Transformando Smartphones Android em Dispositivos Dew para Mitigar Limitações de Offloading em Redes Instáveis

Filipe de Matos[1], **Paulo A. L. Rego**[2], **Fernando Trinta**[2]

[1]GOHaN Research Group – Campus of Crateús – Federal University of Ceará (UFC)
63700-000 – Crateús – CE – Brasil

[2]ATLab – Campus of Pici – Federal University of Ceará (UFC)
60440-554 – Fortaleza – CE – Brasil

filipe.fernandes@crateus.ufc.br, {paulo,fernando.trinta}@dc.ufc.br

Abstract. *Offloading helps overcome mobile devices' computational and energy limits, but its performance depends on an efficient server language and a stable network, which are not always guaranteed. Dew Computing converts client devices into Dew devices, enabling local execution of server processes and prioritizing internal interactions. This work defines the Duplication Management Service (SGD), which transforms Android smartphones into Dew devices and evaluates its impact. Tests with real devices showed that Dew enables tasks previously infeasible, reduces network traffic by up to 2.5x, and computes tasks up to 2.9x faster in high competition scenarios despite consuming up to 19.4x more energy. It is a promising option for unstable networks or low bandwidth.*

Resumo. *O offloading ajuda a superar as limitações de processamento e energia de dispositivos móveis, mas seu desempenho depende de uma linguagem de servidor eficiente e de uma rede estável, o que nem sempre é garantido. A Dew Computing transforma dispositivos clientes em dispositivos Dew, permitindo a execução local de processos de servidor e priorizando interações internas. Este trabalho define o Serviço de Gerenciamento de Duplicação (SGD), que converte smartphones Android em dispositivos Dew e avalia seu impacto. Testes com dispositivos reais mostraram que a Dew viabiliza tarefas antes inviáveis, reduz o tráfego de rede em até 2,5 vezes e realiza tarefas até 2,9 vezes mais rápido em cenários de alta competição, apesar de consumir até 19,4 vezes mais energia. É uma opção promissora para redes instáveis ou com baixa largura de banda.*

1. Introdução

Dispositivos móveis, como *smartphones* e *tablets*, possuem restrições computacionais e energéticas em comparação a *desktops* e servidores. O *offloading*, técnica que consiste no envio de tarefas via rede para serem processadas por máquinas mais poderosas, poupando recursos do dispositivo remetente [De 2016], tem se mostrado eficiente para mitigar tais limitações. No entanto, o *offloading* ainda enfrenta desafios, como a influência da latência da rede em seu desempenho [Pedhadiya et al. 2019, Shakarami et al. 2020].

A constante evolução do hardware dos dispositivos computacionais motivou o surgimento de um novo paradigma pós-Computação em Nuvem, denominado *Dew Computing* [Wang 2015]. A *Dew* propõe trazer partes dos serviços inicialmente alocados na Nuvem, bem como paradigmas pós-Nuvem, para o dispositivo do usuário, assegurando seu

funcionamento mesmo em condições de conectividade limitada ou ausente [Ray 2019]. Para viabilizar a *Dew*, o dispositivo do usuário deve conter componentes que facilitem e automatizem o processo de duplicação, ou seja, a recepção, hospedagem e disponibilização de uma "cópia" dos serviços-alvo na própria máquina do usuário.

No contexto do *offloading* em *smartphones*, um desafio relevante, porém pouco explorado, está relacionado à linguagem de programação utilizada na execução remota das tarefas. A maioria dos estudos nesse área é focada na plataforma Android e na linguagem Java, especialmente devido à popularidade do Android e ao caráter de código aberto da plataforma móvel. Contudo, estudos como [Georgiou et al. 2018, Pereira et al. 2021] apontam que outras linguagens de programação superam o Java em desempenho computacional e eficiência energética, evidenciando as limitações dessa linguagem, sobretudo em dispositivos com recursos restritos.

Para enfrentar esse problema, [de Matos. et al. 2021] combinou as técnicas de *offloading* e de comunicação multilíngue, permitindo que aplicativos clientes, muitas vezes desenvolvidos em linguagens ineficientes, interajam com servidores escritos em linguagens mais eficientes. Essa abordagem, chamada *offloading* multilíngue (MLO), demonstrou que servidores mais eficientes que o Java podem acelerar o processamento de tarefas em até 38% e economizar até 25% de energia do dispositivo cliente. Entretanto, a rede foi o principal limitador, respondendo por até 97% do tempo de resposta. Para lidar com esse problema, [de Matos et al. 2022] criou o Serviço de *Offloading* Multilíngue (SML), que integra o servidor de *offloading* diretamente no *smartphone* cliente, permitindo que o dispositivo execute as demandas do aplicativo independentemente da rede. Os resultados mostraram que mover o processo servidor para o *smartphone* e permitir o autoatendimento reduz em até 87% o tempo de resposta e 25% o consumo de energia em cenários com alto volume de dados, além de diminuir o tráfego de rede em até 97%.

Este trabalho melhora o SML ao remodelar seus componentes, facilitando sua configuração e automatizando seu funcionamento, tornando o *smartphone* Android um dispositivo *Dew* apto a duplicar serviços de *offloading*. Também foram realizados novos testes com uma nova linguagem servidora (Golang) e em cenários mais realistas, incluindo aplicação de filtros em imagens, diferentes filtros e níveis de carga da rede.

2. Fundamentação Teórica

Esta seção discute conceitos básicos para a compreensão da proposta e dos testes realizados: *offloading* multi-linguagem (Seção 2.1) e paradigma *Dew Computing* (Seção 2.2).

2.1. *Offloading* multi-linguagem (MLO)

Em [de Matos. et al. 2021], o MLO foi proposto como uma extensão do *offloading* tradicional, cujo principal diferencial é permitir a interação entre processos implementados em linguagens distintas, aproveitando as características e os benefícios específicos de cada uma. A arquitetura do MLO segue o modelo Cliente-Servidor, no qual o cliente, geralmente um dispositivo móvel, transfere tarefas para servidores mais potentes, que vão desde *cloudlets* locais até infraestruturas em Nuvem. Diferentemente do *offloading* tradicional, o MLO exige a padronização das mensagens para garantir a interoperabilidade entre diferentes linguagens. Ferramentas como gRPC e Apache Thrift viabilizam essa comunicação, assegurando a invocação remota e a serialização eficiente dos dados.

O MLO permite integrar ecossistemas heterogêneos, nos quais cada componente pode ser desenvolvido na linguagem mais adequada às suas funções, otimizando o desempenho e o uso dos recursos computacionais. Tal abordagem é particularmente útil em contextos nos quais dispositivos móveis, sujeitos a restrições computacionais e energéticas, precisam executar suas próprias tarefas e também as de outros dispositivos.

2.2. *Dew Computing*

Dew Computing foi proposto como um novo paradigma computacional que aprimora a relação entre computadores de usuário (PCs) e a Nuvem, baseando-se em dois princípios-chave: independência e colaboração [Wang 2015]. A independência permite que os PCs ofereçam serviços localmente, reduzindo a dependência de conexões de rede instáveis e da própria Nuvem. Já a colaboração otimiza a entrega de serviços ao integrar PC e Nuvem de forma sinérgica, permitindo ajuda mútua entre eles quando existir conexão de rede.

Para viabilizar esse novo paradigma, os PCs atuam como dispositivos *Dew*, configurados para duplicar dados ou serviços da Nuvem, da *Fog* ou da *Edge* [Gushev 2020]. Tal processo de duplicação envolve a transferência de uma cópia, total ou parcial, desses serviços ou dados remotos para o dispositivo *Dew*. Assim, o consumo dos serviços é garantido mesmo sem conectividade, assegurando maior autonomia. Em redes estáveis, o dispositivo *Dew* colabora de maneira automática com a Nuvem, aprimorando a qualidade do serviço, reduzindo a carga nos servidores centrais e melhorando a experiência do usuário com resiliência local e suporte remoto. Embora inicialmente projetado para PCs, o paradigma *Dew* foi expandido para incluir dispositivos como *smartphones* e *smartwatches* em trabalhos recente, como [Garrocho and Oliveira 2020].

Em resumo, a *Dew* otimiza o consumo dos recursos dos dispositivos dos usuários, tornando-os menos dependentes dos paradigmas tradicionais de Nuvem. Ao permitir a operação autônoma, o paradigma reduz os impactos da baixa qualidade de conexão e da instabilidade da rede no processo de *offloading* computacional, proporcionando melhor experiência ao usuário, além de maior resiliência e eficiência ao sistema.

3. Trabalhos Relacionados

Diversos estudos já avaliaram o desempenho de diferentes linguagens de programação na computação local [Pereira et al. 2021, Bugden and Alahmar 2022] ou remota (*i.e.*, via *offloading*) [Georgiou and Spinellis 2019, Araújo et al. 2020] de tarefas, usando aplicações reais e/ou *benchmarks*. Por exemplo, em [Cunha et al. 2024], analisaram-se 20 linguagens, Java e Golang inclusas, considerando consumo energético e tempo de execução. Avaliando as métricas em conjunto, C/C++ se destacaram entre as linguagens compiladas, Julia e Dart entre as interpretadas e C#, Java e JavaScript entre as que operam em Máquinas Virtuais. Separadamente, Haskell obteve maior eficiência energética, enquanto Julia apresentou o menor tempo de execução. Por outro lado, em [De Matos et al. 2023], avaliaram-se o desempenho e a escalabilidade do OML em redes HSDPA e LTE, comparando-o com soluções mono-linguagem. O OML apresentou bom desempenho, com operações sobre matrizes 1000x1000 processadas 34% mais rapidamente no servidor Golang em HSDPA do que localmente, além de boa escalabilidade, com tempos menores no servidor Golang, independentemente da rede ou do número de clientes.

Embora a *Dew* seja um paradigma relativamente recente, já há um número significativo de trabalhos que a utilizam para resolver problemas em diversas áreas do co-

nhecimento [Garrocho and Oliveira 2020, Singh et al. 2023, Yannibelli et al. 2023]. Por exemplo, em [Bera et al. 2023], os autores desenvolveram um sistema para prever a produtividade do solo em um ambiente de *Internet of Agricultural Things* (IoAT), modelado em quatro camadas: *IoT*, *Dew*, *Edge* e *Cloud*. A camada *Dew*, composta por dispositivos com alto poder computacional, acumula e pré-processa os dados dos sensores da camada *IoT*. Os autores conduziram testes emulados que avaliaram o impacto da camada *Dew* em um ambiente IoAT. Os resultados mostraram que a presença da *Dew* reduziu o tempo de resposta em até 70% e o consumo de energia em até 80%.

Os trabalhos do primeiro grupo mostraram que a escolha estratégica de linguagens de programação pode otimizar o desempenho do sistema, economizando recursos computacionais e energéticos, com ou sem *offloading*. Já o segundo grupo destacou que a *Dew* acelera o processamento, reduz o consumo de largura de banda e torna os dispositivos mais autônomos, reduzindo sua dependência de outros paradigmas computacionais e até mesmo da própria Internet, tornando a execução de aplicações mais resiliente e eficiente em cenários com conectividade limitada. Essas descobertas apontam um caminho promissor para combinar tais abordagens, especialmente em contextos nos quais a computação com linguagem nativa e o *offloading* se mostram ineficientes ou até mesmo inviáveis.

Em [de Matos et al. 2022], houve a primeira tentativa de integrar os dois grupos mencionados ao se desenvolver o Serviço de *Offloading* Multi-Linguagem (SML). No entanto, o SML era apenas um protótipo, concebido como uma prova de conceito para verificar a viabilidade de duplicar um processo servidor remoto, escrito em uma linguagem mais eficiente, em um *smartphone* Android e avaliar os benefícios gerais dessa abordagem. Este trabalho aprimora a versão original, refinando a arquitetura do serviço Android para torná-lo mais generalista e adaptável a diferentes aplicações, além de conduzir novos testes mais alinhados com cenários do mundo real.

4. Serviço de Gerenciamento de Duplicação (SGD)

Esta seção define o Serviço de Gerenciamento de Duplicação (SGD), uma versão aprimorada do Serviço de *Offloading* Multi-Linguagem (SML) de [de Matos et al. 2022]. O SGD se diferencia do SML por: i) Implementar um mecanismo automatizado para localizar o Lado Servidor e duplicar o processo; ii) Separar as três principais atividades em componentes distintos; iii) Criar três modelos de estratégias, a serem definidas em tempo de execução do SGD, para guiar o comportamento de cada componente. As Seções 4.1 e 4.2 descrevem, respectivamente, os componentes dos lados Servidor (coloridos em verde) e Cliente (coloridos em roxo) da Figura 1.

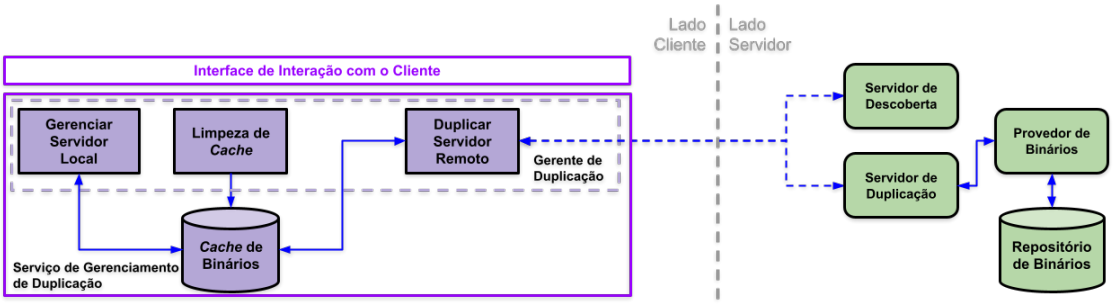

Figura 1. Arquitetura do Serviço de Gerenciamento de Duplicação

4.1. Lado Servidor

O Serviço de Duplicação permite que o Lado Cliente, hospedado em um dispositivo Android, obtenha os binários dos processos servidores. Um binário de processo servidor é um executável utilizado para iniciar o servidor em uma plataforma específica. Portanto, é por meio do *download* e da posterior execução de um desses arquivos na plataforma Android que o processo servidor é duplicado. O Serviço de Duplicação aguarda requisições REST e retorna o binário alvo se a requisição contiver, no mínimo, os identificadores do processo servidor e da plataforma onde o Lado Cliente está sendo executado. O serviço possui dois componentes principais: o Servidor de Duplicação e o Provedor de Binários.

O Servidor de Duplicação, desenvolvido com Flask, oferece o *endpoint /bins* para atender às requisições do Lado Cliente relacionadas à duplicação de processos. As requisições são validadas para garantir que contenham as informações necessárias para que o Provedor de Binários localize o binário alvo no Repositório. Requisições inválidas retornam erro e encerram a duplicação. O Provedor de Binários realiza apenas operações de leitura, e o *endpoint /bins* suporta apenas o método GET do HTTP.

O Serviço de Descoberta fornece ao Lado Cliente, no início da duplicação, o endereço IP da máquina que hospeda o Lado Servidor, permitindo a interação com o Serviço de Duplicação. Para facilitar sua localização, ele foi inserido em um grupo *Multicast* conhecido, escutando em uma porta padrão. Assim, para estabelecer uma conexão com o Serviço de Descoberta, o Lado Cliente envia uma mensagem para o grupo *Multicast* e recebe o IP da máquina servidora em resposta. Nenhum módulo de terceiros foi utilizado no desenvolvimento deste serviço.

4.2. Lado Cliente

Como a pesquisa é voltada para *smartphones* Android, optou-se por implementar o Lado Cliente como um serviço nessa plataforma, visando proporcionar o máximo de autonomia a ele. Nesse contexto, foi proposto o SGD, que, uma vez configurado, age de forma quase independente do aplicativo Android que o instanciou.

O SGD tem dois componentes fundamentais. A *Cache* de Binários é uma pasta no *smartphone* onde são armazenados os binários dos servidores duplicados ou em processo de duplicação. É essencial que o SGD tenha privilégios de leitura, escrita e, principalmente, de execução dos arquivos persistidos nessa pasta, uma vez que ele deverá salvar e executar os binários dos processos servidores nela armazenados. O Gerente de Duplicação, por sua vez, controla as ações relacionadas à duplicação: 1) Duplicar Servidor Remoto; 2) Administrar Servidor Local; e 3) Limpar a *Cache* de Binários.

A ação de Duplicar Servidor Remoto envolve salvar o binário do processo servidor na *Cache* de Binários. Primeiro, decide-se quando iniciar o procedimento, considerando o contexto do *smartphone* (como consumo de recursos) e/ou do ambiente (como latência da rede). Em seguida, busca-se a máquina servidora via Serviço de Descoberta. Após obter o endereço, realiza-se o *download* do binário via Serviço de Duplicação, salvando-o na *Cache*. Tal ação é repetida sempre que o binário alvo não estiver na *Cache*.

A ação de Administrar Servidor Local envolve identificar o melhor momento para iniciar e encerrar um processo servidor local, bem como executar essas ações. A inicialização de um processo servidor local exige que seu binário já esteja na *Cache*, indicando que a ação de Duplicar Servidor Remoto ocorreu previamente. Com base no

contexto interno do *smartphone*, a atividade avalia periodicamente se é adequado iniciar ou encerrar um processo servidor, executando a ação conforme necessário.

Finalmente, a ação de Limpar a *Cache* de Binários envolve excluir binários do sistema de arquivos do *smartphone*, com o objetivo de poupar recursos computacionais do dispositivo. Tal ação avalia periodicamente se o binário do processo servidor pode ser removido da *Cache*. A decisão considera se o binário está em uso e pode levar em conta o contexto do *smartphone*, como se o armazenamento está cheio ou quase saturado.

Cada atividade é executada por uma *thread* específica, instanciada durante a inicialização do Gerente de Duplicação. O comportamento de cada *thread* é determinado dinamicamente durante a configuração do Gerente de Duplicação, por meio de estratégias. Além das estratégias, o Gerente de Duplicação também precisa saber a frequência com que cada atividade será realizada. Assim, durante a configuração do serviço, também devem ser informados os intervalos de tempo em que cada *thread* será escalonada para execução. Por padrão, as atividades ocorrem a cada N minutos, onde N foi inicialmente definido como 1, exceto a de Limpar a *Cache*, realizada a cada N/3 minutos. Toda a etapa de configuração ocorre por meio da Interface de Interação com o Cliente.

A versão atual do SGD inclui três estratégias para cada uma das atividades mencionadas. A estratégia de Duplicar Servidor Remoto cria uma cópia do processo servidor sempre que o binário do servidor alvo não está presente na *Cache* de Binários. Já a estratégia de Administrar Servidor Local inicia o servidor quando seu binário está na *Cache*, mas o servidor ainda não está em execução. Por fim, a estratégia de Limpar a *Cache* de Binários executa a limpeza sempre que o processo servidor não está ativo no *smartphone*. Perceba que cada estratégia executa suas ações sem considerar o contexto do *smartphone* ou da rede, atualmente. Isso acontece porque, na versão atual do SGD, cada estratégia tem apenas um papel ilustrativo. A expectativa é que, em trabalhos futuros, sejam desenvolvidas estratégias reais de tomada de decisão, capazes de determinar os momentos mais adequados para executar as ações estipuladas para cada atividade.

5. Experimentos e Resultados

O ambiente experimental (Tabela 1) foi composto por quatro dispositivos: três *smartphones* clientes e um *notebook* servidor, que atuou como *Cloudlet*. Esses dispositivos foram conectados por uma rede sem fio, organizados em uma topologia estrela e posicionados a poucos centímetros entre si. O *notebook* hospedou ambos os serviços do Lado Servidor, os binários necessários para a duplicação e o processo servidor de *offloading*, enquanto os *smartphones* executavam a aplicação usada nos experimentos e o SGD, configurado com estratégias simplificadas, para replicar o processo servidor remoto localmente. Entendem-se como estratégias simplificadas aquelas desenvolvidas da forma mais simplista possível.

Foi escolhida a aplicação BenchImage [Rego et al. 2017], adaptada para submeter tarefas para processamento através do *framework* Apache Thrift e para iniciar o SGD. Cliente e servidores de *offloading* foram desenvolvidos com os mesmos algoritmos, em Java e Golang, respectivamente, sem paralelismo, garantindo a execução idêntica das tarefas. As tarefas consistiam em aplicar os filtros *GrayScale* e *Pencil* na imagem *SkyLine* de baixa (1 MP) e de alta (8 MP) resolução. O primeiro foi escolhido por ser um filtro básico, enquanto o segundo foi escolhido por aplicar dois filtros (*Gaussian* e *Sobel*) em sequência, o que o torna mais complexo, exigindo mais recursos do dispositivo.

Objetivo(s)	Realizar experimentos de Prova de Conceito com a arquitetura e avaliar os impactos que ela pode proporcionar no processamento de tarefas em uma aplicação móvel.
Sistema (dispositivos utilizados)	**MotorolaGPlay/Cliente** Qualcomm Snapdragon (1.2GHz, Quad Core), 2GB RAM e Android 7, **SamsungJ5/Cliente** Qualcomm Snapdragon (1.2GHz, Quad Core), 1.5GB RAM e Android 6, **MotorolaE6/Cliente** MediaTek (2GHz, Octa Core), 2GB RAM e Android 9, **Notebook/Cloudlet** com Intel Core i7, 12GB RAM e Ubuntu 22.04, **Roteador Netgear WGR612** para construir uma rede WiFi 2,4 GHz exclusiva entre os dispositivos.
Fatores/ Níveis	Cenários (Dispositivos Juntos e e Dispositivos Separados) Tipos de Processamento (*Local*, *Dew* e *Cloudlet*), Linguagens de Níveis de Programação (Golang e Java), Resolução de imagem (1 MP e 8 MP) e Filtro de imagem (*GrayScale* e *Pencil*).
Iterações	Cada experimento foi realizado 50 vezes para cada combinação de Fatores/Níveis; Dentre os resultados, foram removidos os *outliers* e escolhidas as 40 amostras mais rápidas.
Métricas avaliadas	Tempo de Reposta (tempo necessário para aplicar o filtro na imagem), Consumo de Rede (quantidade de *bytes* transmitidos na rede durante o processamento) e Consumo de Energia (energia dispendida para aplicar o filtro na imagem).

Tabela 1. Detalhes sobre a configuração dos experimentos realizados

Também foi introduzido um novo mecanismo de computação de tarefas, a abordagem *Dew*, onde cada tarefa é processada por um processo servidor recém-duplicado no *smartphone* através do SGD. Uma vez em execução, o processo servidor recém-duplicado atende às requisições do aplicativo cliente através da interface de *loopback* e não realiza mais ações na rede externa. Os demais mecanismos são: *Local* (realizada pelo próprio aplicativo cliente) e *Cloudlet* (realizada pelo processo servidor hospedado no *notebook*).

Foram avaliadas as métricas Tempo de Resposta, Consumo de Energia e Consumo de Rede. O Tempo de Resposta considera apenas o tempo de aplicação do filtro, excluindo o tempo de duplicação, que leva cerca de 50s na abordagem *Dew*, segundo testes preliminares. O Consumo de Energia também considera apenas a energia gasta para aplicar o filtro, enquanto o Consumo de Rede mede os *bytes* transmitidos nas abordagens *Dew* e *Cloudlet*. A abordagem *Local* foi desconsiderada por não utilizar a rede. O Tempo de Resposta foi medido na aplicação, o Consumo de Energia via *dumpsys* e o Consumo de Rede através do Wireshark monitorando a porta padrão no servidor.

Os experimentos foram repetidos 50 vezes com o auxílio da ferramenta Ebserver [Oliveira et al. 2023]. *Outliers* (fora da faixa interquartil) foram excluídos e as 40 entradas restantes mais rápidas foram avaliadas. Também foram aplicados testes estatísticos e *post-hoc* para garantir a precisão das conclusões: Kruskal-Wallis seguido de Nemenyi para Tempo de Resposta e Consumo de Energia, e Mann-Whitney para Consumo de Rede. Os resultados são apresentados e discutidos conforme os dois cenários a seguir.

5.1. Cenário 1: Dispositivos Separados

Este cenário comparou as abordagens *Local*, *Dew* e *Cloudlet* no Samsung J5 sem concorrência mútua, garantindo mais recursos na rede e no servidor para operações remotas. A rede não foi isolada de interferências externas, possivelmente causando retransmissões por colisões de sinais. Uma rede com boa largura de banda disponível e um servidor subutilizado favorecem o *offloading* (adotada pela abordagem *Cloudlet*), que depende bastante da transmissão de dados para ser eficiente. Já a abordagem *Dew* se beneficia disso apenas durante a duplicação do processo, operando localmente depois.

A Tabela 2 resume os resultados deste cenário. Em relação ao Tempo de Resposta,

a abordagem *Dew* não superou a *Cloudlet*, independentemente do filtro ou resolução da imagem, como era esperado. Em ambientes com baixa disputa, o *offloading* da *Cloudlet* deve a ser mais eficiente, uma vez que a transmissão de dados tende a ser mais rápida. Além disso, a computação da tarefa acontece no *notebook* na *Cloudlet*, com hardware superior comparado ao *smartphone* da *Dew*, o que afeta a velocidade de computação.

CENÁRIO 1		**Tempo de Reposta (em segs)**			**Consumo de Energia (em Joules)**			**Consumo de Rede (em Kbytes)**	
Filtro	**Res**	**Loc**	**Dew**	**Clo**	**Loc**	**Dew**	**Clo**	**Dew**	**Clo**
GrayScale	**1 MP**	14,12 ±0,03	2,31 ±0,01	1,55 ±0,01	9,69 ±0,42	2,51 ±0,04	0,56 ±0,03	3006,32 ±144,40	1000,68 ±0,62
	8 MP	-	16,99 ±0,03	10,94 ±0,05	-	9,38 ±0,08	0,98 ±0,07	3086,93 ±2,64	7001,15 ±0,01
Pencil	**1 MP**	23,89 ±0,04	4,76 ±0,01	2,01 ±0,01	17,79 ±1,54	3,52 ±0,01	0,59 ±0,03	3078,01 ±0,73	1181,11 ±0,43
	8 MP	-	35,20 ±0,04	15,42 ±0,34	-	19,19 ±0,10	1,02 ±0,02	3083,08 ±1,27	8001,12 ±0,01

Tabela 2. Resultados do *smartphone* SamsungJ5 no Cenário 1

Se por um lado a abordagem *Dew* não superou a *Cloudlet*, por outro ela foi melhor que a *Local* em todas as configurações. Em alguns casos, a abordagem *Dew* computou tarefas que a abordagem *Local* não conseguiu devido a problemas de memória. Tais resultados confirmam a literatura, que aponta o Java (usado no cliente) como mais lento e mais consumidor de recursos do que o Golang (usado no servidor duplicado).

Quanto ao Consumo de Energia, a abordagem *Cloudlet* mostrou-se mais vantajosa, como esperado, pois a *Dew* exige maior participação do *smartphone* ao computar a tarefa, aumentando o consumo de energia, enquanto na *Cloudlet* o dispositivo apenas aguarda o resultado. O Tempo de Resposta reduzido na *Cloudlet* também contribui para sua eficiência. Além disso, a *Dew* foi mais econômica que a *Local*, embora não tenha sido possível comparar para imagens de 8 MP devido ao alto Tempo de Resposta da *Local*.

Quanto ao Consumo de Rede, a abordagem *Dew* apresentou desempenho quase idêntico para todas as resoluções e filtros, pois a interação com a rede ocorre apenas durante a duplicação. O *download* consumiu recursos de rede semelhantes, com pequenas variações devido a interferências externas potencialmente. Para imagens de 8 MP, a *Dew* transmitiu metade dos dados da *Cloudlet*. Para 1 MP, a *Cloudlet* enviou um terço dos dados da *Dew*. Esse comportamento está relacionado ao tamanho das imagens e do binário, pois quanto maior a resolução, maior a quantidade de dados necessária, ao passo que o binário tem tamanho fixo. Esses resultados apontam que a *Dew* pode economizar recursos e reduzir o congestionamento da rede. Os testes estatísticos e *post-hoc* confirmaram diferença significativa entre os valores da Tabela 2, reforçando os argumentos desta seção.

5.2. Cenário 2: Dispositivos Juntos

O segundo cenário comparou as abordagens *Dew* e *Cloudlet* em um ambiente onde os três *smartphones* iniciam o processamento simultaneamente, o que aumenta a disputa e reduz a largura de banda disponível. Além disso, a sobrecarga no *notebook*, que processa as tarefas de três dispositivos, pode tornar o *offloading* mais lento, tornando o cenário mais desafiador para a *Cloudlet*. A *Dew* também é afetada pelo compartilhamento da rede,

porém menos que a *Cloudlet*, pois as ações na rede ocorrem apenas durante a duplicação, quando os três *smartphones* disputam o acesso ao Serviço de Duplicação.

A Tabela 3 resume os resultados obtidos neste cenário. Focando apenas no Tempo de Resposta, nota-se que, ao contrário do cenário anterior, a abordagem *Dew* teve o melhor desempenho em todas as configurações avaliadas em comparação à *Cloudlet*. Tal resultado era esperado, pois o compartilhamento da rede e do *notebook* pelos *smartphones* gera disputas de recursos, afetando o desempenho da *Cloudlet*. A diferença de desempenho entre as abordagens *Local* e *Dew* permaneceu significativa, embora tenha diminuído levemente, pelo mesmo motivo apresentado no cenário anterior.

CENÁRIO 2		**Tempo de Reposta (em segs)**			**Consumo de Energia (em Joules)**			**Consumo de Rede (em Kbytes)**	
Filtro	**Res**	**Loc**	**Dew**	**Clo**	**Loc**	**Dew**	**Clo**	**Dew**	**Clo**
GrayScale	**1 MP**	14,12 ±0,03	2,27 ±0,01	8,39 ±0,01	9,69 ±0,42	2,83 ±0,01	0,57 ±0,01	3815,89 ±2,10	1002,10 ±0,45
	8 MP	-	16,80 ±0,03	50,32 ±0,98	-	9,35 ±0,05	1,17 ±0,02	3905,08 ±157,70	8124,25 ±4,20
Pencil	**1 MP**	23,89 ±0,04	4,73 ±0,01	11,76 ±0,35	17,79 ±1,54	3,88 ±0,05	0,61 ±0,01	3814,49 ±2,24	1183,67 ±0,47
	8 MP	-	35,06 ±0,05	62,37 ±0,99	-	19,11 ±0,17	1,38 ±0,07	3835,49 ±2,56	9095,00 ±5,60

Tabela 3. Resultados do *smartphone* SamsungJ5 no Cenário 2

Já em relação a métrica de Consumo de Energia, ao contrário do Cenário 1, a abordagem com o menor Tempo de Resposta (*Dew*) não foi a que consumiu menos energia. Na abordagem *Dew*, o *smartphone* cliente hospeda também o processo servidor, exigindo um papel ativo para processar as tarefas, o que aumenta o consumo de energia do dispositivo. Em contrapartida, a espera passiva na abordagem *Cloudlet* economiza energia, embora seja significativamente mais demorada. Também nota-se que a *Dew* consome menos energia que a *Local*, devido ao uso de uma linguagem de programação mais eficiente.

Por fim, em relação ao Consumo de Rede, não houve diferenças significativas em relação às observações discutidas no Cenário 1, exceto pelo aumento no tráfego de dados em todas as configurações. Isso se deve à maior disputa no meio compartilhado, com mais *smartphones* usando a rede simultaneamente. Quanto maior a concorrência, mais chances de colisões e retransmissões de pacotes, o que tende a elevar o Consumo de Rede. Como no cenário anterior, os testes estatísticos e *post-hoc* evidenciam diferenças significativas na Tabela 3, reforçando os argumentos desta seção.

5.3. Considerações Finais

A arquitetura do SGD, apesar de sua flexibilidade e robustez, apresenta desafios que devem ser considerados. Um dos principais é a necessidade de desenvolvimento separado para cliente e servidor, possivelmente utilizando linguagens distintas. Essa característica pode aumentar a complexidade do desenvolvimento e manutenção do sistema. No entanto, estratégias como a Engenharia Orientada a Modelos (MDE) ou o uso de Modelos de Linguagem Ampla (LLMs) para tradução de código podem mitigar esse obstáculo.

Além disso, a duplicação do processo servidor no dispositivo móvel pode expor a ataques, como injeção de código e falhas de segurança. Para mitigar esse risco,

recomenda-se usar servidores confiáveis e autenticados, além de realizar o *download* seguro com protocolos criptográficos robustos, como o TLS (*Transport Layer Security*), para garantir a confidencialidade e integridade do binário durante a fase de duplicação.

Outro desafio identificado foi o elevado tempo de duplicação do processo servidor, que pode comprometer a eficiência do sistema, especialmente em cenários dinâmicos e em aplicações sensíveis à latência. Diante disso, propõem-se estudos voltados à criação e à avaliação experimental de soluções concretas para mitigar o atraso causado por esse processo. Entre as estratégias promissoras, destacam-se o uso de técnicas como *cache* distribuída e a duplicação proativa de processos com base na previsão de demanda, apoiada, por exemplo, por modelos inteligentes de tomada de decisão. Em ambos os casos, é essencial buscar um equilíbrio entre a rápida disponibilização do processo e o consumo de recursos do dispositivo do usuário, que, em geral, apresenta limitações nesse aspecto.

Por outro lado, os resultados indicaram que a *Dew*, viabilizada pelo SGD, tem vantagens relevantes. No cenário com alta disputa por recursos, ela superou o *offloading* em velocidade, desde que o processo servidor estivesse previamente duplicado no *smartphone*. Além disso, ela reduziu o impacto da transmissão de dados, essencial para aplicações com comunicação contínua, como jogos. Esses achados destacam a *Dew* como uma alternativa promissora para soluções como as de [Robaina and Fiorese 2023], permitindo execução eficiente sem depender da conectividade com a rede.

Os experimentos também confirmaram achados do estudo anterior, mostrando que a *Dew* mantém um consumo de rede estável, poupando largura de banda em situações de grande volume de dados de entrada e saída. Porém, o maior consumo energético do dispositivo móvel em comparação com o *offloading* precisa ser melhor investigado, pois pode ser uma restrição séria para dispositivos com baterias menores, como *smartwatches*.

Em resumo, os testes confirmaram e expandiram observações anteriores, mostrando que a abordagem *Dew* do SGD amplia as capacidades do *smartphone*, especialmente em redes sobrecarregadas ou instáveis. Contudo, desafios como os altos tempos de duplicação e consumo energético precisam de mais exploração. Estudos futuros devem focar na otimização desses aspectos e na avaliação em aplicações de maior escala, consolidando sua contribuição para a computação distribuída.

6. Conclusão e Trabalhos Futuros

Embora o *offloading* ajude a mitigar as restrições dos dispositivos móveis, ele ainda enfrenta desafios, como a latência de rede, que afeta seu desempenho. Em um trabalho anterior [de Matos et al. 2022], foi desenvolvido um protótipo de serviço Android, chamado Serviço de *Offloading* Multi-Linguagem (SML), que permite que dispositivos móveis assumam processos servidores de *offloading* originalmente hospedados em máquinas remotas, reduzindo os impactos da rede ao realizar a interação localmente.

Este trabalho aprimorou o SML, preservando seu objetivo original, mas reformulando sua arquitetura interna para torná-lo mais flexível às necessidades de diferentes aplicativos. A principal mudança foi a introdução do conceito de estratégias, que permite definir dinamicamente o comportamento dos componentes centrais do SGD em tempo de execução. Tal abordagem proporciona maior controle e personalização, otimizando o desempenho do sistema em diferentes cenários e requisitos de aplicação. Os testes realizados em um novo contexto e cenário reforçaram observações do trabalho anterior e

também revelaram novos resultados sobre os impactos da abordagem *Dew*. Por exemplo, em uma rede congestionada, a *Dew* reduziu o tempo de resposta das tarefas em até 3,7 vezes em relação à abordagem que usa o *offloading*. Além disso, foi identificado que a solução do SGD ampliou a capacidade dos *smartphones* Android, permitindo que, ao duplicar processos servidores desenvolvidos em linguagens mais eficientes do que a adotada no aplicativo cliente, não só houvesse uma redução no tempo de resposta e no consumo de energia do dispositivo móvel, como também fosse possível computar tarefas que anteriormente eram inviáveis com a linguagem nativa.

Como trabalhos futuros, sugere-se o estudo e o desenvolvimento de estratégias que considerem o contexto do *smartphone* (como a carga de bateria disponível) e da rede externa (como o nível de intensidade do sinal) para determinar os melhores momentos para que o SGD possa agir. Também se pretende conduzir uma pesquisa focada na criação de estratégias e no uso de *caching* para reduzir o tempo de duplicação e mitigar os efeitos negativos da ação no desempenho da solução *Dew*. Por fim, realizar testes com novos dispositivos, linguagens servidoras e aplicativos continua sendo bastante promissor.

Referências

Araújo, M., Maia, M. E. F., Rego, P. A. L., and De Souza, J. N. (2020). Performance analysis of computational offloading on embedded platforms using the gRPC framework. In *8th International Workshop on ADVANCEs in ICT Infrastructures and Services (ADVANCE 2020)*, pages 1–8.

Bera, S., Dey, T., Mukherjee, A., and Buyya, R. (2023). E-cropreco: a dew-edge-based multi-parametric crop recommendation framework for internet of agricultural things. *The Journal of Supercomputing*, 79:11965–11999.

Bugden, W. and Alahmar, A. (2022). The safety and performance of prominent programming languages. *International Journal of Software Engineering and Knowledge Engineering*, 32:713–744.

Cunha, S. a., Silva, L., Saraiva, J. a., and Fernandes, J. a. P. (2024). Trading runtime for energy efficiency: Leveraging power caps to save energy across programming languages. In *Proceedings of the 17th ACM SIGPLAN International Conference on Software Language Engineering*, SLE '24, page 130–142, New York, USA. ACM.

De, D. (2016). *Mobile Cloud Computing: Architectures, Algorithms and Applications*. CRC Press.

de Matos, F., Oliveira, W., Castor, F., Rego, P., and Trinta, F. (2022). Multi-language offloading service: An android service aimed at mitigating the network consumption during computation offloading. In *Proceedings of the Brazilian Symposium on Multimedia and the Web*, page 329–338, New York, US. ACM.

de Matos., F., Rego., P., and Trinta., F. (2021). An empirical study about the adoption of multi-language technique in computation offloading in a mobile cloud computing scenario. In *Proceedings of the 11th International Conference on Cloud Computing and Services Science - CLOSER,*, pages 207–214. INSTICC, SciTePress.

De Matos, F., Rego, P. A. L., and Trinta, F. (2023). Evaluating offloading scalability using a multi-language approach on cellular networks. In *2023 IEEE 20th Consumer Communications & Networking Conference*, pages 125–130, Piscataway, US. IEEE.

Garrocho, C. T. B. and Oliveira, R. A. R. (2020). Counting time in drops: views on the role and importance of smartwatches in dew computing. *Wireless Networks*, 26:3139–3157.

Georgiou, S., Kechagia, M., Louridas, P., and Spinellis, D. (2018). What are your programming language's energy-delay implications? In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories*, pages 303–313, New York, US. ACM.

Georgiou, S. and Spinellis, D. (2019). Energy-delay investigation of remote inter-process communication technologies. *Journal of Systems and Software*, 162:1–14.

Gushev, M. (2020). Dew computing architecture for cyber-physical systems and iot. *Internet of Things*, 11:1–9.

Oliveira, W., Moraes, B., Castor, F., and Fernandes, J. P. (2023). Ebserver: Automating resource-usage data collection of android applications. In *2023 IEEE/ACM 10th International Conference on Mobile Software Engineering and Systems*, pages 55–59, Piscataway, US. Institute of Electrical and Electronics Engineers.

Pedhadiya, M. K., Jha, R. K., and Bhatt, H. G. (2019). Device to device communication: A survey. *Journal of Network and Computer Applications*, 129:71–89.

Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., and Saraiva, J. (2021). Ranking programming languages by energy efficiency. *Science of Computer Programming*, 205:1–30.

Ray, P. P. (2019). Minimizing dependency on internetwork: Is dew computing a solution? *Transactions on Emerging Telecommunications Technologies*, 30:1–13.

Rego, P. A., Costa, P. B., Coutinho, E. F., Rocha, L. S., Trinta, F. A., and Souza, J. N. d. (2017). Performing computation offloading on multiple platforms. *Computer Communications*, 105(C):1–13.

Robaina, G. and Fiorese, A. (2023). Gaming on the edge: Uma arquitetura de computação na borda para jogos em dispositivos móveis. In *Anais do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 574–587, Porto Alegre, RS, Brasil. SBC.

Shakarami, A., Shahidinejad, A., and Ghobaei-Arani, M. (2020). A review on the computation offloading approaches in mobile edge computing: A game-theoretic perspective. *Software: Practice and Experience*, 50:1719–1759.

Singh, P., Gaba, G. S., Kaur, A., Hedabou, M., and Gurtov, A. (2023). Dew-cloud-based hierarchical federated learning for intrusion detection in iomt. *IEEE Journal of Biomedical and Health Informatics*, 27:722–731.

Wang, Y. (2015). Cloud-dew architecture. *International Journal of Cloud Computing*, 4:199–210.

Yannibelli, V., Hirsch, M., Toloza, J., Majchrzak, T. A., Zunino, A., and Mateos, C. (2023). Speeding up smartphone-based dew computing: In vivo experiments setup via an evolutionary algorithm. *Sensors*, 23:1–22.