Estudo Comparativo entre Ferramentas de Automação Web via RPA

Paulo Salgado Filho*
Carlos Ferraz*
pdssf@cin.ufpe.br
cagf@cin.ufpe.br
Universidade Federal de Pernambuco
Recife, Pernambuco, Brasil

Abstract

The world is experiencing an unprecedented digitalization process. Digital solutions to everyday problems are nothing new, but the COVID-19 pandemic leveraged them in an almost imposing way. The legal process, medical consultations, and even much of the traditional office work today have some digital correlation, and in the digital world, robots can automate several examples of repetitive work. The term RPA (Robotic Process Automation) is used to designate software that performs tasks previously done manually. Currently, various tools allow for automated workflows on the internet, and many companies are built around these solutions. Two of these tools are well known: Puppeteer and Selenium. This work conducted a comparative study between these two technologies, pointing out positive and negative aspects and showing situations where there may be an advantage for one or the other. The results obtained in two different experiments show that in case 1, which deals with the automation of filling out forms on the Web, the tools perform in a statistically equivalent way, while in case 2, referring to data extraction and report generation, Puppeteer presents significantly better results. This work verifies that among the factors that can be decisive in the performance difference is how the interface between the written code and the Web browser is made.

Keywords: RPA, automation, Puppeteer, Selenium

1 Introdução

A computação tem se tornado cada vez mais presente no cotidiano das pessoas e empresas. Atualmente, são raros os casos de indústrias ou comércios que funcionem sem um auxílio do mundo digital, seja através de websites, software para controle de recursos, gerenciamento de pessoas, meios de pagamentos, a lista é enorme e a tendência é de crescimento. Uma outra forma, não tão conhecida, mas extremamente útil para as empresas é a automação de processos, ou como é mais conhecida, Automação Robótica de Processos. Como o nome sugere, a prática busca automatizar um procedimento, antes feito manualmente, através do emprego de robôs.

O termo RPA (*Robotic Process Automation*) é usado para designar software que executam tarefas feitas anteriormente de forma manual [5], isto é, os robôs são programados para serem capazes de usar a mesma interface que seres humanos, simulando as ações de um operador do sistema, eliminando assim a necessidade de desenvolvimento de *APIs* ou da realização de grandes mudanças nos sistemas já utilizados. Normalmente o principal objetivo da aplicação da automação de processos é eliminar trabalhos repetitivos e padronizados, muito comuns em escritórios [11].

Atualmente existem dezenas de software voltados para a automação de processos. Este trabalho busca comparar, de forma quantitativa, avaliando o consumo de recursos de CPU e memória, duas bibliotecas de código aberto (*open-source*) voltadas para a automação web, isto é, automação de processos executados em um navegador, tais quais: preenchimento de formulários, extração de dados de páginas da web, teste de websites, etc. São elas: *Selenium* [7] e *Puppeteer* [6]; e avaliar em que situações possuímos alguma vantagem na escolha de uma determinada biblioteca em favor de outra. Para isto, simulamos rotinas de preenchimento de formulários e extração de dados usando ambas as ferramentas e comparamos os resultados obtidos, que mostraram valores estatisticamente semelhantes no consumo de recursos para a situação de preenchimento de formulários, porém foi observado uma grande diferença para a extração de dados.

2 Fundamentação

Para a realização da comparação entre as ferramentas [8], os robôs foram desenvolvidos usando cada uma delas, de modo que cada robô realize as mesmas tarefas, seguindo os mesmos passos, na mesma ordem e com os mesmos parâmetros. Foram considerados dois cenários neste experimento com as principais aplicações de RPA: (1) a automação de preenchimento de formulários e (2) a extração de dados

*Both authors contributed equally to this research.

In: XXI Workshop de Ferramentas e Aplicações (WFA 2022), Curitiba, Brasil. Anais Estendidos do Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia). Porto Alegre: Sociedade Brasileira de Computação, 2022.

ISSN 2596-1683

da web. As rotinas desenvolvidas foram inseridas em contêineres Linux contendo apenas o mínimo necessário para a execução do processo, assim pôde-se fazer um monitoramento mais adequado com relação ao consumo de CPU, consumo de memória e I/O de rede. Cada rotina desenvolvida foi executada 30 vezes, e os dados de consumo de recurso foram colhidos usando a ferramenta de leitura nativa do *Docker*. Foi então feito um estudo do consumo médio de CPU (em %) e consumo médio de memória (em Mib) para cada uma das execuções. Mediana, variância e desvio padrão também foram levados em consideração para a confiabilidade dos dados apurados.

2.1 Selenium

Selenium é atualmente uma das mais usadas ferramentas para automação de processos na web, apesar de não ter sido criada originalmente para este propósito, mas sim para testes de software. Seu funcionamento se dá a partir da automação do navegador via um software de integração, chamado *webdriver*. O webdriver é responsável por traduzir os comandos escritos em código para rotinas interpretadas pelo navegador. Outra notável característica de Selenium é o fato de existirem bibliotecas em diversas linguagens de programação. Para este trabalho usaremos a versão em *Python*, por ser, nos dias de hoje, a linguagem interpretada mais usada por quem usa Selenium [3].

2.2 Puppeteer

Puppeteer, ao contrário do Selenium, surgiu com propósitos gerais de automatizar o que quer que seja feito através de um navegador, como: testar aplicações, gerar capturas de tela, *web scraping* (extração de dados de sites da web), etc. Puppeteer já vem com uma instância do navegador *Chromium* em seus arquivos, não sendo necessário realizar a instalação de software adicionais para seu uso. Sua interação com o navegador se dá a partir do *DevTools Protocol*, um protocolo de comunicação criado pela Google [4] que permite interação direta com o navegador.

2.3 Trabalhos relacionados

As ferramentas avaliadas neste trabalho possuem diversas aplicações, e por isso existem outros trabalhos que fazem esta comparação em situações diferentes, a exemplo de [12] que usa *Selenium* e *Puppeteer* para a construção de testes automatizados e verifica o tempo de execução deles. Por sua vez, o artigo [9] faz uma comparação de consumo de recursos de memória e CPU apenas para extração de dados. Em termos de percentual de uso de CPU, o *Puppeteer* usou menos recursos do processador em comparação com o *Selenium*. No entanto, para uso de memória, o *Puppeteer* consumiu mais que o *Selenium*. Em média, a duração do script *Selenium* para terminar o processo de *scraping* foi um pouco menor. Em [8] é feita a avaliação de diversas ferramentas de automação aplicadas em *web scraping*. As ferramentas foram escolhidas com base em fatores como semelhança e popularidade.

3 Etapas dos algoritmos

Para a execução do robô foi criado um contêiner usando uma imagem *python 3.8* para o robô em Selenium e `node:14-slim` para o robô em Puppeteer como base, em seguida, o ambiente é atualizado para receber os pacotes mais recentes para o ambiente. Por fim instalamos as dependências de código da aplicação. Para uma execução mais eficiente, desativamos a interface gráfica, e o download de imagens.

3.1 Automação de preenchimento de formulários

A automação de preenchimento de fomulários é largamente utilizada em grandes empresas, onde há grandes volumes de dados em trânsito e procedimentos padronizados. Através dela é possível alcançar grandes ganhos em produtividade, padronização e bem-estar do funcionário [10]. Preenchimento de formulários é usado quando há uma fonte de informações e um destino que receberá estas informações, mas não há uma integração direta entre eles [1]. Nestes casos, uma pessoa qualificada deve ler as informações e fazer a interface, manualmente preenchendo os dados. O algoritmo 1 traz uma rotina para essa situação.

Algoritmo 1: Preenchimento de formulários

Dados: Lista de informações a serem inseridas.
Resultado: formulários preenchidos e submetidos.

```
1 Carregue a lista de informações;
2 Acesse a página da web;
3 para Cada entrada da lista faça
4     Preencha os dados no formulário;
5     Submeta o formulário;
```

3.2 Automação de captura de dados e geração de relatórios

Uma outra aplicação da automação, talvez a mais conhecida, se dá na captura e compilação de dados da web, prática conhecida com *web scraping* ou *crawling*. As aplicações para este tipo de automação são inúmeras: desde a criação de bases de dados para estudos, geração de relatórios em empresas e até mesmo geração de índices de busca na web [2]. Em nosso experimento, criamos um *Scraper* com o objetivo de acessar as páginas da conferência *WebMedia* (o Simpósio Brasileiro de Sistemas Multimídia e Web) e compilar dados das sessões técnicas para fins de estudo. O algoritmo 2 utilizado busca gerar um relatório das sessões técnicas apresentadas no evento.

Para fins de acurácia do experimento e redução de variáveis que possam causar incertezas na leitura do consumo de recursos o número de páginas visitadas se limitou a apenas uma, evitando assim incertezas causadas por demora de

Algoritmo 2: Extração de dados de uma página web

Resultado: Dados compilados e estruturados.

```
Acesse a página da web;
Liste todos os parágrafos da página;
para cada parágrafo listado faça
    Identificar chair da sessão;
    Extrair título dos artigos;
    Extrair autores dos artigos;
Compilar os dados em uma tabela em CSV;
```

resposta por parte do servidor de cada página (que podem estar hospedadas em servidores diferentes) ou atrasos de pacotes. A página da edição *WebMedia 2019*[1] foi selecionada para este experimento por possuir o maior número de artigos e sessões técnicas, evidenciando de forma mais clara a existência de alguma diferença notável de desempenho.

4 Resultados

Nesta seção, serão apresentados os resultados obtidos a partir de execuções dos métodos apresentados anteriormente. Os resultados são apresentados na forma de (<média> ± <desvio padrão>).

Os experimentos foram realizados em uma máquina com a seguinte configuração:

- Processador AMD Ryzen 7 3750H;
- Memória RAM 16 GB 2400 MHz DDR4;
- Placa gráfica Radeon Vega Mobile Gfx 2.30 GHz.

Neste experimento consideramos as versões mais estáveis das ferramentas utilizadas: Node.Js 14, Python 3.8 e Docker 20.10.

O consumo medido se dá em valores percentuais para processamento (CPU), denotando o quanto está sendo requisitado de um core da CPU. Um consumo de 100% de CPU significa que está sendo consumida a capacidade total de processamento de 1 core, 200% estaria, portanto, consumindo 2 cores, e assim por diante. O consumo de memória, por sua vez, é medido em mebibytes (Mib) e denota a quantidade de espaço alocado em tempo de execução.

4.1 Caso 1: Automação de preenchimento de formulários

O gráfico 1 mostra o consumo médio de processamento medido em valores percentuais. Para a rotina automatizada em *Puppeteer* tivemos um consumo médio de (162 ± 4)% de CPU, enquanto que o valor para a mesma rotina automatizada em *Selenium* foi de (158 ± 9)%, o que implica igualdade (equivalência) estatística. O consumo médio de memória, por sua vez, pode ser visto no gráfico 2. O valor observado foi de (143 ± 7) MiB para a rotina desenvolvida em *Puppeteer* e de (145 ± 4) MiB para a rotina desenvolvida em *Selenium*. Mais uma vez, os valores médios das execuções estão dentro do intervalo de incerteza, demonstrando igualdade estatística.

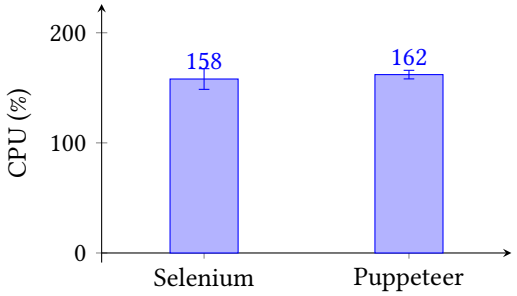

Figure 1. Consumo de CPU (%) em preenchimento de formulários

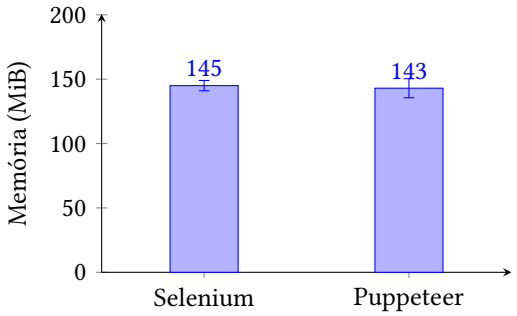

Figure 2. Consumo de memória (MiB) em preenchimento de formulários

4.2 Caso 2: Automação de extração de dados e geração de relatórios

Diferentemente do experimento anterior, onde foi observada igualdade estatística nas execuções, o gráfico 3 evidencia uma grande disparidade entre o consumo médio de CPU para a extração de dados usando *Puppeteer* (6 ± 1)% e a mesma situação usando *Selenium* (109 ± 1)%, evidenciando uma superioridade de desempenho ao se utilizar *Puppeteer*.

A discrepância de valores para este experimento, ilustrada no gráfico da Figura 3, pode ser explicada pela diferença de construção das ferramentas analisadas. *Puppeteer* foi construída a partir de tecnologias mais recentes e com maior compatibilidade com o navegador utilizado, o *Chrome*. Outro fator que pode ser determinante na diferença de desempenho é a forma como é feita a interface entre o código escrito e o navegador; o protocolo *DevTools*, nativo dos navegadores baseados em *Chrome*, torna a interface entre navegador e código muito mais leve (caso do *Puppeteer*) em comparação com a utilização de um *webdriver* (caso do *Selenium*).

De forma semelhante, o gráfico 4 nos mostra uma diferença considerável no consumo de memória entre ambas as

[1]Disponível em https://webmedia.org.br/2019/programacao-em-html

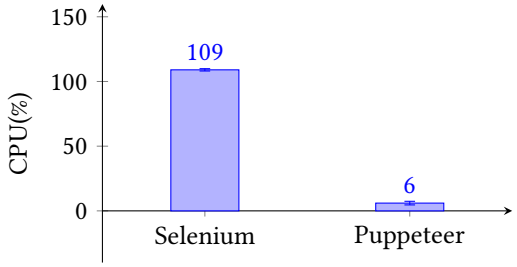

Figure 3. Consumo de CPU (%) em *Web crawlers*

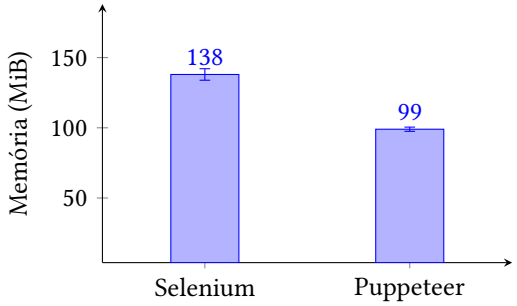

Figure 4. Consumo de memória (MiB) em *Web crawlers*

ferramentas, com o *Puppeteer* apresentando um consumo médio de memória de (99 ± 2) MiB, enquanto *Selenium* obteve um consumo de (138 ± 4) MiB. Desta forma, observa-se que *Puppeteer* consumiu 29% a menos de memória do que o *Selenium*, o que reforça a superioridade de desempenho por parte do *Puppeteer* para a construção de *WebCrawlers*[2].

5 Conclusão

Neste trabalho foi realizado experimentos de automação de processos via RPA com o objetivo de comparar o desempenho de duas bibliotecas utilizadas para a construção de robôs: Puppeteer e Selenium. Os casos de uso envolveram automatizar o preenchimento e submissão de formulários, e a extração de dados, dois dos mais comuns casos de uso, onde se aplica a automação devido à repetitividade do processo. As implementações foram avaliadas de maneira quantitativa, identificando diferenças no consumo de recursos.

Os resultados obtidos mostraram que para o caso de automação de preenchimento de formulários, ambas as ferramentas apresentam desempenho semelhante para consumo de CPU e de memória. O mesmo resultado, porém, não é obtido quando aplicamos as ferramentas em um trabalho de *WebScraping*, onde podemos observar uma diferença significativa de desempenho favorecendo a ferramenta Puppeteer.

A diferença entre os valores observados nos dois experimentos, por sua vez, é fruto da natureza da ação realizada: o preenchimento de formulários demanda muito mais recursos da aplicação, uma vez que é preciso simular periféricos como *clicks* do mouse e entradas de teclado, e, mesmo envolvendo menos seletores, foram observados valores maiores de consumo de recursos em ambas as ferramentas. Já a extração de dados envolve percorrer todos os elementos de uma página e fazer leitura de seus valores, uma atividade bem menos onerosa para a máquina. A grande diferença, por sua vez, entre os valores observados para as duas ferramentas se deve à quantidade de seletores lidos na página. Os experimentos realizados no trabalho em questão foram úteis para elucidar diferenças de desempenho para as ferramentas de automação que estão entre as mais usadas atualmente, entretanto, outros casos de uso e variáveis precisam ser observados para uma conclusão mais assertiva.

Por fim, trabalhos futuros devem incluir a medição do tempo de execução nos experimentos; a utilização de outros recursos disponíveis nas ferramentas e a utilização de formulários maiores com mais riqueza de entrada de dados.

[2]passo a passo para a reprodução do experimento em: https://github.com/pdssf/selenium-puppeteer-comparison

References

[1] Santiago Aguirre and Alejandro Rodriguez. 2017. Automation of a Business Process Using Robotic Process Automation (RPA): A Case Study. In *Applied Computer Sciences in Engineering*, Juan Carlos Figueroa-García, Eduyn Ramiro López-Santana, José Luis Villa-Ramírez, and Roberto Ferro-Escobar (Eds.). Springer International Publishing, Cham, 65–71.

[2] Nilani Algiryage, Gihan Dias, and Sanath Jayasena. 2018. Distinguishing Real Web Crawlers from Fakes: Googlebot Example. In *2018 Moratuwa Engineering Research Conference (MERCon)*. 2018 Moratuwa Engineering Research Conference (MERCon), Siri Lanka, 13–18. https://doi.org/10.1109/MERCon.2018.8421894

[3] Boni García, Micael Gallego, Francisco Gortázar, and Mario Munoz-Organero. 2020. A Survey of the Selenium Ecosystem. *Electronics* 9, 7 (2020), 1. https://doi.org/10.3390/electronics9071067

[4] Google Inc. 2020. Chrome DevTools Protocol. Disponível em: https://chromedevtools.github.io/devtools-protocol/.

[5] Can Kaya, Mete Turkyilmaz, and Burcu Birol. 2019. Impact of RPA Technologies on Accounting Systems. *Muhasebe ve Finansman Dergisi* 1, 1 (04 2019), 235–250. https://doi.org/10.25095/mufad.536083

[6] Google LLC. 2022. Puppeteer. Disponível em https://developers.google.com/web/tools/puppeteer.

[7] Ryan Mitchell. 2018. *Web Scraping with Python*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

[8] Emil Persson. 2019. Evaluating tools and techniques for web scraping. Disponível em: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-271206.

[9] Zurina Saaya. 2021. Extracting Academic Publication Data Using Web Automation Tools. *Journal of Advanced Computing Technology and Application (JACTA)* 3, 1 (Jun. 2021), 18–25. https://jacta.utem.edu.my/jacta/article/view/5223

[10] Leslie Willcocks, Mary Lacity, and A Craig. 2015. Robotic process automation at telefónica O2. *MIS Q Exec* 15, 1 (2015), 21–35.

[11] Leslie Willcocks, Mary Lacity, and A Craig. 2015. Robotic process automation at Xchanging. *MIS Q Exec* 15, 3 (2015), 2–6.

[12] Dmitry Zhyhulin, Kostiantyn Kasian, and Mykola Kasian. 2022. Combined method of prioritization and automation of software regression testing. In *2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*. IEEE Ukraine Section IE/PE/PEL Joint Chapter, Ukraine, 751–755. https://doi.org/10.1109/TCSET55632.2022.9767034