Medindo o Desempenho de Código JavaScript: Experimentos com Diferentes Algoritmos e Interpretadores

Lucas Pacheco Silveira[1], **Gustavo Rissetti**[1], **Andrea S. Charão**[2]

[1] Instituto Federal Farroupilha – Campus São Vicente do Sul

[2]Universidade Federal de Santa Maria

Resumo. *JavaScript é uma linguagem popular em aplicações web, sujeita a variações de desempenho dependendo do interpretador utilizado. Neste trabalho, investiga-se esta questão, realizando-se testes de execução de três algoritmos diferentes para uma mesma tarefa, em diferentes interpretadores, usando a API High Resolution Time. Os resultados foram notavelmente diferentes, reforçando a importância de testes de desempenho em aplicações JavaScript.*

1. Introdução

JavaScript é uma linguagem de programação interpretada e amplamente empregada para o desenvolvimento de aplicações web [Richards et al. 2010]. Sua utilização permite aos desenvolvedores a criação de páginas web dinâmicas, com efeitos visuais e animações que enriquecem a interação com o usuário.

O código JavaScript, para ser executado, necessita de um interpretador que consiga transformar o código escrito em JavaScript e executá-lo corretamente. Existem diferentes interpretadores embutidos em navegadores web modernos, utilizando diferentes técnicas de interpretação e otimização. Isso pode levar a um desempenho diferente para um mesmo código JavaScript, em diferentes interpretadores.

Neste trabalho, investiga-se esta questão quantitativamente, por meio de medições de tempos de execução. Os experimentos são descritos e discutidos nas seções 3 e 4, após uma breve apresentação da linguagem JavaScript, na seção 2.

2. Linguagem JavaScript

JavaScript é uma linguagem orientada a objetos projetada em 1995 por Brendan Eich, na época trabalhando no desenvolvimento do navegador Netscape, para permitir a expansão de *sites* web com códigos executáveis do lado do cliente (navegador do usuário), exigindo menos processamento na parte do servidor [Richards et al. 2010].

Praticamente todos os navegadores web atuais suportam a linguagem JavaScript. Os interpretadores utilizados podem ser divididos em dois níveis: o nível mais básico, que apenas recebe o código e o executa diretamente; e o de nível mais aprimorado, para trabalhar com *layouts*, que recebe o código JavaScript, transforma-o em *bytecode*, retorna-o para o compilador que realiza uma análise e otimiza o código para mostrá-lo no navegador ou aplicação [Garsiel e Irish 2011].

Os principais navegadores web possuem seus próprios motores JavaScript com suas otimizações. Dentre eles, estão os motores JavaScript V8, SpiderMonkey e Chackra. A Google desenvolve o motor JavaScript V8, escrito em C++ e anexado ao seu

navegador Google Chrome, que pode ser utilizado separadamente e anexado a outras aplicações [Google 2015]. O motor SpiderMonkey é desenvolvido em C/C++, pela fundação Mozilla e utilizado amplamente pelos seus produtos, como o navegador Mozilla Firefox. O motor Chackra, desenvolvido pela Microsoft em C++, é utilizado em seu navegador Microsoft Edge e outras plataformas [Microsoft 2016].

Quando se deseja avaliar o desempenho de um código em JavaScript por meio de experimentos com medições de tempo, deve-se levar em conta que os resultados podem ser bastante diferentes após a primeira execução, pois o navegador otimiza a resposta para funções já executadas. Neste caso, usar a média dos tempos observados pode levar a resultados incorretos, então sugere-se que seja utilizada a mediana [Bengtsson 2015], obtendo assim um resultado mais confiável.

3. Experimentos

Uma função em JavaScript pode ser implementada e executada de diversas maneiras, porém produzindo o mesmo resultado. A escolha do melhor algoritmo pode ser feita utilizando os testes de desempenho para observar qual se comporta de maneira mais eficiente, exigindo menos da máquina do cliente. Neste trabalho, utiliza-se como referência o problema de encontrar a soma de todos os números que são divisíveis por 3 e 5 entre 0 e um número N.

Para realizar medições de tempo, foi utilizada uma Interface de Programação de Aplicativos (API) chamada High Resolution Time (tempo de alta resolução) que fornece o tempo atual em resolução de sub-milissegundos sem interferência da hora que está no relógio do sistema. Foi escolhido utilizar esta ferramenta pois ela apresenta valores precisos e é de fácil utilização. O ambiente de teste utilizado foi um computador com configuração de um processador i7-3610QM com quatro núcleos, com frequência de 2.3 GHz e 8 GB de memória RAM DDR3. Os motores JavaScript utilizados foram os anteriormente citados, que acompanham os navegadores Google Chrome, Mozilla Firefox e Microsoft Edge.

Foram utilizados três algoritmos com lógicas diferentes, porém retornando o mesmo valor e executando a mesma tarefa (adaptados do trabalho de [Bates 2016]). Todos os algoritmos foram testados no mesmo ambiente de sistema e sobre as mesmas abordagens de teste de desempenho. Realizou-se a medição de tempo de execução dez vezes e obteve-se o valor da mediana destes valores, para evitar problemas devido às otimizações que os navegadores realizam.

O primeiro algoritmo testado, mostrado na Figura 1, foi desenvolvido para primeiro verificar se o número é divisível por três ou cinco. Caso isto aconteça, ele o armazena em um vetor para posteriormente somar todos os valores do vetor e retornar o resultado. O segundo algoritmo, mostrado na Figura 2, usa um laço de repetição mas não emprega vetores para guardar os valores que são divisíveis por 3 e 5, armazenando a soma de todos os valores em uma variável que é retornada ao final da função. Como terceira opção, decidiu-se encontrar algum modo de calcular estes valores sem utilizar laço de repetição e verificação de condições. Para isso, no algoritmo da Figura 3, realiza-se apenas cálculos de divisão e multiplicação sobre os valores que são divisíveis por três e cinco.

```
function arrayPushSoma(n) {
   var array = [];
   var resultado = 0;
   for (var i = 1; i < n; i ++) {
      if (i % 3 == 0 || i % 5 == 0) {
         array.push(i);
      }
   }
   for (var num of array) {
      resultado += num;
   }
   return resultado;
}
```

Figura 1: Primeiro algoritmo

```
function whileSoma(n) {
   var soma = 0;
   while (n >= 1) {
      n--;
      if (n%3==0||n%5==0) {
         soma += n;
      }
   }
   return soma;
}
```

Figura 2: Segundo algoritmo

```
function multSilgarth(N) {
   var tres = Math.floor(--N / 3);
   var cinco = Math.floor(N / 5);
   var quinze = Math.floor(N / 15);
   return (3 * tres * (tres + 1)   +
           5 * cinco * (cinco + 1) -
           15 * quinze * (quinze + 1))/2;
}
```

Figura 3: Terceiro algoritmo

4. Resultados

Nos experimentos com o primeiro algoritmo, observou-se pouca diferença entre os navegadores da Mozilla e Google, porém o navegador da Microsoft mostrou-se muito mais lento. Os motores JavaScript dos navegadores da Mozilla e Google obtiveram os resultados, respectivamente, de 11,68 e 15,98 ms de tempo de execução, enquanto o navegador Edge demonstrou-se onze vezes mais lento comparado ao Chrome, com 172,60 ms.

Nos experimentos com o segundo algoritmo, sem a utilização de vetores, novamente obteve-se resultados semelhantes entre os 2 primeiros navegadores, porém este segundo algoritmo foi em média cinco vezes mais rápido que o algoritmo anterior. Os tempos obtidos para os navegadores Chrome, Mozilla e Edge foram, respectivamente,

3,08, 2,54, 72,22 ms. Novamente, o navegador Edge mostrou-se mais lento, enquanto o motor JavaScript da Mozilla mostrou-se mais rápido que os demais.

Nos experimentos com o terceiro algoritmo, que não utiliza nenhum laço de repetição, vetores ou condições, observou-se grande diferença dos resultados em comparação com os algoritmos anteriores. Este último algoritmo é notavelmente mais eficiente, porém usa uma lógica menos trivial. Os tempos nos navegadores Google Chrome e Mozilla Firefox foram idênticos, com 0,005 ms para retornar o resultado, enquanto o navegador Edge, diferentemente dos testes anteriores, mostrou-se ligeiramente mais rápido ao executar o terceiro algoritmo, com tempo de execução igual a 0,0049 ms. Sem a utilização da API High Resolution Time, esta diferença poderia não ser perceptível.

De forma geral, observou-se uma grande diferença entre os tempos de execução dos três algoritmos analisados. Notou-se uma diferença considerável entre os motores JavaScript de cada navegador. Os motores JavaScript da Google e da Mozilla demonstraram facilidade ao trabalhar com laços de repetições e condições, enquanto o motor da Microsoft obteve resultados ligeiramente melhores no algoritmo sem essas estruturas.

5. Conclusão

Nos experimentos realizados, os primeiros algoritmos demonstraram diferenças significativas entre os navegadores, enquanto o último algoritmo analisado, além de ser 3440 vezes mais rápido no navegador Edge, obteve resultados semelhantes entre todos os navegadores, garantindo execução rápida em todos eles.

Assim, os resultados dos experimentos realizados neste trabalho reforçam a importância de se considerar aspectos de desempenho no desenvolvimento em JavaScript. Enquanto alguns navegadores possuem melhores otimizações para alguns recursos da linguagem, outros apresentam dificuldades para realizar as mesmas tarefas, então é importante levar isto em consideração para se desenvolver códigos melhores.

Referências

Bates, J. (2016). What i learned from writing six functions that all did the same thing. Disponível em: http://126kr.com/article/6ybscyr2o2d.

Bengtsson, P. (2015). Measuring JavaScript functions' performance. Disponível em: https://www.sitepoint.com/measuring-javascript-functions-performance/.

Garsiel, T. e Irish, P. (2011). How browsers work: Behind the scenes of modern web browsers. Disponível em: https://www.html5rocks.com/en/tutorials/internals/howbrowserswork/.

Google (2015). Chrome v8: Google's high performance, open source, javascript engine. Disponível em: https://developers.google.com/v8/.

Microsoft (2016). Chakracore: the core part of the chakra javascript engine that powers microsoft edge. Disponível em: https://github.com/Microsoft/ChakraCore.

Richards, G., Lebresne, S., Burg, B., e Vitek, J. (2010). An analysis of the dynamic behavior of javascript programs. *SIGPLAN Not.*, 45(6):1–12.