Estratégias de Tolerância a Falhas em Computação Móvel na Nuvem

Euclides Catumbela
Universidade da Beira Interior, Portugal
Instituto de Telecomunicações (IT), Portugal
euclides.catumbela@ubi.pt

Paula Prata
Universidade da Beira Interior, Portugal
Instituto de Telecomunicações (IT), Portugal
pprata@di.ubi.pt

***Resumo*—Apesar de os periféricos móveis possuírem cada vez mais capacidade de computação e armazenamento, a ligação da computação móvel com a computação na nuvem (cloud) é também, cada vez mais, forte. Aplicações móveis que processem ou partilhem grandes quantidades de dados usam a nuvem para superar a limitação de recursos imposta por `smartphones` e `tablets`. Estes sistemas trazem novos desafios em termos de tolerância a falhas. Por um lado funcionam com baterias cuja carga tem duração limitada e por outro lado, a mobilidade do utilizador pode dificultar a obtenção de conectividade contínua e com largura de banda invariável como seria desejável. Neste trabalho propomos e avaliamos mecanismos de tolerância a falhas para dois tipos de falhas comuns em computação móvel na nuvem: Falha da carga da bateria e falhas na ligação à rede.**

Palavras-chaves: Computação Móvel, Computação em núvem, Firebase, Tolerancia a falhas.

***Abstract*—Although mobile peripherals are increasingly capable of computing and storage, the link between mobile computing and cloud computing is also growing stronger. Mobile applications that process or share large amounts of data use the cloud to overcome the resource constraints imposed by mobile devices. These systems bring new challenges in terms of fault tolerance. On the one hand they work with batteries whose charge is of limited duration and on the other hand the mobility of the user may make it difficult to obtain continuous connectivity and with invariable bandwidth as would be desirable. In this work we propose and evaluate fault tolerance mechanisms for two types of common faults in mobile cloud computing: Battery charge failure and network connection failure.**

Index Terms: Mobile Computing, Cloud Computing, Firebase, Fault Tolerance.

I. Introdução

Assiste-se, na presente era, uma grande evolução, nunca antes vista, dos sistemas e tecnologias de informação. Esta evolução engloba todas as áreas das tecnologias de informação transformando, sobretudo, o modo como acedemos à informação, sem perder de vista a facilidade e a simplicidade no tratamento da informação de programadores para utilizadores.

A computação móvel (MC - *Mobile Computing*) tem feito parte, naturalmente, desta evolução, sendo que os dispositivos móveis (e.g., smartphone e tablet) são, cada vez mais, uma parte essencial do quotidiano do homem como ferramentas de comunicação mais eficazes e convenientes não delimitadas pelo tempo e lugar [1]. O rápido crescimento da MC torna-se uma poderosa tendência no desenvolvimento de tecnologia de informação, bem como nos campos de comércio e indústria [2]. Se antes um telemóvel era utilizado, apenas, para fins de comunicação, com esta evolução, os dispositivos móveis são utilizados para múltiplas funções. Estas funções (e.g., internet, sensores, redes sociais baseadas em localização, etc) têm trazido, à tona, uma das principais dificuldades que a MC enfrenta, as limitações de hardware. Dispositivos móveis possuem limitações em seus recursos de hardware (e.g., vida útil da bateria, armazenamento e banda larga) e recursos de comunicação (e.g., mobilidade e segurança), a lacuna entre a procura por executar tarefas complexas e a disponibilidade de recursos limitados têm aumentado todos os dias [3], [4], [5] .

A computação em nuvem (CC - *Cloud Computing*) tem sido amplamente reconhecida como a infraestrutura de computação da que que permite minimizar, ou até mesmo anular as limitações da MC, podendo fornecer vários serviços [1], [3]. A CC permite que os utilizadoes utilizem infraestruturas, plataformas e pacotes de software oferecidos por provedores de serviços em nuvem a custos acessíveis. [1].

A CC inclui três modelos principais de serviços, que são Infraestrutura como Serviço (IaaS - *do inglês, Infrastructure as a Service*), Plataforma como Serviço (PaaS - *do inglês, Plataform as a Service*) e Software como Serviço ou (SaaS - *do inglês, Software as a Service*) [6].

A CC, através de seus serviços, tem sido uma escapatória para a MC ultrapassar as suas limitações de hardware, dando origem, desta forma, à computação móvel na nuvem (MCC - *Mobile Cloud Computing*).

A. Estrutura do Documento

Este documento é constituído por cinco secções: a primeira faz uma breve introdução sobre aspectos relacionados com a computação móvel e sua evolução até chegar a utilizar serviços de nuvem; a segunda apresenta o estado da arte sobre MCC e aspetos ligados a tolerância a falhas em computação móvel na nuvem; na terceira propomos estratégias de tolerância a falhas para aplicações móveis; na quarta, descrevemos o ambiente experimental onde avaliamos estas estratégias e apresentamos

os resultados obtidos; finalmente, a quinta secção contém a discussão dos resultados e as conclusões.

II. Computação Móvel na Nuvem

O MCC é o mais recente modelo de computação distribuída que estende a vantagem de se utilizar CC para *smartphones* [7]. Estas vantagens reduzem as limitações, anteriormente mencionadas, e dá aos utilizadores de *smartphones* maior capacidade de armazenamento e de processamento.

A Conferência Internacional IEEE sobre Computação, Serviços e Engenharia Móveis em Nuvem define MCC como [8]:

'Computação móvel na nuvem, de forma simples, refere-se a um infraestrutura onde tanto o armazenamento e o processamento de dados pode acontecer fora do dispositivo móvel. Aplicações móveis na nuvem deslocam o poder de computação e o armazenamento de dados dos telemóveis para a nuvem'.

Nem todas as aplicações precisam, necessariamente, de uma nuvem para o seu normal funcionamento. As aplicações para nuvem como explicado em [9] encaixam-se em áreas em que se precisa grandes quantidades de armazenamento e processamento de dados, como processamento de imagens, processamento de linguagem natural, partilha de GPS, partilha de acesso à Internet, aplicações de dados de sensores, pesquisa, computação coletiva e pesquisa de multimédia.

De acordo com [10] os serviços típicos para um cliente em MCC são:

- sync: este serviço sincroniza todas as mudanças feitas na aplicação com o servidor;
- push: faz a gestão das alterações de estado que são enviados pelo servidor. Este acrescenta a experiência do utilizador e não requer que o utilizador constantemente a verificar se houve alguma alteração no servidor;
- offlineApp: este serviço é responsável da por criar uma coordenação inteligente entre serviços de baixo nível como o Sync e Push. Este serviço dispensa o programador de fazer o código para que sua aplicação se sincronize com o servidor. Este serviço decide o melhor mecanismo de sincronização para o estado atual.
- network: Este serviço gere o canal de comunicação que se precisa para receber notificações Push do servidor.
- database: Este serviço gere o armazenamento dos dados para as aplicações móveis.
- interApp Bus: Este serviço é um serviço de baixo nível o qual faz a coordenação/comunicação entre o servidor e as aplicações instaladas no dispositivo.

A Figura 1 mostra que, de acordo com [11], a MCC é composta de três partes principais: o dispositivo móvel, os meios de comunicação sem fio e uma infraestrutura de nuvem. Estes últimos fornecem serviços de armazenamento, processamento e mecanismos de segurança para os dispositivos móveis.

A utilização do processamento e armazenamento remotos, designada por "*offloading*", pode melhorar o desempenho e, ao mesmo tempo, economizar energia da bateria, proporcionando uma melhor experiência ao utilizador. Segundo o trabalho

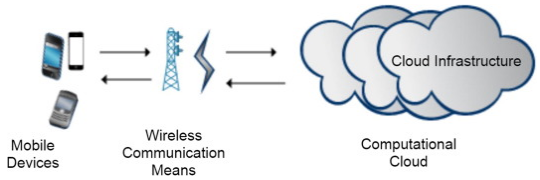

Figura 1. Visão geral da MCC **Fonte:** [11]

apresentado em [12], *Computation offloading*, é um paradigma ou solução para melhorar a capacidade de serviços móveis através da migração das tarefas pesadas para servidores poderosos em nuvens. No entanto, em algumas situações, o "*offloading*"pode ser prejudicial [1]. Estes prejuízos, podem acontecer por não ser necessário uma emigração, dada a pouca quantidade de dados para processamento ou armazenamento, ocupando de maneira desnecessária a nuvem, e por outro lado, por não se prever modelos de tolerância a falhas, durante o planeamento do *offloading*.

A. Tolerância a falhas em MCC

A tolerância a falhas é um aspecto, extremamente, importante em MCC, ainda mais do que em uma nuvem convencional, por causa da natureza móvel dos dispositivos [9], ou seja, como descrito em [13] "a mobilidade é inerentemente perigosa". Diversos factores podem ocasionar uma falha na comunicação entre o dispositivo móvel e a nuvem. A falta de ligação pode acontecer devido à mobilidade do utilizador quando os dispositivos entram e saem de uma rede. Ficar sem carga de bateria, perda de sinal da rede ou falhas de hardware são, entre outros, alguns dos fatores comuns [9].

Estas falhas na utilização de MCC, são tão reais que merecem uma atenção importante por parte dos programadores ou de empresas que, eventualmente, queiram efectuar um *offloading* dos seus serviços móveis. Por esta razão, Marinelli, descreveu, como as principais características de um sistema distribuído móvel as seguintes [14]: **tolerância a falhas**, escalabilidade, privacidade e interoperabilidade do hardware. O mesmo autor descreve, também, que durante a implementação deste tipo de sistemas devem-se ter em conta a carga da bateria, a banda larga utilizável, CPU e a memória do dispositivo, o tempo de execução das tarefas e o armazenamento dos dados. E são estes os principais aspectos que procuramos estudar neste documento.

III. Estratégias de Tolerância a Falhas

Como caso de estudo para ilustrar a implementação de estratégias de tolerância a falhas desenvolvemos no *Android Studio*[1] uma aplicação móvel exemplo que tem como objetivo, efetuar perguntas de diferentes áreas ou categorias ao utilizador do dispositivo móvel, tipo *Quiz*. Cada jogador irá ter associada uma pontuação correspondente aos acertos das questões dentro de cada categoria. Estas perguntas e as

[1] diposnível em https://developer.android.com/studio

respetivas respostas são armazenadas na nuvem utilizando a *Realtime-Database*[2] da plataforma Firebase.

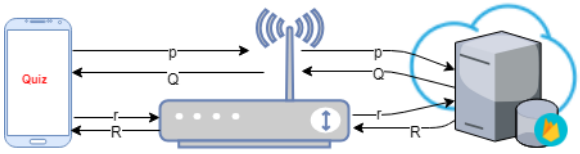

Figura 2. Arquitetura física da aplicação.

A execução deste tipo de aplicações, como MCC, ilustrada na Figura 2, funciona da seguinte maneira: o utilizador na posse de um dispositivo e, necessariamente, ligado à rede faz um pedido **p** de questões ao servidor. O servidor, por sua vez, responde com um conjunto de questões **Q** que são enviados ao utilizador. O utilizador, na posse de **Q**, dá solução as questões e envia as respostas **r** ao servidor. O servidor compara as respostas dadas pelo utilizador com as que tem guardadas calculando o número de respostas corretas. A seguir, o servidor regista o resultado **R** na base de dados e envia ao utilizador.

Para que este processo funcione, sem sobressaltos, há a necessidade de ter uma ligação constante entre o dispositivo cliente e o servidor.

A. *Modelo de falhas*

Durante a execução da aplicação as falhas mais comuns são as seguintes:

- baixo nível de bateria;
- conectividade fraca.

Neste trabalho analisamos estes dois tipos de falhas e deixamos para trabalho futuros falhas na própria aplicação o comportamento da aplicação quando o número de utilizadores em simultâneo aumenta.

B. *Estratégias de tolerância a falhas - soluções propostas*

1) Baixo Nível de Bateria: como um dispositivo móvel opera com recursos finitos de energia em sua bateria, a energia é um dos principais recursos que precisam ser utilizados com cuidado [9], [16] .

Para superar a possível falha de descarga do dispositivo trabalhamos com *checkpoints*[3]. Esta solução permitirá ao utilizador continuar a usar a aplicação assim que tenha carregado o seu dispositivo ou, eventualmente, tenha acesso a outro dispositivo.

Como se pode observar na Figura 3, à medida que se trabalha na aplicação, do dispositivo ***D1.1***, é executado um algoritmo de verificação do nível de bateria de 3 em 3 minutos

[2]O Firebase Realtime Database é um banco de dados hospedado na nuvem. Os dados são armazenados como JSON e sincronizados em tempo real com todos os clientes conectados [15]. Disponível em https://console.firebase.google.com/

[3]Checkpoint é um método de superação de falhas que consiste em guardar resultados parciais para ser possível recuperar de, eventuais, interrupções não previstas durante a execução de determinada tarefa.

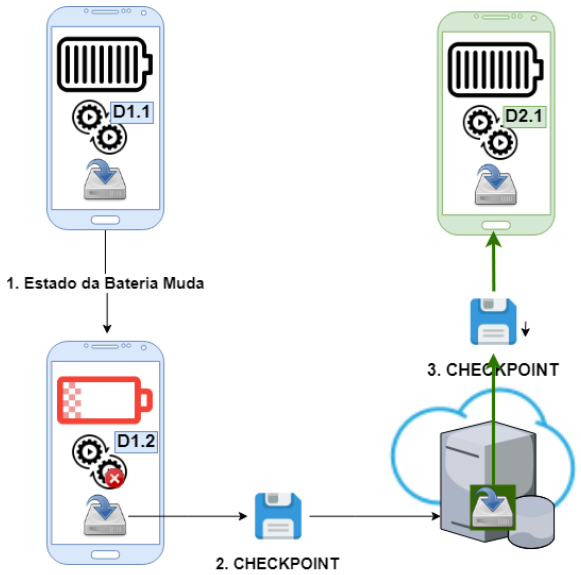

Figura 3. Arquitectura para recuperar de uma descarga de bateria.

(ou outro valor a decidir pelo utilizador) e, enquanto isto, em paralelo guarda-se o progresso do utilizador, referente ao *Quiz* em curso, num ficheiro de texto no dispositivo(*checkpoint*). Se se observar que o nível de bateria é baixo, no mesmo dispositivo ***D1.2***, a aplicação despoleta uma mensagem ao utilizador, a indicar o descarregamento do dispositivo, e o ficheiro de *checkpoint* é enviado ao servidor (seta 2), indexando o ficheiro ao utilizador. A seguir, o utilizador pode efetuar o login a partir de um segundo dispositivo ***D2.1***, ou do mesmo se entretanto recarregou a bateria. O ficheiro de *checkpoint* é enviado do servidor ao dispositivo onde o utilizador está ligado. Assim, o utilizador pode retornar as suas atividades, recomeçando no estado em que ficou antes da descarga.

2) Falha de Ligação ou Ligação Fraca à Rede: A queda de ligação constitui um dos principais desafios da MCC. Em uma nuvem móvel, as localizações geográficas dos utilizadores não são fixas. No entanto, também é vital que as base de dados móveis contenham políticas para proteger contra a perda de dados e, ao mesmo tempo, garantir que esteja em conformidade com as restrições de mobilidade [9].

Solucionamos esta falha, ativando a permanência em disco, ou seja, logo que certa parte dos dados viaja do servidor para o dispositivo, funções necessárias para execução, de tal tarefa, são anexadas aos dados. Desta forma, o utilizador pode efetuar as suas atividades, como se estivesse ligado ao servidor. À medida que se fazem alterações de dados, estes são armazenados em cache no dispositivo, organizados numa fila de acordo com a ordem de chegada.

A Figura 4 ilustra um fluxograma da forma como tratamos a falha de ligação entre o dispositivo e servidor. Logo que **inicia a aplicação**, a aplicação faz o **pedido ao servidor**. O **servidor não só envia os dados**, como também as **funções** necessárias para o tratamento destes dados ao dispositivo cliente. Quando o utilizador **efetua as suas operações e solicita persistência**,

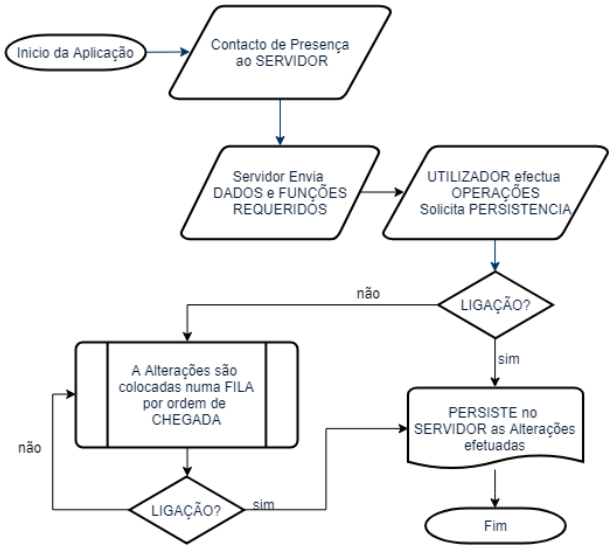

Figura 4. Fluxograma dos passos para recuperar de uma falha de rede

o **algoritmo verifica a ligação ao servidor**, caso haja ligação ele **envia e persiste os dados no servidor**, caso não haja ligação estas alterações são armazenadas numa **fila na cache do dispositivo cliente**. Assim que o sinal de comunicação, entre o dispositivo cliente e o servidor, é **reestabelecido**, os dados existentes na **fila** são enviados para que o servidor possa fazer a devida sincronização.

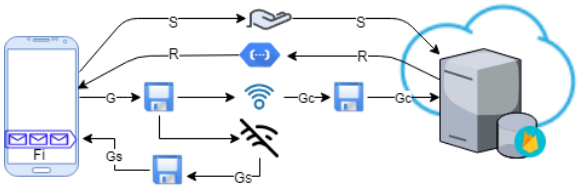

Figura 5. Arquitetura para recuperar de uma falha de rede.

O funcionamento deste algoritmo pode ser facilmente entendido, observando a Figura 5. Quando o dispositivo cliente efetua um determinado pedido **S**, é-lhe dado, pelo servidor, a resposta ao pedido **R**, bem como, as funções necessárias para a consumação de **S**. A seguir, após o tratamento dos dados, quando o dispositivo cliente desejar efetuar qualquer tipo de persistência **G**, é verificada a existência ou não de ligação com o servidor. Caso exista esta ligação a persistência é feita no servidor, **Gc** - guardar **com** ligação, e, caso contrário, os dados são persistidos numa fila **Fi** na cache do dispositivo cliente **Gs**-guardar **sem** ligação.

IV. Ambiente Experimental e Resultados

Após a elaboração dos modelos de tolerância a falhas, estes foram testados usando um computador portátil onde, com o Android Studio, foram criadas duas máquinas virtuais MV1 e MV2. As características deste computador são as seguintes:

- processador: Intel(R) i7-6500U CPU 2.50GHz 2.60GHz;
- RAM: 8GB;
- SSD 240GB;
- S.O: x64.

Foram ainda utilizados dois dispositivos móveis, um tablet (Máquina Física 1 - MF1) e um telemóvel (MF2). As características das duas máquinas virtuais e das duas máquinas físicas constam da Tabela 1:

Tabela I
Características das máquinas utilizadas nos testes.

Máquina	CPU	RAM	Versão do Android
MV1	n.a.*	1GB	7.0 API 24
MV2	n.a.*	1.5	7.0 API 24
MF1	8x ARM A53 @ 1,59GHz	2GB	8.1.0 API 27
MF2	4x Samsung M1 @ 2.60GHz 4x ARM A53 @ 1,59GHz	3.5GB	8.1 API 26

** - não aplicável, as máquinas virtuais no Android Studio utilizam a CPU da máquina em que estão instalados.*

A. Testes

Os testes aplicados foram centrados no desempenho da aplicação, medindo o tempo de transferência e de persistência dos dados entre o dispositivo móvel e o servidor, ou seja procuramos medir a eficiência da aplicação na sua execução normal e quando submetida em situações de falhas.

Sendo a aplicação um *Quiz*, o teste consistiu em medir o tempo de pedido das perguntas ao servidor, o tempo de resposta do servidor, bem como o tempo de persistência das respostas do utilizador.

B. Execução sem implementação dos mecanismos de tolerância a falhas

Durante a execução dos testes, sem aplicação de nenhum dos mecanismos de tolerância a falhas propostos na secção 3, obtiveram-se os seguintes resultados:

Tabela II
Tempos de execução sem mecanismos de tolerância a falhas.

Máquina	Tempo
MV1	6ms
MV2	4ms
MF1	3ms
MF2	2ms

De notar que em todos os testes efetuados, a medição dos tempos de execução não inclui o tempo em que o utilizador responde ao Quiz, pois este tempo depende de cada utilizador e não está relacionado com os mecanismos de tolerância a falhas.

C. Baixo Nível de Bateria

Para medirmos o grau de eficiência do nosso método de recuperação por *checkpoints* decidimos somar os tempos desde o inicio até ao fim do processo, utilizando a seguinte fórmula:

$$t_T = t_{CC} + t_{EC} + t_{RC}$$

onde:

- t_T, é o tempo total desde o inicio ao fim do processo;
- t_{CC}, é o tempo de criação do *checkpoint*;
- t_{EC}, é o tempo de envio do *checkpoint*;
- t_{RC}, é o tempo de recuperação do *checkpoint* no segundo dispositivo ou no primeiro após carregamento.

Os resultados obtidos são apresentados na Tabela III.

Tabela III
Tempos de execução com o mecanismo de tolerância falha da bateria.

Máquina	t_T	t_{EC}	t_{RC}	t_T
MV1	7ms	6ms	9ms	22ms
MV2	6ms	4ms	7ms	17ms
MF1	5ms	4ms	5ms	14ms
MF2	3ms	4ms	4ms	11ms

D. Falha de Ligação ou Ligação Fraca à Rede

Como a solução para a falha de ligação é feita em duas etapas, decidimos avaliar o desempenho, a partir do momento em que se deteta a falha de ligação, quantificando o tempo de gravação na fila, e, a seguir, o tempo de sincronização dos dados pelo servidor, assim que a ligação é retomada. Assim, como se pode observar na fórmula a seguir, o tempo total foi o resultado da soma dos dois tempos:

$$t_T = t_G + t_S$$

onde:

- t_T, é o tempo total desde o inicio ao fim do processo;
- t_G, é o tempo de gravação na fila;
- t_S, é o tempo de sincronização entre o servidor e o dispositivo móvel, assim que a ligação é retomada;

Os resultados obtidos são apresentados na Tabela IV.

Tabela IV
Tempos de execução com o mecanismo de tolerância a falha de ligação.

Máquina	t_G	t_S	t_T
MV1	3ms	5ms	8ms
MV2	3ms	6ms	9ms
MF1	2ms	3ms	5ms
MF2	1ms	3ms	4ms

V. Discussão e Conclusões

A. Impacto da Tolerância a Falhas

Os métodos apresentados neste documento procuram, de uma maneira geral, tolerar eventuais falhas durante a utilização de sistemas com MCC.

Como se pode observar na Tabela II, o desempenho da normal da aplicação possui tempos bastante aceitáveis sendo condicionados, apenas, com a velocidade da rede em que se esteja ligado. Olhando para os tempos, estes revelam que com a utilização de diversas versões do *Android* os resultados são muito parecidos.

Os tempos totais resultados da aplicação do método nível de bateria, na Tabela III, são, consideravelmente, maiores que os da execução normal. Obtiveram-se estes resultados porque a execução obedeceu três etapas: a etapa de criação do *checkpoint* (t_{CC}), etapa de envio do *checkpoint* (t_{EC}) e a etapa de recuperação do *checkpoint* (t_{RC}). Cada uma destas etapas, apresenta tempos muito parecidos com os da execução normal, podendo-se perceber, entretanto, que os tempos totais de execução deste método são, aproximadamente, três vezes maiores que os da execução normal.

Na Tabela IV são apresentados os tempos parciais e totais da execução do algoritmo de recuperação de falha de ligação implementado. Observando a tabela, um aspeto que se tem, logo, em conta é que os tempos de gravação na fila (t_G) são, bastante, baixos inclusive quando comparados aos tempos da execução normal (Tabela I). Estes tempos justificam-se, porque a fila em que os dados são gravados quando há alguma falha de ligação, encontra-se na cache do próprio dispositivo, logo, ao contrário da execução normal, os dados não precisaram de percorrer pela rede para serem persistidos. Outro elemento a ter em conta ainda na Tabela IV é que o tempo de sincronização (t_S) dos dados no servidor são bastantes parecidos com os da execução normal, o que mostra que utilizando este método, o tempo total de execução é, aproximadamente, a metade do tempo de execução normal, mais o tempo de sincronização dos dados no servidor.

B. Conclusões

Este artigo faz um estudo à computação móvel na nuvem muito focado nos possíveis casos de falhas que se podem encontrar durante a sua execução. Estas vão desde as limitações de hardware dos dispositivos móveis até a situações que não dependam diretamente do dispositivo que é a sua comunicação com o servidor.

Apesar de existirem várias, durante a execução destes tipo de sistemas, procuramos solucionar, neste documento, a falha de ligação à rede e, também, a falha que ocorre quando há um descarregamento da bateria, propondo dois modelos de tolerância a falhas, em que os testes tiveram o principal foco no desempenho das soluções quando comparados com a execução normal da aplicação.

C. Trabalhos Futuros

Pode se dar o caso, que durante a utilização da aplicação, mais de um dispositivo cliente encontrarem-se em situações de falhas. Pretendemos então trazer, para trabalho futuro, uma análise desempenho das duas soluções quando estiverem vários dispositivos em situações de falhas, simultaneamente.

De forma a tornar as nossas soluções o mais abrangente possível queremos, também para trabalhos futuros, trabalhar com ficheiros mais pesados e quantificar a eficiência dos algoritmos.

Agradecimentos

Projeto parcialmente financiado por:

- Operação Centro-01-0145-FEDER-000019 – C4 – Centro de Competências em Cloud Computing, co-financiada

pelo Programa Operacional Regional do Centro (CENTRO 2020), através do Sistema de Apoio à Investigação Científica e Tecnológica – Programas Integrados de IC&DT;
- FCT através de fundos nacionais e quando aplicável cofinanciado pelo FEDER, no âmbito do Acordo de Parceria PT2020 no âmbito do projeto UID/EEA/50008/2019.

REFERÊNCIAS

[1] W. Y. Huang and S. C. Lo, "Channel assignment study for multi-Channel multi-radio wireless mesh networks," *Journal of Internet Technology*, vol. 10, no. 4, pp. 345–352, 2009.

[2] M. Satyanarayanan, "Mobile computing: the next decade," *School of Computer Science*, vol. 11, no. 6, pp. 1081–1087, 2013.

[3] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation Offloading for Service Workflow in Mobile Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 12, pp. 3317–3329, 2015.

[4] M. Abdullah-Al-Shafi, A. N. Bahar, and S. Saha, "Mobile on-demand computing: The future generation of cloud," *International Journal of Future Generation Communication and Networking*, vol. 9, pp. 161–178, 12 2016.

[5] K. Kumar, J. Liu, Y. H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[6] A. R. Suraj, S. J. Shekar, and G. S. Mamatha, "A Robust Security Model for Cloud Computing Applications," *7th IEEE International Conference on Computation of Power, Energy, Information and Communication, ICCPEIC 2018*, pp. 18–22, 2018.

[7] Z. Zhang and S. Li, "A survey of computational offloading in mobile cloud computing," *Proceedings - 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, MobileCloud 2016*, pp. 81–82, 2016.

[8] "IEEE International Conference on Mobile Cloud Computing, Services, and Engineering."

[9] S. W. . R. W. Fernando, Niroshinie; Loke, "Future generation computer systems," *FGCS*, 30 May 2013. [Online]. Available: http://www.elsevier.com/locate/fgcs

[10] G. openmobster, "O Mundo da Computação Móvel na Nuvem," *https://code.google.com/archive/p/openmobster/*, vol. 91, pp. 399–404, 2017.

[11] K. Akherfi, M. Gerndt, and H. Harroud, "Mobile cloud computing for computation offloading: Issues and challenges," *Applied Computing and Informatics*, vol. 14, no. 1, pp. 1–16, 2018. [Online]. Available: https://doi.org/10.1016/j.aci.2016.11.002

[12] A. Morichetta, B. Re, and F. Tiezzi, "Runtime computation of optimal offloading scheduling," in *2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, March 2018, pp. 73–78.

[13] M. Satyanarayanan, "Fundamental challenges in mobile computing," in *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '96. New York, NY, USA: ACM, 1996, pp. 1–7. [Online]. Available: http://doi.acm.org/10.1145/248052.248053

[14] E. E. Marinelli, "Hyrax : Cloud Computing on Mobile Devices using MapReduce," *MS Thesis*, vol. 0389, no. September, pp. 1–123, 2009. [Online]. Available: http://www.contrib.andrew.cmu.edu/ emarinel/masters_thesis/emarinel_ms_thesis.pdf

[15] "Firebase realtime database," https://firebase.google.com/docs/database/, accessed: 2019-03-10.

[16] J. R. Lorch and A. Jay Smith, "Software strategies for portable computer energy management," *Personal Communications, IEEE*, vol. 5, pp. 60 – 73, 07 2016.