Distribuição multi-nível de carga de processamento para dispositivos IoT e smartphones

Leandro Noman Ferreira[1], **José Marcos Nogueira**[1], **Daniel Fernandes Macedo**[1]

[1]Departamento de Ciência da Computação – Instituto de Ciências Exatas
Universidade Federal de Minas Gerais (UFMG)

leandronoman@dcc.ufmg.br, jmarcos@dcc.ufmg.br, damacedo@dcc.ufmg.br

Resumo. *O termo offloading indica a ação de se alterar o local de processamento de uma atividade computacional. Os objetivos de se usar offloading são reduzir o tempo de processamento de aplicativos, diminuir o consumo energético dos dispositivos e eventualmente possibilitar execução de tarefas que não seriam possíveis em dispositivos de recursos reduzidos. Este trabalho apresenta um arcabouço, chamado de MLOOF, de offloading para smartphones e para dispositivos IoT. A transferência de processamento é feita dos dispositivos para servidores próximos (cloudlet), o que possibilita a redução da latência e aumento da vazão da rede, quando comparados com a transferência para servidores na nuvem. O sistema foi avaliado experimentalmente e os resultados apontam que a estratégia do offloading em três níveis consegue atingir os objetivos de redução de tempo de processamento (em até 74%) e de consumo de energia (em mais de 90%).*

Abstract. *The term offloading indicates the action of changing the processing location of a computational activity. The purpose of using offloading is to reduce the processing time of applications, reduce the power consumption of the devices and eventually enable the execution of tasks that would not be possible on devices with reduced resources. This paper presents an offloading framework, called MLOOF, for smartphones and IoT devices. The offloading process is done from the devices to nearby servers (cloudlet), which enables the reduction of latency and increase in network throughput when compared to the offloading to servers in the cloud. The system was evaluated experimentally and the results show that the strategy of offloading in three levels achieves the goals of reducing processing time (up to 74%) and energy consumption (more than 90%).*

1. Introdução

Os dispositivos móveis têm poder de processamento e capacidade de bateria limitados por natureza [Mao et al. 2016], o que gera uma necessidade de se aumentar a eficiência energética dos aparelhos e sua capacidade computacional, frente à sempre crescente demanda das aplicações [Kumar et al. 2013]. Um dos grandes problemas em sistemas de internet das coisas e em dispositivos móveis é a quantidade limitada de recursos [Mao et al. 2016], [Satyanarayanan 1996]. Seja por falta de memória, poder de processamento ou restrições

Os autores agradecem às agências CAPES, CNPq e Fapemig pelo apoio, seja na forma de bolsas de estudo ou auxílio a pesquisa.

impostas pela bateria dos dispositivos, algumas aplicações são complexas demais ou mesmo inviáveis de serem executadas nesses ambientes. Mesmo quando os dispositivos conseguem executar os programas, eles podem ser obrigados a executar de forma mais lenta [Kumar and Lu 2010] ou gastar muita carga da bateria.

Algumas soluções foram propostas para suprir a necessidade de processamento e, ao mesmo tempo, melhorar a eficiência energética desses dispositivos. Grande parte dessas propostas envolve a utilização de Computação em Nuvem - ou Cloud Computing - para realizar o processamento que seria feito no próprio dispositivo - seja esse dispositivo um smartphone ou dispositivo IoT. Cloud Computing é um paradigma de computação que permite o acesso facilitado a recursos computacionais compartilhados através de uma rede de computadores de forma ubíqua, conveniente e sobre demanda, que possam ser reservados e liberados de forma rápida com mínimo esforço administrativo ou interação com o provedor de serviços [Mell et al. 2011].

Mesmo a utilização da Computação em Nuvem pode não ser suficiente para algumas aplicações que tenham restrições muito grandes no tempo de execução - nas quais o processamento deve ser feito em tempo real, por exemplo. Altas latências provenientes da WAN (Wide Area Network) podem se tornar obstáculos consideráveis para tais aplicações. O conceito de *cloudlet* pode ser visto como a miniaturização de um data center localizado na borda da rede (Edge Computing), tendo como objetivo trazer a *cloud* para mais perto dos dispositivos, representando uma camada intermediária em uma hierarquia de três camadas: dispositivo – *cloudlet* – *cloud*. Dessa forma, é possível atender às demandas por respostas interativas e em tempo real com baixa latência e alta largura de banda utilizando o acesso sem fio à *cloudlet* [Satyanarayanan et al. 2009].

Podemos utilizar as tecnologias em nuvem para alavancar as capacidades de dispositivos com recursos limitados, como smartphones e dispositivos IoT. São utilizadas técnicas para transferência do processo de computação de um dispositivo para outro - de um computador pessoal ou um smartphone para a nuvem, por exemplo. Dessa forma, é possível economizar energia dos dispositivos mais limitados e até mesmo diminuir o tempo de execução das aplicações, tornando viável a execução de programas que requerem alta intensidade de computação que, à primeira vista, poderiam executar em dispositivos móveis [Kumar and Lu 2010]. Tais técnicas são chamadas de *offloading*.

Na literatura, a quantidade de trabalhos que utilizam soluções de *offloading* para dispositivos móveis [Kumar et al. 2013] tem aumentado, sendo a evolução da qualidade das redes móveis [Bhalla and Bhalla 2010] um dos fatores que possibilitam o sucesso de tais soluções. A evolução dos dispositivos móveis (como smartphones) tem possibilitado sua utilização para a realização de tarefas mais complexas que exigem maior poder de processamento mas que, por consequência, consomem mais energia. A melhora na qualidade e capacidade das redes sem fio e a maior preocupação com a duração das baterias de dispositivos móveis criam um ambiente favorável para a aplicação de técnicas de *offloading*, que podem levar à diminuição do consumo energético dos aparelhos e ainda aumentar indiretamente seu poder de processamento.

[El Baz 2014] apresenta algumas aplicações na área de internet das coisas que estão fortemente conectadas a problemas de computação de alto desempenho. A primeira diz respeito à administração de um edifício inteligente, onde milhares de sensores coletam informação em tempo real, e para a qual soluções de problemas combinatórios devem

ser empregadas no controle de todo o sistema. Outra aplicação é relacionada a logística, na qual dispositivos móveis são utilizados para informar sobre entregas bem efetuadas ou incidentes, como falhas em motores dos veículos de entrega e tráfego lento. Os dispositivos móveis também iniciam computações relacionadas a problemas de roteamento para tratarem de incidentes em tempo real, que devem ser realizadas em uma estrutura computadorizada mais robusta. Em todas essas situações uma solução de *offloading* pode aumentar a responsividade da aplicação ou diminuir o gasto energético dos dispositivos.

Este trabalho apresenta a concepção de uma solução de *offloading* para IoT e smartphones utilizando uma infraestrutura de nuvem multi-nível, com servidores em nuvem e *cloudlets* para diminuir a latência e diminuir o tempo de resposta do sistema. Esse trabalho é, em certo sentido, uma evolução do trabalho ULOOF (User-level Online Offloading Framework) [Neto et al. 2018], que trata de *offloading* em estrutura de dois níveis. No presente caso, a proposta adiciona um nível à estrutura do sistema (dispositivo - *cloudlet* - nuvem), e pode ser utilizado com dispositivos IoT.

Este texto está organizado como se segue. A Seção 2 apresenta e discute os trabalhos da literatura, com foco na problemática de *offloading*. A Seção 3 apresenta a solução proposta, em termos dos componentes principais, incluindo o motor para tomada de decisão sobre o *offloading* e a instrumentação necessária para se obter os dados que alimentam o motor de decisão. A Seção 4 apresenta a avaliação da proposta, focalizando nas possíveis vantagens a serem alcançadas com o *offloading* em três níveis. A última seção conclui o texto.

2. Trabalhos Relacionados

A maioria dos trabalhos encontrados na literatura sobre *offloading* de processamento possui foco exclusivo em smartphones, com o objetivo de aumentar a eficiência energética dos aparelhos, tendo em vista que a bateria é um dos principais gargalos dos dispositivos móveis. Dispositivos IoT podem ter capacidade energética e poder de processamento ainda menores do que smartphones atuais, o que faz do *offloading* uma técnica com potencial de aplicação muito grande nesses dispositivos. Nesta seção serão apresentados alguns trabalhos relacionados que proveem soluções em *offloading* para dispositivos móveis.

O trabalho [Cuervo et al. 2010] propôs um *framework* baseado na tecnologia .NET chamado MAUI, no qual o foco é a economia de energia. O MAUI utiliza um *profiler* para medir o consumo energético de uma aplicação e realizar estimativas para comparar o gasto energético ao executar o código no próprio aparelho e o gasto energético para transmitir os dados e códigos para a execução remota. Os autores avaliaram seu *framework* utilizando três aplicações, revelando que *offloading* de computação não apenas economiza energia, mas pode acelerar a execução das aplicações.

O trabalho [Kemp et al. 2010] propôs Cuckoo, um *framework* simples para *offloading* de computação para a plataforma Android. Ele suporta a execução de código local e remota, baseado apenas na disponibilidade do servidor, sempre realizando o *offloading* de um método quando o servidor remoto está disponível. O Cuckoo também implementa uma biblioteca para gerenciar a comunicação entre dispositivo e servidor remoto.

O trabalho [Chun et al. 2011] propôs o CloneCloud, que realiza uma partição do código da aplicação em um conjunto de pontos de execução, de forma automatizada, utilizando análise estática. Para realizar as partições, são utilizadas informações sobre a

qualidade da rede, velocidade de CPU e consumo de energia. Esses pontos de execução são determinados de forma que as partições sejam executadas no ambiente de execução mais eficiente, seja no próprio dispositivo ou em um clone do dispositivo na nuvem. Essa técnica também leva em consideração se o que está sendo otimizado é o tempo de execução da aplicação ou a quantidade de energia consumida pelo dispositivo. O *framework* decide em tempo de execução se uma *thread* da aplicação deve ser migrada para o servidor na nuvem, caso as predições apontem economia de consumo energético ou no tempo de execução da aplicação.

O artigo [Kosta et al. 2012] propôs ThinkAir, que possui um controlador de execução para decidir se a execução de um método deve ser realizada remotamente, baseado no tempo de execução, consumo de energia e no custo para utilizar a infraestrutura de um servidor na nuvem. O tempo de execução e consumo energético são modelados utilizando informação de execuções passadas. Sua *API* permite a especificação de quais métodos podem ser executados remotamente e o *framework* decide se a execução será local ou remota para cada chamada desses métodos. Um gerenciador de conexões foi desenvolvido para receber e executar o código remoto, bem como para retornar os resultados.

O trabalho [Shi et al. 2014] propôs COSMOS, um *framework* que utiliza uma estratégia gulosa para a decisão de *offloading*. Sempre que uma tarefa que pode ser executada remotamente é iniciada, o controlador determina se a execução remota será benéfica ou não. O controlador utiliza o tamanho dos argumentos dos métodos, largura de banda de *upload*, tamanho do resultado e largura de banda de *download* na decisão. As predições são atualizadas ao final de cada execução, ajustando as larguras de banda de *upload* e de *download* previstas.

O trabalho [Flores et al. 2017] propôs HyMobi, um sistema de *offloading* híbrido que utiliza servidores em nuvem, *cloudlets* e comunicação D2D (*Device-to-Device*) para economizar energia de smartphones Android. No HyMobi os dispositivos são organizados em uma ”comunidade” na qual a realização de computação para outros dispositivos gera créditos. Os aparelhos podem consumir créditos para executar métodos de suas aplicações em outros dispositivos. Não são feitas predições de consumo energético ou de tempo de execução dos métodos. As decisões de *offloading* são tomadas com base na pontuação de cada dispositivo participante da rede.

Os autores de [Neto et al. 2018] propuseram ULOOF, um *framework* para *offloading* de processamento de smartphones para a nuvem, criado para ser transparente e não intrusivo, funcionar de forma simples, sem a necessidade de realizar constantes leituras do nível de energia do dispositivo, e levar a localização do usuário em consideração no seu processo de decisão. O programador que utiliza o ULOOF deve apenas importar a biblioteca do *framework* e utilizar marcações em sua aplicação, anotando quais são os métodos candidatos ao *offloading*. Após a compilação da aplicação, um pós-compilador modifica os métodos anotados, integrando a lógica do processo de *offloading* à aplicação.

A Tabela 1 apresenta um comparativo das principais características presentes nos trabalhos da literatura.

3. A Solução Proposta

O sistema proposto é conceitualmente simples, organizado em uma hierarquia de três níveis. As três entidades principais são os dispositivos clientes e os servidores na nuvem

Tabela 1. Tabela de comparação de recursos dos trabalhos.

Framework	**Intrusividade**	**Motor de Decisão**	***Cloudlets***	**IoT**
MAUI	Baixa	Impreciso	Não	Não
Cuckoo	Alta	Não Possui	Não	Não
CloneCloud	Média	Offline	Não	Não
AIOLOS	Alta	Sim	Não	Não
ThinkAir	Média	Impreciso	Não	Não
COSMOS	Alta	Sim	Não	Não
HyMobi	Média	Não Possui	Sim	Não
ULOOF	Baixa	Sim	Não	Não
Sistema proposto (MLOOF)	Baixa	Sim	Sim	Sim

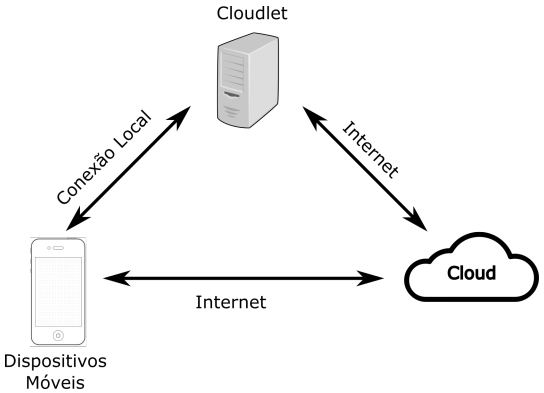

Figura 1. Esquematização do sistema, composto por dispositivos móveis, servidor na cloudlet e servidor na nuvem.

e os servidores na *cloudlet*, todas com capacidade de processamento e comunicação. Os dispositivos clientes executam originalmente as aplicações dos usuários. Os servidores na nuvem e na *cloudlet* têm, por definição, capacidade de executar as mesmas aplicações que executam no dispositivo, mas o fazem apenas quando solicitados. Para a comunicação entre essas entidades, considera-se que existe uma infraestrutura de rede, por meio da qual os dispositivos se conectam à *cloudlet* com baixa latência e possivelmente altas taxas de dados, o que pode ser essencial para aplicações críticas no tempo (requisitos restritos de tempo real). De maneira semelhante, os dispositivos podem se conectar diretamente à nuvem para fazer o *offloading*, entretanto com requisitos de tempo menos restritos, visto que a nuvem pode estar situada em qualquer lugar, o que pode levar a altos atrasos de comunicação. A Figura 1 mostra uma esquematização do sistema proposto.

Cada dispositivo é responsável por estimar a demanda de uma certa computação, antes da sua execução, considerando o tempo de processamento e da transmissão de dados necessária em caso de execução remota, bem como o consumo de energia. Há um processo de decisão que define se a execução de um método (ou a computação) será executado local ou remotamente. Essa decisão é tomada pelo dispositivo, utilizando informações de execuções passadas para estimar e comparar os custos envolvidos nas execuções local e remota. Havendo decisão de execução remota do método, ela ocorrerá preferencialmente na *cloudlet*, por razões de eficiência, ou na nuvem, caso o servidor na *cloudlet* esteja sobrecarregado.

Este trabalho é uma extensão do ULOOF (User-level Online Offloading Framework) [Neto et al. 2018], um *framework* de *offloading* de smartphones Android para a nuvem. O *framework* foi ampliado para rodar em dispositivos IoT e foram desenvolvidos os servidores que rodam nas *cloudlets*. As *cloudlets* permitem a utilização do arcabouço em aplicações que possuam fortes restrições no tempo de execução, que seriam inviáveis de serem executadas na presença da alta latência existente na comunicação com servidores fisicamente distantes dos dispositivos.

As próximas seções explicam com maiores detalhes o papel de cada componente do sistema, denominado MLOOF (Multi-level User Online Offload Framework).

3.1. Dispositivo

O dispositivo móvel (smartphone ou dispositivo IoT) pode executar código de uma aplicação utilizando seus próprios recursos (execução local) ou enviar uma requisição pela internet para que esse código seja executado em uma outra máquina (execução remota). Optando-se pela execução local, o dispositivo utiliza seus recursos computacionais para rodar a aplicação, o que demora uma certa quantidade de tempo e consome certa energia do dispositivo. Optando-se pela execução remota, o dispositivo envia pela rede a informação de qual código deve ser executado e quais dados devem ser utilizados, e aguarda o resultado. Na execução remota, gasta-se tempo para realizar a comunicação entre dispositivo e servidor e para a computação no servidor, bem como consome-se energia para enviar e receber os dados pela rede. O tempo gasto e a energia consumida em ambos os casos dependem de diversas fatores, como a qualidade da rede utilizada, o consumo energético do processador e do rádio dos dispositivos, o poder de processamento do dispositivo e do servidor e a distância entre dispositivo e servidor.

Utilizando informações coletadas sobre execuções passadas no dispositivo, é possível realizar predições de tempo de execução e consumo energético de futuras execuções. Essas predições são importantes para que a decisão sobre onde o código deve ser executado possa ser tomada. Caso a estimativa de tempo de execução local seja maior do que o tempo da execução remota, a decisão de realizar o *offloading* leva à economia de tempo. Caso a estimativa de consumo energético para enviar/receber os dados pela rede seja maior do que a estimativa de consumo para executar o código localmente, a não realização de *offloading* deve economizar energia.

3.1.1. Instrumentação

O *framework* faz predições de tempo de execução e consumo energético de futuras execuções de métodos utilizando um histórico de execuções passadas. Duas execuções de um mesmo método com argumentos similares devem consumir quantidades similares de recursos. Caso o método não se comporte de forma previsível, as primeiras execuções poderão ter predições erradas, mas o histórico será atualizado com os valores das próximas execuções e as próximas predições serão mais precisas.

Um dos aspectos importantes deste *framework* de *offloading* é o baixo impacto no desempenho. Para que esse objetivo seja cumprido, não são feitas medições de consumo energético durante a execução da aplicação. As medições de consumo são feitas de forma *offline* e as informações são codificadas no *framework* de forma estática. Durante a

execução da aplicação, são monitorados o consumo de CPU e a quantidade de dados enviados e recebidos pela rede. Esses valores são usados para estimar o consumo imediato de energia em tempo de execução.

O consumo de CPU é medido utilizando a métrica de *CPU ticks* provido pelo Linux, correspondente à carga exercida no processador. O processador do dispositivo é um recurso compartilhado entre vários processos rodando no sistema, o que faz de *CPU ticks* uma ótima unidade de medida para calcular a taxa de utilização do processador por um método específico, já que o sistema possui um contador de *ticks* separado para cada processo. *CPU ticks* é uma métrica normalizada em cem *ticks* por segundo. Para calcular a taxa de utilização do processador consumida por um método, basta calcular a diferença de *ticks* depois e antes de executar o método e dividir pelo tempo gasto.

A energia gasta para transmitir informações pela rede é medida de acordo com a quantidade de bytes transmitidos. Em tempo de execução, a quantidade de bytes transmitidos é utilizada para estimar o consumo energético causado pela utilização da rede.

3.1.2. Motor de Decisão

O motor de decisão é o módulo do *framework* responsável por executar cálculos de previsão e por decidir quando executar um método localmente e quando executá-lo remotamente. Ele utiliza informações de histórico de execuções, qualidade da rede e estimativas de custo energético para decidir se é mais vantajoso realizar o *offloading* ou executar o método localmente. Esse motor de decisão é o mesmo do *Uloof*, mas uma versão aprimorada com maior precisão está em desenvolvimento.

O motor de decisão trabalha com funções de utilidade de energia e tempo, ponderadas por uma constante α. A constante $0 \leq \alpha \leq 1$ representa uma inclinação para economia de energia ou para diminuição do tempo de execução. Quanto mais perto de zero, maior a inclinação para economizar energia, e quanto mais próxima de um, maior a inclinação para aumentar a responsividade da aplicação. As equações são apresentadas a seguir:

$$L(M) = \alpha * tl(M) + (1 - \alpha) * el(M)$$
$$R(M) = \alpha * tr(M) + (1 - \alpha) * er(M)$$

Nas equações, *M* indica o método que está sendo avaliado, $L(M)$ é a utilidade da execução local e $R(M)$ é a utilidade da execução remota. As funções $tl(M)$ e $tr(M)$ representam as estimativas de tempo de execução local e remota, respectivamente, e as funções $el(M)$ e $er(M)$ indicam as estimativas de consumo energético na execução local e remota, respectivamente. Dessa forma, a decisão escolhe o método de execução de acordo com a função de utilidade de menor custo:

$$Offloading(M) = R(M) < L(M)$$

3.2. Cloudlet

O servidor da *cloudlet* é o recurso principal, ou alternativa preferencial, utilizado pelos dispositivos para a realização do *offloading*. Se o servidor recebe uma requisição

para executar determinado método com determinados argumentos, ele executa o método e transmite o resultado de volta para o dispositivo. Caso não haja nenhuma *cloudlet* disponível ou a *cloudlet* não possua capacidade para atender à requisição, o dispositivo pode se conectar diretamente a um servidor na nuvem.

Ao receber um pedido de *offloading*, o servidor da *cloudlet* primeiramente se comunica com o servidor na nuvem e verifica se existe uma versão mais atualizada da aplicação. Caso a *cloudlet* não possua o código da aplicação ou exista uma versão mais atualizada, o servidor baixa o código de um servidor na nuvem e dá continuação ao processo de *offloading*.

3.3. Nuvem

O servidor na nuvem serve como segunda alternativa para a realização do *offloading* por parte dos dispositivos. Ademais, tem a função de guardar os códigos das aplicações que rodam nos dispositivos, para que ele possa executá-los caso haja uma requisição de *offloading* por parte dos dispositivos, e para que ele possa enviar a aplicação para as *cloudlets* caso elas ainda não possuam o código ou o código esteja desatualizado.

3.4. Implementação

Os programas do arcabouço de *offloading*, servidor da *cloudlet* e servidor da nuvem foram implementados na linguagem de programação Java. A linguagem Java permite a execução de um mesmo código em variadas plataformas de hardware e diferentes sistemas operacionais, incluindo smartphones Android, computadores pessoais e outros hardwares rodando o sistema operacional Linux.

O arcabouço de *offloading* é composto por quatro módulos principais: (1) O Gerenciador de Rede, que armazena as informações e realiza predições sobre a qualidade da rede; (2) O Gerenciador de Execução, que armazena o tempo de execução dos métodos e realiza predições de execuções futuras; (3) O Módulo de Energia, que estima o consumo de energia do processador e da utilização da rede; (4) O Motor de Decisão, que utiliza os outros três módulos para optar pela execução remota ou local dos métodos. Caso a decisão seja executar o *offloading*, o motor de decisão realiza o processo de comunicação com o servidor, envia os dados e aguarda a chegada da resposta. Caso a opção escolhida seja a de executar o método localmente, o motor de decisão faz a chamada do método, o método executa e retorna o resultado obtido.

Aplicações desenvolvidas em Java podem ser facilmente modificadas para utilizar o arcabouço de *offloading*. O desenvolvedor escolhe métodos candidatos ao *offloading* e os anota com a marcação *@OffloadCandidate*. O código já anotado e compilado é então modificado por um pós-compilador, que modifica os métodos anotados para adicionar a lógica de *offloading*, que inclui código para realizar a instrumentação do dispositivo em tempo de execução, criação do histórico de execução dos métodos, motor de decisão e do processo de *offloading* propriamente dito.

Quando um dispositivo cliente se conecta a um servidor de uma *cloudlet* e realiza uma requisição de *offloading*, o servidor deve executar o código requisitado, que faz parte da aplicação rodando no dispositivo cliente. Essa aplicação deve estar presente em algum servidor na nuvem a priori, dessa forma o servidor da *cloudlet* se comunica com o servidor da nuvem para baixar o código da aplicação, caso ainda não o possua, ou para

atualizar o código para uma versão mais recente. Em posse da aplicação, o servidor pode executar o código de um método requisitado pelo dispositivo e enviar o resultado pela rede. O servidor da nuvem pode atender os dispositivos clientes da mesma forma feita pelos servidores das *cloudlets*, porém os dispositivos dão prioridade às *cloudlets* por estarem fisicamente próximas aos dispositivos (normalmente na mesma rede local), o que possibilita uma conexão de melhor qualidade (maior largura de banda e menor latência).

Os aplicativos utilizados para avaliação do sistema também foram desenvolvidos em Java e são explicados com mais detalhes na próxima seção.

4. Avaliação

O objetivo da avaliação é verificar se e quanto a solução proposta atende aos propósitos de liberação de carga e melhoria no tempo de execução de determinadas tarefas de um dispositivo. Para isso, foram executados experimentos em ambiente real com o objetivo de avaliar o impacto que a estratégia de *offloading* pode causar na execução da aplicação, em especial identificar as vantagens de se utilizar servidores em *cloudlets*.

Para um dispositivo que executa uma aplicação, um arcabouço de *offloading* pode ser útil para: economizar energia consumida pelo dispositivo; aumentar de forma indireta o poder de processamento do dispositivo; diminuir o tempo de execução de tarefas computacionais da aplicação. Dessa forma, as **métricas** a serem avaliadas pelos experimentos são:

1. *Tempo de execução* do método, definido como o tempo transcorrido entre a tomada de decisão pelo motor de decisão e obtenção do resultado no dispositivo. No caso de execução remota, inclui o tempo de transmissão dos parâmetros, execução no servidor remoto (*cloudlet* ou nuvem) e de transmissão dos resultados para o dispositivo. No caso de execução local, é apenas o tempo de execução do método no dispositivo;
2. *Consumo de energia* pelo dispositivo, definido como o gasto energético para a execução do método em questão, definido como, para execução local, o consumo desde o início e até o final da execução da tarefa e, para execução remota, o consumo de energia para transmissão de dados (parâmetros de entrada) e de recepção dos resultados.

4.1. Cenários de teste

Diferentes tipos de aplicações com comportamento de execução distintos sofrem impactos diferentes quando executados em uma plataforma remota. Durante o processo de *offloading*, uma aplicação que executa muito processamento em uma pequena quantidade de dados não é tão afetada por uma conexão ruim entre dispositivo e servidor, mas é afetada caso o servidor tenha baixo poder de processamento. De forma semelhante, uma aplicação que trabalha com uma quantidade grande de dados e pouco processamento precisa de uma boa qualidade de rede mas não necessita de um servidor muito poderoso. Por esses motivos, os aplicativos desenvolvidos para os experimentos foram projetados para terem diferentes combinações de carga de processamento e volume da dados a transmitir.

Os cenários de teste permitem avaliar os impactos do *offloading* e o benefício causado pela *cloudlet* no sistema. Foram comparados tempo de execução e consumo

Tabela 2. Cenários de experimentação, para execução local, na cloudlet e na nuvem

Cenários	Carga de processamento	Volume de dados (parâmetros e resultados)	Transações de Rede
A	Alta	Baixo	Poucas
B	Baixa	Alto	Poucas
C	Baixa	Baixo	Muitas

energético de duas aplicações com características distintas (uma consome muito processamento, e outra trabalha com muitos dados). As aplicações são executadas localmente nos dispositivos, realizando o *offloading* para uma *cloudlet* e realizando o *offloading* para um servidor na nuvem. A Tabela 2 apresenta os cenários escolhidos.

Para a execução do cenário de testes A (cenário que requer alta carga de processamento mas transmite pequeno volume de dados pela rede), foi desenvolvido um aplicativo que executa o algoritmo de Fibonacci recursivo. O algoritmo recebe como entrada um número inteiro positivo e calcula o número de Fibonacci de acordo com o seguinte algoritmo:

Algorithm 1: Algoritmo Fibonacci Recursivo

```
1 Function Fibonacci(N):
2     if N <= 1 then
3         Retorna N;
4     else
5         Retorna Fibonacci(N - 1) + Fibonacci(N - 2);
6     end
7 return
```

Para a execução do *offloading* desse aplicativo, é necessário enviar poucos bytes de informação pela rede, que identifica o número de Fibonacci a ser calculado, porém a carga de processamento necessária é grande, por se tratar de um algoritmo com custo exponencial. A complexidade assintótica desse algoritmo é, aproximadamente, $\theta(1.6^n)$.

O cenário de testes B apresenta baixa carga de processamento em uma grande quantidade de dados que devem ser transmitidos pela rede durante o processo de *offloading*. Para realizar os experimentos, foi desenvolvida uma aplicação que transforma uma imagem em sua escala de cinza. A imagem é enviada ao servidor (*cloudlet* ou nuvem) que aplica o filtro de escala de cinza e devolve a imagem filtrada ao dispositivo.

4.2. Experimentos

Para execução dos experimentos de avaliação da proposta, foi utilizado como dispositivo móvel o microcontrolador Raspberry Pi 3 Model B v1.2. Ele possui um processador Quad Core 1.2GHz Broadcom BCM2837 de 64 bits, 1GB de memória RAM, conexão de internet *Wi-Fi* e *Ethernet*, e roda *Raspbian*, um sistema operacional baseado na distribuição *Debian* do Linux. Todos os experimentos foram repetidos no mínimo dez vezes.

Para realizar medições de consumo energético do Raspberry Pi foi criado um hardware separado que, quando colocado entre a fonte de energia e o Raspberry, realiza medições instantâneas de tensão e corrente. Os dados coletados são processados por

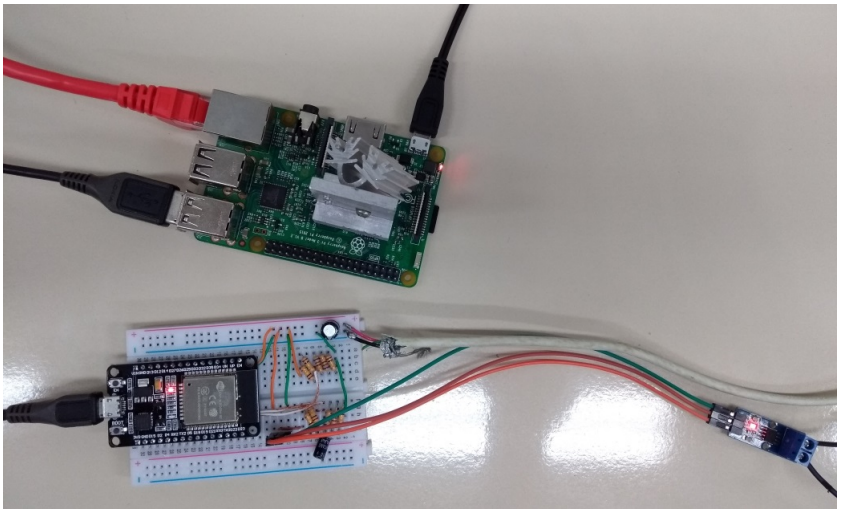

Figura 2. Arranjo para medição de energia consumida (parte inferior da figura). Dispositivo Raspberry Pi (parte superior da figura).

um microcontrolador e transferidos para outra máquina via interface USB para análise. Esse hardware foi criado utilizando-se um kit de desenvolvimento NodeMCU equipado com o microcontrolador ESP-WROOM-32 e um sensor de corrente que emprega o circuito integrado ACS712ELCTR-05B-T. A Figura 2 mostra, na parte inferior, o arranjo montado para medição de energia consumida no dispositivo Raspberry Pi, representado na parte superior da imagem.

No papel de *cloudlet* foi utilizado um laptop que possui um processador Intel Core i3-4005U 1.7GHz, 4GB de memória RAM, rodando o sistema operacional Ubuntu versão 16.04. A *cloudlet* está conectada via *Wi-Fi* à mesma rede local do Raspberry, a um *hop* de distância. No papel de nuvem foi utilizado um PC com processador AMD FX-4300 3.80GHz, 8GB de memória RAM, rodando o sistema operacional Ubuntu versão 16.04. A nuvem está conectada à internet via interface *Ethernet*, e se encontra a quinze *hops* de distância do dispositivo IoT. Durante os experimentos, a latência entre dispositivo e nuvem foi de 170ms na média, e a latência entre dispositivo e *cloudlet* foi de 9ms na média.

4.3. Resultados e Análise

A Figura 3 apresenta os resultados dos testes feitos para o cenário de testes A. O gráfico à esquerda apresenta as médias de tempo de execução do método quando executado localmente e quando há o *offloading* para o servidor da *cloudlet* e para o servidor da nuvem. Como o tempo de processamento do algoritmo aumenta com o aumento da entrada, a energia consumida pelo dispositivo durante a execução local também aumenta. Com a execução remota, porém, a energia consumida se mantém constante, porque não há variação significativa da quantidade de dados transmitidos pela rede quando a entrada do algoritmo varia. O gráfico à direita da Figura 3 apresenta as médias de consumo energético do dispositivo durante a execução remota e local do método.

Para calcular números de Fibonacci pequenos (menores do que 35, nos testes realizados) é mais interessante executar a computação no próprio dispositivo, porque tanto o tempo de execução quanto o consumo energético são menores quando comparados ao

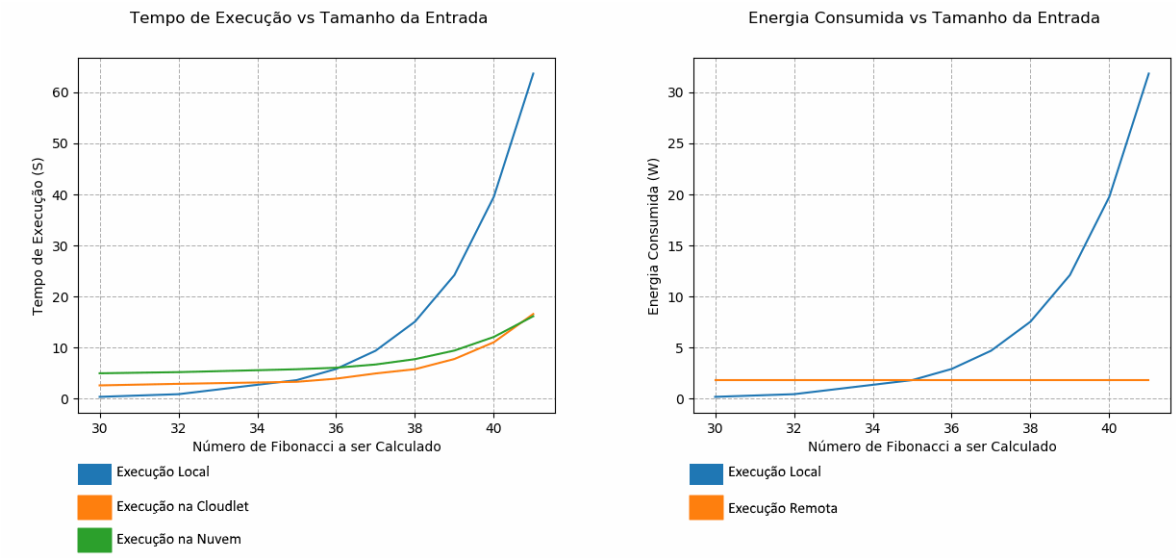

Figura 3. Cenário de testes A: À esquerda, comparação dos tempos de execução do algoritmo de Fibonacci quando executado no Raspberry Pi, no servidor da *cloudlet* e no servidor da nuvem. À direita, consumo energético do dispositivo durante a execução remota e local do método.

tempo e consumo energético de enviar os dados pela rede e aguardar o resultado. Para valores maiores, o consumo energético da computação local supera o consumo de enviar os dados pela rede, e o tempo de execução no hardware do Raspberry supera o tempo de execução no hardware dos servidores mais o tempo de comunicação entre eles. Nos testes realizados, o tempo de execução no dispositivo chega a ser quatro vezes maior do que o tempo de execução remoto, e a energia consumida chega a ser dezessete vezes maior. Para calcular números de Fibonacci maiores, essa diferença seria ainda maior. Ao optar por realizar o *offloading* para *cloudlets*, o tempo de execução chega a ser 49,4% menor do que ao realizar o *offloading* para o servidor da nuvem. Para números de Fibonacci muito grandes, o servidor da nuvem supera o tempo de execução da *cloudlet*, por se tratar de uma máquina com maior poder de processamento. Naturalmente, esses valores dependem das capacidades e disponibilidades dos servidores remotos: quanto maior a capacidade, melhor o resultado.

O gráfico à esquerda da Figura 4 apresenta os resultados dos testes feitos para o cenário de testes B. O gráfico apresenta as médias de tempo de execução ao realizar o *offloading* para a *cloudlet* e para a nuvem, ao variar o tamanho dos dados transmitidos representando imagens. O ponto fundamental desse experimento é o tempo de transmissão das imagens, que é menor para o servidor da *cloudlet* por estar na mesma rede local do Raspberry. O tempo de execução local, bem como o o tempo de processamento dos dados nos servidores, é irrelevante se comparado ao tempo de execução remoto, considerando transmissão e processamento, e por isso não foi apresentado no gráfico. O tempo de execução remoto é, em grande parte, devido à utilização da rede.

O gráfico à direita da Figura 4 apresenta o tempo de execução de diversas chamadas consecutivas do algoritmo de Fibonacci (cenário C). Vários aplicativos com restrições de tempo real necessitam executar métodos em intervalos curtos de tempo, dependendo de suas restrições. Utilizando *cloudlets* no lugar de servidores na nuvem temos ganhos de tempo significativos (até 74% de ganho nos testes realizados), o que pode ser a diferença

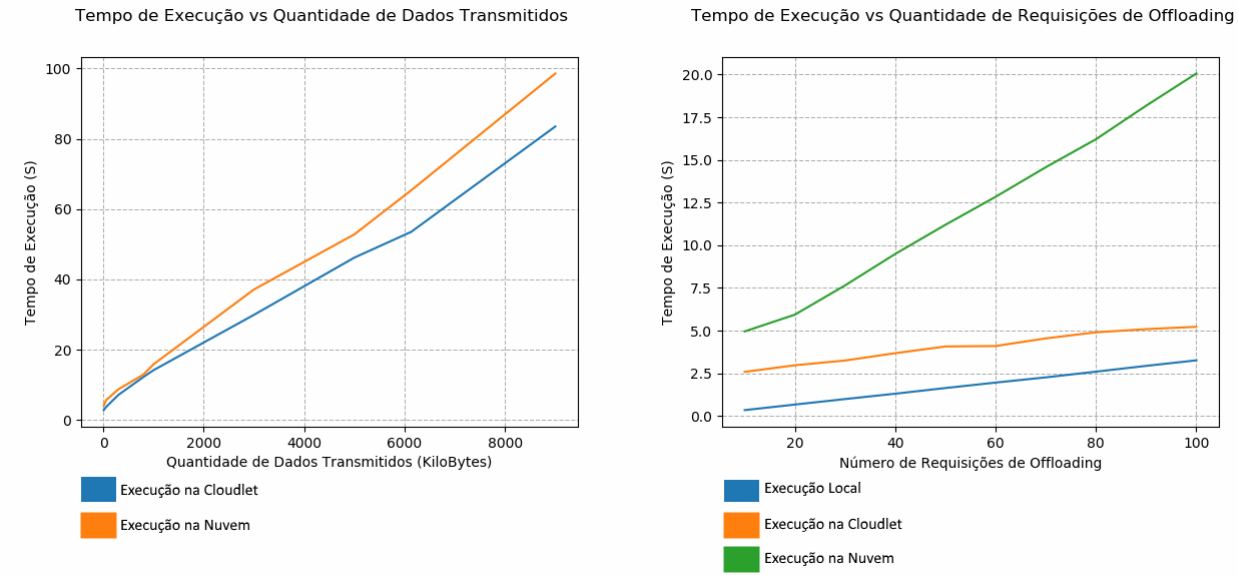

Figura 4. Cenário de testes B, à esquerda: tempos de execução ao transmitir imagens de tamanhos variados para o servidor da *cloudlet* e para o servidor da nuvem. Cenário de testes C, à direita: tempos de execução ao realizar diversas execuções do mesmo método em sequência, simulando uma aplicação com restrições de execução em tempo real.

entre possibilitar ou não a utilização de técnicas de *offloading* nessas aplicações.

5. Conclusão

Este trabalho apresentou o sistema MLOOF, uma proposta de estratégia para descarregamento multi-nível de carga de dispositivos IoT e celulares, como uma evolução de trabalhos da literatura. A partir de dados históricos sobre a execução de determinada tarefa, bem como de dados de contexto, um motor de decisão situado em um dispositivo decide onde é o melhor local para executar a tarefa, se localmente, no servidor na cloudlet ou no servidor na nuvem. Os resultados até então obtidos por experimentação com dispositivos reais mostram que a estratégia para descarregamento em três níveis (dispositivo, cloudlet e nuvem) traz resultados promissores em termos de redução de tempo de execução das tarefas (até 74% de redução) ou consumo de energia dos dispositivos (mais de 90% de economia). A evolução deste trabalho está considerando um aperfeiçoamento no motor de decisão, experimentos com diversos dispositivos no mesmo ambiente, o projeto de um base de dados distribuída de históricos de execução de tarefas e experimentações mais abrangentes utilizando a estrutura do *testbed* do projeto FUTEBOL (EU-BR 3rd Call).

6. Agradecimentos

Os autores agradecem às agências CAPES, CNPq e Fapemig pelo apoio, seja na forma de bolsas de estudo ou auxílio a pesquisa. Agradecimentos também ao aluno de iniciação científica, Gustavo Guedes de Azevedo Barbosa, que projetou e criou o sistema de medição de energia, essencial para a realização dos experimentos desse trabalho.

Referências

Bhalla, M. R. and Bhalla, A. V. (2010). Generations of mobile wireless technology: A survey. *International Journal of Computer Applications*, 5(4).

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., and Patti, A. (2011). Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM.

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. (2010). Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM.

El Baz, D. (2014). Iot and the need for high performance computing. In *Identification, Information and Knowledge in the Internet of Things (IIKI), 2014 International Conference on*, pages 1–6. IEEE.

Flores, H., Sharma, R., Ferreira, D., Kostakos, V., Manner, J., Tarkoma, S., Hui, P., and Li, Y. (2017). Social-aware hybrid mobile offloading. *Pervasive and Mobile Computing*, 36:25–43.

Kemp, R., Palmer, N., Kielmann, T., and Bal, H. E. (2010). Cuckoo: A computation offloading framework for smartphones. In *MobiCASE*, pages 59–79. Springer.

Kosta, S., Aucinas, A., Hui, P., Mortier, R., and Zhang, X. (2012). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Infocom, 2012 Proceedings IEEE*, pages 945–953. IEEE.

Kumar, K., Liu, J., Lu, Y.-H., and Bhargava, B. (2013). A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140.

Kumar, K. and Lu, Y.-H. (2010). Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56.

Mao, Y., Zhang, J., and Letaief, K. B. (2016). Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal on Selected Areas in Communications*, 34(12):3590–3605.

Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing.

Neto, J. L. D., Yu, S.-y., Macedo, D. F., Nogueira, J. M. S., Langar, R., and Secci, S. (2018). Uloof: a user level online offloading framework for mobile edge computing. *IEEE Transactions on Mobile Computing*.

Satyanarayanan, M. (1996). Fundamental challenges in mobile computing. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 1–7. ACM.

Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4).

Shi, C., Habak, K., Pandurangan, P., Ammar, M., Naik, M., and Zegura, E. (2014). Cosmos: computation offloading as a service for mobile devices. In *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*, pages 287–296. ACM.