UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Tolerância a Falhas em Computação Móvel na Cloud

Euclides João Mujanga Catumbela

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientadora: Profª. Doutora Paula Prata

Covilhã, Junho de 2019

.

Dissertação elaborada no Instituto de Telecomunicações - Delegação da Covilhã e no Departamento de Informática da Universidade da Beira Interior por Euclides João Mujanga Catumbela, Licenciado em Engenharia Informática pelo Instituto Superior Politécnico Independente-Angola, sob orientação da Doutora Maria Paula Prata de Sousa, Investigadora do Instituto de Telecomunicações e Professora Auxiliar do Departamento de Informática da Universidade da Beira Interior, e submetida à Universidade da Beira Interior para discussão em provas públicas.

Trabalho integrado em projeto parcialmente financiado por:

- Programa Integrado de IC&DT "C4 – Centro de Competências em Cloud Computing" com contrato CENTRO-01-0145-FEDER-000019;
- FCT - Fundação para a Ciência e a Tecnologia através do projeto número UID/EEA/50008/2019.

Dedicatória

Ao Manuel Catumbela
e à Feliciana Catumbela,
meus pais.

Agradecimentos

Factores materiais, socias, financeiros, e, sobretudo, a intervenção, indirecta e directa, de várias pessoas, muito nos influenciaram para tornar possível a realização do trabalho. Quero expressar os meus agradecimentos:

- à Deus, pela dom da vida, e por conceder-me a oportunidade de efectuar este trabalho, com saúde;
- à professora Paula Prata, minha orientadora, por quem desenvolvi uma profunda admiração intelectual e pessoal fruto da coerência e precisão das suas orientações na estruturação e implementação deste trabalho. As suas orientções permitiram-me, também adquirir conhecimentos que, certamente, ficarão para sempre. Pelo seu acompanhamento, incentivo e pronta disponibilidade, expresso a minha especial gratidão;
- à minha querida família, pais, irmãos, tios, primos, amigos, agradeço as várias manifestações de motivação, incentivo e, sobretudo, de carinho;
- à Escola Superior Pedagógica do Bengo e ao Instituto Nacional de Gestão de Bolsas de Estudo de Angola – INAGBE, estou inteiramente grato pela atribuição da bolsa que permitiu a realização deste ciclo académico;
- aos meus professores e colegas de mestrado, da Universidade da Beira Interior - UBI, a minha eterna gratidão, pela recepção e sobretudo pela disponibilidade, quando o assunto foi partilha de conhecimentos.

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, estudamos as tecnologias de nuvem, desde suas principais aplicações, até aos desafios que actualmente enfrentam. Exploramos os mecanismos de persistência local de dados *offline* do Azure e do Firebase. A seguir, propusemos dois mecanismos de tolerância a falhas em computação móvel na nuvem: falha da carga da bateria e falhas na ligação à rede. E, por fim, avaliamos e comparamos os mecanismos explorados e propostos.

Palavras-chave

Computação Móvel, Computação em Nuvem, Tolerância a Falhas, Falha de Bateria, Falha de Conectividade, Azure, Firebase.

Abstract

Although mobile peripherals are increasingly capable of computing and storage, the link between mobile computing and cloud computing is also increasingly strong. Mobile applications that process or share large amounts of data use the cloud to overcome the resource constraints imposed by smartphones and tablets. These systems bring new challenges in terms of fault tolerance. On the one hand, they operate on batteries whose charge has a limited duration and, on the other hand, the mobility of the user can make it difficult to obtain continuous connectivity and with invariable bandwidth as would be desirable. In this work, we study cloud technologies, from their main applications, to the challenges they currently face. We studied the Azure and Firebase persistence mechanisms of offline data. Next, we propose fault-tolerance mechanisms for two types of common cloud computing failures: battery failure and network failure. And, finally, we evaluate and compare the mechanisms proposed by the native mechanisms (Azure and Firebase).

Keywords

Mobile Cloud Computing, Cloud Computing, Fault Tolerance, Battery Failure, Connectivity Failure, Azure, Firebase.

Conteúdo

Lista de Figuras

Lista de Tabelas

List of Algorithms

Lista de Siglas e Acrónimos

AI	Inteligência Artificial (Artificial Intelligence)
API	Interface de Programação de Aplicações (Application Programming Interface)
AWS	Amazon Web Service
BaaS	Backend as a Service
CC	Computação na Nuvem (Cloud Computing)
CRUD	Create Read Update Delete
EC2	Amazon Elastic Compute Cloud
EUA	Estados Unidos da América
IaaS	Infrastructure as a Service
INAGBE	Institituto Nacional de Bolsas de Estudo - Angola
JSON	JavaScript Object Notation
MC	Computação Móvel (Mobile Computing)
MCC	Computação Móvel na Nuvem (Mobile Cloud Computing)
MF	Máquina Física
ML	Machine Learning
MVC	Model View Controller
NIST	Instituto Nacional de Tecnologias e Padrões (National Institute of Standards and Technology's)
OMC	Oracle Mobile Cloud
PaaS	Plataform as a Service
SaaS	Software as a Service
UBI	Universidade da Beira Interior
VM	Máquina Virtual

Capítulo 1

Introdução

Atualmente as aplicações para dispositivos móveis parecem invadir todos os aspetos da nossa vida, desde áreas de puro entretenimento (como os mais variados jogos) até aplicações para marcar consultas ou exames médicos, passando por aplicações de astronomia que nos mostram mapas do céu em tempo real ou aplicações para monitorizar o nosso exercício físico. A computação móvel (Mobile Computing - MC) é, assim, cada vez mais, parte integrante do nosso quotidiano fazendo dos dispositivos móveis ferramentas de comunicação eficazes não delimitadas pelo tempo e lugar [Sat96]. Se antes um telemóvel era usado apenas como meio de comunicação, agora são utilizados para as mais diversas funções: internet, sensores, redes sociais baseadas na localização, etc. [BJGP16].

O número de utilizadores de dispositivos e de aplicações móveis vem aumentando a cada dia. As aplicações têm na nuvem uma escapatória para as suas limitações, portanto é comum, nos dias de hoje, utilizar um dispositivo móvel com necessidade de se manter constantemente ligado a rede. As estatísticas, apresentadas em [CLJ18], mostram que no ano de 2018, o número de utilizadores móveis no mundo é de 4,9 bilhões, o que significa que 66% da população mundial já tem um telemóvel. Na Europa, 78 em cada 100 habitantes têm um smartphone, o mundo é eminentemente móvel e a tendência continua. O tráfego móvel global deverá crescer cerca de 8 vezes entre 2015 e 2020. Em 2016, 61% dos usuários disseram que olharam para o seu dispositivo móvel nos primeiros 5 minutos depois de acorda.

Os dispositivos móveis, apesar de terem cada vez mais recursos, apresentam no entanto alguns constrangimentos [DHTZ15, AASBS16] . Por um lado, existem os limites de autonomia e mobilidade impostos pela carga da bateria e pela possibilidade de acesso à rede. Por outro lado, algumas aplicações mais complexas como, por exemplo, processamento de imagens ou algoritmos de inteligência artificial podem exigir maior capacidade de cálculo e/ou mais capacidade de armazenamento. A computação na nuvem (Cloud Computing – CC), outra tecnologia em grande evolução na última década aparece como uma solução natural para ultrapassar as limitações da MC. Surgiu assim o conceito de Computação Móvel na Nuvem (Mobile Cloud Computing - MCC) [KLLB13]. Plataformas de nuvem são cada vez mais usadas quer para ultrapassar as restrições de recursos da MC, como também para armazenar os sistemas de backend das aplicações móveis. Por muito simples que uma aplicação seja, necessita na maioria dos casos de um servidor para armazenar dados partilhados.

1.1 Motivação e Objetivos

Embora que, pelas previsões, os dispositivos móveis estarão a dominar o futuro dos dispositivos de computação, os dispositivos móveis e suas aplicações ainda são restringidos por algumas limitações, como a duração da bateria, o potencial de processamento e a capacidade de memória

dos dispositivos móveis [AGH18].

Este trabalho tem como objetivo estudar estratégias para tolerar falhas em aplicações móveis na nuvem. Em particular, explorar mecanismos que permitam lidar com falhas na ligação entre o periférico móvel e o servidor na nuvem quando o utilizador envia ou recebe dados e também quando há uma falha de bateria.

Como plataforma de estudo pretende-se desenvolver uma aplicação móvel cujos dados são armazenados e eventualmente processados numa nuvem caso a conectividade seja fraca ou o nível de carga da bateria seja baixo.

1.2 Abordagem

A fiabilidade é um aspecto extremamente importante para qualquer aplicação. Os utilizadores esperam que as suas aplicações funcionem corretamente e de forma contínua, mas quando falamos de aplicações móveis surgem desafios, novos em termos de tolerância a falhas [Ter16].

Para realizar este trabalho:

- desenvolvemos uma aplicação móvel e utilizamos o Firebase como servidor;
- desenhámos e implementamos um algoritmo para tolerar falhas de conectividade a rede;
- desenhámos e implementamos um algoritmo para tolerar descarga de bateria de dispositivo móvel;
- reaproveitámos a aplicação e exploramos o mecanismo de tratamento de dados *offline* do Firebase;
- reaproveitamos e adequamos a aplicação ao Xamarin e exploramos o mecanismo de sincronização de dados *offline* do Azure;
- comparamos os tempos de execução obtidos dos dois algoritmos, com os dos mecanismos de sincronização *offline* das duas plataformas (Azure e Firebase).

1.3 Contribuições

O trabalho desenvolvido para esta dissertação deu origem às seguintes contribuições para a área computação móvel na nuvem:

- O estado da arte da computação móvel na nuvem;
- um algoritmo de tolerância a falhas de bateria de um dispositivo móvel;
- um algoritmo de tolerância a falhas de conexão de rede de um dispositivo móvel;
- um artigo científico com os resultados parciais obtidos: *Euclides Catumbela e Paula Prata*. Estratégias de Tolerância a falhas em Computação Móvel na Nuvem. 3CTP - Conferência de Ciências da Computação Tendências e Paradigmas, 17 de Maio de 2019 .

1.4 Organização da Dissertação

Esta dissertação está organizada da seguinte forma:

- Capítulo 1 - Introdução - apresenta breve introdução sobre aspetos relacionados com a MC, as motivações do trabalho, a abordagem utilizada, as contribuições e a estrutura do documento.
- Capítulo 2 - Estado da Arte - apresenta o revisão bibliográfica da CC, MCC e aspetos ligados a tolerância a falhas em MCC.
- Capítulo 3 - Tolerância a Falhas de Bateria e Falhas de Ligação à Rede - apresenta propostas de estratégias de tolerância a falhas para aplicações móveis, e uma comparação com soluções existentes, nomeadamente os mecanismos de tratamento de dados *offline* do Firebase e Azure.
- Capítulo 4 - Implementação, Testes, Resultados e Discussão - descreve o ambiente em que avaliamos as estratégias e apresenta os resultados obtidos durante as experimentações das soluções apresentadas no capítulo 3.
- Capítulo 5 - Conclusões - apresenta as principais conclusões do trabalho desenvolvido, e uma perspetiva de trabalhos futuros

Capítulo 2

Estado da Arte

Neste capítulo são revistos os principais conceitos do trabalho: - computação na nuvem, suas características essenciais, seus principais modelos de serviço, seus modos de implementação, bem como seus benefícios; - computação móvel na nuvem, sua arquitetura, introduzimos o conceito de computação *offloading* (incluindo *frameworks* existentes), suas aplicações e os problemas e desafios que actualmente enfrenta; - Uma revisão de estudos realizados sobre tolerância a falhas em computação móvel na nuvem, seus mecanismos e alguns modelos de sincronização *offline* de Backend as a Service (BaaS) existentes; - por fim, uma conclusão do capítulo.

2.1 Computação na Nuvem

A CC tem sido amplamente reconhecida como a infraestrutura de computação que permite minimizar, ou até mesmo anular as limitações da MC, podendo fornecer vários serviços [Hoa09, DHTZ15].

A CC permite que os utilizadores utilizem infraestruturas, plataformas e pacotes de software oferecidos por provedores de serviços em nuvem a custos acessíveis [Hoa09]. Pode dizer-se que a CC surgiu como o paradigma que tentou realizar o sonho de fornecer recursos de computação (servidores, armazenamento e rede) como um serviço, em que os utilizadores só pagam os recursos que precisam e pelo tempo que os usam.

O Instituto Nacional de Tecnologias e Padrões dos EUA (*National Institute of Standards and Technology's* - NIST) apresenta a seguinte definição para CC [MG11]: *"A computação em nuvem é um modelo que permite acesso, através de uma rede de comunicação, a um conjunto recursos partilhados, de computação flexível (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e libertados com pouco esforço de gestão ou interação com o provedor de serviços."*

Esta definição prevê um modelo de nuvem composto de cinco características principais, três modelos de serviço e quatro modelos de implementação [MG11], como ilustrado na Figura 2.1.

2.1.1 Principais Características da Computação na Nuvem

As principais caraterísticas da CC são [PSMS15, MG11, Sas13]: auto-atendimento sob procura, acesso amplo à rede, agrupamento de recursos, rápida elasticidade, serviços sob medida.

2.1.1.1 Auto-atendimento sob Procura

A nuvem permite a utilização de recursos a pedido sem intervenção do fornecedor. Um consumidor pode provisionar unilateralmente recursos de computação, como tempo de servidor e

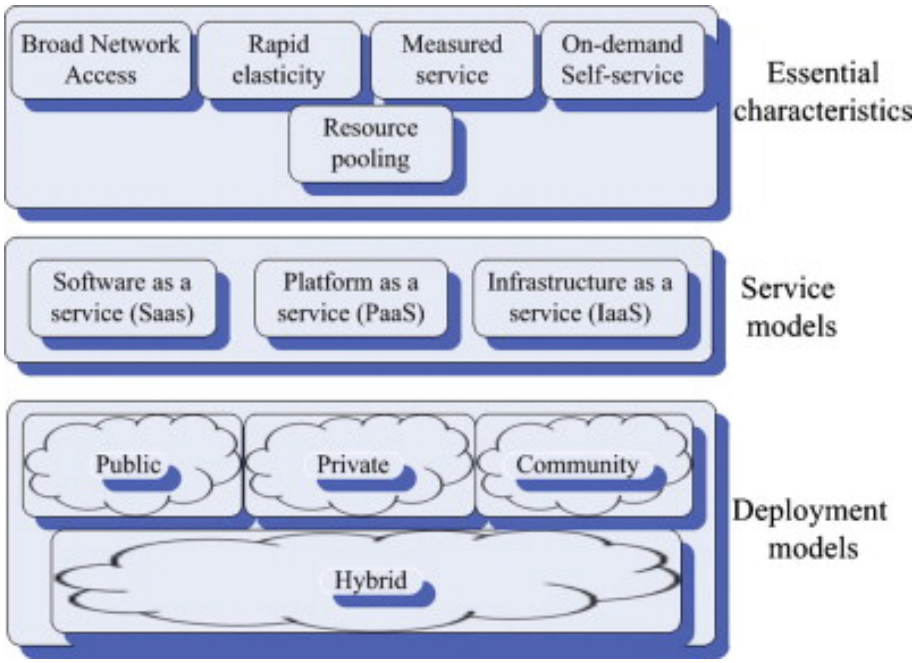

Figura 2.1: Modelos de Serviço da Computação na Nuvem **Fonte:** [AKV15]

armazenamento de rede, conforme necessário, automaticamente, sem exigir interação humana com cada provedor de serviços [MG11]. Recursos de computação incluem poder de processamento, armazenamento, máquinas virtuais, etc. [Sas13].

2.1.1.2 Acesso Amplo à Rede

As funcionalidades da nuvem são acedidas através dos protocolos *standard* da *internet*. Os recursos estão disponíveis na *internet* e são acedidos por meio de mecanismos padrão que promovem o uso de plataformas heterogêneas (por exemplo, *smartphone*, *tablets*, *PCs* e *workstations*)[Sas13].

2.1.1.3 Agrupamento de Recursos

Os recursos de computação do provedor são agrupados para atender a vários consumidores usando um modelo de multialojamento, com diferentes recursos físicos e virtuais atribuídos e reatribuídos dinamicamente de acordo com a procura dos consumidores [MG11]. Há um senso de independência de localização em que o cliente geralmente não tem o controlo ou conhecimento sobre a localização exata dos recursos fornecidos, mas pode especificar a localização em um nível mais alto de abstração (por exemplo, país, província ou *datacenter*). Isto é chamado de *multi-tenure* onde, por exemplo, um servidor físico pode ter algumas máquinas virtuais tendo um lugar com clientes distintos [PSMS15].

2.1.1.4 Rápida Elasticidade

Os serviços da nuvem são dimensionáveis de forma dinâmica de acordo com as necessidades do utilizador. Os recursos podem ser solicitados e libertados elasticamente, em alguns casos automaticamente, para escalar rapidamente para fora e para dentro de acordo com a procura [MG11]. Para o cliente, os recursos para sí disponíveis geralmente parecem ilimitados e podem ser utilizados em qualquer quantidade e a qualquer momento.

2.1.1.5 Serviço sob Medida

A utilização dos recursos da nuvem pode ser medida e monitorizada. Os sistemas em nuvem controlam e optimizam automaticamente a utilização de recursos aproveitando um mecanismo de medição em algum nível de abstração apropriado ao tipo de serviço (por exemplo, armazenamento, processamento, largura de banda e contas de utilizadores activadas). A utilização de recursos pode ser monitorizada, controlada e relatada, fornecendo transparência tanto para o provedor quanto para o utilizador do serviço [MG11]. A utilização de recursos é medida pela monitorização do uso de armazenamento, horas de CPU, uso banda da larga etc. Todos os provedores de serviços de nuvem aplicam estas métricas, mas cada um fornece aos utilizadores serviços em um nível diferente de abstração [PSMS15], que é uma alternativa para as empresas consumidoras.

2.1.2 Modelos de Serviço da Computação na Nuvem

Os três modelos de serviços principais propostos pela NIST, em [MG11], são: software como um serviço (*Software as a Service* – SaaS), plataforma como um serviço (*Platform as a Service* - PaaS) e infraestrutura como um serviço (*Infrastructure as a Service* – IaaS).

2.1.2.1 Software as a Service (SaaS)

Possibilita ao consumidor a capacidade de usar aplicações em execução em uma infraestrutura de nuvem, principalmente num navegador *Web*, para aceder software que se ofereça como serviço. O consumidor não tem controlo e, por consequência, não consegue efectuar alterações de *framework*, servidores, rede, sistemas operativos, armazenamento ou até mesmo capacidades de aplicações individuais, com a possível excepção de alterar definições de aplicações específicas de clientes particulares [PSMS15].

2.1.2.2 Platform as a Service (PaaS)

Possibilita ao consumidor a capacidade de construir aplicações numa infraestrutura de nuvem. As aplicações são produzidas utilizando um conjunto de linguagens de programação e ferramentas que são suportadas pelo provedor PaaS. O consumidor não controla a estrutura que está por trás da nuvem, isto é rede, servidores, sistemas operativos ou armazenamento, mas tem controlo das aplicações disponibilizadas e, possivelmente, sobre configurações do ambiente de desenvolvimento das aplicações. Da mesma forma que no modelo SaaS, os clientes não precisam de ter o controlo das infraestruturas para ter suas aplicações a funcionar numa PaaS [PSMS15].

2.1.2.3 Infrastructure as a Service (IaaS)

Possibilita ao consumidor a capacidade de adquirir processamento, armazenamento, rede e outros recursos fundamentais de computação de um provedor de IaaS. Permite que o consumidor implemente e execute qualquer software, que pode incluir sistemas operativos, serviços e aplicações. O cliente tem controlo dos sistemas operativos, do armazenamento, aplicações implementadas. Diferente do modelo PaaS, o modelo IaaS é de baixo nível e permite aos clientes a utilização mais pormenorizada de máquinas virtuais. O Iaas oferece aos clientes mais adaptabilidade do que o PaaS, pois permite que ao cliente usar qualquer pilha de produtos sobre o sistema operativo [PSMS15].

2.1.2.4 Modelos de Serviço: Visão Geral

Como nas secções anteriores, as principais diferenças entre os modelos de serviços oferecidos pela CC está, sobretudo, no nível de abstração do acesso a estrutura de nuvem: A IaaS constitui a camada de mais baixo nível da nuvem, oferecendo através de tecnologias de virtualização recursos de computação, rede e armazenamento. Um dos exemplos comerciais mais conhecido é a infraestrutura da Amazon, EC2. Uma PaaS oferece um conjunto de serviços para o desenvolvimento de aplicações, como por exemplo, linguagens de programação, ambientes integrados de desenvolvimento, servidores de base de dados, etc. São exemplos de PaaS, a Google App Engine e a Microsoft Windows Azure. Finalmente o modelo de SaaS constitui a camada mais elevada da nuvem e fornece software sem a necessidade de instalar e executar as aplicações em máquinas do utilizador. São exemplos de SaaS, a Dropbox[1], o Salesforce[2], o Google Docs[3] e muitos outros serviços largamente utilizados.

A tabela 2.1, adaptada de [Pra12], apresenta uma comparação entre as áreas de actuação dos modelos de serviços disponíveis na CC.

Tabela 2.1: Comparação entre os modelos de serviço da CC. Adaptado de : [Pra12]

Modelos	Utilizadores	Áreas adequadas
SaaS	Utilizadores finais	- Aplicações onde há interação significativa entre a organização e o mundo exterior. - Aplicações que possuem uma necessidade significativa de acesso Web ou móvel. - Software que deve ser utilizado apenas para uma necessidade de curto prazo. - Software com alta procura.
PaaS	Programadores de Aplicações	- Situação em que vários programadores trabalharão no desenvolvimento de um projecto ou em que outras partes externas precisarão interagir com o processo de desenvolvimento. - Os programadores desejam automatizar os serviços de teste e implementação.
IaaS	Arquitetos de rede	- Pode ser aplicado onde a procura é muito volátil. - Para novas organizações sem capital para investir em hardware. - Para empresas com crescimento rápido de recursos humanos, hardware seria um problema. - Há pressão sobre a organização para limitar as despesas de capital e passar para as despesas operacionais. - Para uma linha específica de negócios, teste ou necessidades. de infra-estrutura temporária.

2.1.3 Modos de Implementação da Computação na Nuvem

Modos de Implementação da CC referem-se à forma como esta será apresentada, do ponto de vista de acesso, aos utilizadores. A CC pode ser implementada como uma nuvem privada, comunitária, pública e híbrida [MG11].

[1]Site oficial: https://www.dropbox.com/
[2]Site oficial: https://www.salesforce.com/
[3]Site oficial: https://docs.google.com/

2.1.3.1 Nuvem Privada

A infraestrutura de nuvem é provisionada para uso exclusivo por uma única organização que inclui vários consumidores (por exemplo, unidades de negócios). Este modo pode ser de propriedade, gerido e operado pela organização, por terceiros ou por alguma combinação deles, e pode estar disponível dentro ou fora das instalações do provedor de serviços da nuvem.

2.1.3.2 Nuvem Comunitária

A infraestrutura em nuvem é provisionada para uso exclusivo por uma comunidade específica de consumidores de organizações que partilham preocupações (por exemplo, missão, requisitos de segurança, políticas e considerações de conformidade). Este modo pode ser de propriedade, gerido e operado por uma ou mais organizações da comunidade, um terceiro ou uma combinação deles, e pode existir dentro ou fora das instalações.

2.1.3.3 Nuvem Pública

A infraestrutura de nuvem é provisionada para uso aberto pelo público em geral. Ela pode ser de propriedade, gerida e operada por uma organização comercial, académica ou governamental ou por alguma combinação deles. Este modo existe nas instalações do provedor de nuvem.

2.1.3.4 Nuvem Híbrida

A infraestrutura de nuvem é uma composição de duas ou mais infraestruturas de nuvem distintas (privada, comunitária ou pública) que permanecem como entidades exclusivas, mas unidas por uma tecnologia padronizada ou proprietária que permite a portabilidade de dados e aplicações.

2.1.4 Computação na Nuvem: Visão Geral

Com a CC deixa-se de se estar preocupado com recursos físicos, pelo contrário, pode-se utilizar a *internet* em qualquer lugar e momento [SS11]. Muitas das características da CC acabam por ser, ao mesmo tempo, vantagens de sua utilização.Dentre as vantagens da computação em nuvem está a possibilidade de acesso aos dados e aplicações de qualquer lugar, desde que haja conexão de qualidade com a *internet*, trazendo assim, a possibilidade de mobilidade e flexibilidade dos serviços disponibilizados aos utilizadores [PN11].

O acesso faz-se através dos protocolos da *internet*, e portanto independentemente da localização e do tipo plataforma ou dispositivo (*smartphone, tablet, laptop*, etc) usado pelo cliente levou à rápida generalização da CC. Quando uma aplicação de um dispositivo móvel usa serviços da nuvem estamos perante um modelo de MCC [Fer13, ZL16].

2.2 Computação Móvel na Nuvem

A MC consiste em sistemas computacionais distribuídos em diferentes dispositivos que comunicam-se entre si através de uma rede de comunicação sem fio, o que permite a mobilidade desses aparelhos [Gom15].

O rápido crescimento da MC torna-se uma poderosa tendência no desenvolvimento de tecnologia de informação, bem como nos campos de comércio e indústria [Sat13]. 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 tem aumentado todos os dias [DHTZ15, AASBS16, KLLB13]. O MCC é o mais recente modelo de computação distribuída que estende a vantagem de se utilizar CC para *smartphone* [ZL16].

A computação móvel está a tornar-se, cada vez mais, necessária devido ao aumento da variedade de computadores móveis e também ao desejo de possuir uma conexão contínua na rede, independentemente da localização física do nó [Pra16]. A CC, através de seus serviços, tem sido uma escapatória para a MC ultrapassar as suas limitações de hardware, dando origem, desta forma, à MCC.

K. Akherfi et al., em [AGH18], apresenta que MCC, de forma simples, refere-se a uma infraestrutura onde tanto o armazenamento e como 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.

Sharma et al. descreve, em [SS13], a MCC como um novo paradigma para aplicações móveis em que o processamento e o armazenamento de dados são movidos do dispositivo móvel para plataformas de computação poderosas e centralizadas localizadas em nuvens. Estas aplicações centralizadas são acedidas pela conexão sem fio com base em um cliente nativo ou navegador da Web nos dispositivos móveis.

Alternativamente Z. Zhang et al., em [ZL16], escreve que o desenvolvimento da MCC fornece soluções em nível de software para mitigar as restrições de recursos de *hardware*. Continua, que a MCC é o mais recente modelo de computação distribuída que estende a visão computacional de nuvens computacionais para os dispositivos móveis e utiliza os serviços de processamento de aplicações de nuvens computacionais para o processamento de aplicações intensivas.

A MCC apresenta-se assim como um paradigma de computação que combina a MC e a CC em que o processamento e o armazenamento, devido as limitações dos dispositivos móveis, passam a ser efetuados na nuvem.

2.2.1 Arquitetura da Computação Móvel na Nuvem

A Figura 2.2 mostra que, 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.
Nem todas as aplicações precisam, necessariamente, de uma nuvem para o seu normal funcionamento. As aplicações para nuvem, como explicado em [Fer13], encaixam-se em áreas em que se precisa grandes quantidades de armazenamento e processamento de dados, como processa-

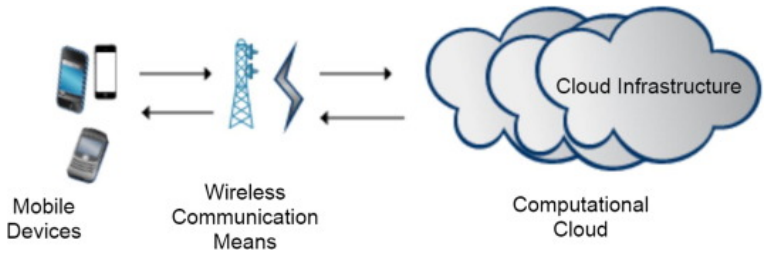

Figura 2.2: Visão geral da MCC **Fonte:** [AGH18]

mento 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.

2.2.2 Computação *Offloading*

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 apresentado em [MRT18], *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 [Hoa09]. Estes prejuízos, podem acontecer por não ser necessário uma migração, dada a pouca quantidade de dados para processamento ou armazenamento, ocupando de maneira desnecessária a nuvem, e por outro lado, por não se preverem modelos de tolerância a falhas, durante o planeamento do *offloading*.

Computação *offloading* é a tarefa de enviar a intensa carga de computação de componentes de aplicações para um servidor remoto [AGH18]. Em [DS16b] são apresentados vários modelos e *frameworks* para *offloading* em MCC, e em [AGH18] são discutidos quais os desafios deste modelo de computação. Alguns mecanismos para particionar aplicações de computação intensiva entre o dispositivo móvel e a nuvem são estudados nos trabalhos [SAGH14, MRT18].

Processamento intensivos em recursos executados localmente, num dispositivo móvel, levam mais tempo para serem executados, pois o poder de processamento e a memória principal do dispositivo móvel são limitados e não escalonáveis. Se o mesmo processamento for executado na nuvem, podem levar menos tempo devido à alta capacidade de processamento da infraestrutura de nuvem. [DS16b]

2.2.2.1 Fases do *offloading*

Em [AGH18], K. Aherfi et al. apresenta três passos básicos para se efectuar o *offloading*:

1. **Particionamento** da aplicação: é o primeiro e muito importante passo para o computação *offloading*. Ele divide a aplicação em partes *offloadable* e *non-offloadable* (transferíveis e não transferíveis, respetivamente), significando quais componentes devem permanecer no dispositivo móvel e quais migrar para o servidor de nuvem. A decisão de um componente ser transferível pode ser tomada com base em informações diferentes. O programador

pode anotar partes da aplicação, por exemplo, por meio de uma API especial, como *offloadable*. As peças intensivas de computação candidatas a *offloading* também podem ser identificadas pela análise do código-fonte em combinação com a previsão de desempenho ou por meio do perfil da aplicação. Se o particionamento for feito em tempo de desenho da aplicação, as duas técnicas terão uma precisão limitada, uma vez que não levam em consideração o contexto de execução real da aplicação.

2. **Preparação** da aplicação: neste passo executam-se todas as ações necessárias para componentes transferíveis para permitir seu uso em aplicações móveis. Isto inclui a seleção de um servidor remoto, a transferência e a instalação do código, como o início de processos *proxy* que recebem e executam tarefas em nome do *smartphone*. Além da transferência do código, os dados também podem ser transferidos para preparar a execução remota.

3. **Decisão do *offloading***: é a etapa final, antes que a execução remota seja iniciada, para componentes transferíveis. Se um componente remoto instalado é usado ou não na aplicação, normalmente depende do contexto de execução. Se a decisão for tomada em tempo de execução, informações mais precisas estarão disponíveis, por exemplo, o *smartphone* pode até não ter uma conexão sem fio ou o consumo de energia para transferir os dados para a execução remota pode simplesmente ser muito alto. Sempre que a situação muda, a transferência pode ser adaptada. Essa decisão de tempo de execução, induz alguma sobrecarga que normalmente não está presente caso a decisão seja tomada em tempo de desenho da aplicação. Preferências do utilizador, especificações do servidor, especificações da aplicação, especificações do dispositivo móvel e especificações da rede, são os aspectos que influenciam directamente na decisão do *offloading*, como ilustrado na figura 2.3.

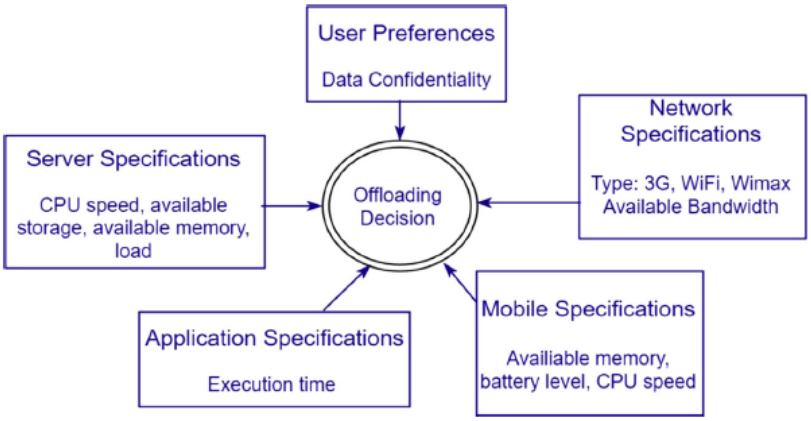

Figura 2.3: Aspectos que afectam a decisão de se efectuar um *offloading* **Fonte:** [AGH18]

2.2.2.2 *frameworks* de *Offoading*

O tempo de transmissão para transferir o processamento e recuperar os resultados é um fator importante e determinante para um *offloading* ser benéfico[DS16b]. Conforme a decisão de se efectuar a transferência da carga computacional para o servidor, podem-se distinguir dois tipos de *frameworks* para *offloading*: estático e dinámico [ZL16]. No primeiro, todas as etapas apresentadas são executadas em tempo de desenho, antes que a aplicação seja executada no

dispositivo móvel. Enquanto que nos *frameworks* dinámicos, pelo menos, a decisão final de transferir uma computação é tomada em tempo de execução.

K. Akherfi et al. apresenta, em [AGH18], dois grandes mecanismos para se efectuar um *offloading*:

1. *frameworks* baseados em clonagem de máquinas virtuais, a imagem completa do dispositivo móvel é capturada e armazenada no servidor de nuvem. Durante o *offloading*, a execução do dispositivo móvel é suspensa e transferida para o clone da VM na nuvem.

2. *frameworks* baseados em *offloading* do código, descarregam o código, invocando uma chamada de procedimento remoto usando anotações, compilação especial ou modificação binária.

Apresentamos, a seguir, alguns *frameworks* existentes na literatura, e suas principais abordagens.

1. **Phone2Cloud**, apresentado em [XDL+14], tem como objetivo melhorar a eficiência energética dos *smartphone* e, ao mesmo tempo, melhorar o desempenho da aplicação, reduzindo seu tempo de execução. Os autores implementaram o protótipo do Phone2Cloud no ambiente Android e Hadoop[4].

   Phone2Cloud é um *framework* de *offloading* semi-automático. Para executar aplicações na nuvem e receber os resultados, estas precisam ser modificadas manualmente durante a etapa de preparação para possibilitar a execução em servidores na nuvem [AGH18]. A decisão de *offloading* é baseada numa análise estática, considerando o limite de tolerância a *delay* do utilizador. Para aplicações tolerantes ao *delay*, o *framework* usa um modelo para esperar conectividade *WiFi*.

   Os autores se concentram em conduzir uma análise totalmente quantitativa e os resultados experimentais mostraram que o Phone2Cloud pode efetivamente economizar energia para *smartphone* e reduzir o tempo de execução das aplicações [XDL+14].

2. **Cuckoo**, apresentado em [KPKB12], é um *framework* para *offloading* de *smartphone*. Este *framework* move as aplicações do dispositivo móvel para o servidor nuvem utilizando o *Java Stub Model*.

   Desenvolvido no Eclipse, o Cuckoo oferece um modelo de programação preparado para ambientes móveis, como aqueles em que a conectividade com servidores é instável. Suporta execução local e remota e agrupa ambos os códigos em um único pacote. O Cuckoo integra-se com ferramentas de desenvolvimento existentes que são familiares aos programadores (e.g. Java) e automatiza grandes partes do processo de desenvolvimento [KPKB12].

3. **Jade**, H. Quian e D. Andresen apresentam, em [QA17], o "Jade: Reducing Energy Consumption of Android App", em português Jade: Reduzindo o Consumo de Energia da Aplicação Android, - é um *framework* que teve como objetivo maximizar os benefícios do *offloading*

[4] O Apache Hadoop é uma plataforma grátis e de código aberto. É usado para no mercado da análise de dados, como também em diferentes setores de armazenamento e processamento de dados. A primeira versão do Hadoop, ou seja, o Hadoop 1.0, foi lançada pelo Apache, em 2011 [sBD18]

com reconhecimento de energia para aplicações móveis, minimizando o peso dos programadores na criação de aplicações deste tipo [AGH18]. O Jade ajusta dinamicamente a estratégia de *offloading*, adaptando-se à variação da carga de trabalho, aos custos de comunicação e ao status de energia em uma rede distribuída de dispositivos Android e não Android. O Jade fornece a API Jade que tem como objectivo minimizar a carga sobre os programadores para criar aplicações com capacidade de *offloading*. A avaliação mostra que a Jade pode efetivamente reduzir até 39% do consumo médio de energia para aplicações móveis, melhorando o desempenho [QA17].

4. **CloneCloud**, Chun et al. apresenta, em [CIM+11], o *framework* CloneCloud que visa melhorar a vida útil da bateria e o desempenho no dispositivo móvel, transferindo componentes intensivos para servidores em nuvem.

   Os autores do CloneCloud focam-se na aplicação de VMs de camada de aplicação, como a Java VM, DalvikVM da plataforma Android e .NET da Microsoft. Optaram por utilizar as VMs da camada de aplicações, já que elas são amplamente utilizadas em plataformas móveis. Além disso, com o modelo de VM da camada de aplicação se consegue manipular executáveis de aplicações e migrar seus pedaços para dispositivos de computação de arquiteturas divergentes, até mesmo arquiteturas de conjuntos de instruções diferentes (por exemplo, *smartphone* baseados em ARM e servidores baseados em x86) [CIM+11].

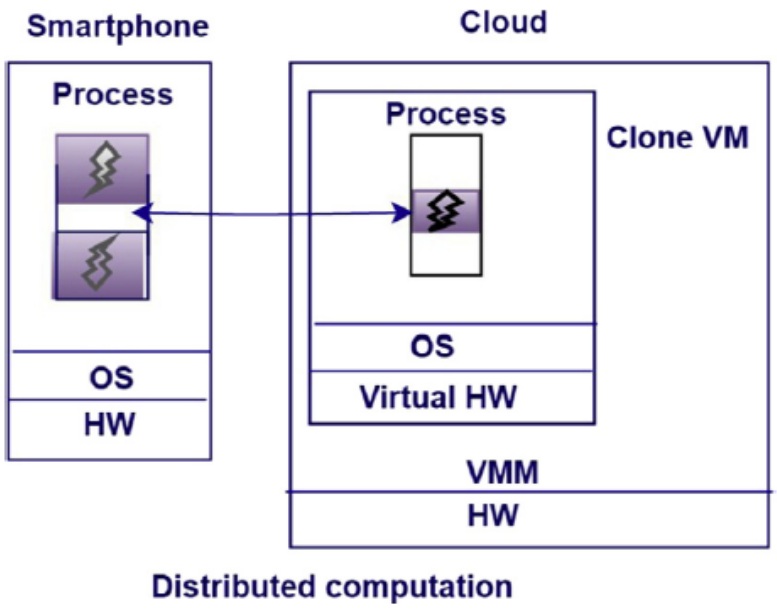

Figura 2.4: Modelo de funcionamento do CloneCloud **Fonte:** [CIM+11]

A Figura 2.4 ilustra o modelo de execução do CloneCloud. Inicialmente, é criado um duplicado do software do *smartphone* na nuvem. O estado do *smartphone* e do clone é sincronizado periodicamente ou a pedido. Após a execução dos componentes transferidos, os resultados da execução no clone são reintegrados novamente no estado do *smartphone*. O CloneCloud utiliza o *offloading* dinâmico e é baseado na migração da instância de VM para o servidor de nuvem [AGH18].

5. **EECOF**, M. Shiraz et al. apresentam, em [SGS+14], o Energy Efficient Computational *offloading framework* (em português, *frameworks* de *offloading* com Eficiência Energética). Os autores propõem um *framework* com eficiência energética para o processamento de aplicações móveis intensivos em MCC. O EECOF se concentra em alavancar serviços de processamento de aplicações de datacenters em nuvem com instâncias mínimas de migração de código da aplicação em tempo de execução. As instâncias da migração do componente de tempo de execução são minimizadas implementando-se o *offloading* de tarefa computacional, como o principal procedimento de transferência, em vez da migração intensiva de componentes.

   O EECOF incorpora o modelo SaaS com o modelo IaaS para reduzir as instâncias da migração de componentes em tempo de execução. O *framework* EECOF utiliza uma decisão dinámica, com uma estratégia de particionamento.

6. **ACOF**, Jaya Ashok Suradkar et al. apresentam, em [AKS13], o Autonomous Computation *offloading framework* for Android Using Cloud (em português, *framework* de *offloading* Autônomo para Android Usando Nuvem), - Os autores propõem um *framework* para tornar uma aplicação autônoma reduzindo o tempo de execução e o consumo de energia.

   ACOF avalia uma aplicação para encontrar os problemas da aplicação, a fim de medir o desempenho do *offloading*. Quando a tarefa exige consumo e tempo máximo de energia, a tarefa será transferida para a nuvem. Além disso, os autores propõem o particionamento de aplicações para um *offloading* que fornece uma quantidade menor de taxa de decisão para transferência falsa do que os métodos anteriores [AKS13]. O ACOF utiliza uma decisão estática e o é baseado em clonagem de máquinas virtuais.

2.2.3 Aplicações da Computação Móvel na Nuvem

Diversas áreas têm sentido a intervenção das técnicas de *offloading* quer seja, para melhorar o desempenho dos serviços como também para a expansão do serviço, do ponto de vista geográfico. Diversos investigações vêm sido efectuadas e as áreas que mais se destacam em aplicações de técnicas de *offloading* são: comércio eletrónico, *m-learning*, inteligência artificial (AI, do inglês *artificial intelligence*), processamento de gráficos e imagens, saúde e jogos.

2.2.3.1 Comércio Eletrónico Móvel

Com o notável crescimento do desenvolvimento de aplicações móveis, e com o amplo crescimento do sector de vendas de produtos online, foi natural que estes serviços, fortemente complementares, se unissem. É muito comum, hoje em dia, encontrar aplicações móveis de vendas online em que o processamento ou armazenamento de produtos e pedidos encontram-se fora do dispositivo móvel. Portanto o sector da comércio eletrónico apresenta-se como um grande beneficiário deste paradigma de computação.

Um dos principais exemplos desta união é o modelo de negócio que a empresa Amazon decidiu adoptar. A Amazon estendeu os seus serviços para aplicações móveis, tanto que com as suas estruturas de aplicação é possível gerir compras, bem como guardar preferências dos milhares de clientes que possuem.

2.2.3.2 *M-learning*

O termo *m-learning* foi criado com base na aprendizagem electrónica (*e-learning*), acrescentando o conceito de mobilidade (m-learning). O M-Learning é definido como aprendizagem a qualquer hora e em qualquer lugar utilizando dispositivos portáteis, como *smartphones, PDAs, tablets e laptops* [LSX15]. A *m-learning* possibilita aos intervenientes do processo de ensino e aprendizagem uma partilha de informação a estudantes e professores mesmo estando distantes, e a partir de qualquer lugar.

No entanto, as aplicações tradicionais de *m-learning* têm limitações em termos de alto custo de dispositivos e rede, baixa taxa de transmissão de rede e recursos educacionais limitados [AGH18]

2.2.3.3 Inteligência Artificial

Recentes pesquisas em MCC e em AI criaram uma oportunidade perfeita para criar soluções escaláveis em problemas da AI. A seguir apresentamos dois exemplos da união de tecnologias móveis com a AI.

É proposto, em [NKLL10], um trabalho que visa reduzir o tempo de computação em robôs usando uma técnica de transferência de carga. O sistema é projetado para reconhecimento e rastreamento de objetos em movimento em tempo real utilizando a computação *offloading*.

Outro exemplo é apresentado em [Sin18], onde se construiu um sistema que faz diagnósticos de doenças de plantas para agricultores através de uma plataforma colaborativa escalável baseada na nuvem. A plataforma é acessível por meio de uma aplicação móvel, que permite que os utilizadores façam upload de imagens, de várias partes de sua fábrica, e recebam a doença da planta diagnosticada automaticamente, em tempo real. Eles também podem ver o mapa de "densidade de doença" na sua região, mostrando a distribuição geográfica de doenças. A imagem enviada é classificada por um mecanismo AI, na categoria apropriada de doença para a qual uma solução de método conhecido mais conhecida é fornecida ao indivíduo. Simultaneamente, a localização geográfica da imagem e um carimbo de hora são usados para marcar a presença da doença em particular naquele local. Uma densidade coletiva de doenças armazenadas numa base de dados em nuvem é exibida em um mapa para mostrar sua localização em relação ao utilizador. Isso permite que o utilizador tome medidas preventivas baseadas em doenças em sua vizinhança e sirva de alerta para qualquer epidemia em expansão. Os componentes principais na arquitetura deste sistema são ilustrados na Figura 2.5 com os seguintes componentes: aplicação móvel, classificador de doença, treinador de redes neuronais, base de dados de treino e uma interface especializada.

2.2.3.4 Processamento de Gráficos

As aplicações de processamento de gráficos consomem muito tempo de processamento e grande tempo de execução em dispositivos móveis, especialmente quando são utilizadas imagens de tamanho muito elevado. O *offloading* do processamento de imagens do dispositivo móvel para a nuvem economiza recursos móveis consegue enviar respostas em tempo útil. O desenvolvimento da computação paralela na nuvem aumenta a velocidade e a precisão do algoritmo de processamento de gráficos [ATS15].

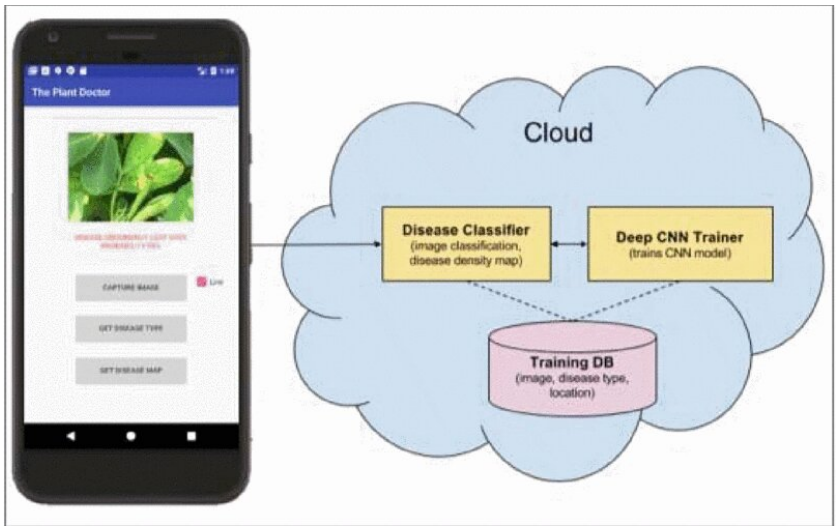

Figura 2.5: Arquitetura do sistema proposto em [Sin18]

.

O trabalho sugerido em [HMZ+14] examina os *trade-offs* que emergem da execução local da carga de trabalho e alguns em servidores de nuvem remotos. Extração e correspondência são dois recursos cruciais na classificação de imagens. O artigo analisa a possibilidade de executar os recursos mencionados anteriormente em um dispositivo móvel usando diferentes cenários.

2.2.3.5 Saúde

Aplicações móveis de saúde referem-se ao propósito de aplicar o MCC em aplicações médicas e minimizar as limitações do tratamento médico tradicional (por exemplo, pequeno armazenamento físico, segurança e privacidade e erros médicos) [Hoa09]. A saúde móvel (*m-healthcare*) fornece aos utilizadores móveis ajuda conveniente para aceder recursos (por exemplo, registos de saúde do paciente) com facilidade e eficiência. Além disso, a *m-healthcare* oferece aos hospitais e organizações de saúde uma variedade de serviços a pedido em nuvens, em vez de possuir aplicações independentes em servidores locais.

Um exemplo deste tipo de serviço é o PowerSense, apresentado em [MCF+11]. PowerSense é um módulo extensível para o Aplicações Móvel Detector de Dengue que facilita proativamente na gestão de processos para aplicações móveis assistidas por serviço da web.

2.2.3.6 Jogos

O jogo para telemóvel (*m-game*) é um mercado potencial que gera receita para os provedores de serviços. O *m-game* pode descarregar completamente o mecanismo do jogo, exigindo grande recurso computacional (por exemplo, renderização gráfica) para o servidor na nuvem, e os jogadores apenas interagem com a interface do ecrã em seus dispositivos [Hoa09].

Um exemplo deste tipo de aplicações pode ser encontrado em [WD10] em que o objetivo do trabalho apresentado foi satisfazer as restrições de comunicação e computação do Cloud Mobile Gaming, usando uma técnica de adaptação que garante uma boa experiência de jogo em dispositivos móveis.

2.2.4 Problemas e Desafios da Computação Móvel na Nuvem

Apesar da MCC ser uma solução inovadora e bastante vantajosa, unindo os benefícios de se utilizar a cloud computing à utilização da computação móvel, ela enfrenta alguns problemas e possui alguns desafios por concretizar. Apresentamos nesta secção alguns dos principais problemas e consequentes desafios da MCC actualmente.

Vários autores escrevem sobre problemas que a MCC enfrenta. As suas abordagens tem, na sua maioria, convergido visto que são sempre os mesmos a ser apresentados. H. T. Dinh et al. apresentam em [Hoa09] uma abordagem sistemática, dividindo os problemas em lado da computação móvel (baixa largura de banda, disponibilidade e heterogeneidade); e lado da computação (computação em *offloading*, segurança, eficiência no acesso aos dados e o reconhecimento do contexto da nuvem móvel).

Como alternativa, em [Fer13], Fernando et al. apresenta uma taxonomia dos principais problemas da MCC. Como pode se observar, na Figura 2.6, esta taxonomia é composta por 6 camadas: operacional, utilizador final, nível de serviço, segurança e privacidade, contexto e gestão de dados.

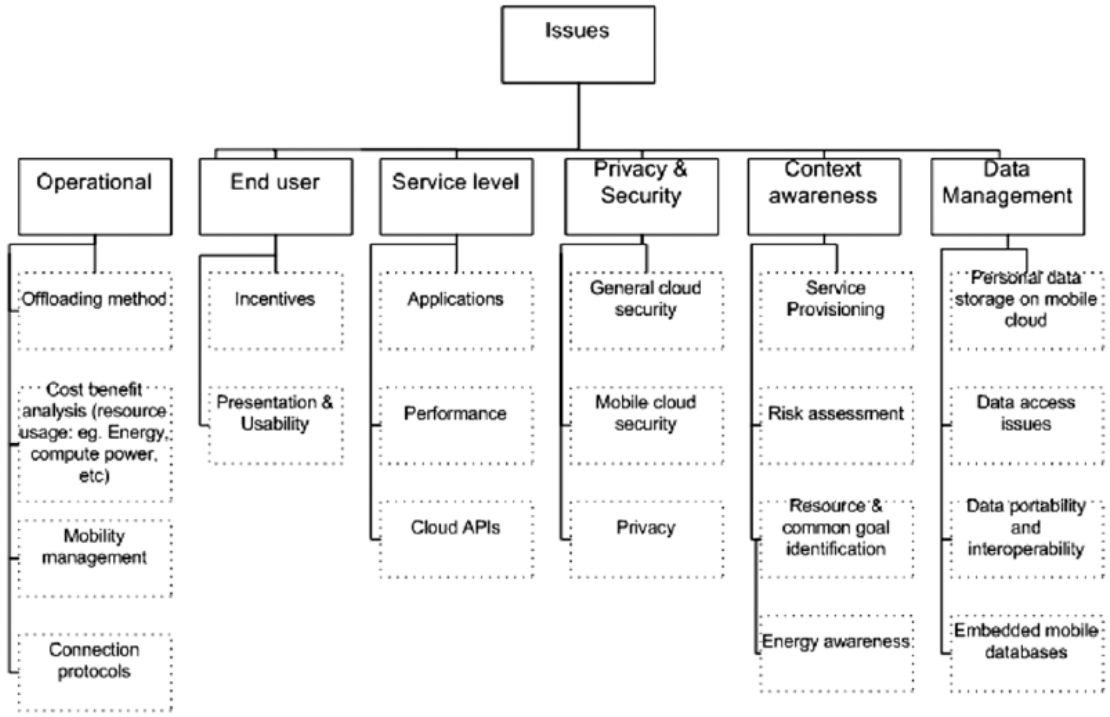

Figura 2.6: Uma taxonomia sobre problemas em computação móvel na nuvem **Fonte:** [Fer13]

1. A parte operacional tem a ver com a parte logística da implementação de computação móvel na nuvem. É nesta etapa que se definem os métodos de *offloading* a implementar; faz-se o estudo do custo e benefício da implementação do sistema; faz-se também a gestão da mobilidade dos dispositivos móveis; e, por fim, definem-se os protocolos de conexão entre os dispositivos móveis e nuvem.

2. A parte do utilizador final, um sistema em MCC deverá estar preocupado com questões ligadas a requisitos não funcionais do sistema, e não sendo menos importantes o sistema

deverá proporcionar ao utilizador incentivos para a sua utilização, uma apresentação simples e usabilidade necessária.

3. Os níveis de serviço estão relacionadas com a arquitetura do MCC. Garantem uma separação dos níveis de aplicações, os desafios prendem-se em melhorar o desempenho das APIs de nuvens utilizadas em MCC.

4. No geral, a segurança e a privacidade são requisitos muito importantes para se garantir confidencialidade de um sistema. Esta camada dentro da MCC preocupa-se, de um modo geral, com a segurança da plataforma nuvem e móvel, bem como com a privacidade dos dados dos utilizadores do sistema.

5. O Contexto, em que se aplica o sistema, é também um desafio. Deve-se ter consciência do contexto em que o sistema de MCC é implementado. Para isto deve-se efectuar uma previsão de serviços, uma avaliação dos riscos, identificação de objectivos comuns dos recursos, do ponto de vista institucional, e, por fim, ter consciência da energia.

6. Gestão de dados, um MCC deve ter a capacidade de poder armazenar dados pessoais, gerir os problemas no acesso aos dados, garantir a portabilidade e interoperabilidade dos dados e capacidade de armazenar dados de forma local, isto é no dispositivo móvel.

2.3 Tolerância a Falhas em Computação Móvel na Nuvem

Neste novo ecossistema de nuvem e periféricos móveis, aspectos como segurança, privacidade e fiabilidade ganham especial importância e enfrentam novos desafios. Se a segurança contra falhas intencionais é fundamental para garantir a adesão dos utilizadores a serviços geridos por fornecedores a quem confiam os seus dados [RSJSSM18], a confiabilidade, isto é, a garantia de fiabilidade e de disponibilidade, mesmo que ocorram falhas acidentais, dos sistemas móveis assume igual importância. Há mais de duas décadas que Satyanarayanan afirmava [Sat96]: “a mobilidade é inerentemente arriscada”.

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 [Fer13]. 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 [Fer13].

Estas falhas na utilização de MCC, são tão reais que merecem a atenção dos programadores ou de empresas que, eventualmente, queiram efectuar um *offloading* dos seus serviços móveis. Por esta razão, Marinelli descreveu, em [Mar09], que as principais características para um sistema distribuído móvel são: tolerância a falhas, escalabilidade, privacidade e interoperabilidade do hardware. O mesmo autor descreve, 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.

Para lidar com falhas no processo de *offloading*, existem vários algoritmos tolerantes a falhas que tiram partido dos recursos da nuvem. Em [DHTZ15] é proposto um modelo baseado em algoritmos genéticos para optimização de estratégias de *offloading*, onde falhas durante a computação são recuperadas através de mecanismos de *checkpointing*. Em [SBL17], propõem-se um mecanismo dinâmico de tolerância a falhas por replicação. Em [DS16a] é apresentada uma arquitectura de MCC tolerante a falhas, baseada no modelo de resistência à doença do corpo humano, introduzindo módulos de monitorização, resposta, conhecimento (guarda quais as máquinas virtuais disponíveis) e memória (guarda estratégias de reescalonamento). Recentemente Lee e Gil, em [LG19], propuseram algoritmos de escalonamento tolerantes a falhas baseados em checkpointing e replicação.

Aplicações móveis que requerem computação intensiva, como processamento de linguagem natural ou sistemas de apoio à decisão, são candidatos óbvios a *offloading* na nuvem, no entanto, qualquer aplicação móvel que precise de um servidor partilhado pelos vários utilizadores pode usar a nuvem para o alojar. Surgiu o conceito de "Mobile Backend as a Service" (MBaaS), [Wil12], para designar plataformas de nuvem que permitem alojar aplicações móveis e fornecem um conjunto de funcionalidades como autenticação de utilizadores, suporte para notificações, suporte para analisar o comportamento das aplicações, e sistemas de base de dados entre outros. Se uma plataforma deste tipo fornece suporte quer para aplicações móveis quer para aplicações web, diz-se um "Backend as a Service" (BaaS).

Vários factores podem causar falhas de comunicação entre o utilizador e o dispositivo móvel ou entre este e a nuvem: desde a mobilidade do dispositivo ao entrar ou sair de uma rede até ao esgotar da carga na bateria, ou outras falhas de hardware ou software [Fer13]. Empresas ligadas ao desenvolvimento de aplicações móveis em nuvem têm estado, cada vez mais, preocupadas com este aspecto, tal que, actualmente, muitos serviços BaaS possuem mecanismos para suporte *offline*.

2.3.1 Serviços BaaS e Mecanismos de Suporte *Offline*

Quando a quantidade de dados é considerável e o número de clientes aumenta, torna-se necessário pensar em soluções remotas que nos permitam centralizar e sincronizar os dados entre vários clientes. Como tal, a maioria das aplicações móveis precisa de um servidor *backend* para executar tarefas, tais como autenticar utilizadores, sincronizar seus dados em vários dispositivos e gerir estes dados remotamente. No entanto, a criação dos servidores requer um conjunto de requisitos e competências que grande parte dos programadores não possui. Felizmente existem várias plataformas online de serviços, BaaS, que disponibilizam um conjunto de serviços que vão libertar o programador para outras tarefas relacionadas com a lógica da aplicação [Que18].

Actualmente existem diversas empresas que fornecem serviços de BaaS dentre as quais podemos trazer as seguintes: Stamplay, RapidAPI, Back4app, AWS, Azure e Firebase.

2.3.1.1 Amazon Web Services

A Amazon Web Services, ou simplesmente AWS[5], é uma subunidade da Amazon que fornece plataforma de computação em nuvem a pedido para vários tipos de utilizadores, como empresas, instituições, utilizadores comerciais, etc. Os recursos e serviços mais comuns disponibilizados pela AWS são computação, rede, armazenamento, base de dados, serviços de aplicações, etc [SD18].

O Amazon Elastic Compute Cloud (EC2), que é um modelo IaaS, oferece serviços de computação completos. Esse serviço é fornecido por servidores virtuais controlados por uma API baseada em um *hypervisor* chamado Xen. Ele também fornece um recurso, o Amazon Elastic Beanstalk, que fornece uma PaaS de plataforma como serviço, para hospedagem de aplicações.

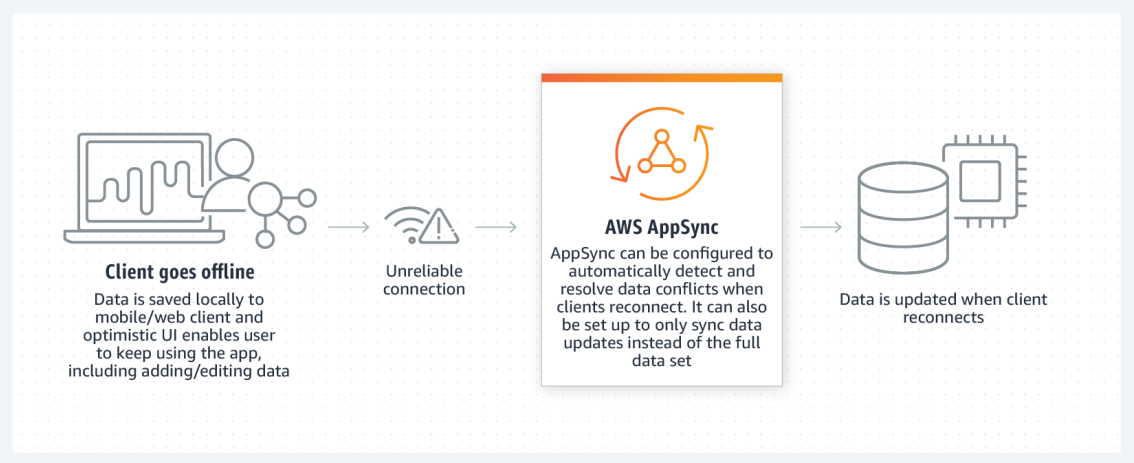

Figura 2.7: AWS AppSync *Offline*. **Fonte:** [Ama19a]

As linguagens suportadas pela AWS são: Interface da Linha de Comando (ILC) da AWS, JavaScript, Python, PHP, .NET, Ruby, Java, Go, Node.js, C++ [Ama19b].

O AWS AppSync oferece suporte a um modelo de programação *offline* em que, além de disponibilizar os dados da aplicação no modo *offline*, o modelo permite que utilizadores adicionem e atualizem esses dados. Quando o dispositivo estiver *offline*, a interface da aplicação será atualizada automaticamente com os dados *offline*. O *AppSync* permite definir como os dados são armazenados em cache *offline* e como o *AppSync* deve gerir as atualizações da cache em diferentes condições da rede. O modo de funcionamento do AWS AppSync *Offline* encontra-se ilustrado na Figura 2.7.

2.3.1.2 Oracle Cloud

A Oracle apresenta-se como uma solução de nuvem pública do ramo empresarial. Dentre vários serviços que disponibiliza, destaca-se a possibilidade de aceder a bases de dados diretamente da nuvem através de conexões de rede padrão, com um ambiente completo de desenvolvimento e implementação.

[5]Site oficial: https://aws.amazon.com/

Os programadores de aplicações móveis podem usar os recursos *Data Offline* e *Sync* para criar aplicações cliente que permitam a utilizadores executar tarefas críticas quando estiverem *offline*.

Podem-se utilizar APIs, ilustradas na Figura 2.8, para criar aplicações que armazenam em cache recursos REST para uso *offline* e, em seguida, sincronizam todas as alterações *offline* com o servidor quando o dispositivo fica *online* novamente.

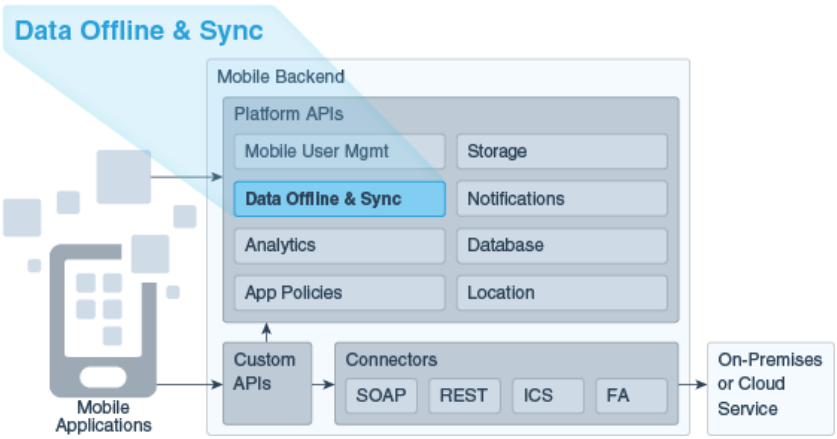

Figura 2.8: APIs do OMC e Mecanismo Offline. **Fonte:** [Ora19]

Na Tabela 2.2 apresentamos os principais mecanismos de execução *offline* do Oracle Mobile Cloud.

Tabela 2.2: Mecanismos de sincronização do Oracle Mobile Cloud. Fonte: [Ora19]

API	**Plataforma**	**Características**
Sync Express	- Cordova - Javascript	- Sincronização básica. - Funciona com qualquer REST API[6] em que o nome do recurso alterna entre substantivos no plural e identificadores de recursos singulares (rid), como / items / $\{rid\}$ / subitems / $\{rid\}$. - Requer alterações mínimas no código existente. - Funciona com qualquer estrutura JavaScript. - Quando o dispositivo se reconecta, envia solicitações de mudança um objeto de recurso por vez. - Sempre sobrescreve a versão do servidor do objeto.
Synchronization	- Android - iOS	- Sincronização robusta. - Funciona com APIs personalizadas compatíveis com sincronização. - Quando o dispositivo se reconecta, envia todas as alterações em uma solicitação. - Fornece opções para o que fazer se a versão do servidor de um objeto for alterada enquanto as edições forem feitas *offline* (do servidor, do cliente, preservação de conflitos). - Fornece opções por quanto tempo armazenar objetos de recursos no dispositivo, quando atualizar dados do servidor e quais recursos podem ser editados quando estiverem *offline*. - Sincroniza automaticamente com a plataforma de armazenamento.

2.3.1.3 RapidAPI

O RapidAPI [7] funciona como um mercado de APIs para programadores encontrarem, conectarem e gerirem suas conexões de API [Rapc]. No RapidAPI é possível encontrar as APIs de que se precisa para projetos em desenvolvimento, incorporar APIs de aplicações e controlar a utilização de todas as APIs, por meio de um único painel. Caso uma API já estiver criada, pode-se utilizar o RapidAPI para disponibilizá-la para outros de programadores que utilizam APIs por meio do RapidAPI.

Como pode se observar, na Figura 2.9, existe dentro da arquitectura do RapidAPI, um APIProvider que será o responsável pela publicação de API no Mercado de API e, por sua vez, o programador faz o uso do Mercado de API para encontrar ou conectar soluções para suas aplicações.

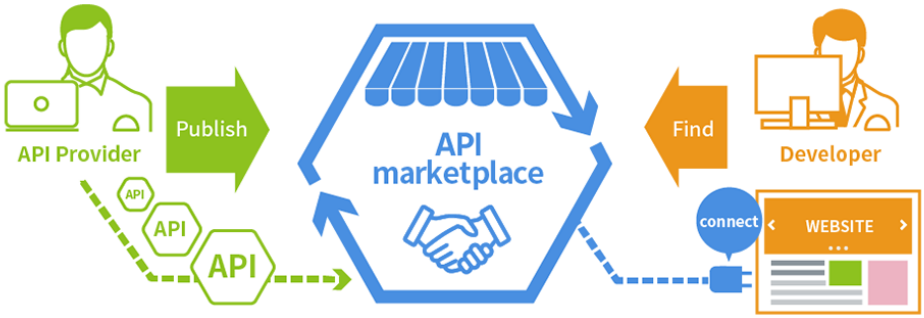

Figura 2.9: Arquitectura do RapidAPI **Fonte:** [Rapb]

O RapidAPI tem muitos SDKs diferentes que permitem utilizar todas as APIs, das aplicações dos programadores, no seu mercado. Independentemente da linguagem de programação, o RapidAPI possui um SDK para permitir usar facilmente nas aplicações. As linguagens de programação actualmente suportadas são: Node.js, PHP, Python, Ruby, Java, Objective-C e .NET [Rapa].

2.3.1.4 Firebase

O Firebase[8] é um BaaS de serviços móveis e web criado pela Firebase Inc., em 2011, sendo adquirido pela Google, em 2014 [LYL+18].

O Firebase disponibiliza vários produtos, como Cloud Firestore, Kit de Machine Learning (ML) (versão Beta), Cloud Functions, Autenticação, Hospedagem, Cloud Storage, Realtime Database, Crashlytics, Monitorização de desempenho, Test Lab, Google Analytics, Cloud Messaging, Remote config, Dynamic Links [Goo19].

O Firebase disponibiliza um serviço de persistência para as suas aplicações. Este recurso armazena em cache uma cópia dos dados que a aplicação utiliza activamente. Assim, a aplicação pode aceder os dados quando o dispositivo estiver desconectado. É possível gravar, ler, detectar e consultar os dados armazenados em cache. Quando o dispositivo for conectado novamente, o Cloud Firestore sincroniza todas as alterações locais feitas pela aplicação com dados armazenados remotamente. A Figura 2.10 ilustra o funcionamento deste processo, onde pode-se ver

[7] Site oficial: https://rapidapi.com/
[8] Site oficial: https://firebase.google.com

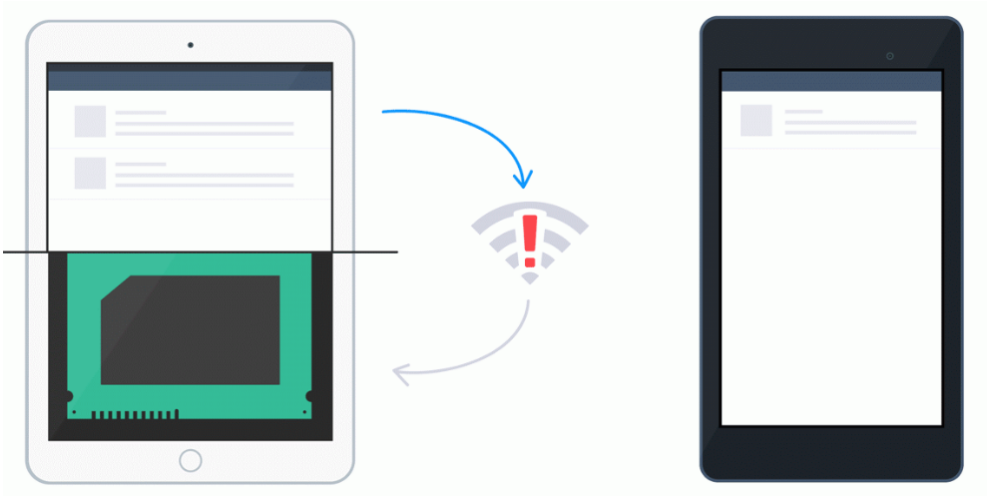

Figura 2.10: Armazenamento Offline Firebase. Adaptado de [Jon15]

o armazenamento local dos dados pela falta de conexão a rede.

2.3.1.5 Azure Cloud Platform

A Plataforma de Serviços Azure[9] da Microsoft é um grupo de tecnologias de nuvem, cada uma fornecendo um conjunto específico de serviços para programadores de aplicações. A Plataforma de Serviços do Azure pode ser utilizada tanto por aplicações em execução em nuvem quanto por aplicações executados em sistemas locais [SN16]. É do utilizador, a liberdade para criar, gerir e implementar aplicações numa rede global em grande escala através das suas ferramentas e arquiteturas preferidas.

Actualmente as principais áreas de actuação da Azure são: *internet* das Coisas, Inteligência artificial, Blockchain, Aplicações na cloud híbrida, DevOps, Móvel, Comércio eletrónico, Governação do Azure, Computação confidencial, Desenvolvimento e teste, Business inteligence, Macrodados e análise, Armazém de dados moderno, Aplicações SaaS empresariais, Cópia de segurança e arquivo, Recuperação após desastre, Marketing digital, Multimédia digital, Computação de alto desempenho, Aplicações de microsserviços, Jogos [DGGF16].

Na funcionalidade Aplicações Móveis do Serviço de Aplicações do Azure podem-se criar aplicações para várias plataformas e nativas para iOS, Android, Windows ou Mac, armazenar dados de aplicações na cloud ou localmente, autenticar clientes, enviar notificações *push* ou adicionar lógica de backend personalizada em C# ou Node.js.

No azure é possivel criar aplicações iOS, Android e Windows, com as seguintes possibilidades [Mic19]:

- Difusão via push com a segmentação de clientes.
- Início de sessão único com o Azure Active Directory.
- Dimensionamento automatico para suportar milhões de dispositivos.
- Integração nas redes sociais com o Facebook, o Twitter e o Google.

[9]Site oficial: https://portal.azure.com/

- As aplicações podem funcionar *offline* e em sincronização.

O Azure possui um mecanismo de suporte *offline*, que permite criar aplicações robustas que permanecem úteis mesmo quando surgem problemas de rede, para que os seus clientes possam criar e alterar dados quando estão *offline*. Isto melhora a capacidade de resposta das aplicações colocando os dados do servidor em cache localmente no dispositivo.

2.3.1.6 Back4app

O Back4App[10] é uma plataforma BaaS desenvolvida pelo Parse Open Source, onde se podem criar aplicações, hospedar e manter controlo total sobre seu *Backend*. Com a utilização desse serviço, os utilizadores não precisam criar e realizar a manutenção de servidores para hospedar a sua aplicação. A plataforma é responsável pela gestão e monitorização das aplicações criadas na mesma. Todos os sistemas são hospedados na Amazon. Além de garantir o funcionamento das aplicações, o Back4app também dispõe de outros mecanismos para auxiliar os programadores, como a configuração e envio de notificações para *smartphone* e a criação de APIs personalizadas [Lei17].

O Back4app utiliza o Node.js, como linguagem de programação, e o MongoDB, como base de dados NoSQL[11].

2.4 Conclusões

Considera-se que as aplicações em MCC estão, cada mais, presentes no quotidiano. Neste capítulo foram revistos os principais conceitos do trabalho, trazendo uma abordagem sistemática desde as limitações que MC actualmente enfrenta e como a CC tem procurado supri-las através das suas características essenciais. O que permitiu apurar que as aplicações móveis têm estado a evoluir, e a ligação com os serviços da nuvem vê-se necessária, trazendo vários benefícios e por consequência enormes desafios. Existem vários *framework*s, com diferentes abordagens, para facilitar o processo de migração das aplicações. E diversas grandes empresas têm estado a investir fortemente na criação de BaaS, cada vez mais, completos e abraangentes, para facilitar no desenvolvimento de aplicações em MCC.

[10] Site oficial: https://www.back4app.com/

[11] NoSQL é um termo genérico que representa base de dados não relacionais.

Capítulo 3

Tolerância a Falhas de Bateria e Falhas de Ligação à Rede

Neste capítulo é apresentada a aplicação exemplo usada para avaliar os mecanismos de tolerância a falhas estudados e propostos. A seguir apresentamos o modelo das principais falhas que ocorrem em aplicações móveis na nuvem. Continuamos, apresentando os mecanismos de tratamentos de dados *offline* do Firebase e Azure, seus modos de funcionamento e suas formas de acesso aos dados *offline*. Por fim, apresentamos duas propostas de tolerância a falhas em MCC, nomeadamente falha de bateria e de conectividade de rede.

3.1 Aplicação Móvel Exemplo

3.1.1 Descrição da Aplicação

Como caso de estudo, para ilustrar a implementação de estratégias de tolerância a falhas, projetamos 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. A pontuação, os dados de cada utilizador, as perguntas e as respetivas respostas são armazenadas em uma infraestrutura de nuvem.

3.1.2 Arquitetura Física da Aplicação

A execução deste tipo de aplicações, como MCC, ilustrada na Figura 3.1, 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.

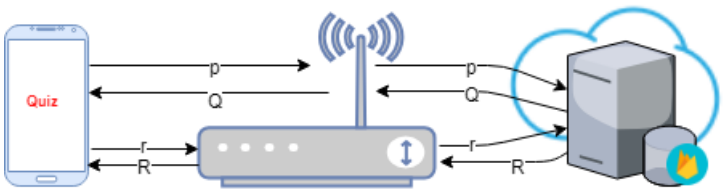

Figura 3.1: Arquitetura física da aplicação

3.1.3 Arquitetura Lógica da Aplicação

Na Figura 3.2 é apresentada a arquitetura lógica da aplicação desenvolvida. É baseada na arquitetura MVC[1](Model View Controller), sendo que os modelos foram persistidos no Firebase Realtime Database[2], as interfaces gráficas foram construídas com o XML, as interações e validações foram programadas utilizando a linguagem de programação Java.

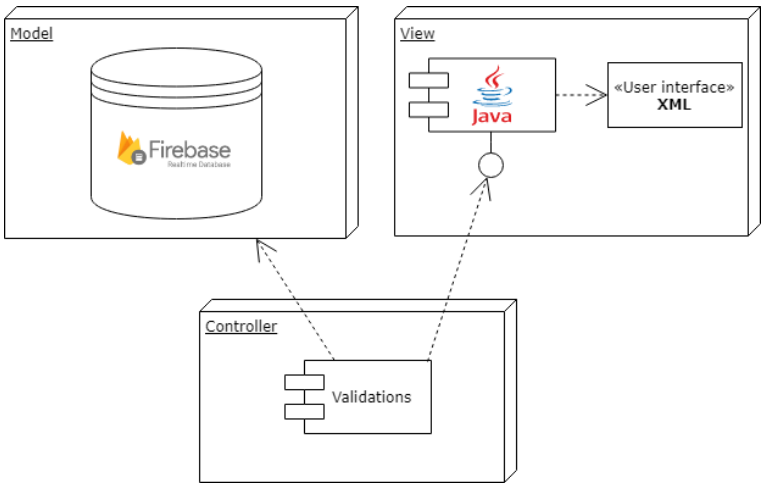

Figura 3.2: Arquitetura Lógica da Aplicação

3.1.3.1 Armazenamento e Estrutura dos Dados

Como ilustrado na arquitetura lógica da aplicação, os dados são armazenados no Firebase Realtime Database. Para tal, criamos uma base de dados NoSQL com um esquema contendo os seguintes nós principais: BackupBattery, Category, Questoes, Questoes_Score, Ranking e Users. A seguir, apresentamos a finalidade e estrutura de cada nó principal.

- BackupBattery: para registar o instante em que o utilizador vê a sua bateria descarregada, e guardar o *checkpoint*. Este nó é constituído pelos seguintes nós filhos: - *checkpoint* - guarda o local em que utilizador esteve na altura da descarga; - *nomeUtilizador*, guarda o nome do utilizador em sessão. A sua estrutura é ilustrada na Figura 3.5.

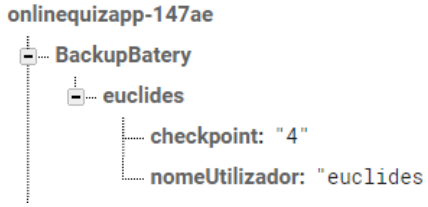

Figura 3.3: Estrutura do nó BackupBattery

[1]MVC é um padrão de arquitetura de software para implementar interfaces de utilizador. O MVC divide uma determinada aplicação de software em três partes interligadas (modelo, representando os dados; visão, a representar as interfaces do utilizador; e controlos, para validar as entradas do utilizador). [Rom14]

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

- Category: regista as categorias das perguntas disponíveis para o jogo. Possui os seguintes nós filhos: - *id*, utilizado como identificador da categoria; - *image*, para guardar a imagem da categoria; - *name*, regista o nome da categoria. A sua estrutura é ilustrada na Figura 3.5.

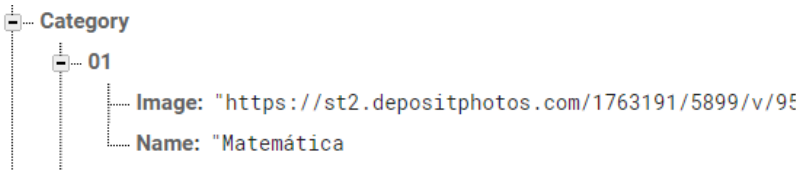

Figura 3.4: Estrutura do nó Category

- Questoes: regista todas as questões do *Quiz*. Este nó é constituído pelos seguintes nós filhos: - *CategoryId*, guarda o identificador da categoria da pergunta; - *IsImageQuestion*, guarda um valor booleano, que indica se a pergunta é uma imagem ou não; - *Questao*, guarda o texto da pergunta ou o link da imagem com a pergunta; - *Resposta(A,B,C,D)*, guardam as opções de resposta para a pergunta; - *RepostaCorrecta* - guarda a opção com a resposta correcta. A sua estrutura é ilustrada na Figura 3.5.

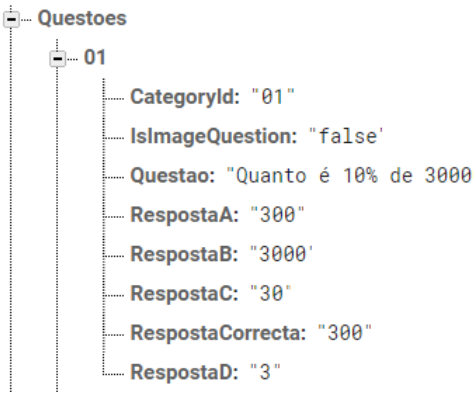

Figura 3.5: Estrutura do nó Question

- Questoes_Score: guarda a pontuação do utilizador na categoria selecionada. É constiuido pelos seguintes nós filhos: - *CategoryId*, regista o id da categoria; - *categoryName*, guarda o nome da categoria; - *questao_score*, guarda o nome do utilizador e id da categoria; - *score*, guarda a pontuação do utilizador; - *utilizador*, guarda o nome do utilizador em sessão. A sua estrutura é ilustrada na Figura 3.6.

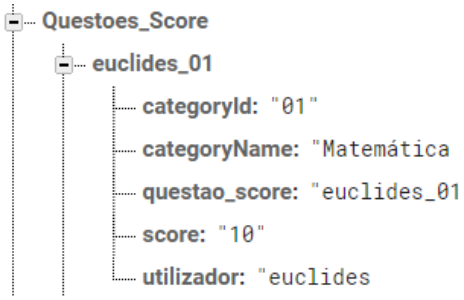

Figura 3.6: Estrutura do nó Questoes_Score

- Ranking: guarda as pontuação geral de cada utilizador. É constituído pelos seguintes nós filhos: - *nomeUtilizador*, guarda o nome do utilizador; *score*, guarda a pontuação geral do utilizador. A sua estrutura é ilustrada na Figura 3.7.

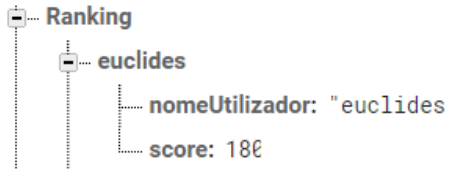

Figura 3.7: Estrutura do nó Ranking

- Users: regista todos utilizadores da aplicação. É constituído pelos seguintes nós filhos: - *email*, guarda o email do utilizador; -*nomeUtilizador*, guarda o nome do utilizador; - *password*, guarda a palavra-passe do utilizador. A sua estrutura é ilustrada na Figura 3.8.

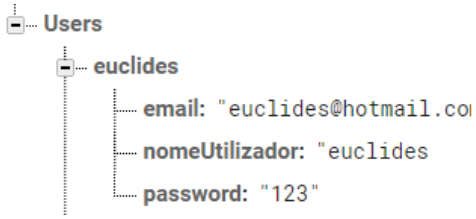

Figura 3.8: Estrutura do nó Users

3.1.4 Modelo de Falhas

Os aspectos como segurança, privacidade e fiabilidade ganham principal importância e enfrentam novos desafios. A segurança contra falhas intencionais é fundamental para garantir a adesão dos utilizadores a serviços geridos por fornecedores a quem confiam os dados. Neste trabalho deixamos para trabalhos futuros falhas na própria aplicação e no comportamento da aplicação quando o número de utilizadores em simultâneo aumenta e analisamos os dois tipos de falhas seguintes:

- Baixo nível de bateria, uma das principais limitações dos dispositivos móveis é exatamente a pouca capacidade de conservação de energia. Dispositivos móveis estão sujeitos a altas descargas de bateria quando estes executam tarefas pesadas, ou quando estão permanentemente ligados a rede.

- Falha na ligação a rede: diversos factores podem ocasionar uma falha na comunicação entre o dispositivo móvel e o servidor. Um caso mais frequente, é devido a mobilidade do utilizador quando os dispositivos entram e saem de uma rede.

Para lidar com estas duas falhas, procuramos soluções já existentes no mercado da MCC (em particular estudamos os mecanismos de tratamento de dados *offline* do Firebase e do Azure) e propusemos duas estratégias. As secções, a seguir, apresentam com mais detalhes o funcionamento destas soluções.

3.2 Mecanismo de Tratamento de Dados *Offline* do Firebase

O Firebase oferece mecanismos para que as aplicações possam funcionar mesmo se a aplicação perder temporariamente a conexão de rede. Porque o Firebase oferece ferramentas para a persistência local de dados, gestão de presença e tratamento de latência.

3.2.1 Persistência Local de Dados

Neste trabalho foi utilizada a funcionalidade persistência em disco que consiste na capacidade das aplicações do Firebase administrarem automaticamente interrupções temporárias de rede. Os dados em cache permanecem disponíveis *offline*, e o Firebase reenvia quaisquer actualizações assim que a conectividade de rede é restaurada. Quando a persistência em disco está activada, a aplicação grava os dados localmente no dispositivo para que possa manter o estado enquanto estiver *offline*, mesmo que o utilizador ou o sistema operativo seja reiniciado.

A activação da persistência em disco pode ser feita com a seguinte linha de código:

```
FirebaseDatabase.getInstance().setPersistenceEnabled(true);
```

Ao ativar a persistência, quaisquer dados que o cliente do Firebase Realtime Database sincronizar *online* permanecerão no disco e estarão disponíveis *offline*, mesmo quando o utilizador ou o sistema operativo reiniciar a aplicação. Isto significa que a aplicação funciona como se estivesse *online* utilizando os dados locais armazenados em cache.

3.2.2 Modo de Funcionamento

O cliente do Firebase Realtime Database mantém automaticamente uma fila de todas as operações de gravação executadas enquanto a aplicação está *offline*. Quando a persistência é ativada, essa fila também é mantida no disco para que todas as gravações estejam disponíveis quando o utilizador ou o sistema operativo reiniciar a aplicação. Quando a aplicação recupera a ligação à rede, todas as operações são enviadas para o servidor do Firebase Realtime Database.

Caso a aplicação esteja a utilizar o Firebase Authentication, o cliente do Firebase Realtime Database mantém o token de autenticação do utilizador entre as reinicializações da aplicação. Se o token de autenticação expirar enquanto a aplicação estiver *offline*, o cliente suspende as operações de gravação até a aplicação reautenticar o utilizador. Caso contrário, as operações de gravação poderão falhar devido às regras de segurança.

3.2.3 Especificação da Sincronização e Limite de Dados

O Firebase Realtime Database sincroniza e armazena uma cópia local dos dados para os *listeners* activados. Além disso, é possível manter locais específicos em sincronia. Um exemplo é o código abaixo que mostra a configuração da sincronização apenas para o nó *Users*.

```
DatabaseReference scoresRef = FirebaseDatabase.getInstance().getReference("users");
scoresRef.keepSynced(true);
```

O cliente do Firebase Realtime Database faz o download automático dos dados neste local e mantém-os sincronizados, mesmo que a referência não tenha *listeners* activados. A sincronização pode ser desativada novamente com a linha de código que se segue.

```
scoresRef.keepSynced(false);
```

O Firebase, por padrão, mantém os dados *offline* até 10 MB de dados armazenados em cache. Se o cache ultrapassar o tamanho configurado, o Firebase Realtime Database limpará os dados que foram menos utilizados. Os dados mantidos em sincronização não são limpos em cache [Goo19].

3.2.4 Acesso aos Dados *Offline*

Se a persistência estiver activada, o Firebase Realtime Database armazena os dados retornados de uma consulta para uso *offline*. Nas consultas criadas *offline*, o Firebase Realtime Database continua funcionando com os dados já carregados. Se os dados solicitados não foram carregados, o Firebase Realtime Database os carrega a partir do cache local. Quando a ligação de rede estiver novamente disponível, os dados serão carregados e refletirão a consulta.

Por exemplo, o código abaixo consulta os últimos quatro jogadores do *Quiz*, ordenados pela pontuação:

```
DatabaseReference scoresRef = FirebaseDatabase.getInstance().getReference("ranking");
scoresRef.orderByValue().limitToLast(4).addChildEventListener(new ChildEventListener() {
    @Override
    public void onChildAdded(@NonNull DataSnapshot snapshot, String previousChild) {
        Log.d(TAG, snapshot.getKey() + " do Jogador é " + snapshot.getValue());
    }
});
```

Se o utilizador perder a conexão, ficar *offline* ou reiniciar a aplicação. Enquanto estiver *offline*, a aplicação consulta os dois últimos itens no mesmo local, como ilustrado no código seguinte. Essa consulta retorna com êxito os dois últimos itens porque a aplicação carregou todos os quatro itens na consulta acima.

```
scoresRef.orderByValue().limitToLast(2).addChildEventListener(new ChildEventListener() {
    @Override
    public void onChildAdded(@NonNull DataSnapshot snapshot, String previousChild) {
        Log.d(TAG, snapshot.getKey() + " do Jogador é " + snapshot.getValue());
    }
});
```

No exemplo anterior, o cliente do Firebase Realtime Database gera eventos *child added* (filho adicionado) para os dois jogadores com pontuação mais alta usando os dados em cache. Porém, isso não gera um evento *value* (valor), já que a aplicação nunca executou essa consulta enquanto estava *online*.

Se a aplicação tivesse que solicitar os últimos seis itens enquanto estivesse *offline*, ele receberia eventos *child added* para os quatro itens em cache imediatamente. Quando o dispositivo fica *online* novamente, o cliente do Firebase Realtime Database é sincronizado com o servidor e recebe os dois eventos *child added* finais e os eventos *value* para a aplicação.

3.2.5 Gestão de Transações *Offline*

Todas as transações que são executadas enquanto a aplicação está *offline* são colocadas em fila. Assim que a aplicação recupera a conectividade de rede, as transações são enviadas para o servidor do Realtime Database.

As transações não são guardadas entre as reinicializações da aplicação [Goo19]. Mesmo com a persistência ativada, as transações não são mantidas entre as reinicializações da aplicação. Portanto, as transações feitas *offline* nem sempre são confirmadas pelo Firebase Realtime Database. Para melhorar a experiência do utilizador, a aplicação precisa mostrar que a transação ainda não foi guardada no Firebase Realtime Database. Outra opção é, o programador, garantir que a aplicação a memorize e a execute depois de ser reiniciada.

3.3 Mecanismo de Tratamento de Dados *Offline* do Azure

A sincronização de dados *offline* é um recurso do Azure Mobile Apps que possibilita aos programadores criarem aplicações que são funcionais mesmo sem uma conexão de rede. Quando a aplicação está no modo *offline*, pode-se criar e modificar dados, que estão armazenados localmente. Assim que estiver *online* novamente, poderá fazer-se a sincronização das alterações locais com o backend do Azure Mobile App. O recurso também inclui suporte para detectar conflitos quando o mesmo registo é alterado no cliente e no backend. Estes conflitos podem ser tratados no servidor ou no cliente [DGGF16].

3.3.1 Armazenamento Local

O armazenamento local é a camada de persistência de dados no dispositivo do cliente. Os SDKs do cliente do Azure Mobile Apps fornecem uma implementação de armazenamento local padrão. No Windows, Xamarin e Android, é baseado no SQLite. No iOS, é baseado em Core Data. O Android e o iOS vêm com uma versão do SQLite no próprio sistema operativo do dispositivo, portanto, não é necessário fazer referência à sua própria versão do SQLite.

Os programadores de aplicações móveis também podem implementar o seu próprio armazenamento local. Por exemplo, caso o programador deseje armazenar dados em um formato criptografado no cliente móvel, é possível definir um armazenamento local que utilize SQLCipher para criptografia [DGGF16].

3.3.2 Acesso aos Dados

Os princípios básicos da edição de dados *offline* com o Azure são os seguintes:

- Todas as operações CRUD de dados ocorrem em um base de dados SQLite local.
- Os dados são enviados apenas para a nuvem quando são ”*push*ed”(enviados).
- Os dados, para uma tabela específica, são recuperadas da nuvem quando são ”*pull*ed”(recebidos).
- Um *pull* sempre iniciará automaticamente um *push*.

3.3.2.1 O Processo *Push*

Enviar os dados para o Azure significa fazer alterações locais e aplicá-las ao Azure. Independentemente de se estar *online* ou não, todas as edições dos dados ocorrem numa base de dados SQLite local. Quando estamos prontos para enviar (*push*) para a nuvem, apenas os dados que foram alterados são enviados. Observa-se que os dados em todas as tabelas serão enviados para o Azure (para consistência de dados), não apenas dados de uma única tabela.

3.3.2.2 O Processo *Pull*

Quando falamos em puxar dados (*pull*), queremos dizer receber dados do Azure e colocá-los no dispositivo. A primeira coisa que deve acontecer antes de se colectar os dados do Azure será enviar todas as alterações feitas localmente para a nuvem. Isso significa que qualquer operação de *pull* invoca automaticamente um *push*, isto acontece para manter os dados no Azure consistentes. Depois que os dados são enviados, os dados solicitados são extraídos. Observa-se que são apenas dados para uma única tabela (e provavelmente filtrados por isso) - nem todos os dados em todas as tabelas no Azure.

3.3.3 Modo de Funcionamento

Para aceder o terminal */tables*, os SDKs do cliente do Azure Mobile fornecem interfaces como IMobileServiceTable (SDK do cliente .NET) ou MSTable (cliente iOS). Essas APIs se conectam diretamente ao backend do Azure Mobile App e falham se o dispositivo cliente não estiver conectado a uma rede.

Para suportar o uso *offline*,a aplicação deve utilizar as APIs da tabela de sincronização, como *IMobileServiceSyncTable*(SDK do cliente .NET) ou *MSSyncTable* (cliente iOS). Todas as mesmas operações CRUD (Criar, Ler, Atualizar, Eliminar) funcionam com as APIs da tabela de sincronização, excepto que agora elas são lidas ou gravadas em um armazenamento local. Antes que qualquer operação da tabela de sincronização possa ser executada, o armazenamento local deve ser iniciado.

A base de dados de armazenamento local é inicializado quando *ToDoActivity.OnCreate()* executa *ToDoActivity.InitLocalStoreAsync()*. Esse método cria uma base de dados local SQLite usando a classe *MobileServiceSQLiteStore* fornecida pelo SDK do cliente do Azure Mobile Apps.
O método *DefineTable* cria uma tabela no armazenamento local que corresponde aos campos no tipo fornecido, *ToDoItem* nesse caso. O tipo não precisa incluir todas as colunas que estão na base de dados remota. É possível armazenar apenas um subconjunto de colunas, como se encontra no código seguinte.

```
// ToDoActivity.cs
  private async Task InitLocalStoreAsync()
  {
     // Código para inicializar o armazenamento no SQLite
     string path = Path.Combine(System.Environment
         .GetFolderPath(System.Environment.SpecialFolder.Personal), localDbFilename);

     if (!File.Exists(path))
     {
         File.Create(path).Dispose();
     }

     var store = new MobileServiceSQLiteStore(path);
     store.DefineTable<ToDoItem>();

     // Uses the default conflict handler, which fails on conflict
     // To use a different conflict handler, pass a parameter to InitializeAsync.
     await client.SyncContext.InitializeAsync(store);
  }
```

O membro *toDoTable* de *ToDoActivity* é do tipo *IMobileServiceSyncTable*, em vez de *IMobileServiceTable*. O *IMobileServiceSyncTable* direciona todas as operações de criação, leitura, atualização e eliminação(CRUD) para a base de dados de armazenamento local.

O programador decide quando as alterações são enviadas ao backend do Azure Mobile App chamando *IMobileServiceSyncContext.pushAsync()*. O contexto de sincronização ajuda a preservar as relações de tabela, rastreando e enviando alterações em todas as tabelas que uma aplicação cliente modificou, quando o *pushAsync* é chamado.

O código fornecido chama *ToDoActivity.SyncAsync()* para sincronizar sempre que a lista *ToDoItem* for atualizada ou um item inteiro for adicionado ou completado. O código é sincronizado após cada alteração local.

No código abaixo, todos os registos na tabela *ToDoItem* remoto são consultados, mas também é possível filtrar os registos passando um *ID* de consulta e consulta para *pushAsync*.

```
// ToDoActivity.cs
  private async Task SyncAsync()
  {
     try {
         await client.SyncContext.pushAsync();
         await toDoTable.pullAsync("allTodoItems", toDoTable.CreateQuery()); // query ID
             is used for incremental sync
     } catch (Java.Net.MalformedURLException) {
         CreateAndShowDialog (new Exception ("Erro ao ciar o Mobile Service. Verifica o
             URL"), "Erro!");
     } catch (Exception e) {
         CreateAndShowDialog (e, "Erro!");
```

```
    }
}
```

3.4 Estratégias de Tolerância a Falhas Propostas

Nesta secção apresentamos duas propostas com estratégias para tolerar falhas de baixo nível de bateria e falha de ligação ou ligação fraca à rede.

3.4.1 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 [Fer13, RLJS16] .Para lidar com a descarga da bateria, implementamos um mecanismo de *check pointing* (ou salvaguarda de estado) isto é, um mecanismo que vai guardar dados de resultados parciais e que permitirá, em caso de falha, a aplicação recuperar a partir do último *checkpoint*. Esta solução permitirá ao utilizador continuar a usar a aplicação assim que tenha recarregado o seu dispositivo ou, eventualmente, tenha acesso a outro dispositivo. Este processo é composto por duas fases a fase de criação e envio do *checkpoint*, e a fase de recuperação.

Algorithm 1 Criação e Envio Checkpoint

```
begin
bs: battery status
n: battery percentage to be set by developer
point: point of checkpoint
user: current user
pwd: password of user
Login(user, pwd)
for bs >= n do
  Current Operations
  checkpoint =SaveCheckpoint (user, point).
Send(checkpoint) to server (BatteryBackupTable)
STOP app
end
```

A fase de criação e envio do *checkpoint* encontra-se detalhado no Algoritmo 1. Após o inicio de sessão é despoletado um mecanismo de verificação periódico do estado de bateria (bs), enquanto se guarda o *checkpoint* e se efectua as operações normais na aplicação. Quando o verificador identificar baixa carga de bateria, interrompe a execução do utilizador, notifica o utilizador, e envia o *checkpoint* ao servidor.

A segunda fase, de recuperação do *checkpoint*, encontra-se detalhada no Algorithm 2: após o inicio de cada sessão é feita uma verificação, na tabela $BatteryBackupTable$, para localizar um possível *checkpoint*. Caso exista um registo com o nome do utilizador, a aplicação retorna ao ponto em que o utilizador estava, antes da descarga. Caso não exista nenhum registo, a aplicação continua a sua execução normal.

O mecanismo é ilustrado na Figura 3.9. Quando o utilizador inicia a sua sessão, no dispositivo $D1.1$, é executado um algoritmo de verificação do nível da bateria de 3 em 3 minutos (ou

Algorithm 2 Recuperar Checkpoint

Recovery($user$, $point$)
begin
$user$: current user
pwd: password of user
checkpoint: data in $BatteryBackupTable$
$checkpoint = (user, point)$
Login($user$, pwd)
if $user$ exists in $BatteryBackupTable$ **then**
  $point = checkpoint(point)$
  Return to $point$
else
  Current Operations
end

outro valor a decidir pelo utilizador). Enquanto isso, em simultâneo, guarda-se no dispositivo o progresso do utilizador, referente ao $Quiz$ em curso. Quando no dispositivo em causa o teste da bateria indica que o nível de carga é abaixo de um certo valor (15% é o valor actual, mas poderá ser parametrizado pelo utilizador), a aplicação despoleta uma mensagem ao utilizador (dispositivo no estado $D1.2$) e o ficheiro de *checkpoint* é enviado ao servidor ($seta2$). Para prosseguir o jogo, o utilizador pode fazer login num outo dispositivo (ou no mesmo, se entretanto recarregou a bateria (estado $D2.1$ do dispositivo). Quando faz login, o ficheiro de *checkpoint* é enviado do servidor ao dispositivo do utilizador, podendo recomeçar no estado onde interrompeu anteriormente.

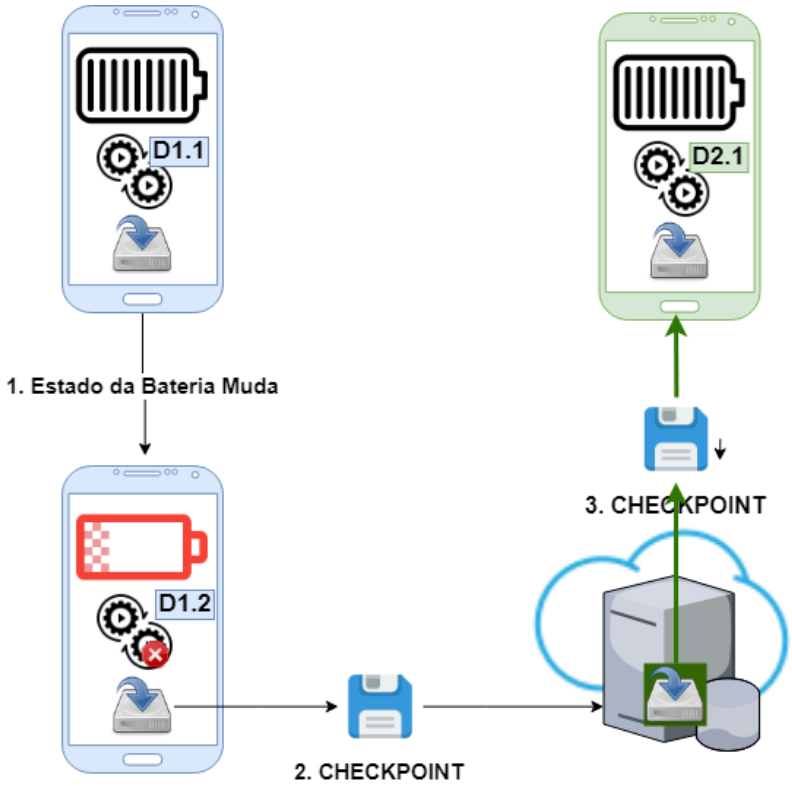

Figura 3.9: Fluxograma dos passos para recuperar de uma falha de rede

3.4.2 Falha de Conectividade

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 [Fer13].

Para tolerar falhas de conectividade, implementamos um mecanismo de persistência local. Quando o utilizador efetua as suas operações e solicita persistência, o algoritmo verifica a ligação ao servidor. Se há ligação, envia e persiste os dados no servidor. Caso contrário, as alterações são armazenadas numa fila na cache do dispositivo do cliente. Quando a ligação é restabelecida, os dados existentes na fila são enviados para que o servidor possa fazer a devida sincronização. O modelo do mecanismo é ilustrado na Figura 3.10.

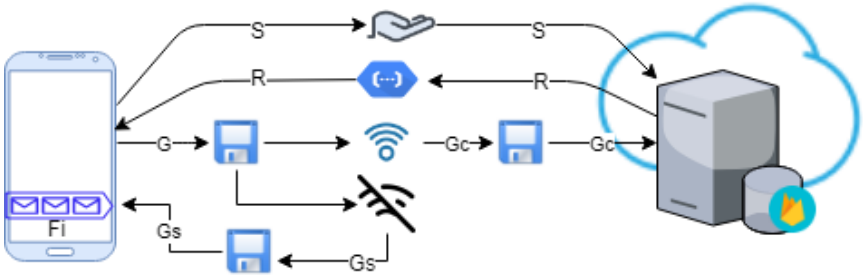

Figura 3.10: Arquitetura física Falha de Conectividade

Quando o dispositivo cliente efetua um pedido S, o servidor envia a resposta ao pedido, R, bem como todo o estado necessário para a execução local de S. A seguir, após o tratamento dos dados, quando o cliente desejar executar qualquer tipo de persistência, G, é verificada a ligação com o servidor. Se existe ligação a persistência é feita no servidor, Gc - guardar com ligação, caso contrário os dados são persistidos numa fila, Fi, na cache do dispositivo cliente, Gs - guardar sem ligação.

3.5 Conclusões

Apresentamos, neste capítulo, o modo de funcionamento dos mecanismos de persistência local de dados do Azure e Firebase, a propostas de tolerância a falhas de conectividade, entre o o dispositivo móvel e a nuvem, e a de tolerância falhas de bateria, do dispositivo móvel. Para implementar as soluções desenvolveu-se duas aplicações Android, uma no Android Studio, hospedada no Azure, e outra no Xamarin, hospedada no Azure. Os resultados desta implementação são apresentados no capítulo 4.

Capítulo 4

Ambiente Experimental, Testes e Resultados

Neste capítulo, mostramos o ambiente experimental onde testamos e avaliamos as soluções apresentadas no Capítulo 3. Testamos os mecanismos de persistência do Firebase, do Azure e as estratégias de falha de bateria e de rede. Por fim, efetuamos uma discussão do impacto dos mecanismos implementados.

4.1 Ambiente Experimental

Os mecanismos de tolerância a falhas foram implementados e avaliados em termos de funcionalidade e tempo de execução adicional simulando a ocorrência de falhas. Para isso, foi usado um conjunto de máquinas que se descreve de seguida. Um computador portátil com processador Intel(R) i7-6500U a 2,50GHz, 8GB de RAM e sistema operativo Windows de 64 bits. Nesta máquina foram criadas com o Android Studio, duas máquinas virtuais VM1 e VM2, para simularem máquinas físicas. Foram também usados dois dispositivos móveis físicos, um *tablet* (Máquina Física 1 – MF1) e um telemóvel (MF2). As características das duas máquinas virtuais e das duas máquinas físicas apresentam-se na tabela 4.1.

Tabela 4.1: Características das máquinas utilizadas nos testes

Máquina	acsCPU	RAM	Versão do Android
VM1	n.a.*	1GB	7.0 API 24
VM2	n.a.*	1.5	7.0 API 24
MF1	8x ARM A53 @ 1,59GHz	2GB	8.1 API 27
MF2	4x Samsung M1 @ 2.60GHz 4x ARM A53 @ 1,59GHz	3.5GB	8.1 API 2

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

4.2 Testes e Resultados

Para avaliar as soluções apresentadas utilizamos dois tipos de jogos: o *Quiz* em que as perguntas se apresentam em forma de textos, e a mesma aplicação em que as perguntas se presentam em forma de imagens (de no máximo 300KB). Efectuamos os dois testes para analisar a eficiência (o tempo de resposta) das soluções quando se tem baixo ou alto tempo de comunicação entre o dispositivo móvel e o servidor.

Apresentamos os tempos de execução normal da aplicação com o armazenamento no Firebase e no Azure, os *overheads* de execução com os mecanismos de persistência activados, e os *overheads* que as nossas soluções, utilizando, como base, a execução normal do Firebase, pelo facto

de utilizarmos o Firebase como a infraestrutura de nuvem.

4.2.1 Execução Normal da Aplicação

Para determinarmos o tempo adicional (*overhead*) introduzido pelos mecanismos de tolerância a falhas implementados, começamos por medir o tempo de execução sem qualquer mecanismo e depois com cada um dos mecanismos. Nessa medição não foi incluído o tempo que o jogador leva a responder ao questionário pois este depende do tempo de reflexão de cada utilizador. Foi medido o tempo de transferência dos dados e o tempo de persistência. Para cada situação foram realizadas três execuções apresentando-se o tempo médio obtido. Foram testadas duas situações, questionários com perguntas de texto simples e questionário com perguntas que incluem imagens.

A Tabela 4.2 mostra o tempo total de execução quando não existe nenhum mecanismo de tolerância a falhas, para questionários de texto e questionários de imagens no Firebase. O tempo total tem início quando o jogador submete as respostas, isto é, a sua resolução do $Quiz$, e termina quando o servidor devolve a pontuação obtida.

Tabela 4.2: Resultados com a execução normal no servidor Firebase

Máquinas	Tempo (ms) questionário de texto	Tempo (ms) quesionário de imagens
VM1	87	153
VM2	85	149
MF1	63	108
MF2	51	89

A Tabela 4.3 mostra o tempo total de execução quando não existe nenhum mecanismo de tolerância a falhas, para questionários de texto e questionários de imagens no Azure. O tempo total tem início quando o jogador submete as respostas, isto é, a sua resolução do $Quiz$, e termina quando o servidor devolve a pontuação obtida.

Tabela 4.3: Resultados com a execução normal no servidor Azure

Máquinas	Tempo (ms) questionário de texto	Tempo (ms) quesionário de imagens
VM1	95	156
VM2	90	160
MF1	81	111
MF2	70	86

Os resultados do Azure, mostraram-se mais lentos em relação aos do Firebase, com excepção do tempo de questionário de imagens da MF2.

4.2.2 Execução com os Mecanismos de Persistência Activados

Para simularmos a falha de rede, para ambos os casos (Firebase e Azure), interrompemos a conexão das máquinas com os respectivos servidores. Após isto, voltou-se a ligar a conexão e

analisou-se o comportamento das soluções.

A Tabela 4.4 mostra , para o mesmo tipo de questionários, os tempos de execução, obtidos da mesma forma mas quando foi usado o mecanismo de persistência do Firebase. Para efeitos de medição do tempo adicional de execução, este mecanismo foi ativado imediatamente antes da conexão à base de dados do servidor. A contagem do tempo foi interrompida no momento em que se detectou a falha de rede, até ao momento em que a rede voltou a estar disponível, ignorando assim o tempo em que aplicação ficou *offline*.

Apresentam-se os tempos de execução, e, para cada caso, o tempo de execução adicional em percentagem (*overhead*) em relação à execução sem persistência no Firabase.

Tabela 4.4: Tempos de execução com persistência local do Firebase activada

Máquinas	Tempo (ms) questionário de texto (%*overhead*)	Tempo (ms) questionário de imagens (%*overhead*)
VM1	103 (24.1%)	190 (29.2%)
VM2	101 (24.7%)	201 (29.7%)
MF1	80 (29.9%)	140 (27.3%)
MF2	57 (26.7%)	101 (30.3%)

Caso haja falha de rede, por causa da persistência activada, os dados ficam armazenados localmente, e assim que a aplicação, através do verificador de presença, detecta a disponibilidade da rede e efectua a persistência dos dados *offline* para o Firebase Realtime Database. Quando há falha de bateria, o comportamento deste mecanismo é diferente da falha de rede. Por se tratar de uma aplicação sequencial, caso estivermos a meio de um jogo a e a bateria descarrega, perdemos o jogo, em execução.

A Tabela 4.5 mostra, para o mesmo tipo de questionários, os tempos de execução, obtidos da mesma forma mas quando foi usado o mecanismo de persistência do Azure. Para efeitos de medição do tempo adicional de execução, este mecanismo foi ativado imediatamente antes da conexão à base de dados do servidor. Como no caso anterior, a contagem do tempo foi interrompida no momento em que se detectou a falha de rede, até ao momento em que a rede voltou a estar disponível, ignorando assim o tempo em que aplicação ficou *offline*.

Apresentam-se os tempos de execução, e para cada caso o tempo de execução adicional em percentagem (*overhead*) em relação à execução sem persistência no Azure.

Tabela 4.5: Tempos de execução com persistência local do Azure activada

Máquinas	Tempo (ms) questionário de texto (%*overhead*)	Tempo (ms) questionário de imagens (%*overhead*)
VM1	124 (30.5%)	203 (30%)
VM2	116 (28.9%)	201 (25.6%)
MF1	99 (22.2%)	147 (32.4%)
MF2	88 (25.7%)	113 (31.4%)

Quando há falha de rede, com o mecanismo de persistência do Azure activado, os dados ficam armazenados nas tabelas do SQLite, e assim que a aplicação detecta a disponibilidade da rede

efectua a persistência dos dados *offline* para o Azure. Quando há falha de bateria, o comportamento deste mecanismo é diferente da falha de rede. Por se tratar de uma aplicação sequencial, caso estivermos a meio de um jogo e a bateria descarrega, perdemos o jogo, em execução.

4.2.3 Execução com os Mecanismos Propostos

4.2.3.1 Falha da Bateria

A simulação da falha de bateria para as VMs apresentou-se mais flexível, em relação às MFs, por causa da possibilidade que as VMs oferecem na manipulação a percentagem de carga. Assim foi efectuada esta simulação deixando o dispositivo descarregar, até uma percentagem que pode ser definida pelo programador, e recuperando o *checkpoint*, no carregamento do mesmo dispositivo, ou na eventual utilização de um segundo dispositivo.

Para calcular o tempo de execução do mecanismo de suporte à falha de carga da bateria, o tempo de execução medido tem os seguintes componentes:

t_{CC}- tempo de criação do *checkpoint*;
t_{RC}- tempo de recuperação do *checkpoint* (transferência de dados e actualização);

Sendo t_{Total}= t_{CC} + t_{RC}, o tempo adicional de execução (*overhead*) introduzido pelo mecanismo. O tempo de envio do *checkpoint* é equivalente ao tempo de persistência dos dados que também existiria se o *checkpoint* não fosse criado.

A Tabela 4.6 mostra os tempos de execução obtidos para o mecanismo de tolerância a falhas de bateria. Os *overheads* representam a soma do t_{Total}, pelos tempos de execução normal do Firebase, apresentados na Tabela 4.2.

Tabela 4.6: Tempos de execução adicional em ms do mecanismo de falha de bateria

Máquinas	**Tempo (ms) questionário de texto (%*overhead*)**			**Tempo (ms) questionário de imagens (%*overhead*)**		
	t_{CC}	t_{RC}	t_{Total}	t_{CC}	t_{RC}	t_{Total}
VM1	17	19	**36 (43.4%)**	25	36	**61 (41.5%)**
VM2	18	21	**39 (48.2%)**	27	37	**64 (41.3%)**
MF1	16	17	**33 (52,4%)**	23	24	**47 (42.7%)**
MF2	11	13	**24 (53.3%)**	21	22	**43 (56.6%)**

Quando o verificador do estado detecta o baixo nível de bateria, este cria um último *checkpoint* e envia para o servidor. Sequencialmente, quando houver bateria suficiente, ou eventualmente se tenha acesso a outro dispositivo móvel, após o inicio de sessão é detectada a existência do *checkpoint* e é recuperada a sessão anterior. Neste caso, para os mecanismos de persistência de dados local do Azure e Firebase, significaria a perca de um jogo.

4.2.3.2 Falha na Conectividade

Para simularmos a falha de rede, e testarmos o nosso mecanismo de tolerância a falha de conectividade, interrompemos a conexão das máquinas com os respectivos servidores. Após isto, voltou-se a ligar a conexão e analisou-se o comportamento das soluções, quantificando-se os

tempos de execução.

O mecanismo de tolerância a falhas de ligação à rede é feito em duas etapas, tempo de gravação na fila e tempo de sincronização dos dados pelo servidor assim que a ligação é retomada. Como o tempo de sincronização também ocorreria se não houvesse falha de ligação, o tempo adicional exigido pelo mecanismo é apenas o tempo de gravação na fila (t_G). A Tabela 4.7 mostra os resultados obtidos quando uma falha de conectividade foi simulada, em termos de tempo adicional absoluto e percentagem em relação à execução sem mecanismo de tolerância à falha, apresentados na Tabela 4.2

Tabela 4.7: Tempo de execução em milissegundos para o caso de falha de conectividade

Máquinas	**Texto**		**Imagem**	
	t_G	***overhead***	t_G	***overhead***
VM1	30	**36.1%**	64	**43.5%**
VM2	29	**35.8%**	62	**40%**
MF1	23	**36.5%**	45	**40.9%**
MF2	22	**48.9%**	42	**55.3%**

4.3 Resultados: Visão Geral

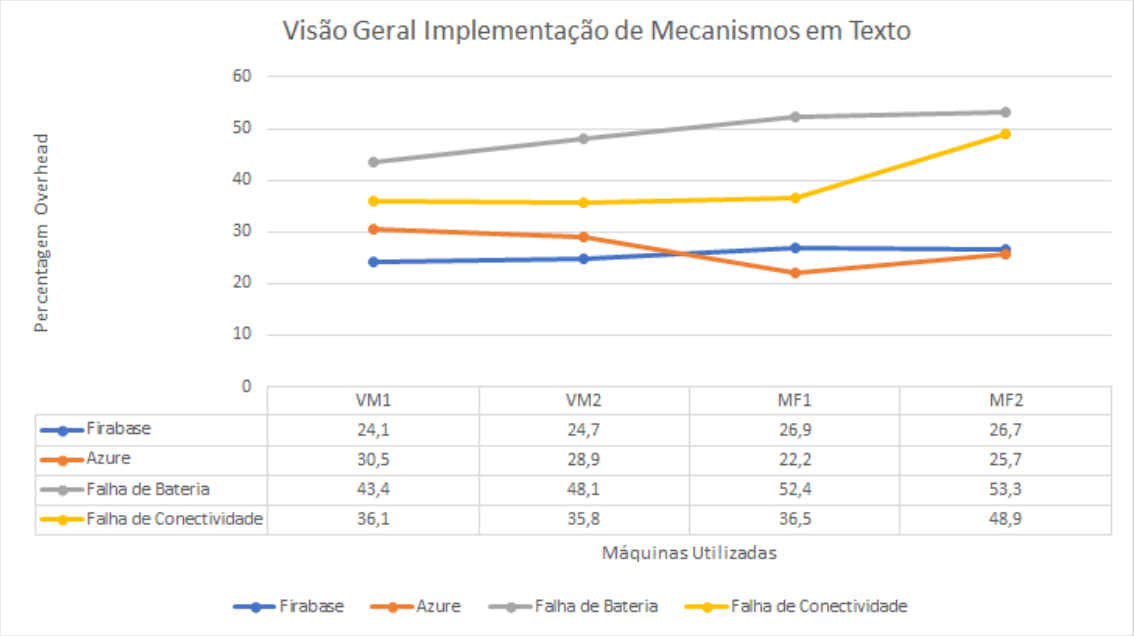

Figura 4.1: Visao geral da implementação dos mecanismos em textos

Os resultados apresentados na Figura 4.1 representam os *overheads*, de todos os mecanismos utilizados neste trabalho, num *Quiz* com textos. Nota-se que há um *overhead* menor nos mecanismos nativos (Firebase e Azure), entre 24% e 30%, em relação aos mecanismos propostos nesta dissertação, que estão entre 36% e 53%.

Os resultados apresentados na Figura 4.2 representam o *overheads* de todos os mecanismos

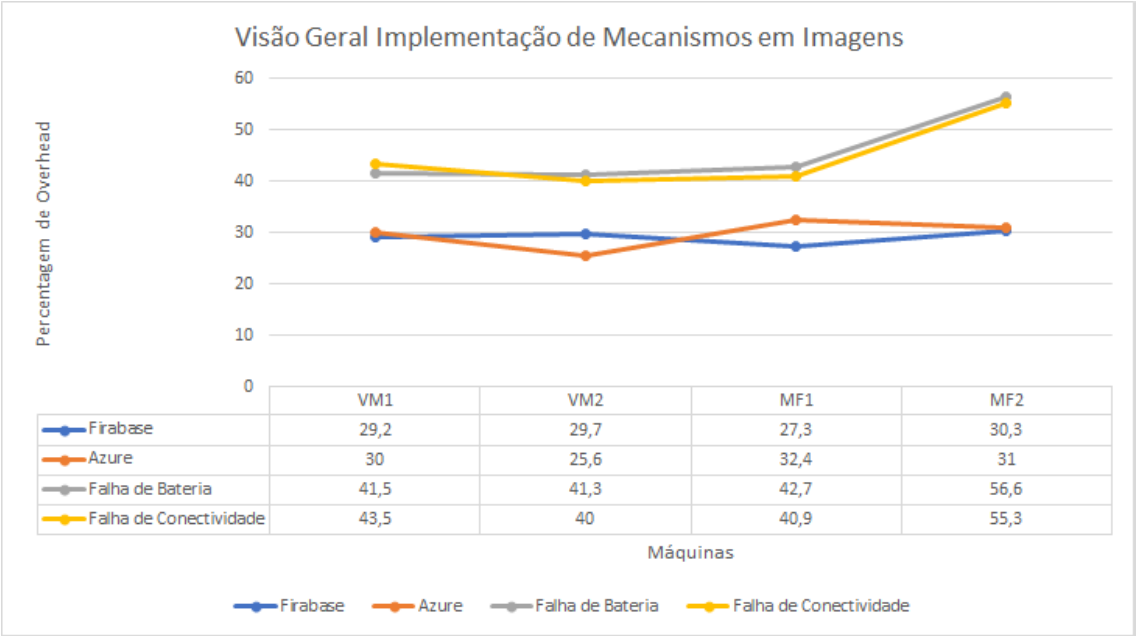

Figura 4.2: Visao geral da implementação dos mecanismos em imagens

utilizados neste trabalho, num *Quiz* com imagens. Nota-se que há um *overhead* menor nos mecanismos nativos (Firebase e Azure), entre 25% e 33%, em relação aos mecanismos propostos nesta dissertação, que estão entre 34% e 56%.

Para compararmos directamente o *overhead* do nosso mecanismo com o *overhead* do mecanismo de persistência do Firebase e Azure, teríamos de medir o tempo de salvaguarda do estado da aplicação na cache do sistema e o tempo de sincronização após a falha. Não tendo acesso ao código fonte do Firebase e Azure, tal não é possível. Por outro lado, medir tempos totais de execução com falhas, implicaria automatizar o processo de simulação de falhas. Pelos tempos obtidos, o nosso mecanismo parece ter um custo mais elevado, porém, tem no entanto a vantagem de apenas guardar as operações que o jogador fez desde o início do jogo em vez de todo o estado da aplicação. O facto de o nosso mecanismo inibir a continuação do jogo após a criação do *checkpoint*, pode ser visto também como uma vantagem. O jogador não corre o risco de ficar a meio de um jogo por falta de bateria e ainda tem a possibilidade de recomeçar, a partir do ponto em que tinha parado, tal não é previsto no mecanismo de persistência local dos dados do Azure e Firebase.

Na Tabela 4.7, podemos observar que o nosso mecanismo, para tolerar o caso de falha na ligação, apresenta um *overhead* de cerca de 40% para questionários de texto e de cerca de 50% para imagens. Podemos considerar valores razoáveis visto que os mecanismos próprios a plataforma estão com valores entre 25% a 35%, para textos e imagens, para além de serem todos valores em milissegundos. Neste mesmo caso, o estado do jogo foi guardado na cache do periférico e por isso bastante mais rápido do que o armazenamento do *checkpoint*.

4.4 Conclusões

Neste capítulo apresentamos o ambiente experimental e os resultados dos mecanismos estudados e propostos. Os mecanismos implementados neste capítulo, apresentaram percentagens de *overhead* maiores do que os mecanismos de persistência do Azure e do Firebase. Porém, analisando o contexto em que aplicamos o experimento consideramos estes *overheads* aceitáveis. Minimizações e/ou otimizações são necessárias para que os mecanismos propostos se aproximem ou melhorem, cada vez mais, daqueles que chamamos de mecanismos nativos.

Capítulo 5

Conclusões

Neste capítulo, apresentamos as principais conclusões do trabalho e a projecção de alguns trabalhos futuros.

5.1 Principais Conclusões

Os sistemas em MCC estão, cada vez mais, presentes no quotidiano do ser humano. Devido às suas vantagens, numerosos serviços têm estado a adoptar esta tecnologia. Actualmente, apesar das suas vantagens, a MCC apresenta alguns desafios. Estes desafios têm vindo a ser ultrapassados com a criação de frameworks e, sobretudo, com o aparecimento de plataformas BaaS, que cada vez mais incorporam serviços vitais ao desenvolvimento de aplicações em MCC, como autenticação, armazenamento, segurança, processamento de dados, suporte a dados *offline* entre outros.

Utilizamos um *Quiz*, como a aplicação exemplo para testar soluções de tolerância a falhas. Estas soluções foram desde estudar mecanismos de tratamento de dados *offline* do Firebase e do Azure, como também uma apresentação de propostas para tolerar falhas de ligação a rede e descargas inesperadas de bateria. A solução, da falha de bateria, mostrou-se menos genérica, pois adequa-se mais a aplicações que possuem uma característica interactiva (ex. Jogos). Enquanto que o tratamento de falhas de conectividade apresenta-se mais genérica, podendo servir para qualquer tipo de aplicação em MCC.

A utilização dos métodos propostos nestes trabalhos apresentam-se como alternativa aos métodos nativos e possibilitam ainda ao programador uma implementação menos abstrata, fornecendo acesso a todos os passos da implementação do mecanismo. Para além disto, os possíveis custos associados a utilização do Azure e Firebase, apresentam-se como desvantagens, na utilização dos BaaS.

Utilizando o mecanismo de tolerância a falhas de bateria, o jogador não corre o risco de ficar a meio de um jogo por falta de bateria e ainda tem a possibilidade de recomeçar, a partir do ponto em que tinha parado. Tal não acontece nos mecanismos de persistência local de dados do Azure e Firebase.

Ficou ilustrado que as nossas propostas adicionaram, de um modo geral , de cerca de 15% de *overehead* em relação aos mecanismos nativos das plataformas. Realçando as vantagens da nossa proposta, como o facto de possuir um mecanismo mais manipulável em relação aos dos BaaS, e possibilidade, no caso da falha bateria, de iniciar sessão num outro dispositivo.

5.2 Trabalhos Futuros

Apresentam-se como trabalhos futuros:

- a optimização dos algoritmos de modo a encontrar, tempos mais próximos ou melhores que os mecanismos nativos;
- estudar mecanismos de persistência de outras plataformas BaaS, para se obter resultados mais conclusivos;
- uma análise do desempenho das duas soluções, quando vários dispositivos estiverem em situações de falhas, simultaneamente;
- estudar outros mecanismos para tolerância a falhas de aplicações em MCC, para além da falha de bateria e conectividade à rede.

Bibliografia

[AASBS16] Md Abdullah-Al-Shafi, Ali Newaz Bahar, and Sajeeb Saha. Mobile on-demand computing: The future generation of cloud. *International Journal of Future Generation Communication and Networking*, 9:161–178, 12 2016. 1, 10

[AGH18] Khadija Akherfi, Micheal Gerndt, and Hamid Harroud. Mobile cloud computing for computation offloading: Issues and challenges. *Applied Computing and Informatics*, 14(1):1–16, 2018. Available from: https://doi.org/10.1016/j.aci.2016.11.002. xiii, 2, 10, 11, 12, 13, 14, 16

[AKS13] Mayank Arora, Mala Kalra, and Sarabjeet Singh. Acof: Autonomous computation offloading framework for android using cloud. *2013 2nd International Conference on Information Management in the Knowledge Economy*, pages 143–149, 2013. 15

[AKV15] Mazhar Ali, Samee U. Khan, and Athanasios V. Vasilakos. Security in cloud computing: Opportunities and challenges. *Information Sciences*, 305:357 – 383, 2015. Available from: http://www.sciencedirect.com/science/article/pii/S0020025515000638. xiii, 6

[Ama19a] Amazon. AWS AppSync, 2019. Available from: https://aws.amazon.com/pt/appsync/. xiii, 21

[Ama19b] Amazon. Documentação da AWS, 2019. Available from: https://docs.aws.amazon.com/index.html?nc2=h{_}ql{_}doc. 21

[ATS15] M. Ayad, M. Taher, and A. Salem. Mobile gpu cloud computing with real time application. In *5th International Conference on Energy Aware Computing Systems Applications*, pages 1–4, March 2015. 16

[BJGP16] Laura Briz, Juan Juanes, and Francisco García-Peñalvo. *Preface of Handbook of Research on Mobile Devices and Applications in Higher Education Settings*, pages xxii–xxvi. 10 2016. 1

[CIM+11] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM. Available from: http://doi.acm.org/10.1145/1966445.1966473. xiii, 14

[CLJ18] S. Criollo-C, S. Luján-Mora, and A. Jaramillo-Alcázar. Advantages and disadvantages of m-learning in current education. In *2018 IEEE World Engineering Education Conference (EDUNINE)*, pages 1–6, March 2018. 1

[DGGF16] Craig Dunn, Kevin Gallahan, Glenn Gailey, and Cory Fowler. Offline Data Sync in Azure Mobile Apps, 2016. Available from: https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-offline-data-sync. 24, 33

[DHTZ15] Shuiguang Deng, Longtao Huang, Javid Taheri, and Albert Y. Zomaya. Computation Offloading for Service Workflow in Mobile Cloud Computing. *IEEE Transactions on Parallel and Distributed Systems*, 26(12):3317–3329, 2015. 1, 5, 10, 20

[DS16a] Naga Raju Dasari and V Saritha. Architecture for fault tolerance in mobile cloud computing using disease resistance approach. *IJCNIS*, 8, 2016. 20

[DS16b] Shantanu Deshmukh and Rinku Shah. Computation offloading frameworks in mobile cloud computing : a survey. pages 1–5, 03 2016. 11, 12

[Fer13] Seng W. ; Rahayu Wenny Fernando, Niroshinie; Loke. Future generation computer systems. *FGCS*, 30 May 2013. Available from: `http://www.elsevier.com/locate/fgcs`. xiii, 9, 10, 18, 19, 20, 36, 38

[Gom15] Gomes M Gomes Da Silva V, Silva de Souza R. Desafios da computação móvel e usabilidade em sala de aula. *Revista de Estudios e Investigación en Psicología y Educación*, (13):181, 2015. Available from: `https://repositorium.sdum.uminho.pt/bitstream/1822/40003/1/Viviane-Ranniery{_}Gomes-Galaico-2015.pdf`. 9

[Goo] Google. Firebase realtime database. `https://firebase.google.com/docs/database/`. Accessed: 2019-03-10. 28

[Goo19] Google. Produtos do Firebase, 2019. Available from: `https://firebase.google.com/products`. 23, 32, 33

[HMZ+14] J. Hauswald, T. Manville, Q. Zheng, R. Dreslinski, C. Chakrabarti, and T. Mudge. A hybrid approach to offloading mobile image classification. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8375–8379, May 2014. 17

[Hoa09] Ping Wang Hoang T. Dinh, Chonho Lee, Dusit Niyato. Channel assignment study for multi-Channel multi-radio wireless mesh networks. *Journal of Internet Technology*, 10(4):345–352, 2009. 5, 11, 17, 18

[Jon15] Jonny Dimond. The Firebase Blog: Announcing Mobile Offline Support, 2015. Available from: `https://firebase.googleblog.com/2015/05/announcing-mobile-offline-support{_}93.html`. xiii, 24

[KLLB13] Karthik Kumar, Jibang Liu, Yung Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, 2013. 1, 10

[KPKB12] Roelof Kemp, Nick Palmer, T Kielmann, and Henri Bal. Cuckoo: A computation offloading framework for smartphones. volume 76, 01 2012. 13

[Lei17] Rodrigo Freitas Leite. SELETO: Sistema de Motorista Particular. Technical report, UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, 2017. Available from: `https://www.lume.ufrgs.br/bitstream/handle/10183/168957/001048293.pdf?sequence=1{&}isAllowed=y`. 25

[LG19] JongHyuk Lee and JoonMin Gil. Adaptive fault-tolerant scheduling strategies for mobile cloud computing. *The Journal of Supercomputing*, 01 2019. 20

[LSX15] C. Looi, D. Sun, and W. Xie. Exploring students' progression in an inquiry science curriculum enabled by mobile learning. *IEEE Transactions on Learning Technologies*, 8(1):43–54, Jan 2015. 16

[LYL+18] W. Li, C. Yen, Y. Lin, S. Tung, and S. Huang. Justiot internet of things based on the firebase real-time database. In *2018 IEEE International Conference on Smart Manufacturing, Industrial Logistics Engineering (SMILE)*, pages 43–47, Feb 2018. 23

[Mar09] Eugene E Marinelli. Hyrax : Cloud Computing on Mobile Devices using MapReduce. *MS Thesis*, 0389(September):1–123, 2009. Available from: http://www.contrib.andrew.cmu.edu/{~}emarinel/masters{_}thesis/emarinel{_}ms{_}thesis.pdf. 19

[MCF+11] Jerrid Matthews, Max Chang, ZhiNan Feng, Ravi Srinivas, and Mario Gerla. Powersense: Power aware dengue diagnosis on mobile phones. In *Proceedings of the First ACM Workshop on Mobile Systems, Applications, and Services for Healthcare*, mHealthSys '11, pages 6:1–6:6, New York, NY, USA, 2011. ACM. Available from: http://doi.acm.org/10.1145/2064942.2064951. 17

[MG11] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. Technical report, U.S. Department of Commerce, 2011. Available from: http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf. 5, 6, 7, 8

[Mic19] Microsoft. Serviço de Aplicações Móveis | Microsoft Azure, 2019. Available from: https://azure.microsoft.com/pt-pt/services/app-service/mobile/. 24

[MRT18] 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)*, pages 73–78, March 2018. 11

[NKLL10] Y. Nimmagadda, K. Kumar, Y. Lu, and C. S. G. Lee. Real-time moving object recognition and tracking using computation offloading. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2449–2455, Oct 2010. 16

[Ora19] Oracle Mobile Cloud. Data Offline and Sync, 2019. Available from: https://docs.oracle.com/en/cloud/paas/mobile-cloud/mcsua/data-offline-and-sync.html{#}GUID-7C2A0D49-F898-4886-9A6A-4FF799F776F4. xiii, xv, 22

[PN11] Paulo HC Pedrosa and Tiago Nogueira. Computação em nuvem. 3, 2011. 9

[Pra12] Anupama Prasanth. Cloud computing services: A survey. *International Journal of Computer Applications*, 46:975–8887, 05 2012. xv, 8

[Pra16] Pranav Patil. Mobile Computing: Issues and Limitations. *International Journal of Computer Science and Mobile Applications*, 8:1–6, 2016. Available from: www.ijcsma.com. 10

[PSMS15] D. Puthal, B. P. S. Sahoo, S. Mishra, and S. Swain. Cloud computing features, issues, and challenges: A big picture. In *2015 International Conference on Computational Intelligence and Networks*, pages 116–123, Jan 2015. 5, 6, 7

[QA17] Hao Qian and Daniel Andresen. Jade: Reducing energy consumption of android app. *International Journal of Networked and Distributed Computing*, 3:150–158, 2017. Available from: https://doi.org/10.2991/ijndc.2015.3.3.2. 13, 14

[Que18] Ricardo Queirós. *Android Profissional: Desenvolvimento Moderno de Aplicações*. FCA - Editora de Informática, Lda, 2018. 20

[Rapa] RapidAPI. Getting Started with RapidAPI SDKs. Available from: https://docs.rapidapi.com/docs/getting-started-with-rapidapi-sdks. 23

[Rapb] RapidAPI. Rakuten RapidAPI. Available from: https://api.rakuten.co.jp/en/. xiii, 23

[Rapc] RapidAPI. What is RapidAPI. Available from: https://docs.rapidapi.com/docs. 23

[RLJS16] Jacob R. Lorch and Alan Jay Smith. Software strategies for portable computer energy management. *Personal Communications, IEEE*, 5:60 – 73, 07 2016. 36

[Rom14] W. Romsaiyud. Applying mvc data model on hadoop for delivering the business intelligence. In *2014 Twelfth International Conference on ICT and Knowledge Engineering*, pages 78–82, Nov 2014. 28

[RSJSSM18] A R Suraj, Sneha Janani Shekar, and G S Mamatha. A robust security model for cloud computing applications. pages 018–022, 03 2018. 19

[SAGH14] Muhammad Shiraz, Ejaz Ahmed, Abdullah Gani, and Qi Han. Investigation on runtime partitioning of elastic mobile applications for mobile cloud computing. *The Journal of Supercomputing*, 67(1):84–103, Jan 2014. Available from: https://doi.org/10.1007/s11227-013-0988-6. 11

[Sas13] P Sasikala. Research challenges and potential green technological applications in cloud computing. Technical Report 1, 2013. Available from: http://cs691vrbsky.cs.ua.edu/2014/Papers/ChallengeGreen.pdf. 5, 6

[Sat96] M. Satyanarayanan. Fundamental challenges in mobile computing. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '96, pages 1–7, New York, NY, USA, 1996. ACM. Available from: http://doi.acm.org/10.1145/248052.248053. 1, 19

[Sat13] Mahadev Satyanarayanan. Mobile computing: the next decade. *School of Computer Science*, 11(6):1081–1087, 2013. 10

[sBD18] G. s. Bhathal and A. S. Dhiman. Big data solution: Improvised distributions framework of hadoop. In *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 35–38, June 2018. 13

[SBL17] Philip Stahl, Jonatan Broberg, and Björn Landfeldt. Dynamic fault-tolerance and mobility provisioning for services on mobile cloud platforms. pages 131–138, 04 2017. 20

[SD18] K. Swedha and T. Dubey. Analysis of web authentication methods using amazon web services. In *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–6, July 2018. 21

[SGS+14] Muhammad Shiraz, Abdullah Gani, Azra Shamim, Suleman Khan, and Raja Wasim Ahmad. Energy efficient computational offloading framework for mobile cloud computing. *Journal of Grid Computing*, 13:1–18, 2014. 15

[Sin18] K. K. Singh. An artificial intelligence and cloud based collaborative platform for plant disease identification, tracking and forecasting for farmers. In *2018 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 49–56, Nov 2018. xiii, 16, 17

[SN16] P. Subhashini and S. Nalla. Data retrieval mechanism using amazon simple storage service and windows azure. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 412–414, March 2016. 24

[SS11] Weiguang Song and Xiaolong Su. Review of mobile cloud computing. 05 2011. 9

[SS13] Amit K Sharma and Priyanka Soni. Mobile Cloud Computing (MCC): Open Research Issues. *International Journal of Innovations in Engineering and Technology (IJIET)*, 2(1):24–27, 2013. 10

[Ter16] D. Terry. Toward a new approach to iot fault tolerance. *Computer*, 49(8):80–83, Aug 2016. 2

[WD10] Shaoxuan Wang and Sujit Dey. Rendering adaptation to address communication and computation constraints in cloud mobile gaming. pages 1–6, 12 2010. 17

[Wil12] A. Williams. Kii Cloud Opens Doors For Mobile Developer Platform With 25 Million End Users | TechCrunch, 2012. Available from: `https://techcrunch.com/2012/10/11/kii-cloud-opens-doors-for-mobile-developer-platform-with-25-million-end-users/`. 20

[XDL+14] Feng Xia, Fangwei Ding, Jie Li, Xiangjie Kong, Laurence T. Yang, and Jianhua Ma. Phone2cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. *Information Systems Frontiers*, 16(1):95–111, Mar 2014. Available from: `https://doi.org/10.1007/s10796-013-9458-1`. 13

[ZL16] Zhaosheng Zhang and Shuyu 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*, pages 81–82, 2016. 9, 10, 12