COMPARANDO A EFICIÊNCIA DE APLICATIVOS ANDROID NATIVOS COM APLICATIVOS BASEADOS EM JAVASCRIPT

KELLERSON KURTZ[1]; MARCELO NOGUEZ[2]; LISANE BRISOLARA[3]

[1]*Universidade Federal de Pelotas – kkurtz@inf.ufpel.edu.br*
[2]*Universidade Federal de Pelotas – mnnoguez@inf.ufpel.edu.br*
[3] *Universidade Federal de Pelotas – lisane@inf.ufpel.edu.br*

1. INTRODUÇÃO

No desenvolvimento de aplicações móveis, tradicionalmente, são empregadas abordagens nativas de desenvolvimento em que as linguagens de programação são definidas para cada plataforma alvo. Aplicações nativas possuem vantagens como o melhor uso de recursos providos pela plataforma como gráficos 3D e sensores do dispositivo, no entanto exigem reescrita de código para serem portadas para outras plataformas móveis.

Por outro lado, tecnologias *web*, que utilizam linguagens como JavaScript, Node.js e PHP, estão ganhando rapidamente o interesse da comunidade de desenvolvedores para dispositivos móveis (SERRANO et al., 2013) (OLIVEIRA, 2016). Este interesse deve-se principalmente ao fato destas tecnologias permitirem que uma única implementação possa ser compartilhada entre diferentes plataformas, evitando especificidades de cada sistema operacional (SO) (AMATYA e KURTI, 2013), sendo conhecidas como tecnologias multiplataforma.

Devido a limitação dos recursos disponíveis nos dispositivos móveis, métodos que possibilitam migrar processamento e armazenamento de dados para um servidor remoto ganham também atenção dos desenvolvedores e motivam o emprego das tecnologias web neste domínio. As aplicações móveis assim desenvolvidas empregam uma arquitetura cliente/servidor e assemelham-se a sistemas *web* tradicionais, os quais adotam diferentes tecnologias/linguagens para o *front-end* (lado do cliente) e *back-end* (lado do servidor). Usualmente, o *front-end* é desenvolvido em HTML5 ou JavaScript, enquanto o *back-end* adota PHP e, recentemente, Node.js (N.FOUNDATION, 2017), um ambiente multiplataforma que utiliza JavaScript em tempo de execução.

Diferentemente das nativas, aplicações móveis baseadas nestas tecnologias são executadas no navegador e esta nova camada pode gerar uma sobrecarga, impactando assim negativamente na sua eficiência. Além disso, aplicações com arquitetura cliente/servidor, como é o caso das baseadas em Node.js, necessitam de comunicação com o servidor. Dependendo do volume de informação transferida, esta comunicação pode reduzir o desempenho e aumentar do consumo de energia. Em contraste, aplicações executadas localmente (nativa e JavaScript) tendem a consumir mais processamento se comparadas a implementações em Node.js.

Estudos comparativos entre aplicações nativas e multiplataforma foram publicados nos últimos anos. No entanto, sua maioria considera apenas critérios como a facilidade de desenvolvimento, usabilidade ou a experiência do usuário final, e apenas um pequeno número discute também desempenho e consumo de energia (OLIVEIRA, 2016). Como estas métricas são muito importantes no domínio *mobile*, este artigo possui o objetivo de comparar a eficiência de aplicações Android nativas a aplicações em JavaScript e Node.js através de experimentos. Além disso, o artigo discute também se uma abordagem com arquitetura cliente/servidor pode trazer benefícios na eficiência se comparada a uma abordagem local.

2. METODOLOGIA

Para comparar a eficiência entre distintas abordagens de desenvolvimento, diferentes implementações de uma mesma aplicação foram desenvolvidas: duas implementações com abordagem *web*, uma em JavaScript (local) e a outra em Node.js (cliente/servidor); e uma implementação nativa ao SO Android. A aplicação alvo emprega o algoritmo de ordenação *bubble sort*, que possui complexidade $O(n^2)$, para ordenar um vetor de inteiros. Nos experimentos, o tamanho do vetor de entrada foi variado entre 100, 1.000, 10.000 e 100.000 elementos e foi inicializado em ordem decrescente, gerando o pior caso de execução do algoritmo. Assim, os experimentos exploram a variação da carga de processamento, com o aumento do tamanho da entrada, visando analisar o seu impacto na eficiência temporal e energética nas diferentes implementações. O tamanho do vetor impacta também na comunicação em implementações Node.js e permitem avaliar o impacto do emprego da arquitetura cliente/servidor, como alternativa ao processamento local.

Nos experimentos, foi utilizado o *smartphone Motorola* Moto X, segunda geração, com 2GB de memória RAM, rodando Android versão 6.0. Para comparar a eficiência das diferentes implementações, foram estimados o tempo de execução e o consumo de energia através de experimentos. Como a mensuração varia para cada execução, repetimos cada experimento trinta vezes, para validação estatística dos resultados. A média e o desvio padrão das trinta execuções foram calculadas e o teste *t de Student* foi aplicado para verificar se as médias possuíam uma diferença estatística significante. Para uma avaliação apropriada das métricas, antes de cada execução, todas as outras aplicações do dispositivo foram fechadas e a aplicação foi impedida de rodar em *background*, já que isto pode interferir no desempenho e no consumo estimados.

Para a avaliação de desempenho, o tempo atual é salvo no início e fim de cada execução, e então é subtraído o tempo inicial do final para determinar o tempo total de execução. O método *System.currentTimeMillis* foi empregado para capturar o tempo na execução local no dispositivo e para execuções remotas, uma estratégia para considerar também o tempo de execução no servidor foi aplicada. Para a avaliação de consumo de energia foi utilizado o Powertutor (YANG, 2012), que é um aplicativo Android para estimar consumo. Apesar de ser uma ferramenta de estimativa e possuir uma taxa de erro considerável, de acordo com (FISCHER, 2015), este aplicativo pode ser utilizado para analisar tendências.

3. RESULTADOS E DISCUSSÃO

Esta seção apresenta e discute o estudo comparativo realizado, onde a eficiência de duas diferentes implementações web da aplicação de ordenamento são comparadas à sua versão nativa. A Tabela 1 sumariza os resultados de desempenho das três implementações avaliadas. Os resultados obtidos em execuções com 100 e 1.000 elementos mostram que a implementação com abordagem nativa apresenta menor tempo de execução comparada às versões em JavaScript e Node.js. A versão nativa mostrou-se a mais eficiente das três nos cenários de menor tamanho, com melhorias em torno de 90% e de 76% para 100 e 1.000, respectivamente. No entanto, nos cenários com vetores maiores, a nativa perde desempenho. Com o vetor de 10.000 elementos, a versão nativa ainda apresenta menor tempo de execução quando comparada à JavaScript (mas com ganho de apenas 18%). No entanto, neste cenário, a versão nativa apresenta pior desempenho em relação à versão Node.js.

Tabela 1. Resultados de desempenho dos aplicativos em milissegundos

Tamanho	Implementação	Tempo de Execução (ms)	Desvio Padrão (ms)
100	Nativa	73,03	9,60
	JavaScript	811,67	59,38
	Node.js	709	48,96
1.000	Nativa	194,5	10,61
	JavaScript	897,9	50,97
	Node.js	741,93	51,10
10.000	Nativa	1.830,2	425,68
	JavaScript	2.223,17	109,72
	Node.js	1.439,8	102,17
100.000	Nativa	168.954,33	792,69
	JavaScript	39.263,3	102,17
	Node.js	15.405	881,08

A implementação Node.js obteve o melhor desempenho em cenários com grandes tamanhos de vetores. Com 10.000 elementos, essa implementação obteve em torno de 21% de ganho quando comparada à versão nativa e 35% em relação à JavaScript. Para o vetor de 100.000 elementos, a implementação em Node.js apresentou ganhos de 91% e 61% em relação às versões nativa e JavaScript, respectivamente. Vale destacar que comparada à implementação JavaScript, a versão Node.js apresentou melhor desempenho em todos os cenários avaliados.

O consumo de energia estimados para as versões JavaScript, Node.js e nativa são apresentados na Tabela 2. A implementação nativa apresentou o menor consumo energético com 100 e 1.000 elementos, alcançando entre 99% e 94% de ganho em relação às implementações JavaScript e Node.js, respectivamente. No entanto, com vetores maiores, a implementação nativa perde eficiência, apresentando alto consumo de energia quando comparada às demais versões. Embora as implementações Node.js e JavaScript apresentaram resultados estatisticamente equivalentes nos dois cenários menores, a implementação Node.js alcançou ganhos de 24% e 93% com 10.000 e 100.000 elementos, respectivamente. Além disso, comparada à abordagem nativa, a implementação Node.js apresentou ganhos de 43% e 98% nos dois cenários maiores.

Tabela 2. Resultados de consumo energéticos dos aplicativos

Tamanho	Implementação	Consumo de Energia (mJ)	Desvio Padrão (mJ)
100	Nativa	8,17	2,78
	JavaScript	751,47	167,83
	Node.js	724,3	143,33
1.000	Nativa	43,37	18,44
	JavaScript	789,87	215,32
	Node.js	770,73	189,18
10.000	Nativa	2.026,67	496,84
	JavaScript	1.523,17	345,44
	Node.js	1.155,57	297,68
100.000	Nativa	68.556,67	1.593,02
	JavaScript	18.810	119,58
	Node.js	1.361,87	340,21

4. CONCLUSÕES

Este artigo investigou o impacto do emprego de abordagens multiplataforma no tempo de execução e consumo de energia para um aplicativo de ordenação, apresentando um estudo comparativo entre implementações JavaScript, Node.js e nativa. Nossos experimentos avaliaram a eficiência das três implementações distintas variando a carga de processamento e exploram tanto abordagens baseadas em execuções locais quanto remotas.

Os resultados indicaram que a implementação nativa é mais eficiente em desempenho e energia até uma certa carga de processamento. No entanto, a implementação nativa é altamente impactada pelo processamento, e então esta pode não ser adequada para grandes cargas de processamento. Conforme o a carga de processamento é significativamente aumentada, a versão Node.js apresenta os menores tempos de execução, bem como menores consumos de energia. É importante ressaltar que a eficiência desta versão foi menos impactada pela variação na carga, principalmente em relação ao consumo de energia.

Nossos experimentos indicaram que, entre as abordagens multiplataforma consideradas, Node.js é a melhor opção para larga carga de processamento considerando as duas métricas avaliadas. Isso deve-se a execução remota, que reduz o consumo por transferir o processamento para um servidor, o que explica a melhora alcançada por esta abordagem quando grandes vetores são utilizados.

Este estudo explorou duas abordagens multiplataforma baseadas em tecnologias *web*, porém sem empregar *frameworks* recentemente propostos para garantir portabilidade de aplicativos móveis. Como trabalhos futuros, pretende-se incluir a avaliação de aplicativos gerados por estes *frameworks,* além de avaliar diferentes aplicações, de forma a discutir também como a natureza da aplicação impacta na eficiência alcançada pelas diferentes abordagens.

5. REFERÊNCIAS BIBLIOGRÁFICAS

AMATYA, S.; KURTI, A.. Cross-platform mobile development: challenges and opportunities. **ICT Innovations 2013.** Springer**,** Heidelberg**,** p. 219-229, 2014.

FISCHER, L.; DE BRISOLARA, L. B.; DE MATTOS, J. C. B.. SEMA: An approach based on internal measurement to evaluate energy efficiency of Android applications. **Brazilian Symposium on Computing Systems Engineering (SBESC)**, IEEE. p. 48-53, 2015.

N. FOUNDATION. **"Node.js"**, 2017. Acessado em 01 out. 2017. Online. Disponível em: https://nodejs.org/en/

OLIVEIRA, W., et al. **Native or web? A preliminary study on the energy consumption of android development models**. IEEE 23rd International Conference on. Vol. 1 IEEE 2016.

SERRANO, N.; HERNANTES, J.; GALLARDO, G.. Mobile web apps. **IEEE software**, v. 30, n. 5, p. 22-27, 2013.

YANG, Z. **Powertutor-a power monitor for android-based mobile platforms**, 2012, EECS, University of Michigan**.**