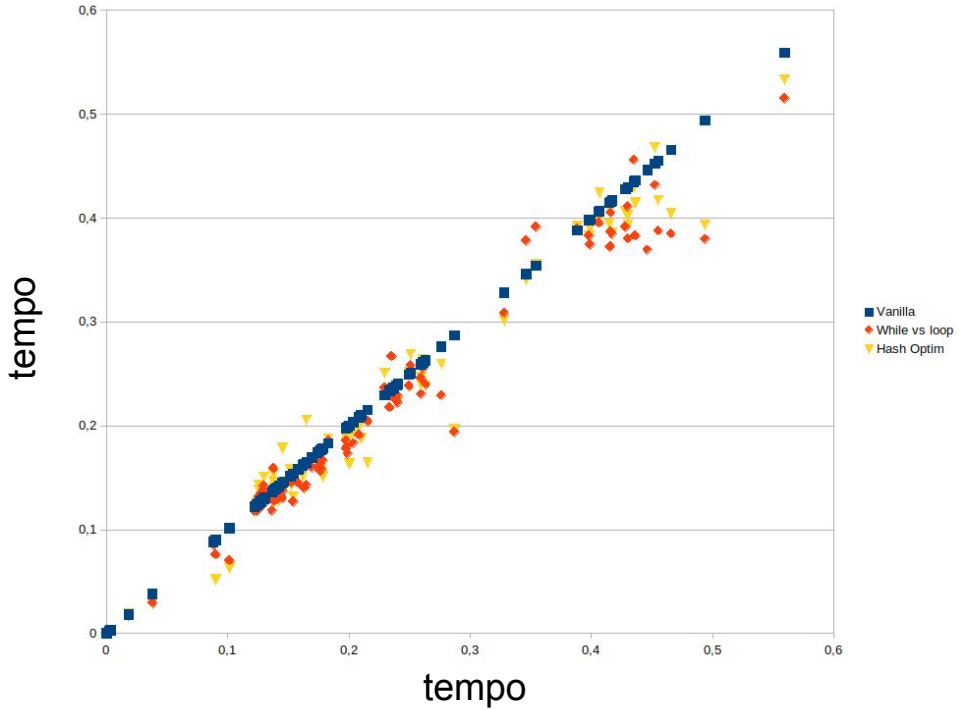
Motivação



Otimizações mais simples

Otimizações mais simples

Otimizações mais simples

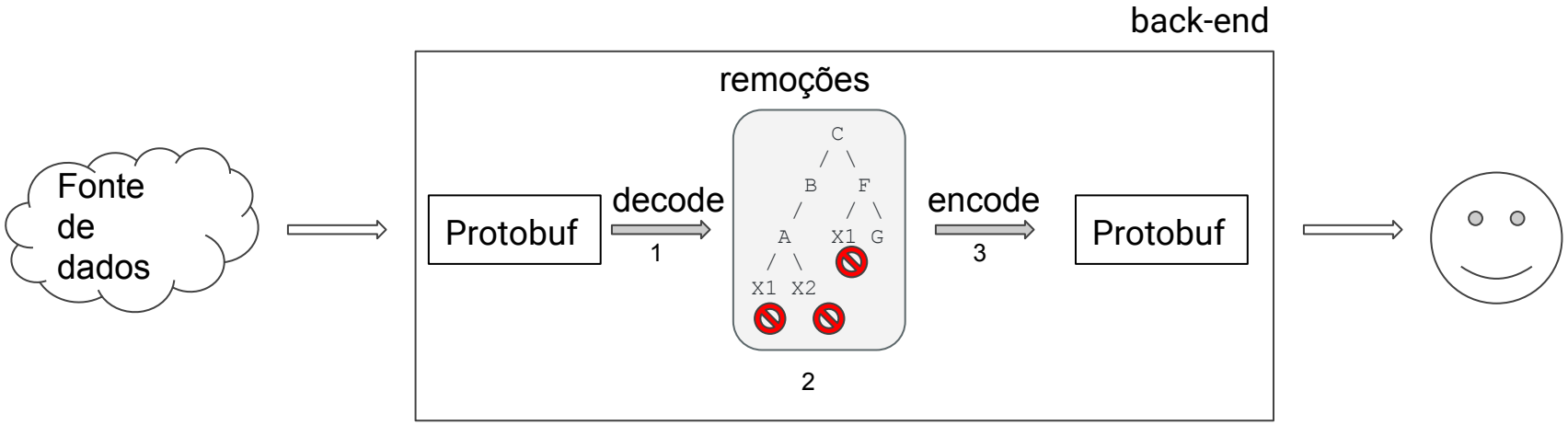


Entenda o problema

Um passo para trás: entenda o problema

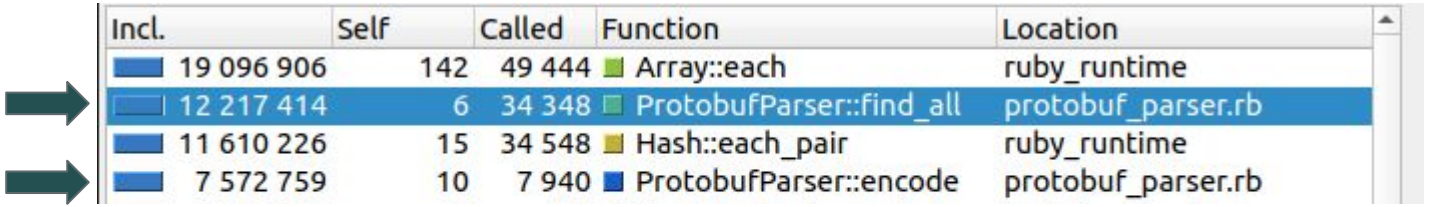
1. Leitura exaustiva: código e testes
2. Desenhar fluxogramas
3. Simular a execução com debugger
4. Q&A com alguém experiente no código

Um passo para trás: entenda o problema



Um passo para trás: entenda o problema

- Profiling

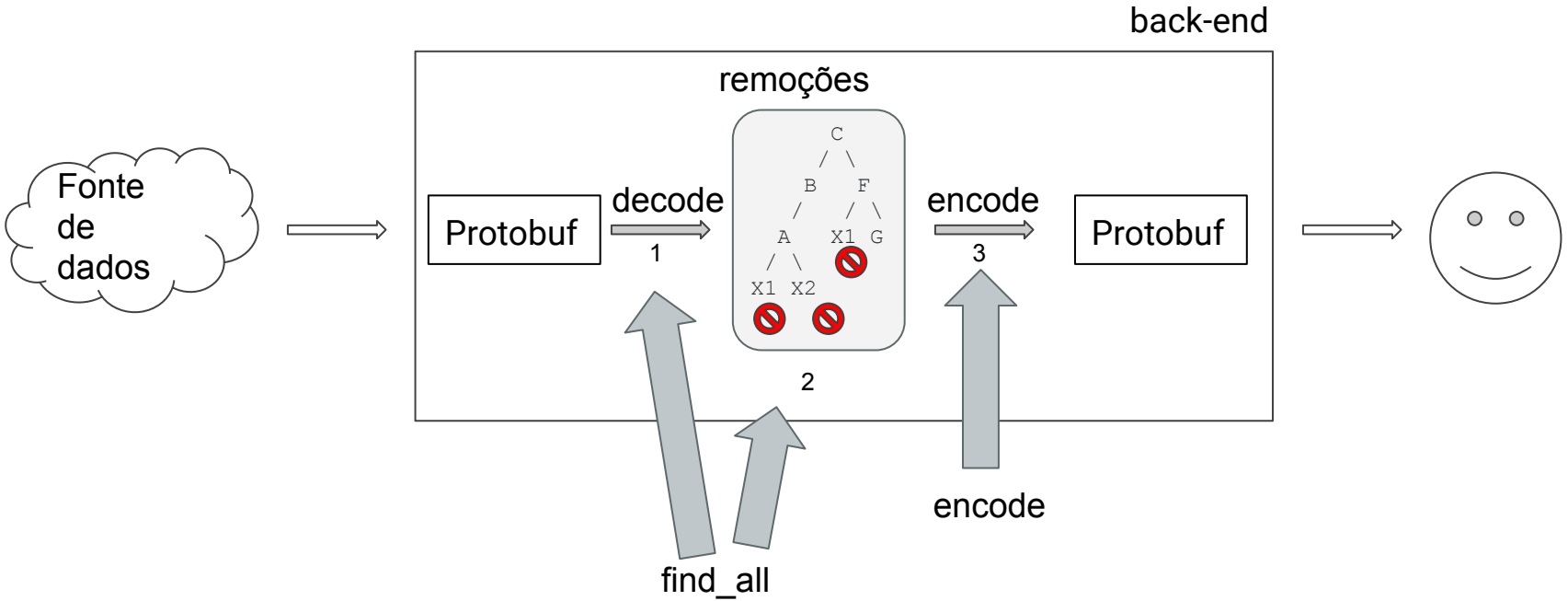


Incl.	Self	Called	Function	Location
19 096 906	142	49 444	Array::each	ruby_runtime
12 217 414	6	34 348	ProtobufParser::find_all	protobuf_parser.rb
11 610 226	15	34 548	Hash::each_pair	ruby_runtime
7 572 759	10	7 940	ProtobufParser::encode	protobuf_parser.rb

Um passo para trás: entenda o problema

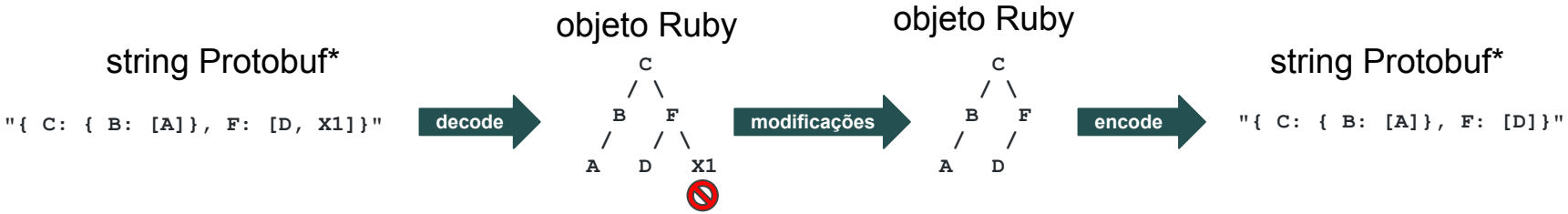
```
Total time:        0.250000   0.000000   0.250000 (  0.251023)
First find_all:    0.140000   0.000000   0.140000 (  0.143514)
Second find_all:   0.010000   0.000000   0.010000 (  0.011192)
Encode time:       0.100000   0.000000   0.100000 (  0.094486)

---

1st find_all: 56%
2nd find_all: 4%
Encode: 37%
Subtotal: 97%
```

Um passo para trás: entenda o problema



Primeira otimização

Primeira otimização: encode



* Uma string Protobuf de verdade é assim:

```
f1: "dsfadsafsaf"
f2: 234
fa: 2342134
fa: 2342135...
```

Primeira otimização: encode

- Será que a árvore **inteira** é modificada?
  - Não!
  - Porém, o **encode** é feito para a árvore inteira

Primeira otimização: encode

strings geradas pelo **decode**

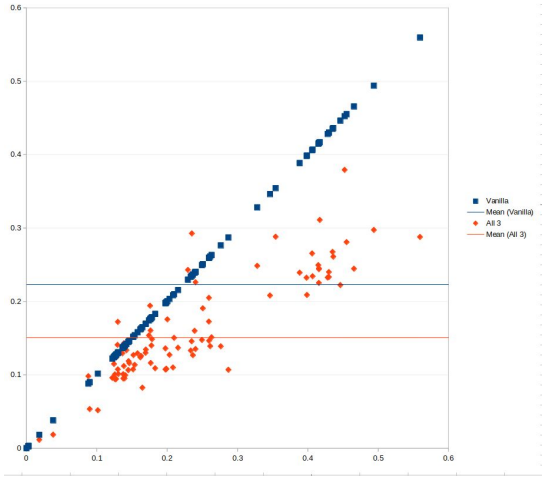
```
C: "{ C: { B: [A]}, F: [D, X]}"
B: "{ C: { B: [A]}, F: [D, X]}"
A: "{ C: { B: [A]}, F: [D, X]}"
F: "{ C: { B: [A]}, F: [D, X]}"
D: "{ C: { B: [A]}, F: [D, X]}"
X: "{ C: { B: [A]}, F: [D, X]}"
```

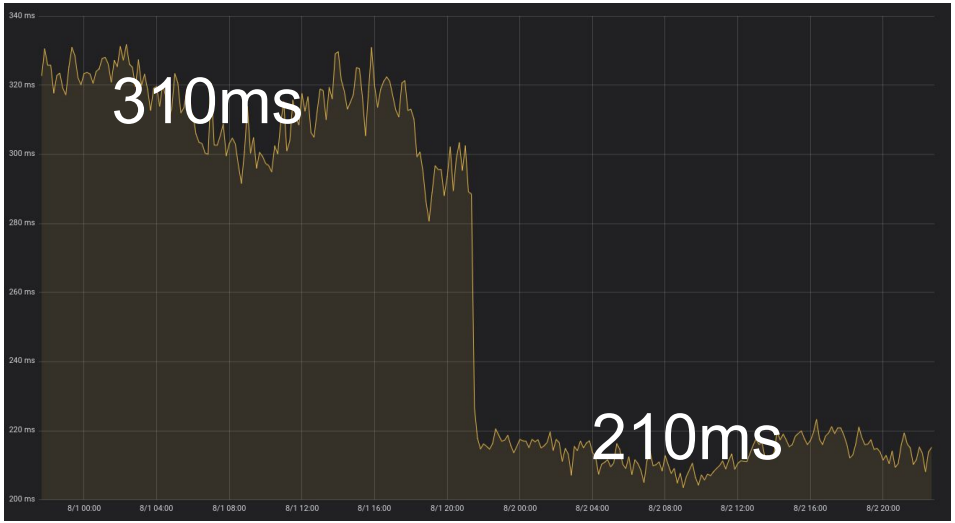
strings geradas pelo **encode**

```
C: "{ C: { B: [A]}, F: [D]}"
B: "{ C: { B: [A]}, F: [D]}"
A: "{ C: { B: [A]}, F: [D]}"
F: "{ C: { B: [A]}, F: [D]}"
D: "{ C: { B: [A]}, F: [D]}"
```

Primeira otimização: resultados

- ~30% de melhora do desempenho

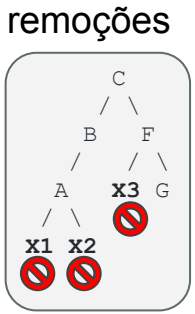




Segunda otimização

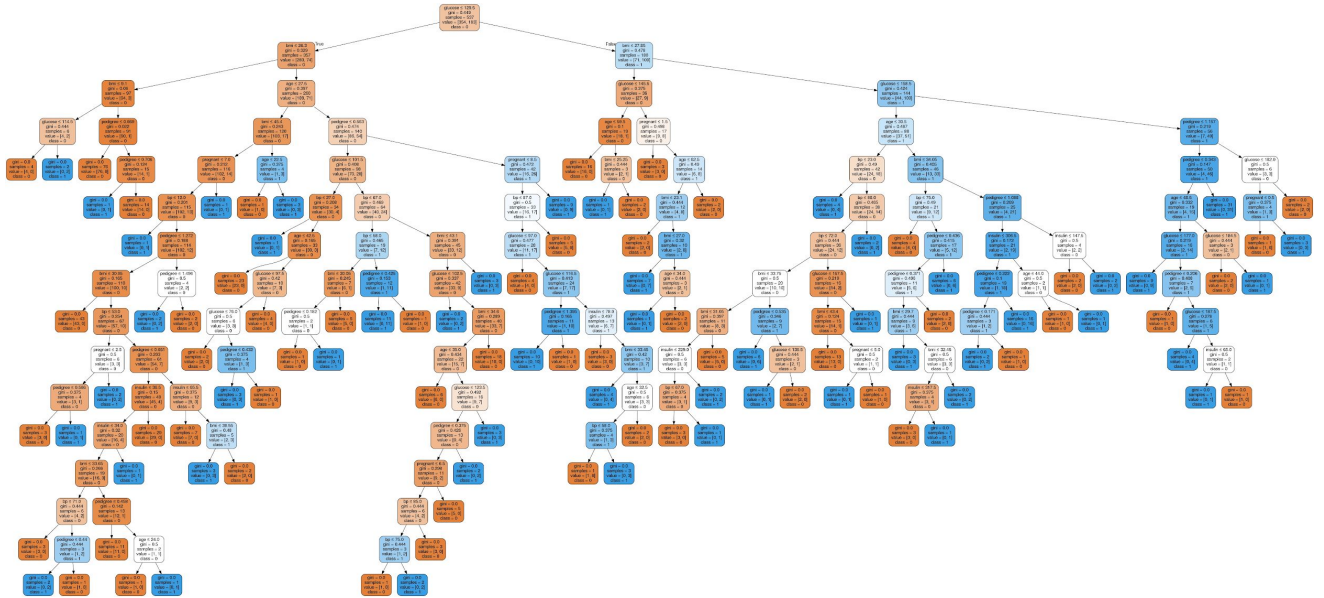
Segunda otimização: find_all

- Cadê os 300%???????????
- find_all:
  - Decodifica uma string Protobuf → objeto Ruby
  - Busca elementos (tree traversal) para remoções na próxima etapa



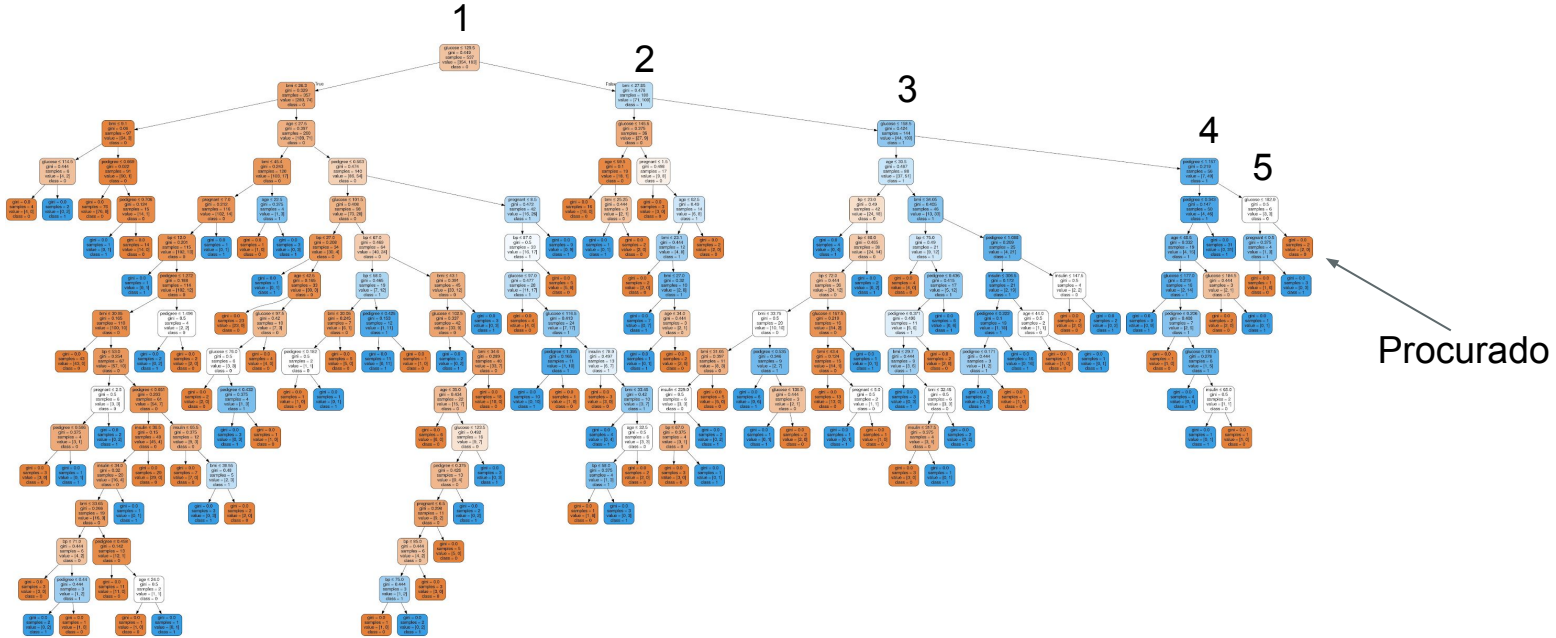
Segunda otimização: find_all

- Atravessar a árvore inteira:
  - Tempo **linear** (complexidade O(n))
  - Cresce de acordo com **quantidade de nós** da árvore



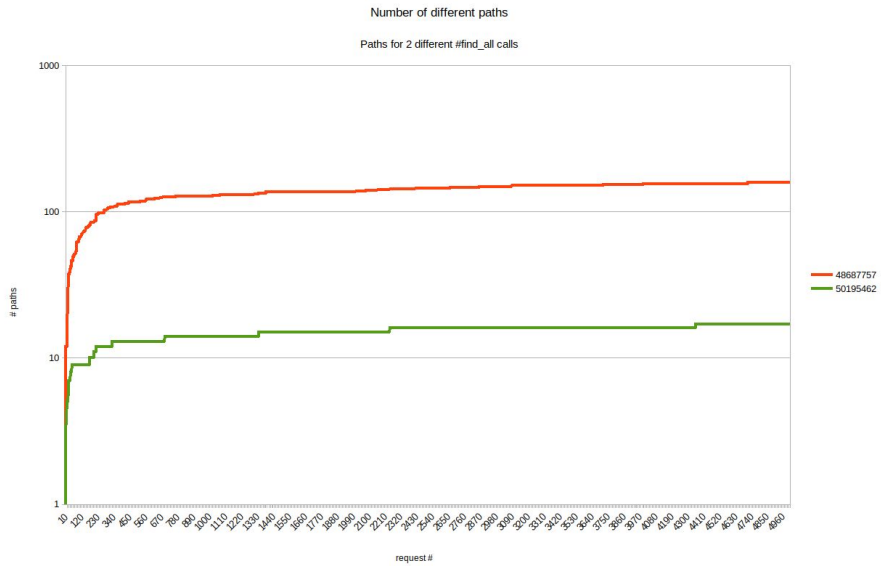
Segunda otimização: find_all

- Atravessar um caminho até o nó:
  - Tempo **logarítimico** (complexidade O(log n))
  - Cresce de acordo com **profundidade** da árvore



Segunda otimização: find_all

- Os caminhos se repetem bastante
  - Logo, podem ser reutilizados
- Podemos estimar o número de caminhos diferentes que existem!



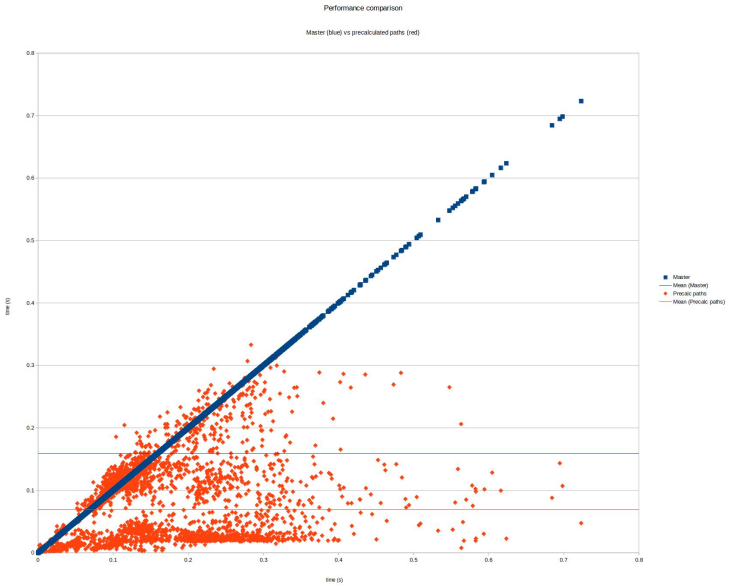
Segunda otimização: find_all

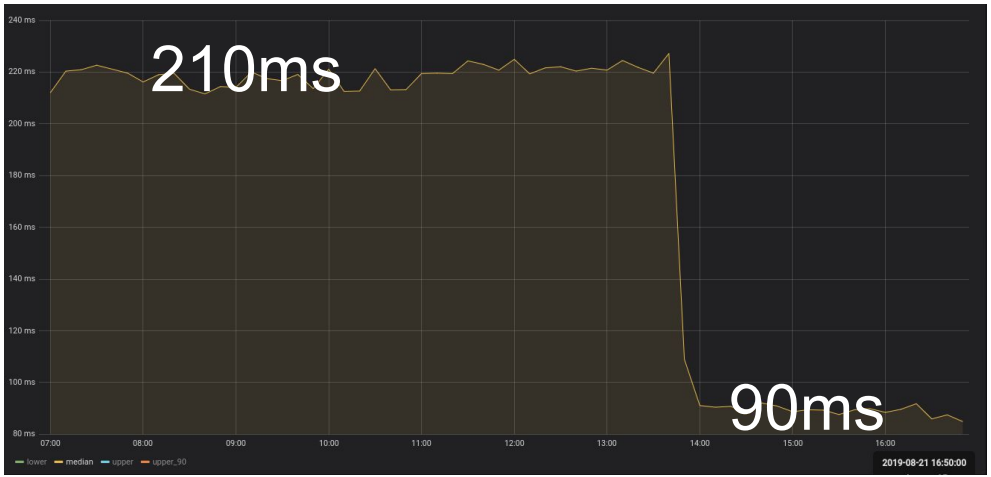
- Descobrimos que:
  - Ter um caminho da raiz até o nó == achar o nó rapidamente (O(log n))
  - Os caminhos até os nós que queremos remover se repetem muito
  - A quantidade de caminhos diferentes aumenta muito lentamente

Segunda otimização: find_all

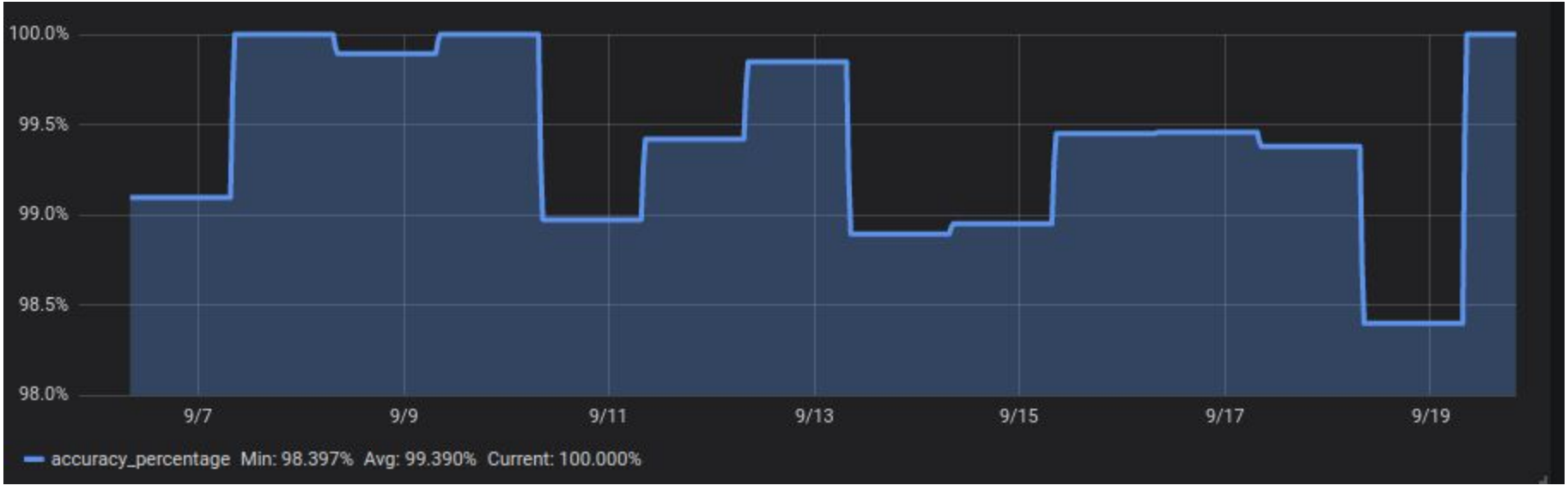
- Descobrimos que:
  - Ter um caminho da raiz até o nó == achar o nó rapidamente (O(log n))
  - Os caminhos até os nós que queremos remover se repetem muito
  - A quantidade de caminhos diferentes aumenta muito lentamente
- Juntando essas informações:
  - Cache!

Segunda otimização: resultados

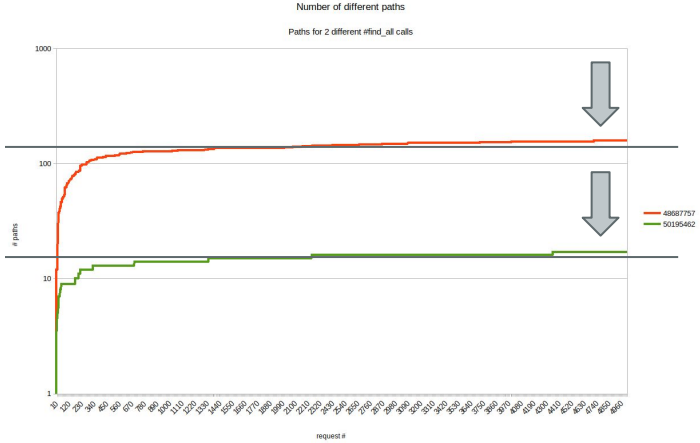




Segunda otimização: trade-offs



Segunda otimização: trade-offs



Resultados finais

Resultados finais

Conclusão

Conclusão

- Procure por trabalho repetido que pode ser evitado
- Reflita se seu sistema pode acertar "só" 99% das vezes
- Reflita se você pode ser mais assíncrono
- **Entenda o problema que quer resolver**

Perguntas