WEBASSEMBLY: UMA ESTRATÉGIA PARA MELHORAR O DESEMPENHO DA APLICAÇÃO JAVASCRIPT EM AMBIENTES IOT

FERNANDO LUIS OLIVEIRA[1];
JÚLIO C. B. MATTOS[2]

[1]*Universidade Federal de Pelotas – fernando.oliveira@inf.ufpel.edu.br*
[2] *Universidade Federal de Pelotas – julius@inf.ufpel.edu.br*

1. INTRODUÇÃO

A Internet das Coisas (*Internet of things* – IoT), se caracteriza por dar aos objetos do cotidiano a capacidade de capturar, processar, armazenar, transmitir e apresentar informações (ALMEIDA, 2018). Os objetos se tornam capazes de interagir com ambientes e geram dados em quantidade, especificidade e variedade, como resultado de suas interações (LACERDA, 2015). Segundo o Transforma Insights (2020), a previsão até o ano 2030 será de um total de 24 bilhões de dispositivos IoT ativos, com uma taxa de crescimento de 11% ao ano.

Por outro lado, a programação de dispositivos IoT requer conhecimentos específicos sobre sistemas embarcados e linguagem de programação com baixo nível de abstração como a linguagem C. No entanto, linguagens de mais alto nível podem facilitar a programação e podem ser mais adequadas ao contexto de dispositivos inteligentes.

O JavaScript (JS) é uma linguagem de programação de alto nível, leve, dinâmica e não tipada. Ao longo do tempo ela amadureceu e se estabeleceu como umas das principais linguagens de programação, pois, a grande maioria dos websites utilizam JS e *frameworks* modernos são projetados baseados nela (DELCEV; DRASKOVIC, 2018). De acordo com Stack Overflow (2020), pelo oitavo ano consecutivo, o JavaScript se manteve como a linguagem de programação mais utilizada e conta com uma comunidade estimada de 12 milhões de desenvolvedores (DEVELOPER ECONOMICS, 2020).

No contexto da programação embarcada é comum responder com uma ação quando um dado evento acontece, esse cenário pode ser simplificado ao adotar o JS como linguagem de programação, pois, ela permite a programação orientada a eventos, ou seja, é possível monitorar, capturar e reagir quando eventos acontecem.

No entanto, o JavaScript é uma linguagem interpretada. Essa característica pode penalizar as aplicações, em termos de desempenho, quando comparadas aos programas compilados e, principalmente, em sistemas embarcados podem consumir mais recursos como CPU, memória e energia. Para esses dispositivos, o consumo da bateria é uma questão essencial. Portanto, para permitir a linguagem JavaScript na programação IoT é necessário implementar técnicas ou ferramentas para melhorar seu desempenho.

O WebAssembly (WASM) é uma proposta que pode melhorar o desempenho de aplicativos JS. O WASM é uma linguagem de baixo nível semelhante ao *assembly*, que promete desempenho mais próximo de linguagens compiladas (HAAS et all., 2017).

Este resumo apresenta a análise do WebAssembly como estratégia para melhorar o desempenho da linguagem JS em ambientes IoT. Experimentos foram realizados a partir da execução de algoritmos utilizando WebAssembly e

JavaScript com base no interpretador nodeJS em uma placa Raspberry Pi. Informações como tempo de execução, memória e consumo enérgico foram utilizadas na avaliação.

2. METODOLOGIA

Para avaliar o WebAssembly como estratégia para melhorar o desempenho das aplicações JavaScript, realizou-se uma série de testes com um conjunto de algoritmos para fins diversos. O objetivo é avaliar se há ganhos de desempenho e quanto isso afeta os dispositivos em termo de consumo de recursos.

Como ambiente de execução, escolheu-se o Raspberry Pi 3 modelo B (RPi), o qual é equipado com um processador ARM quad-core de 1.2GHz e 1GB de memória RAM, quanto ao sistema operacional, adotou-se o Raspberry OS em sua versão básica.

Em relação ao interpretador JavaScript, procedeu-se com o NodeJS (v. 12.18.2). O NodeJS representa o estado da arte em termos de plataforma para execução de JavaScript, sendo amplamente utilizado e possuindo uma ampla gama de ferramentas e *frameworks* para desenvolvimento de aplicações (DELCEV; DRASKOVIC, 2018).

Quanto aos benchmarks, elegeu-se o Ostrich Benchmark Suite (2020). Este *benchmark* facilita a avaliação do JavaScript em relação ao WebAssembly porque oferece a mesma implementação do algoritmo em JS e C. Além disso, Ostrich fornece um arquivo (*make*) para automatizar o processo de construção do módulo WASM, a partir do código C. Dos doze algoritmos disponíveis no *benchmark* utilizaram-se oito, pois, quatro não compilaram no RPI devido à incompatibilidade de versões e arquitetura.

Para avaliar o consumo energético, construiu-se um amperímetro utilizando o microcontrolador Arduino Uno e o sensor de corrente INA219. O RPi precisa de 5V DC e uma corrente de 1A para operar. Para estimar o consumo, a linha de alimentação é interrompida e o sensor de corrente é inserido em linha. O princípio básico é que a corrente que flui através da linha de energia USB da bateria para o Raspberry Pi causa uma atração de tensão, proporcional à corrente consumida pela operação em execução no RPi. Essa corrente é armazenada e processada no Arduino sendo utilizada para estimar o consumo.

Para coletar as informações de desempenho, utilizou-se a ferramenta Perf profiling. Através dessa ferramenta é possível avaliar o desempenho do CPU a partir de contadores de eventos, possibilitando a coletar informações úteis como instruções executadas, alocação de memória ou falhas de acesso à memória cache.

O experimento consiste em avaliar, para cada *benchmark*, a execução do algoritmo em JS e no WebAssembly ambos executados por meio do NodeJS. Cada operação foi repetida 30 vezes para obter consistência dos dados e informações estatísticas mais precisas (BUYTAERT; EECKHOUT, 2007).

3. RESULTADOS E DISCUSSÃO

Para avaliar os resultados, os dados foram segmentados por abordagem (JavaScript/WebAssembly) e agrupados por benchmark. A Tabela 01 apresenta a média das execuções de cada teste, bem como mostra o desvio padrão.

Os ensaios apontam que com WebAssembly o tempo de execução e consumo de energia são otimizados em cerca de 39% em relação à execução do JavaScript. O ganho é dado em razão que o processo de otimização que, no

JavaScript ocorre em tempo de execução e implica uso de CPU, ocorre no WebAssembly em tempo de compilação, portanto, etapas extras com vistas a desempenho não são necessárias.

Tabela 1 Resultados dos experimentos

Benchmark	Tempo de execução (sec.)				Memória (MB)				Energia (J)			
	JavaScript		WASM		JavaScript		WASM		JavaScript		WASM	
	Média	*DP*	*Média*	*DP*	*Média*	*DP*	*Média*	*DP*	*Média*	*DP*	*Média*	*DP*
nw	8,95	0,002	6,62	0,001	15,93	3,765	17,40	0,392	22,75	1,165	18,03	1,236
crc	10,35	0,003	7,86	0,002	3,67	0,147	3,94	0,122	26,30	0,800	19,99	0,874
nqueens	43,51	0,001	27,10	0,001	2,98	0,008	3,18	0,016	105,22	2,721	66,93	4,130
lud	115,30	2,281	70,06	0,032	4,87	0,231	6,04	0,508	270,64	7,984	166,62	1,527
bfs	7,79	0,041	2,24	0,000	12,64	6,844	5,85	3,245	23,77	1,830	6,00	1,113
lavamd	19,33	0,002	11,87	0,001	3,84	0,061	3,33	0,063	44,30	4,435	27,88	5,483
fft	17,25	0,012	11,79	0,004	6,10	1,425	6,41	1,443	43,22	1,249	27,43	2,058
srad	45,95	0,054	22,95	0,064	4,11	0,116	3,93	0,082	107,98	3,296	52,90	2,851

DP: Desvio Padrão

Com relação ao consumo de memória, o WebAssembly gerencia a memória através de um *ArrayBuffer* dimensionável que contém uma matriz linear de *bytes* que é lida e escrita por instruções de memória de baixo nível (HAAS et all., 2017). Portanto, o WASM aloca memória em formato bloco contíguo de memória o que implica em alocações fixas, para os experimentos compilou-se o módulo WASM sem limite de uso de memória, aliado a isso, o controle para liberação de memória é feito de forma a liberar todo bloco de uma vez.

Percebeu-se que o WebAssembly alcançou melhores resultados em situações em que são necessários cálculos massivos, como multiplicação de matrizes ou processamento gráfico. Entretanto, quando as implementações são simples, como por exemplo a soma de dois números, não se torna viável adotá-lo pois, para utilizá-lo o módulo WASM precisa ser carregado, essa carga acarreta custo, por tanto, em situações de baixa complexidade é conveniente utilizar apenas o JavaScript.

4. CONCLUSÕES

Este resumo avaliou se o WebAssembly poderia melhorar o desempenho da linguagem JavaScript no ambiente IoT. Coletou-se informações sobre o consumo de recursos como CPU, memória e bateria a partir da execução de benchmarks.

Embora o JavaScript seja uma linguagem interpretada, ela tem um desempenho satisfatório na maioria dos casos, e sua abordagem orientada a eventos é adequada para implementação no ecossistema IoT. No entanto, ainda há uma grande diferença de desempenho para uma linguagem compilada.

As avaliações apresentadas neste trabalho mostraram que o WebAssembly é uma estratégia eficiente para melhorar o desempenho do JavaScript, sendo superior em todos os itens avaliados, com ênfase na redução do consumo da bateria. Em dispositivos alimentados por bateria cujo recursos são limitados, o consumo de energia pode ser mais significativo do que a velocidade de execução.

Por fim, o WebAssembly é uma tecnologia promissora que pode ser aplicada a diferentes contextos, principalmente porque é um código compilado adequado para o ambiente IoT oferecendo uma boa relação entre desempenho e consumo de recursos.

5. REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, H. C., SIQUEIRA, S. Uma Revisão Sistemática sobre Descrição Semântica na Internet das Coisas. **iSys: Revista Brasileira de Sistemas de Informação.** New York, NY. p. 185–200, 2017. V.11, n.2, p. 43-62, 2018.

BUYTAERT, G. D.; EECKHOUT, L. Statistically rigorous java performance evaluation. **22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications**. New York, NY. p. 57–76, 2007.

DELCEV, S; DRASKOVIC, D. Modern javascript frameworks: A surveystudy. **Zooming Innovation in Consumer Technologies Conference** (ZINC). pp. 106–109, 2018.

HAAS, A; ROSSBERG, D; L. SCHUFF; B. L. TITZER; M. HOLMAN; D. GOHMAN; L. WAGNER; A. ZAKAI; J. BASTIEN. Bringing the web up to speed with webassembly. **38th ACM SIGPLAN Conference on Programming Language Design and Implementation.** New York, NY. p. 185–200, 2017.

KHAN, F; FOLEY-BOURGON, V; KATHROTIA, S. L. E. **Ostrich benchmark suite.** Acessado em 29 jun. 2020. Online. Disponível em: https://github.com/Sable/Ostrich

LACERDA, F; LIMA-MARQUES, M. Da necessidade de princípios de Arquitetura da Informação para a Internet das Coisas. **Perspect. ciênc. inf.** Belo Horizonte, BR. vol.20 n.2. 2018.

STACK OVERFLOW. **Stack overflow annual developer survey 2020.** Acessado em 23 set. 2020. Online. Disponível em: https://insights.stackoverflow.com/survey/2020

DEVELOPER ECONOMICS. **State of the developer nation - 18 edition.** Acessado em 11 ago. 2020. Online. Disponível em: https://www.developereconomics.com

TRANSFORMA INSIGHTS. **The internet of things (iot) market 2019-2030.** Acessado em 23 set. 2020. Online. Disponível em: https://transformainsights.com/news/iot-market-24-billion-usd15-trillion-revenue-2030