Análise de Desempenho Durante Descarregamento Computacional de Aplicações Móveis

Gabriel Magno França da Fonseca[1]

[1]Bacharelado em Engenharia de Software
Instituto de Ciências Exatas e Informática – PUC Minas
Ed. Fernanda. Rua Cláudio Manoel, 1.162, Funcionários, Belo Horizonte – MG – Brasil

gfonseca@sga.pucminas.br

Abstract. *The use of mobile devices is becoming more and more frequent, also increasing the consumption of mobile computing (e.g. cloud computing). Hence, executing computational calculus on external environments gains prominence, as a way to outsource the application data processing. With that in mind, we seek to find the best technology for these cases, where it is possible to save time and resources, such as the battery. The goal of this work is to analyze the performance of computational calculations in different environments and to assess the impact on applications for mobile devices. For that, comparisons are made using the execution of the algorithms in the device itself and in REST and gRPC services, located on servers in the United States of America. The algorithms used in this study were Heap Sort, Bubble Sort and Selection Sort. From the results found, it is noticeable the impact that the complexity of the algorithm causes on battery consumption and application performance, both for remote solutions and for executions on the device, allowing a better understanding of when to use the alternatives presented.*
Keywords*: mobile devices, REST, gRPC, algorithms.*

Resumo. *A utilização de dispositivos móveis vem se tornando cada vez mais frequente, aumentando também o consumo de computação móvel (e.g. computação na nuvem). Com isso, a execução de cálculos computacionais em ambientes externos ao do dispositivo ganha um destaque como forma de terceirizar o processamento de dados das aplicações. Pensando nisso, busca-se encontrar a melhor tecnologia para estes casos, onde é possível economizar tempo e recurso como por exemplo, a bateria. O objetivo deste trabalho é analisar o desempenho da execução de cálculos computacionais em diferentes ambientes, e avaliar o impacto em aplicações para dispositivos móveis. Para isso são realizadas comparações utilizando execução dos algoritmos no próprio dispositvo e em serviços REST e gRPC, localizados em servidores nos Estados Unidos da América. Os algoritmos usados neste estudo foram Heap Sort, Bubble Sort e Selection Sort. Dos resultados encontrados, é perceptível o impacto que a complexidade do algoritmo causa no consumo da bateria e no desempenho da aplicação, tanto para as soluções remotas quanto para execuções no dispositivo, possibilitando melhor entendimento de quando utilizar as alternativas apresentadas.*
Palavras-chave*: Dispositivos móveis, REST, gRPC, algoritmos.*

Bacharelado em Engenharia de Software - PUC Minas
Trabalho de Conclusão de Curso (TCC)

Orientador de conteúdo (TCC I): Laerte Xavier - laertexavier@gmail.com
Orientador acadêmico (TCC I): Lesandro Ponciano - lesandrop@pucminas.br
Orientador do TCC II: Alexandre Teixeira - teixeira@sga.pucminas.br

Belo Horizonte, 01 de junho de 2021.

1. Introdução

A utilização de dispositivos móveis vem se tornando cada vez mais frequente, assim como o número de aplicações por dispositivo também vem crescendo [Liao et al. 2013]. Com o aumento da frequência de uso dos aplicativos, o desenvolvimento de aplicações mais complexas e poderosas pode ser possível, já que esse mercado é alimentado de consumidores que demandam mais dos recursos do dispositivo. Entretanto, os dispositivos móveis inteligentes (SMDs, do inglês *Smart Mobile Devices*) possuem baixa resistência de energia da bateria e velocidade do processador [Gill and Kaur 2016]. A limitação tecnológica dos SMDs pode prejudicar o desempenho das aplicações e do dispositivo, ao efetuar cálculos computacionais pesados durante execução de um aplicativo, requisitando mais da bateria disponível.

Uma das estratégias mais investigadas para economia de energia é o descarregamento computacional (do inglês, *Computation Offloading*), que consiste na realização de cálculos computacionais num ambiente externo, como em nuvem [Chamas et al. 2017]. Com essa descarga de computação, o dispositivo alivia a carga nos seus recursos, possibilitando ao usuário continuar suas ações ao mesmo tempo que contribui para a manutenção da vida útil do aparelho, podendo ainda reduzir o tempo de execução com a utilização dessa abordagem. Além disso, descarregar computação pode ser uma estratégia para otimizar a execução de aplicativos complexos, de forma a reduzir o consumo de recursos do dispositivo, como uso de memória e processador [Gill and Kaur 2016].

Nesse contexto, o objetivo deste estudo é **analisar o desempenho da execução de cálculos computacionais, tanto em ambiente remoto quanto local, e avaliar seu impacto na bateria dos dispositivos**. Para tanto, foram levantadas as seguintes questões de pesquisa (QP):

- QP 1: Qual a influência do volume de dados no tempo de resposta?
- QP 2: Qual a influência da tecnologia utilizada no tempo de resposta?
- QP 3: Qual a diferença no tempo de resposta com execução local ou externalizada?
- QP 4: Qual o impacto causado na bateria se comparadas execuções em ambiente local e remoto?

A partir das análises de desempenho e consumo de bateria, busca-se comparar as soluções caso a caso, para entender os cenários envolvidos. Para tanto, foram utilizadas as métricas de tempo de resposta, vazão de bateria e volume de dados. Os resultados neste estudo podem auxiliar na decisão sobre a tecnologia que será utilizada ao desenvolver uma aplicação móvel, dependendo do contexto em que está inserida.

O restante deste trabalho está distribuído em sete seções. Na Seção 2, são apresentados os principais conceitos utilizados no estudo. Na Seção 3, são apresentados os trabalhos relacionados. Na Seção 4, são apresentadas as metodologias e métodos utilizados no trabalho. Na Seção 5, são apresentados os resultados encontrados. Na Seção 6, estão as discussões de implicações e limitações do trabalho. Por fim, na Seção 7 encontra-se a conclusão sobre o que foi feito e a abertura para trabalhos futuros.

2. Fundamentação Teórica

2.1. Computação em Nuvem e Descarregamento Computacional

A computação em nuvem é um modelo computacional baseado na Internet ou sob demanda, que fornece um ambiente escalável, flexível, seguro e imediato [Gill and Kaur 2016]. Com ela, é possível compartilhar dados entre diferentes ambientes/usuários ou realizar cálculos computacionais em um ambiente externo ao dispositivo. O compartilhamento se deve pela possibilidade de se conectarem duas máquinas diferentes, podendo assim compartilhar os recursos computacionais de uma máquina mais potente com, por exemplo, um dispositivo móvel de baixo potencial.

Descarregamento computacional é uma abordagem utilizada para executar algoritmos ou cálculos computacionais em um ambiente externo ao do dispositivo que o solicita, geralmente em nuvem [Chamas et al. 2017]. Utilizando esta abordagem, é possível poupar o dispositivo de realizar cálculos pesados, enviando os dados para um ambiente em nuvem que realizará todo o processamento dos mesmos. Como demonstrado na Figura 1, os dispositivos móveis são responsáveis por enviar uma requisição a um serviço em nuvem, através do seu provedor de internet que interceptará a comunicação. Ao enviar a requisição com os dados, o dispositivo espera por uma resposta do serviço com os dados já processados.

Os conceitos apresentados anteriormente estão diretamente relacionados, tendo em vista que o descarregamento computacional depende de uma infraestrutura em nuvem para que seja concretizada. Esta infraestrutura é a responsável por receber todos os dados enviados pelo dispositivo que pretende terceirizar o processamento dos mesmos. Ao fim da execução em nuvem, o dispositivo apenas recebe os dados já processados, consumindo seus recursos apenas para instaurar conexão com o ambiente remoto, ao enviar e receber os dados.

2.2. REST

REST (*Representational State Transfer*) é um modelo arquitetural usado para fornecer a representação de uma entidade de aplicação, comunicando através de requisições do Protocolo de Transferência de Hipertexto (HTTP, do inglês *Hypertext Transfer Protocol*) [Kufner and Mařík 2019]. Estas requisições HTTP são feitas no HTTP/1.1, sendo passível de problemas com a latência durante a comunicação. Este estilo arquitetural é também o mais comum dentre as implementações no HTTP/1 [Vogel et al. 2018].

Utilizando a arquitetura REST, as conexões cliente-servidor são estabelecidas por interfaces nomeadas *POST*, *GET*, *PUT* e *DELETE*. Cada interface possui uma responsabilidade no padrão da arquitetura, que deve ser atendido para manter um padrão, como: o *POST* é responsável pela criação de novos objetos/entidades (dados); o *GET* é responsável por recuperar os dados previamente criados; o *PUT* é responsável pela

Figura 1. Arquitetura de descarga computacional
Fonte: Elaborada pelo autor

atualização de informações dos dados; por fim, o *DELETE* é responsável por apagar objetos específicos na base de dados. As comunicações do modelo REST são em formato JSON, como mostrado na Listagem 1, sendo interpretado por ambos, cliente e servidor, desta forma.

```
{
  id: "0d2ae732-6971-44a7-b891-c899a649eeaf",
  nome: "Livro"
}
```

Listagem 1. Resposta JSON

2.3. gRPC

gRPC é um *framework* moderno de Chamada de Procedimento Remoto (RPC, do inglês *Remote Procedure Call*) do HTTP/2, de código aberto e de alto desempenho, que pode ser executado em qualquer ambiente [gRPC Authors 2020]. Com o gRPC, é possível otimizar o tráfego de dados devido a sua serialização, o qual compacta os dados de forma binária antes do envio ao cliente, que possui um interpretador destes dados. Os interpretadores e chamadas de serviço são gerados automaticamente a partir dos arquivos escritos na linguagem proto, que determinam a forma na qual as aplicações irão se comunicar, como mostrado na Listagem 2.

O *framework* gRPC possui um compilador para os arquivos proto, responsável por gerar um arquivo específico para a linguagem escolhida. Por exemplo, se utilizando o gRPC para java, o arquivo escrito em proto será compilado para um arquivo java. No arquivo proto exemplo, mostrado na Listagem 2, existem algumas palavras chaves destacadas, que são elas: *sintax*, que especifica a linguagem proto utilizada; *message*, que define uma mensagem, podendo ser interpretada como a declaração de um objeto, definindo assim os seus campos de valores e tipos, além também de um serial numérico para cada campo (representado pelo número ao lado do mesmo); *service*, que declara uma interface de comunicação utilizando um canal RPC; e *rpc*, que declara a assinatura da requisição.

```
syntax = "proto3";

message UUID {
```

```
    int64 mostSigBits = 1;
    int64 leastSigBits = 2;
  }

  message Produto {
    UUID id = 1;
    string nome = 2;
  }

  message GetProdutoRequest {
    UUID id = 1;
  }

  service ProdutoService {
    rpc GetProduto (GetProdutoRequest) returns Produto;
  }
```

Listagem 2. Exemplo de um arquivo escrito em proto3

3. Trabalhos Relacionados

Nesta seção, são discutidos trabalhos com temáticas semelhantes às apresentadas nesta proposta.

Chamas et al. (2017) analisaram o consumo de energia durante a execução computacional de alguns algoritmos. Para isto, fizeram comparações dentre executar o algoritmo em um ambiente externo e no próprio dispositivo. Dentre as tecnologias do ambiente externo, estão inclusas comparações da arquitetura REST e do sistema gRPC. A solução seguindo o padrão REST apresentou maior economia de energia, conforme o número de objetos aumentava. No presente estudo, os algoritmos de ordenação são similares aos apresentados no supracitado, com foco não apenas no consumo de bateria, mas também abordando o tempo de execução das soluções.

Tang e Li (2015) investigaram o problema de otimização de energia e o tempo para descarregar a computação dos dispositivos mais limitados de recursos. O cenário ao qual se encontram, envolve vários usuários de dispositivos conectados por uma rede. Para minimizar o consumo de bateria, propuseram um jogo utilizando o algoritmo de Nash para decidir um grupo de dispositivos que utilizarão o descarregamento computacional. Com os resultados obtidos, conseguiram encontrar um volume de dados que possibilita uma maior economia de energia dos dispositivos que optam pelo descarregamento computacional. O presente estudo também faz medições quanto ao consumo da bateria durante a descarga, de acordo com o volume de dados fornecido, comparando as tecnologias e a execução local. [Tang and Li 2015]

Gill e Kaur (2016) apresentaram um esquema de descarregamento computacional para resolver um problema onde as aplicações, conforme ficam mais complexas, acabam prejudicando os dispositivos que possuem recursos limitados. Investigaram alternativas para avaliar a viabilidade da execução computacional em ambiente local e remota, utilizando a descarga computacional para a nuvem como uma das possibilidades, com um método de alocação de buffer para trafegar os dados. O problema da mochila e o *Quick sort* foram os algoritmos escolhidos para avaliarem o que foi proposto. Perceberam que o desempenho do dispositivo melhorava conforme o consumo da bateria reduzia ao des-

carregar os dados, com um ganho também no tempo de resposta. Realizaram medições similares às propostas no presente trabalho, como as análises de desempenho e consumo de bateria dos dispositivos durante o processo de descarga computacional.

Jia et al. (2018) apresentaram uma solução para um gargalo encontrado no processo distribuído de trabalho do TensorFlow, uma biblioteca *open-source* responsável por distribuir computação para um ou mais processadores computacionais ou gráficos em uma máquina física [Jia et al. 2018]. Eles compararam a solução básica elaborada pelo TensorFlow, utilizando o sistema gRPC, e sua solução proposta utilizando *Remote Direct Memory Access* (RDMA). Conseguiram obter alguns resultados positivos quanto ao uso da tecnologia, possibilitando um maior desempenho para aliviar os gargalos frequentes, se comparado com a versão original com gRPC. Assim como feito no estudo, o presente trabalho busca também realizar comparações entre tecnologias, tendo o *framework* gRPC em comum, além de comparar com uma implementação em REST e execuções no dispositivo.

Corbel et al. (2016) apresentaram uma comparação de desempenho entre as duas versões do protocolo de comunicação HTTP (1.1 e 2). Nesse estudo, obtiveram como resultados uma maior eficiência por parte do protocolo HTTP/2 sobre o HTTP/1, tanto em questões de *download* de uma página, como também em termos de funcionalidades. Apresentaram então as diferenças dentre os protocolos utilizados no presente estudo, onde as soluções remotas (implementações em gRPC e REST) se comunicam por diferentes versões do protocolo HTTP, podendo apresentar diferença de desempenho entre elas. [Corbel et al. 2016]

4. Materiais e Métodos

O estudo apresentado neste trabalho é classificado como quantitativo e controlado, tendo como base o trabalho de Chamas et al. (2017), além da análise de algumas questões de pesquisa para avaliação de desempenho das soluções. Buscou-se quantificar métricas, de forma a comparar a eficiência da execução de algoritmos computacionais em ambientes remotos, quando solicitados de um dispositivo móvel, e no próprio aparelho. Para a orientação do estudo, foram definidas as seguintes questões de pesquisa:

- QP 1: Qual a influência do volume de dados no tempo de resposta?
- QP 2: Qual a influência da tecnologia utilizada no tempo de resposta?
- QP 3: Qual a diferença no tempo de resposta com execução local ou externalizada?
- QP 4: Qual o impacto causado na bateria se comparadas execuções em ambiente local e remoto?

Para resolução das questões de pesquisa foram estabelecidas as seguintes métricas: tempo de resposta; vazão de bateria; e volume de dados. O restante da seção descreve os procedimentos e métodos para realização do trabalho.

4.1. Procedimentos

O trabalho foi composto por um instrumento de avaliação que é a execução de algoritmos de ordenação em dois ambientes diferentes. O primeiro é o ambiente local, o qual será o próprio dispositivo móvel, provendo informações do consumo de bateria do dispositivo e desempenho da execução dos algoritmos. O segundo é um ambiente em nuvem, sendo duas implementações distintas, uma utilizando REST e a outra gRPC, para

avaliação de desempenho do descarregamento computacional. Ambos os ambientes terão a implementação dos algoritmos clássicos de ordenação demonstrados no Quadro 1, similares aos usados por Chamas et al. (2017).

Quadro 1. Algoritmos de ordenação e suas complexidades

Algoritmo	Melhor caso	Pior caso
Heap Sort	$O(n \log n)$	$O(n \log n)$
Bubble Sort	$O(n)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$

Fonte: Chamas et al., 2017

No instrumento, buscou-se medir o consumo de bateria de um dispositivo móvel e o desempenho da execução dos algoritmos nos ambientes local e remoto. O consumo de bateria é medido durante a execução dos algoritmos no dispositivo e durante o descarregamento computacional, desde o envio dos dados até o seu retorno. Assim como o consumo de bateria, também é avaliado o desempenho da aplicação, medido pelo tempo de resposta dos algoritmos. Para a realização do experimento, foram definidos dois serviços para o ambiente remoto: um REST, por ser o modelo arquitetural mais utilizado [Vogel et al. 2018], e o gRPC, por se tratar de uma tecnologia relativamente nova, lançada em 2016 [gRPC Authors 2020].

Para a execução remota do experimento, foi necessário a implementação dos *web services* em REST, com o Spring Boot, e em gRPC, com o próprio *framework*, responsáveis por executar os algoritmos com os dados recebidos no descarregamento computacional. Em conjunto com estas APIs, desenvolveu-se uma aplicação Android nativa para consumí-las. O aplicativo utiliza a biblioteca *Retrofit*, responsável por criar interfaces HTTP, consumindo do serviço REST. Para consumir a API gRPC, utilizou-se a biblioteca cliente do próprio *framework*. Todas as aplicações foram implementadas com a linguagem de programação Kotlin, como uma forma de padronizá-los e manter um algoritmo idêntico entre elas. A estrutura da aplicação e dos serviços foi baseada no diagrama apresentado na Figura 2.

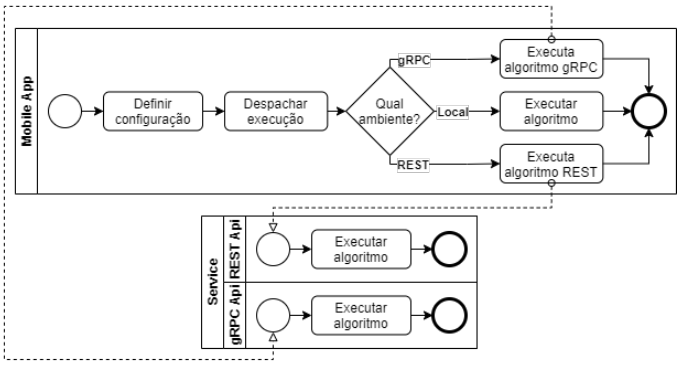

Figura 2. Estrutura do fluxo da aplicação
Fonte: Elaborada pelo autor

Para a execução da aplicação Android, utilizou-se um dispositivo Samsung Galaxy S7 edge com Android 8.0.0 (Oreo), com bateria de 3600mAh, dois processadores Octa Core, Quad-core 2.3 GHz M1 Mongoose e Quad-core 1.6 GHz Cortex-A53 e memória

RAM 4Gb. Para a execução dos *web services*, foi utilizada uma instância de serviço do Kubernetes Engine, na plataforma Google Cloud. A instância foi criada na região us-east1-b e de tipo balanceador de carga, com três nós, 6 unidades de processamento central virtual (vCPUs, do inglês *Virtual Central Processing Unit*) e 12GB de memória. A comunicação com os serviços em nuvem se deu através de um *Wi-Fi* com padrão IEEE 802.11g e velocidade de internet de 200 Mbps.

O consumo de bateria foi medido através da API *Battery Manager* do Android, o qual provê informações da porcentagem, em formato inteiro, da bateria. Para a execução dos algoritmos de ordenação do Quadro 1, utilizou-se um volume de dados com 10.000, 50.000 e 100.000 elementos, os quais possuem os tipos *Int*, *Float* e *Any*, que são, respectivamente, representações de valores inteiros, flutuantes e de objetos. O objeto em questão é composto por uma sequência de 36 caracteres, que representam um Identificador Único Universal (UUID, do inglês *Universally Unique Identifier*) da versão 4, e outros dois campos, um de valor inteiro e outro um ponto flutuante. O pior caso dos algoritmos será dado como uma ordenação decrescente dos elementos, e o melhor, como uma ordenação crescente.

4.2. Métodos

Devido ao número de configurações possíveis para realização dos testes, foi elaborado um processo automatizado para que o experimento pudesse ser realizado de forma mais eficiente. Nisso, os algoritmos foram executados pelo menos cinco vezes para cada configuração, afim de ter uma base de dados mais robusta para as análises. Essa automação foi organizada, para cada configuração, segundo o fluxo apresentado na Figura 3.

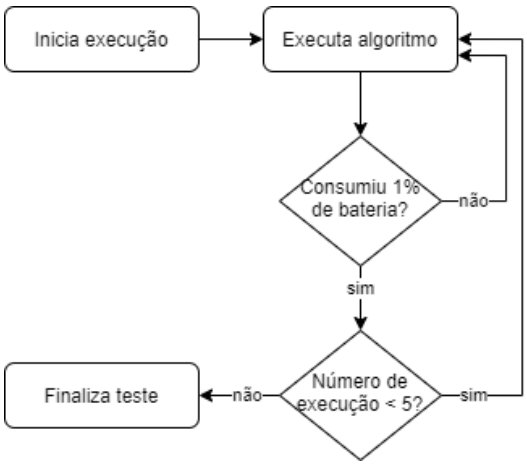

Figura 3. Fluxograma da execução para cada configuração
Fonte: Elaborada pelo autor

Para a realização dos experimentos, foram necessárias adaptações para medição do consumo da bateria, devido à limitação da API utilizada para consultar o nível de bateria do dispositivo, a qual notifica somente números naturais do estado da bateria [Chamas et al. 2017]. Então para que fosse possível recuperar um valor mais próximo ao real para o consumo de bateria, foram realizadas inúmeras execuções até que se consumisse pelo menos 1% da bateria. Caso tenha consumido 1%, esse teste é dado como

finalizado e então calcula-se a média do tempo de execução e do consumo de bateria para esse teste. Após isso, inicia-se mais um teste para a mesma configuração, até que se atinja cinco execuções da mesma. Ao fim de cada teste, os resultados são armazenados em um arquivo de valores separados por vírgula (csv, do inglês *Comma-separated values*).

As execuções de cada configuração, com cinco repetições, foram realizadas até o dispositivo atingir o nível de bateria de 15%, que é o momento ao qual o dispositivo acusa pouca carga. Ao atingir esse nível, o dispositivo é carregado até o seu nível máximo, sendo este 100% de bateria, e então prosseguisse com os experimentos. O tempo de execução e o consumo de bateria são medidos seguindo a ordem de execução dos algoritmos, onde o pontapé inicial é dado pelo envio dos dados para execução do algoritmo, seja ele local ou remoto, e o seu término ao receber o retorno desta chamada.

5. Resultados

Nesta seção são apresentados os resultados encontrados no trabalho. Para atender às questões de pesquisa levantadas, são necessárias as métricas de tempo de execução e do consumo de bateria. Ambas as métricas foram medidas em conjunto para cada configuração do experimento.

Os experimentos foram feitos durante o período de 24 de março a 2 de abril do ano de 2021, no qual houveram algumas instabilidades de rede que podem ter afetado diretamente os resultados obtidos para execuções em ambiente remoto. Os valores em ambiente local, no entanto, não apresentaram o mesmo problema, apesar de que, como apresentado por Chamas et al. (2017), as execuções nesses dispositivos móveis que possuem bateria de lítio tendem a apresentar divergência de resultados quando executados em diferentes níveis de bateria.

Os dados obtidos quanto ao consumo de bateria estão presentes na Tabela 2 (Apêndice A), sendo ordenados de forma crescente com base nos valores obtidos para as execuções dentro do próprio dispositivo (local). Os valores estão expressos em percentual e foram calculados, com cinco execuções para cada configuração, a partir de uma média com exclusão de dois valores extremos, sendo estes o maior e o menor para o intervalo. Os valores destacados são os melhores encontrados para cada configuração, visando facilitar a compreensão e interpretação dos dados.

Os dados obtidos sobre o tempo de execução estão presentes na Tabela 3 (Apêndice B) e, assim como feito na Tabela 2 (Apêndice A), esta possui ordenação crescente a partir dos valores do ambiente local e os melhores valores destacados, para cada configuração, além da mesma metodologia para cálculo das médias. Os dados apresentados nessa tabela estão demonstrados em milissegundos (ms) e são coerentes quanto ao consumo de bateria, possibilitando a correlação dos dados para incrementar o estudo.

6. Discussões de Implicações e Limitações

Visando responder às questões de pesquisa estabelecidas anteriormente, buscou-se levantar dados quanto ao consumo de bateria e o tempo de execução durante o descarregamento computacional e a execução local (no próprio dispositivo) dos algoritmos. Os resultados obtidos apresentam informações relevantes para traçar uma estratégia quanto a solução a ser utilizada, a depender do cenário em que se encontra.

Nesta seção, são discutidas as questões de pesquisa de acordo com os dados presentes nas Tabelas 2 e 3, assim como as limitações referentes ao trabalho. Para fins explicativos, neste trabalho considera-se como uma configuração a combinação do algoritmo utilizado, o seu caso e o tipo de dado. Alguns casos isolados foram selecionados e estão presentes na Tabela 1 para fins demonstrativos, como um resumo dos resultados.

6.1. Qual a influência do volume de dados no tempo de resposta?

Para medir a influência do volume de dados quanto ao tempo de resposta, foi organizado o *data-set* da Tabela 3 (Apêndice B) de acordo com as configurações e o volume de dados. É perceptível a diferença existente dentre as execuções de uma mesma configuração para quantidades diferentes de elementos, onde conforme se aumenta a quantidade de dados a serem ordenados, maior é o tempo que os algoritmos demandam para finalizar. Entretanto, para as execuções remotas, o fator rede pode ter impactado nesta medição, pois são encontrados casos onde o tempo de resposta não correspondeu, necessariamente, a esta regra.

A efeitos de demonstração, a Figura 4 foi criada a partir do percentual do tempo de resposta total para cada ambiente, com base em um cenário isolado apresentado na Tabela 1, sendo possível perceber uma incoerência por parte da solução em REST, a qual apresenta um crescimento inicial e decai ao final, enquanto as demais tenderam a ascender.

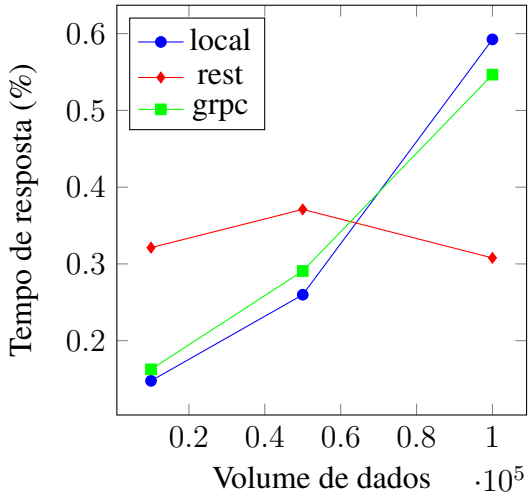

Figura 4. Percentual do tempo de resposta x Volume de dados
Fonte: Elaborada pelo autor

6.2. Qual a influência da tecnologia utilizada no tempo de resposta?

Considerando os dados presentes na Tabela 3 (Apêndice B), percebeu-se que é possível a divisão igualitária do *data-set* em dois grupos: os que foram melhores em ambiente local (em destaque na tabela) e os que não foram. Cada grupo possui exatos 27 elementos cada. No primeiro grupo, das execuções com melhores resultados para execuções no próprio dispositivo, a solução em REST se mostrou superior ao gRPC, sendo melhor em aproximadamente 63% das execuções. Já no segundo grupo, o gRPC teve seu destaque

para as execuções mais demoradas para o dispositivo, sendo superior ao REST em aproximadamente 55.6% das execuções. As diferenças dentre as tecnologias podem ter sido afetadas diretamente por instabilidades de rede, podendo apresentar resultados diferentes se executadas em um ambiente mais controlado.

6.3. Qual a diferença no tempo de resposta com execução local ou externalizada?

Considerando os dados levantados na Tabela 3 (Apêndice B), para as execuções que foram piores em ambiente local, todos possuem complexidade quadrática ($O(n^2)$), dos quais demandam maior poder de processamento, enquanto todos que foram melhores em ambiente local apresentam complexidades melhores, que são o *Heap Sort* ($O(n \log n)$) e o melhor caso do *Bubble Sort* ($O(n)$), conforme apresentados no Quadro 1.

6.4. Qual o impacto causado na bateria se comparadas execuções em ambiente local e remoto?

Considerando agora os dados presentes na Tabela 2 (Apêndice A), é perceptível que, assim como destacado na seção anterior, os algoritmos que possuem menor consumo de bateria para as execuções locais são aqueles com melhores complexidades, segundo o Quadro 1. Com isso, os que se destacaram para as execuções locais foram o *Heap sort*, de complexidade $O(n \log n)$, e o melhor caso do *Bubble sort*, com complexidade $O(n)$. Já as que foram piores para o ambiente local, foram o *Selection sort* e o pior caso do *Bubble sort*, os quais apresentam complexidade $O(n^2)$.

Vale destacar que as execuções remotas apresentaram uma superioridade perceptível para os cenários com maior demanda computacional, onde o consumo de bateria foi bastante reduzido ao descarregar a computação de processamentos pesados, conforme destacados na Tabela 1. Esse destaque também deve ser dado à solução REST que, apesar de ligeira diferença, foi superior ao gRPC em 51,8% dos resultados com pior execução no ambiente local. Já para as configurações que foram melhores em ambiente local, o REST obteve uma diferença ainda mais perceptível sobre o gRPC, sendo melhor em 59,5% dos casos.

Tabela 1. Resumo dos resultados

Seção	Algoritmo	Caso	Tipo	Volume	Local	REST	gRPC
Tempo de Processamento					**ms (desvio padrão)**		
6.1	Bubble	Melhor	Float	10.000	**25 (1)**	4.553,67 (1.981,37)	*1.967,33 (65,61)*
	Bubble	Melhor	Float	50.000	**44 (2,65)**	5.260,33 (2.643,04)	4.517,33 (195,12)
	Bubble	Melhor	Float	100.000	**100,33 (11,02)**	4.364 (344,02)	6.612 (80)
6.2	Heap	Melhor	Any	100.000	**0,01205 (0,00000145)**	0,112 (0,000125)	0,0981 (0,00014)
	Heap	Pior	Any	100.000	**2.635 (10,82)**	4.803.034 (6.792.289,13)	5.034.717 (3.105.868,91)
	Selection	Melhor	Float	10.000	5.955,33 (793,04)	2.892,33 (63,69)	**2.836,67 (42,16)**
	Selection	Pior	Float	10.000	6.120 (90,02)	**3.040 (156,77)**	3.247,67 (135,57)
6.3	Bubble	Melhor	Float	10.000	**25 (1)**	4.553,67 (1.981,37)	1.967,33 (65,61)
	Bubble	Melhor	Int	10.000	**25,67 (5,69)**	2.342,67 (1.422,76)	1.539,67 (46,92)
	Bubble	Melhor	Any	10.000	**212,33 (5,51)**	3.758 (392,31)	5.357 (174,55)
	Bubble	Pior	Float	10.000	48.527 (21.051,94)	**4.387 (75,75)**	4.551 (87,59)
	Bubble	Pior	Int	10.000	207.544 (163.719,63)	4.129 (47,59)	**4.023 (80,29)**
	Bubble	Pior	Any	10.000	1.207.775 (677.226,69)	**6.247 (166,32)**	7.847 (173,51)
Consumo da Bateria					**% (desvio padrão)**		
6.4	Bubble	Melhor	Int	10.000	**0,0000738 (0,000000236)**	0,002289576433 (0,00000189)	0,00342 (0,00000256)
	Bubble	Pior	Int	10.000	0,0893 (0,000116)	**0,00827 (0,00000342)**	0,0102 (0,00000815)
	Bubble	Melhor	Any	100.000	**0,0097 (0,00000911)**	0,0862 (0,0000807)	0,0976 (0,000117)
	Bubble	Pior	Any	100.000	10,333 (0,0115)	0,999999 (0)	**0,5 (0,00000000866)**

Fonte: Elaborada pelo autor

6.5. Limitações

Neste trabalho, o foco em medir execuções em diferentes ambientes possibilitou que valores tenham sido afetados por fatores externos à própria execução dos algoritmos. O

ambiente local (o próprio dispositivo) sofre influências do seu *hardware*, o que pode resultar em diferentes valores se executados em diferentes dispositivos. O ambiente remoto pode sofrer instabilidades na conexão durante a execução, tendo em vista as influências da conexão do usuário e até mesmo o *hardware* de rede do dispositivo.

Os experimentos foram realizados em apenas um aparelho, uma rede e um provedor de serviço, o que pode ter impactado diretamente nos resultados encontrados. Com uma base maior de serviços e dispositivos, seria possível obter uma ideia mais precisa quanto ao real consumo da bateria para os cenários utilizados neste trabalho, assim como do tempo de execução.

7. Conclusões e Trabalhos Futuros

Este trabalho se propôs a implementar e medir o desempenho do uso de descarregamento computacional como solução para dispositivos móveis. Para isso, foram implementados dois serviços em nuvem, um utilizando REST e outro gRPC, para ordenação de elementos. Também foi necessária a implementação de uma aplicação para dispositivos móveis para consumir os serviços e realizar execuções locais dos algoritmos, a fim de levantar dados de consumo de bateria e tempo de resposta para avaliação do desempenho em diferentes ambientes.

As análises mostraram cenários aos quais fariam sentido a implementação do descarregamento computacional, sendo melhor para os cenários onde os algoritmos demandaram mais processamento, como os algoritmos de complexidade quadrática. Além disso, para algoritmos mais eficientes e de baixa complexidade, a melhor alternativa foi utilizar o próprio dispositivo para realizar os cálculos. O volume de dados também não teve tanta influência quanto a complexidade computacional, tendo em vista que todas as execuções que foram piores localmente apresentam a pior complexidade, assim como todas de pior complexidade foram piores localmente.

Considerando apenas o descarregamento computacional, os resultados apontam uma semelhança ao encontrado por Chamas et al. (2017), na qual a solução em REST se mostrou mais econômica que o gRPC, tendo consumido menos bateria do dispositivo para a maioria das requisições. Quanto ao tempo de resposta das soluções, o REST foi superior ao gRPC quando executado os algoritmos mais eficientes, os quais foram mais rápidos se executados no próprio dispositivo. Já o gRPC, obteve um desempenho ligeiramente superior ao REST nos cenários que demandaram maior consumo dos recursos do dispositivo, tendo processado e respondido mais rápido a estas requisições em 55,6% dos casos. Devido a uma diferença muito pequena quanto ao tempo de resposta dentre as soluções, não foi possível apontar uma como ideal, demandando uma maior base de dados para definir a melhor tecnologia.

Para trabalhos futuros, poderiam ser feitas análises a partir de diferentes cenários e mais diversificado, tendo em vista que o presente estudo teve seu foco em apenas um serviço de nuvem, um dispositivo móvel e uma rede. Abranger também o estudo para a utilização de um ambiente controlado, pode revelar os impactos provocados pela rede nos resultados encontrados no presente estudo. Além disso, um estudo baseado no uso de outros modelos de conexão, como 3g e 4g, tendem a apresentar resultados diferentes aos encontrados neste trabalho, tendo em vista que este estudo foi baseado apenas no uso de *Wi-Fi* para a comunicação com os serviços.

Referências

Chamas, C. L., Cordeiro, D., and Eler, M. M. (2017). Comparing rest, soap, socket and grpc in computation offloading of mobile applications: An energy cost analysis. In *2017 IEEE 9th Latin-American Conference on Communications (LATINCOM)*, pages 1–6.

Corbel, R., Stephan, E., and Omnes, N. (2016). Http/1.1 pipelining vs http2 in-the-clear: Performance comparison. In *2016 13th International Conference on New Technologies for Distributed Systems (NOTERE)*, pages 1–6.

Gill, Q. K. and Kaur, K. (2016). A computation offloading scheme for performance enhancement of smart mobile devices for mobile cloud computing. In *2016 International Conference on Next Generation Intelligent Systems (ICNGIS)*, pages 1–6.

gRPC Authors (2020). gRPC: A high-performance, open source universal RPC framework. https://grpc.io/.

Jia, C., Liu, J., Jin, X., Lin, H., An, H., Han, W., Wu, Z., and Chi, M. (2018). Improving the performance of distributed tensorflow with rdma. pages 674–685. International Journal of Parallel Programming.

Kufner, J. and Mařík, R. (2019). Restful state machines and sql database. *IEEE Access*, 7:144603–144617.

Liao, Z.-X., Pan, Y.-C., Peng, W.-C., and Lei, P.-R. (2013). On mining mobile apps usage behavior for predicting apps usage in smartphones. In *Proceedings of the 22nd ACM International Conference on Information Knowledge Management*, CIKM '13, page 609–618, New York, NY, USA. Association for Computing Machinery.

Tang, L. and Li, Q. (2015). Energy and time optimization for wireless computation offloading. In *2015 International Conference on Wireless Communications Signal Processing (WCSP)*, pages 1–5.

Vogel, M., Weber, S., and Zirpins, C. (2018). Experiences on migrating restful web services to graphql. In Braubach, L., Murillo, J. M., Kaviani, N., Lama, M., Burgueño, L., Moha, N., and Oriol, M., editors, *Service-Oriented Computing – ICSOC 2017 Workshops*, pages 283–295, Cham. Springer International Publishing.

Tabela 2. Resultados do consumo da bateria em percentual (%)

Algoritmo	Caso	Tipo	Volume	Local (desvio padrão)	REST (desvio padrão)	gRPC (desvio padrão)
Bubble	Melhor	Float	10.000	**0,0000517 (0,0000000509)**	0,00802 (0,0000388)	0,00481 (0,000000462)
Bubble	Melhor	Int	10.000	**0,0000738 (0,000000236)**	0,002289576433 (0,00000189)	0,00342 (0,00000256)
Heap	Melhor	Float	10.000	**0,000165 (0,00000000367)**	0,00375 (0,0000115)	0,00596 (0,00000181)
Heap	Melhor	Int	10.000	**0,000167 (0,000000168)**	0,00204 (0,00000126)	0,0043 (0,00000916)
Bubble	Melhor	Float	50.000	**0,000171 (0,000000125)**	0,0124 (0,00002)	0,0116 (0,0000133)
Heap	Pior	Int	10.000	**0,000182 (0,0000000157)**	0,00739 (0,000000831)	0,00383 (0,00000370)
Heap	Pior	Float	10.000	**0,000193 (0,000000335)**	0,00824 (0,0000219)	0,00648 (0,00000542)
Bubble	Melhor	Int	50.000	**0,000226 (0,000000628)**	0,00456 (0,000002)	0,0104 (0,00000659)
Bubble	Melhor	Int	100.000	**0,00036 (0,000000328)**	0,0078 (0,00000752)	0,0175 (0,0000123)
Bubble	Melhor	Float	100.000	**0,000382 (0,000000361)**	0,0118 (0,00000361)	0,0184 (0,00000194)
Heap	Melhor	Int	50.000	**0,0009006 (0,00000152)**	0,00521 (0,00000109)	0,014 (0,0000252)
Heap	Pior	Float	50.000	**0,00104 (0,0000000345)**	0,0112 (0,00000253)	0,0119 (0,00000284)
Heap	Melhor	Float	50.000	**0,00104 (0,0000018)**	0,00864 (0,0000159)	0,0269 (0,0000226)
Bubble	Melhor	Any	10.000	**0,00107 (0,000000336)**	0,0114 (0,00000347)	0,0161 (0,0000026)
Heap	Melhor	Any	10.000	**0,00116 (0,00000079)**	0,0111 (0,00000525)	0,0201 (0,00000626)
Heap	Pior	Any	10.000	**0,00125 (0,00000024)**	0,0227 (0,0000266)	0,0227 (0)
Heap	Pior	Int	50.000	**0,00134 (0,000003)**	0,0305 (0,000112)	0,00957 (0,0000076)
Heap	Melhor	Int	100.000	**0,00172 (0,000000163)**	0,00918 (0,00000297)	0,0183 (0,0000199)
Heap	Melhor	Float	100.000	**0,00187 (0,000000616)**	0,015 (0,0000144)	0,0191 (0,0000272)
Heap	Pior	Int	100.000	**0,00212 (0,0000000518)**	0,0266 (0,0000488)	0,0132 (0,00000296)
Heap	Pior	Float	100.000	**0,00229 (0,00000443)**	0,0189 (0,0000244)	0,0167 (0,000016)
Bubble	Melhor	Any	50.000	**0,00494 (0,00000384)**	0,0425 (0,000044)	0,0538 (0,0000453)
Heap	Melhor	Any	50.000	**0,00569 (0,00000512)**	0,0546 (0,0000169)	0,0571 (0,0000642)
Heap	Pior	Any	50.000	**0,00857 (0,0000461)**	0,118 (0,000222)	0,108 (0,000144)
Bubble	Melhor	Any	100.000	**0,0097 (0,00000911)**	0,0862 (0,0000807)	0,0976 (0,000117)
Heap	Melhor	Any	100.000	**0,01205 (0,00000145)**	0,112 (0,000125)	0,0981 (0,00014)
Heap	Pior	Any	100.000	**0,01215 (0,00001014)**	0,156 (0,000392)	0,153 (0,000241)
Selection	Melhor	Int	10.000	0,0169 (0,00000163)	**0,00456 (0,0000066)**	0,00599 (0,00000273)
Selection	Pior	Int	10.000	0,0199 (0,0000465)	**0,00482 (0,00000113)**	0,00679 (0,0000104)
Selection	Melhor	Float	10.000	0,0221 (0,00000736)	**0,00705 (0,00000279)**	0,00745 (0,00000698)
Selection	Pior	Float	10.000	0,0246 (0,000086)	0,00755 (0,00000564)	**0,00733 (0,00000707)**
Selection	Pior	Any	10.000	0,0367 (0,000029)	**0,0147 (0,00000369)**	0,0161 (0,00000704)
Selection	Melhor	Any	10.000	0,0434 (0,0000596)	**0,0201 (0,00000236)**	0,024 (0,0000663)
Bubble	Pior	Float	10.000	0,0756 (0,0000841)	**0,00872 (0,00000561)**	0,0108 (0,00000116)
Bubble	Pior	Int	10.000	0,0893 (0,000116)	**0,00827 (0,00000342)**	0,0102 (0,00000815)
Bubble	Pior	Any	10.000	0,139 (0,000241)	**0,0178 (0,00000181)**	0,0191 (0,00000209)
Selection	Melhor	Int	50.000	0,833 (0,00289)	**0,0172 (0,0000143)**	0,0782 (0,000122)
Selection	Pior	Int	50.000	0,833 (0,00289)	0,0559 (0,0000509)	**0,0493 (0,000029)**
Selection	Pior	Float	50.000	0,833 (0,00289)	0,107 (0,0000642)	**0,0526 (0)**
Selection	Melhor	Float	50.000	0,833 (0,00289)	**0,0367 (0,000029)**	0,116 (0,0000802)
Selection	Melhor	Any	50.000	0,99 (0)	0,0909 (0)	**0,0859 (0,0000437)**
Selection	Pior	Any	50.000	1,33 (0,00577)	**0,0939 (0,0000525)**	0,1007 (0,000101)
Bubble	Pior	Float	50.000	2 (0,0000000346)	0,3055 (0,000481)	**0,111 (0,00000000381)**
Bubble	Pior	Any	50.000	2,333334 (0,0057734)	0,2000002 (0,00000000693)	**0,143 (0,00000000248)**
Bubble	Pior	Int	50.000	2,33334 (0,00577346)	0,189 (0,000192)	**0,107 (0,0000641)**
Selection	Melhor	Int	100.000	3,0000001 (0)	**0,0391 (0,0000307)**	0,333 (0)
Selection	Melhor	Float	100.000	3,000001 (0,0000000346)	**0,0939 (0,0000525)**	0,5 (0,00000000866)
Selection	Pior	Int	100.000	3,000001 (0,0000000346)	0,217 (0,000289)	**0,189 (0,000192)**
Selection	Pior	Any	100.000	3,66667 (0,00577)	0,3333337 (0,0000000114)	**0,306 (0,000481)**
Selection	Pior	Float	100.000	3,66667 (0,0115)	0,5 (0,00000000866)	**0,1999998 (0)**
Selection	Melhor	Any	100.000	4,3334 (0,00577)	**0,2777779 (0,000481)**	0,389 (0,000962)
Bubble	Pior	Int	100.000	8,9999 (0,000000015)	0,8333325 (0,002886748459)	**0,667 (0,00289)**
Bubble	Pior	Float	100.000	9,3333 (0,00577)	1 (0,0000000173)	**0,5000005 (0,0000000173)**
Bubble	Pior	Any	100.000	10,333 (0,0115)	0,999999 (0)	**0,5 (0,00000000866)**

Tabela 3. Resultados do tempo de execução em milissegundos (ms)

Algoritmo	Caso	Tipo	Volume	Local (desvio padrão)	REST (desvio padrão)	gRPC (desvio padrão)
Bubble	Melhor	Float	10.000	**25 (1)**	4.553,67 (1.981,37)	1.967,33 (65,61)
Bubble	Melhor	Int	10.000	**25,67 (5,69)**	2.342,67 (1.422,76)	1.539,67 (46,92)
Bubble	Melhor	Float	50.000	**44 (2,65)**	5.260,33 (2.643,04)	4.517,33 (195,12)
Bubble	Melhor	Int	50.000	**52,67 (5,51)**	1.657 (56,20)	4.325 (248,95)
Heap	Pior	Int	10.000	**54 (1)**	2.782,67 (516,35)	1.548 (56,45)
Heap	Melhor	Int	10.000	**58,33 (3,06)**	774 (54,34)	1.518,67 (73,12)
Heap	Pior	Float	10.000	**60,67 (2,08)**	2.871,33 (655,54)	3.060 (1.229,87)
Heap	Melhor	Float	10.000	**72,67 (11,59)**	1.958,33 (1.527,39)	2.449,33 (117,80)
Bubble	Melhor	Float	100.000	**100,33 (11,02)**	4.364 (344,02)	6.612 (80)
Bubble	Melhor	Int	100.000	**119,67 (16,01)**	3.111 (79,57)	6.350 (530,03)
Bubble	Melhor	Any	10.000	**212,33 (5,51)**	3.758 (392,31)	5.357 (174,55)
Heap	Pior	Int	50.000	**219 (44,40)**	24.256 (27.006,41)	3.553 (28,69)
Heap	Melhor	Any	10.000	**237 (19,08)**	3.755 (34,04)	7.006,33 (301,69)
Heap	Pior	Any	10.000	**251,67 (17,39)**	5.198 (1.504,99)	9.528,67 (783,11)
Heap	Melhor	Int	50.000	**260 (10,58)**	1.863,67 (67,12)	4.859,67 (276,52)
Heap	Melhor	Float	50.000	**276 (51,22)**	2.727,33 (83,81)	9.702,67 (2.735,70)
Heap	Pior	Float	50.000	**303 (4,58)**	4.287,67 (722,85)	5.082,33 (200,14)
Heap	Melhor	Int	100.000	**569 (29,82)**	3.293,33 (162,21)	6.943,33 (386,73)
Heap	Pior	Float	100.000	**582,33 (34,49)**	7.055 (565,62)	6.577 (211,37)
Heap	Pior	Int	100.000	**590 (5)**	8.037 (2.467,46)	6.242,33 (330,86)
Heap	Melhor	Float	100.000	**621,67 (15,53)**	5.060 (14)	7.292,33 (909,90)
Bubble	Melhor	Any	50.000	**986,33 (7,02)**	13.487,33 (27,57)	19.163,67 (1.691,10)
Heap	Melhor	Any	50.000	**1.264,67 (10,07)**	15.922 (725,07)	24.288,67 (5.114,72)
Heap	Pior	Any	50.000	**1.282,67 (68,30)**	1.236.213 (1.589.085,23)	784.764,33 (642.296,55)
Bubble	Melhor	Any	100.000	**1.954,67 (5,03)**	128.102 (56.621,10)	315.355 (244.790,25)
Heap	Melhor	Any	100.000	**2.548,33 (27,32)**	684.113,33 (425.671,24)	347.134 (252.474,84)
Heap	Pior	Any	100.000	**2.635 (10,82)**	4.803.034 (6.792.289,13)	5.034.717 (3.105.868,91)
Selection	Pior	Int	10.000	5.624,67 (192,79)	**2.312 (25,24)**	4.225,67 (2.526,49)
Selection	Melhor	Float	10.000	5.955,33 (793,04)	2.892,33 (63,69)	**2.836,67 (42,16)**
Selection	Pior	Float	10.000	6.120 (90,02)	**3.040 (156,77)**	3.247,67 (135,57)
Selection	Melhor	Int	10.000	6.282,33 (39,55)	**1.840,33 (128,16)**	2.704,33 (41,79)
Selection	Pior	Any	10.000	11.814,67 (1.167,14)	**5.358,33 (265,72)**	7.138,33 (1.793,53)
Selection	Melhor	Any	10.000	13.224,67 (1.370,84)	7.085,67 (110,62)	**6.909 (2.268,12)**
Bubble	Pior	Float	10.000	48.527 (21.051,94)	**4.387 (75,75)**	4.551 (87,59)
Bubble	Pior	Int	10.000	207.544 (163.719,63)	4.129 (47,59)	**4.023 (80,29)**
Bubble	Pior	Any	10.000	1.207.775 (677.226,69)	**6.247 (166,32)**	7.847 (173,51)
Selection	Melhor	Int	50.000	123.211.826 (35.093.553,93)	**7.846,67 (86,43)**	115.056 (75.113,65)
Selection	Melhor	Float	50.000	124.521.855 (35.462.646,39)	**18.028 (834,43)**	549.173 (168.224,70)
Selection	Melhor	Any	50.000	166.671.620 (5.349.289,61)	185.111 (821,76)	**138.066,33 (43.176,29)**
Selection	Melhor	Int	100.000	167.400.432,33 (844.523,01)	**19.595,67 (228,61)**	33.014.974,67 (12.037.709,05)
Selection	Melhor	Float	100.000	170.260.089,33 (826.007,88)	**246.114 (86.691,22)**	83.432.831,33 (5.891.623,70)
Selection	Melhor	Any	100.000	172.555.875 (427.601,84)	**25.972.015,33 (10.175.106,53)**	55.194.577 (23.310.237,67)
Selection	Pior	Float	50.000	186.550.993 (58.265.244,89)	651.893,33 (202.293)	**29.304,67 (1.337,85)**
Selection	Pior	Int	50.000	208.031.028,67 (59.315.697,90)	33.149 (7.506,58)	**28.033 (188,55)**
Bubble	Pior	Int	50.000	209.681.408 (6.812.806,45)	8.822.739 (2.971.424)	**620.149 (193.442,08)**
Selection	Pior	Any	50.000	212.024.425,33 (26.676.422,24)	**276.652,67 (100.405,33)**	559.109 (369.248,77)
Bubble	Pior	Int	100.000	217.912.990 (243.743,45)	170.616.449 (53.199.259,27)	**130.429.856 (36.280.644,35)**
Selection	Pior	Any	100.000	223.220.198 (1.175.762,18)	56.144.367,67 (1.661.295,55)	**48.155.546,67 (17.093.414,56)**
Selection	Pior	Float	100.000	225.687.464,33 (379.903,01)	104.902.868,33 (11.359.569,78)	**13.289.489,33 (81.143,23**
Bubble	Pior	Any	50.000	238.101.712 (33.166.082,05)	10.937.421 (155.812,16)	**2.818.001 (16.340,47)**
Bubble	Pior	Float	50.000	255.583.933 (8.246.468,31)	38.190.852 (13.650.880,80)	**737.270 (1.508,87)**
Bubble	Pior	Float	100.000	265.274.175 (524.958,28)	213.170.830 (14.509.319,18)	**94.582.096 (6.603.862,73)**
Bubble	Pior	Any	100.000	265.461.785 (7.224.694,73)	212.946.867 (17.535.485,49)	**94.111.400 (5.756.500,79)**
Selection	Pior	Int	100.000	281.959.771,33 (1.247.612,24)	15.529.211,33 (6.433.102,87)	**11.040.078,67 (3.812.650,76)**