Universidade Federal de Pernambuco

Centro de Informática

Graduação em Ciência da Computação

COLETA DE LOCALIZACAO EM BACKGROUND: UM ESTUDO SOBRE OTIMIZAÇÃO DA COLETA DE DADOS VISANDO REDUÇÃO DO CONSUMO DE RECURSOS

Gabriel Avelar Falcone de Melo

TRABALHO DE GRADUAÇÃO
Recife

12 de Julho de 2017

Universidade Federal de Pernambuco

Centro de Informática

Gabriel Avelar Falcone de Melo

COLETA DE LOCALIZACAO EM BACKGROUND: UM ESTUDO SOBRE OTIMIZAÇÃO DA COLETA DE DADOS VISANDO REDUÇÃO DO CONSUMO DE RECURSOS

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Prof. Carlos André Guimarães Ferraz*

Recife

12 de Julho de 2017

AGRADECIMENTOS

Gostaria de agradecer à Universidade Federal de Pernambuco, ao Centro de Informática, e a todos os seus funcionários, por todo o apoio e infraestrutura que me ajudaram a desenvolver e completar meus estudos.

Agradeço também a todos os professores pelas orientações durante o curso, em especial ao professor Carlos André Guimarães Ferraz por me auxiliar na confecção deste documento. Agradeço a Airton Sampaio, Alan Gomes, André Ferraz, Denyson Messias, Eduardo Martins, Júlio Rangel, Lucas de Queiroz, e aos demais pela aventura proporcionada na criação da In Loco. Agradecimentos também para Larissa Passos, Luís Medeiros e Pedro Bello pela ajuda na criação e testes necessários para a confecção desse documento.

E finalmente, gostaria de agradecer a Carol, que tanto insistiu para que eu concluísse este trabalho mesmo com todas as adversidades. Serei sempre grato a minha família, a minha mãe, Cláudia, ao meu pai, Telmo, a minha irmã, Mariana, e ao meu irmão, Gustavo, por estarem sempre presentes em minha vida.

RESUMO

Com o domínio dos dados de navegação do mundo online através de *cookies*, e os resultados obtido através dos serviços que utilizam informações a respeito de seus consumidores para prover serviços superiores, percebe-se que existe um outro cenário não abordado, o mundo *offline*. Devido ao atual volume de usuários de *smartphones*, a coleta dos dados de localização, providos por tais aparelhos, torna-se uma abordagem eficaz para solucionar tal dificuldade. O alto consumo de bateria por tais serviços levanta, como um primeiro passo, o desafio de obter essas informações de forma eficiente, sem que o usuário seja prejudicado pela utilização de tais recursos. Este documento tem como objetivo realizar um estudo sobre os meios de obtenção dos dados relacionados à localização física de usuários *Androids*, tendo como foco a utilização mínima dos recursos limitados de dispositivos móveis.

Palavras-chave: android, serviços de localização, lbs, localização em background, bateria

ABSTRACT

After dominating the web navigation data through cookies, and the results achieved by the services that use their user's information to provide superior services, it can be seen that there are another scenery yet to be conquested, the offline world. With the actual number of smartphone users, the location data provided by those devices become an efficient approach to solve this difficulty. The high battery consumption of those services raises, as the first step, the challenge to obtain that information efficiently, without harming the device. This document has the objective to perform a study concerning how to obtain data regarding the physical location of Android's users, focusing on reducing the usage of the device's resources.

Keywords: android, location services, lbs, background location, battery

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

Ao utilizar navegadores, é comum a utilização dos chamados *cookies*, pequenos arquivos que são armazenados durante a navegação do usuário, detectando o caminho que ele percorre entre as diferentes aplicações. Esse rastro permite que cada aplicação que o usuário visite possa prover uma qualidade melhor dos seus serviços, apresentando seu conteúdo e guiando o usuário de uma forma mais eficiente. Um caso de uso seria quando um usuário pesquisa sobre um produto em uma página *web*, através de um navegador, e entra no *site* de uma loja virtual, a página irá dispor em sua tela inicial produtos relacionados ao que ele havia pesquisado anteriormente. Esse processo, no entanto, limita-se ao interesse do ambiente *online*. Com o domínio dos dados de navegação, percebe-se que existe um outro cenário não abordado, o mundo *offline*. Para tal, existem diversos esforços para tentar obter essas informações, seja através de pesquisas de campo, ou de câmeras nos ambientes para detectar a visita de consumidores.

Durante o *Google I/O* de 2017, o maior evento de *Android* do ano, a *Google* anunciou que haviam 2 bilhões de usuários ativos utilizando o sistema operacional [1], e esse volume permite solucionar vários obstáculos que no passado seriam inviáveis. Nesse cenário, uma abordagem começa a ganhar força, utilizando sensores embutidos nos *smartphones* para a obtenção dos dados de localização de seus portadores.

1.1 OBJETIVOS

Este documento realiza um estudo sobre os meios de obtenção dos dados relacionados ao comportamento *offline* do usuário, tendo como foco a utilização mínima dos recursos limitados de dispositivos móveis. O trabalho limita-se à capacidade do sistema operacio-

nal *Android*, devido à maior liberdade de acesso aos seus recursos. Todos os testes serão feitos utilizando *Java*, desconsiderando benefícios adicionais que o desenvolvimento nativo ou utilizando qualquer outra ferramenta possa trazer. O documento aborda possíveis cenários para a obtenção dos dados e tentará chegar ao modo mais eficiente, esclarecendo os benefícios e limitações de cada abordagem.

1.2 ESTRUTURA DO TRABALHO

O trabalho é separado em 5 capítulos, a começar com essa introdução. O segundo capítulo contempla a motivação para este trabalho e os benefícios que essas informações coletadas podem trazer. O terceiro descreve, de forma breve, os componentes de uma aplicação *Android*, tratando dos serviços e sensores interessantes para o projeto. O quarto capítulo aborda as estratégias de coleta junto a iterações para torná-las mais eficientes, com as considerações finais a respeito da qualidade dos dados coletados. No último capítulo encerramos o documento levantando as limitações da solução e os próximos desafios.

CAPÍTULO 2

MOTIVAÇÃO

Entender os hábitos de consumo de uma população é um desafio cobiçado. Seja um banco desejando oferecer crédito para os seus clientes mais confiáveis, ou uma seguradora que gostaria de saber quando um indivíduo está à procura de um novo veículo para lhe oferecer a melhor proposta antes de seus concorrentes. O ambiente digital consegue proporcionar parte dessa visibilidade através das pesquisas que cada usuário realiza, dos sites com os quais ele interage e das compras online realizadas. No entanto, de acordo com a *MarketingLand* [2], a maior parte das compras ainda ocorre no mundo físico.

Informações sobre quantas vezes o indivíduo pratica esportes e que tipo de exercício ele faz, são de difícil obtenção pelos de navegadores. Os dados do mundo *offline* são capazes de determinar o comportamento e interesse de cada pessoa no mundo físico e, a partir dessas informações, fornecer serviços não só focados em vendas, mas que possam melhorar a qualidade de vida de cada pessoa.

A coleta dos dados de forma automática, sem precisar que o usuário esteja ativamente fornecendo essas informações, viabiliza cenários nos quais os serviços ao redor das pessoas podem evoluir para melhor atender cada indivíduo. O atual número de *smartphones*, junto ao poder computacional dos mesmos, fornecem um grande passo para a visão de futuro da *Computação Ubíqua* [3], vislumbrada por Mark Weiser em 1988. Abaixo são listados alguns serviços que podem ser evoluídos com essas novas informações.

- **Avaliação de estabelecimentos comerciais**: Embora existam serviços nos quais o usuário possa avaliar um estabelecimento, como *Yelp* e o *Trip Advisor*, é possível a existência de dados fraudulentos e opiniões que não sejam válidas para aquele indivíduo. Com os dados de navegação dos usuários a esse tipo de estabelecimento, podemos identificar padrões de visitas recorrentes, visitas motivadas por indicações

e também de pessoas relacionadas. Utilizando essas informações é possível gerar uma cenário de avaliações e recomendações muito mais eficaz e com menos necessidade de ações ativas dos usuários. Esses serviços podem ser alimentados com o tipo de estabelecimento que as pessoas costumam visitar, podendo fornecer notas melhores ou piores de acordo com o cada perfil.

- **Mobilidade Urbana**: Os dados de navegação dos usuários podem prover visibilidade sobre as cidades para auxiliar na tomada de decisões como onde construir novas vias ou alteração das frotas de transporte público de acordo com o volume de pessoas.

- **Marketing e publicidade**: Os dados de visita a estabelecimentos fornecem um capacidade de distribuição segmentada de campanhas digitais de acordo com os hábitos de comportamento *offline* de cada usuário. Podendo também, fornecer informações sobre o resultado das campanhas caso o objetivo seja levar o usuário a um ponto de venda. Além disso, é possível ter a visão do público que passa por determinadas regiões em cada horário, viabilizando uma melhor criação de *marketing* utilizando *outdoors*, por exemplo.

2.1 TRABALHOS RECENTES

Denis Huber [4] faz uma descrição das *APIs* de localização tanto de *Android* como *iOS*, fornecendo também valores brutos do consumo de bateria em milliampères no seu trabalho, descrevendo os atributos relacionados à qualidade dos dados provenientes dos serviços de localização das plataformas. Aborda também as formas de obtenção de dados de localização bem como os serviços envolvidos, mencionando outros provedores de localização que podem ser usados além dos serviços da *Google*, como o *Skyhook*, que era utilizado pela própria *Apple* até meados de 2010. O mesmo tema é abordado por *Shaveta Bhatia [5]*, descrevendo a arquitetura de um serviço para obter e exibir a localização em um mapa em tempo real usando a *API* do *Google Maps*. *Yosi Kristian [6]*, exemplifica uma utilização dos serviços de localização para a segurança de aparelhos em caso de perda

ou possível furto, que hoje já é fornecido nativamente pelos próprios serviços da *Google* no *Android*. Isto pode ser consultado inclusive pelo próprio mecanismo de busca do navegador de um computador caso o usuário esteja logado em sua conta do *Google*. Em relação à mensuração do consumo de bateria, o trabalho de *Marcelo Martins [7]*, tem uma proposta interessante de uma arquitetura com um módulo, embarcado dentro do próprio sistema operacional, que reduz o consumo de bateria, alertando inclusive sobre os gastos causados pelos serviços da *Google*. Outro ponto interessante é a mensuração de acesso ao *Wake Lock*, uma *API* do *Android* que fornece recursos de controle ao ciclo de hibernação do processador que, de acordo com o documento, causa aumentos no consumo de bateria.

CAPÍTULO 3

ARQUITETURA E INFRAESTRUTURA ANDROID

Para facilitar a compreensão do estudo a ser realizado, este capítulo tem como objetivo explicar os componentes do *Android* que serão utilizados ou mencionados no próximo capítulo.

3.1 ANDROID

O sistema operacional *Android* foi criado em 2008 [8], seu projeto é *Open Source*, gerenciado pela *Google*, sendo permitido aos fabricantes de dispositivos fazer suas próprias alterações em seus *Androids*, contanto que as modificações atenham-se a um conjunto de diretrizes disponibilizadas pela própria *Google* [9]. Devido à natureza em constante evolução do projeto, temos anualmente uma nova versão do *Android* disponibilizada, em versão beta, no primeiro semestre, sendo a versão final lançada na metade do segundo trimestre, o que gera uma quantidade grande de versões.

O *Android* encontra-se na *API* 25, com a 26 perto de seu lançamento, e devido a este constante fluxo, e às mudanças que cada fabricante pode faz, gera-se uma enorme fragmentação. Hoje, a maioria dos *Androids* não se encontram próximos das versões mais recentes, e não devido à falta de interesse do usuário de atualizar, mas simplesmente porque o aparelho nunca irá receber uma versão mais nova. Essa fragmentação torna-se pior quando as operadoras entram na equação, pois elas também fazem alterações em cima do software do fabricante. O resultado é uma cascata de atualizações, primeiro pela *Google*, cujo resultado precisa ser obtido pelo fabricante, adaptado à sua versão, para só então disponibilizar para seu aparelho e, posteriormente, essa atualização precisa ser obtida pela operadora, que irá repetir o processo, para só então estar disponível para o

usuário final. Além disso, ainda existe uma quantidade muito grande de aparelhos que são lançados no mercado, a um baixo custo, que acabam recebendo nenhuma atenção nesse quesito.

Na tabela 3.1 é possível visualizar a atual distribuição de versões de *Android* coletadas pela *Google* durante uma semana, em junho, referente ao mercado global. Ao lado, a distribuição coletada durante os dias 23 à 25 de junho de uma base brasileira, fornecida pela empresa *In Loco Media* [10]. Esses números evidenciam a forte fragmentação que ocorre nesse sistema operacional.

Versão	Nome da Versão	API	Distribuição Google	Distribuição Brasil
2.3.3 - 2.3.7	Gingerbread	10	0.8%	N/A
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.8%	2.32%
4.1.x	Jelly Bean	16	3.1%	4.38%
4.2.x		17	4.4%	2.70%
4.3		18	1.3%	1.16%
4.4	KitKat	19	18.1%	12.58%
5.0	Lollipop	21	8.2%	7.07%
5.1		22	22.6%	17.10%
6.0	Marshmallow	23	31.2%	44.77%
7.0	Nougat	24	8.9%	9.02%
7.1		25	0.6%	0.18%

Tabela 3.1 Distribuição de aparelhos *android* por versão.

3.2 COMPONENTES BÁSICOS

Uma aplicação *Android* possui quatro componentes básicos: *Activity*, *Service*, *Broadcast Receiver* e *Content Provider* [11]. Para fazer a declaração dos três primeiros, é necessário declará-los no *Android Manifest*, um arquivo de configuração que define todos os componentes que o sistema estará usando. No caso de *Broadcast Receivers*, não é necessário declará-los no documento, mas isso inviabiliza que o sistema operacional consiga se comunicar com esse componente, tornando-o restrito para a aplicação em que ele foi criado.

3.2.1 Activity

Representa uma tela, sendo o ponto de entrada inicial da interação de um usuário com um aplicativo. Uma *Activity* deve ser independente do resto da aplicação, podendo ser chamada por outros componentes para abrir uma tela nova.

3.2.2 Service

Um componente que permite rodar uma operação sem estar associado à uma tela e que pode ser acionado por diferentes telas ou até outros serviços. Uma implementação comum seria um serviço que tocasse uma lista de músicas definida pelo usuário, e essa operação não estaria presa à telas e, caso algum componente precisasse interagir com o serviço para interrompê-lo, teria essa capacidade. Um serviço não precisa estar associado à tela do usuário, podendo executar processamentos que eventualmente alguma outra tela vai consultar. Uma utilização frequente é quando um aplicativo lança um novo serviço para atualizar o seu banco de dados, para que no momento em que o usuário for abrir a primeira tela, já esteja com o conteúdo mais recente sem a necessidade de esperar o carregamento. Esse componente será o principal utilizado nas estratégias de coleta, visto que ele viabiliza operações fora do ciclo de interação do usuário.

A principal utilidade de um *Service* é informar ao sistema operacional que existe uma operação com certo grau de prioridade que está em execução, dando uma maior

flexibilidade ao sistema operacional para lidar com esse componente. Um *Service* pode ser criado com uma entre várias configurações de comportamento para quando o sistema operacional precisar removê-lo da memória. Para simplificar o fluxo, iremos trabalhar apenas com o *Sticky* e com o *Not Sticky*, sendo este o comportamento que solicita ao sistema operacional eventualmente reativar o serviço caso ele seja removido e, o segundo, um comportamento mais simples onde ele aceita ser removido sem solicitar um fluxo adicional.

3.2.3 Broadcast Receiver

É um componente que permite a uma aplicação receber mensagens do sistema operacional ou de outras aplicações. Uma utilização comum é a aplicação de um despertador, que necessita ser acordado em determinado momento para fazer sua notificação sonora. Para isso, a aplicação registra seu *Broadcast Receiver* e faz essa solicitação para que o sistema notifique esse componente quando chegar à hora desejada. Outro cenário ocorre quando a bateria do aparelho entra em estado crítico e, nesse momento, o sistema operacional envia uma mensagem que pode ser recebida por qualquer *Broadcast Receiver* que esteja registrado para receber esse evento.

3.2.4 Content Provider

Componente que gerencia o acesso em conjunto a determinados arquivos no sistema. Um exemplo de *Content Provider* é o acesso compartilhado à lista de contatos salvos em um aparelho, que qualquer aplicativo com as permissões necessárias tem acesso de leitura e escrita sem se ater a cenários de concorrência de serviços para esse recurso.

3.3 CICLO DE VIDA DE UMA APLICAÇÃO

Uma aplicação *Android* roda em seu próprio, e único a priori, processo [12] que é criado quando um dos componentes de uma aplicação é requisitado e ocupa seu espaço na

memória do *Android* até o momento que o sistema operacional esteja precisando de recursos e esse processo esteja em baixo grau de prioridade. A aplicação também poderá ser removida manualmente pelo usuário ou forçadamente no caso de um erro não tratado.

Para determinar que processo está em baixa prioridade e precisa ser removido, existem quatro categorias descritas abaixo em ordem de prioridade.

- ***Foreground Process***: É qualificada dessa forma quando a aplicação está rodando em alguma tela visível para o usuário. Os processos nessa categoria são os últimos na fila de remoção e, caso ocorra, a aplicação encontra-se em um estado crítico e o sistema operacional não consegue manter a tela responsiva.

- ***Visible Process***: Geralmente ocorre quando uma *Activity* está pausada mas não parada. É o caso no momento em que um diálogo é exibido na tela do usuário, mas não preenche a tela toda e, dessa forma, o usuário ainda está ciente da existência daquela *Activity*. Ocorre também quando um *Service* está exibindo algo diretamente para o usuário através de uma notificação.

- ***Service Process***: É o caso de um *Service* comum que está realizando uma operação, mas não está exibindo nada na tela. Nessa categoria, serviços que estão rodando durante um tempo maior vão perdendo prioridade para os novos. No caso de *Androids* com recursos extremamente limitados, esse tipo de serviço dificilmente consegue ser executado por um tempo satisfatório sem ser interrompido.

- ***Cached Process***: Será o caso para qualquer *thread* lançada por uma aplicação que não esteja presa a um dos componentes em uma das categorias acima.

3.4 SENSORES

O *Android* possui um conjunto de sensores para auxiliar a detecção de informações a respeito da situação do aparelho e do ambiente no qual ele está imerso. Os sensores são categorizados em três tipos:

- **Sensores de movimentação**: Quantificar a aceleração e rotação do aparelho.

- **Sensores de ambiente**: Fornecer informações a respeito da temperatura, pressão, iluminação e umidade.
- **Sensores de posicionamento**: Auxiliar na detecção da posição do aparelho em relação a um referencial.

Dos tipos listados acima, apenas os sensores abaixo são úteis para auxiliar na localização do aparelho.

3.4.1 Acelerômetro

Detecta a aceleração do aparelho, em metros por segundo ao quadrado, que é aplicada em cada um dos três eixos do dispositivo. Esse sensor pode ser utilizado para detectar se o aparelho está sendo balançado.

3.4.2 Giroscópio

Mensura a rotação do aparelho, em radianos por segundo, em relação aos três eixos. Possui utilização similar ao acelerômetro, podendo também auxiliar para determinar se o aparelho está na mão ou no bolso do seu usuário.

3.4.3 Campo Magnético

Mensura o ambiente magnético do aparelho nos três eixos. Necessário para a criação de uma bússola, também auxilia no posicionamento do aparelho caso o ambiente possua um comportamento característico.

3.5 GERENCIADOR DE ALARMES

Aplicações *Android* podem ser solicitadas para acordar em um momento futuro. Para isso, utilizamos a *API* chamada *Alarm Manager* [13] que permite o agendamento de alertas para momentos no futuro. Para fins de simplificação, não estaremos utilizando

a nova *API* de Alarmes, chamada *Job Scheduler*. Embora esta seja mais eficiente, ela só está disponível a partir versão 21 do *Android*, o que limitaria muito nosso espaço de trabalho em cerca de 25%, dado as estatísticas de fragmentação exibidas na seção 3.1.

3.6 SERVIÇOS PROVEDORES DE LOCALIZAÇÃO (LBS)

O *Android* possui um conjunto de sensores cuja plataforma dá suporte em seu código original base, o qual pode ser sobrescrito de acordo com o fabricante daquele hardware. Esse tipo de influência não foi considerado na criação desse documento. Para um aparelho poder ser classificado como *Android*, não é necessário possuir nenhum desses sensores, mas é bastante recomendado, visto que diversas das aplicações com alto grau de engajamento utilizam esses sensores. A ausência deles, no entanto, é levada em consideração na análise da capacidade de escalabilidade de cada uma das propostas de coleta de dados.

3.6.1 Satélites

O Sistema de Posicionamento Global [14] usa uma constelação de 24 satélites em órbitas do planeta. A tecnologia funciona calculando a diferença de tempo entre o envio e o recebimento de um sinal, fazendo a triangulação entre vários satélites para determinar a posição do aparelho. Dependendo do aparelho e do fabricante, a utilização da constelação de satélites a ser utilizada pode variar entre o *GPS* tradicional que pertence aos Estados Unidos, o *GLONASS* russo, entre outros. Qualquer subsequente referência a *GPS* neste documento estará desconsiderando o tipo específico de constelação de satélites usado, referindo-se apenas ao provedor de coordenadas geo espaciais que o aparelho está utilizando.

3.6.2 Rede móvel de Telefonia

Utiliza as antenas de transmissão do sinal de telefonia para ter localização aproximada do aparelho. Uma torre de transmissão pode ter até à 20 quilômetros de alcance [14],

dificultando uma boa precisão. Em locais com várias antenas pode-se utilizar os sinais para obter uma precisão melhor, mas ainda é significantemente inferior a localização obtida por *GPS*. Por outro lado, não é necessário *hardware* adicional no *smartphone* para que a localização seja obtida.

3.6.3 Wi-Fi

O termo *Wi-Fi* será utilizado para remeter-se à tecnologia para a criação de redes de acesso local criadas utilizando o padrão *IEEE 802.11*. Embora o serviço seja utilizado para a criação de redes locais, esses dispositivos emitem sinais que podem ser captados pelos aparelhos. Sabendo-se que o aparelho consegue visualizar aquela rede e a potência do sinal que está chegando pelo aparelho, é possível utilizar a qualidade dos sinais para prover uma localização aproximada. Vale ressaltar que em ambientes urbanos a quantidade de sinais de *Wi-Fi* coletados viabiliza uma localização de alta precisão, superando o *GPS* básico de um *Android* em ambientes fechados.

3.6.4 Bluetooth

Padrão para envio de dados de baixa frequência. Por possuir um alcance menor que o *Wi-Fi* comum, pode ser utilizado para localizar da mesma forma com uma precisão maior.

3.7 CATEGORIAS DOS LBS

O Android agrupa os tipos de localização em dois grupos de provedores principais.

3.7.1 GPS Provider

Provedor responsável por fornecer os dados de *GPS* de alta precisão, utilizando os satélites descritos anteriormente. Necessita de um tempo maior para obter a primeira posição devido à necessidade de sincronização dos dados dos satélites. Por usar *hardware* específico,

não depende de conectividade, funcionando bem em lugares abertos como parques e para navegação em ambientes abertos. Até a *API* 23, qualquer utilização do provedor *GPS* faz com que o aplicativo seja classificado como 'alto consumo de bateria' na listagem de aplicativos que utilizam localização.

3.7.2 Network Provider

Provedor que utiliza os serviços das redes próximas, também conhecido como *GPS* assistido. Consome os dados de telefonia, *Wi-Fi* e *Bluetooth* para converter em uma geolocalização. Esse provedor consegue obter a posição mais rapidamente que o *GPS Provider*, porém necessita de conexão para fazer a tradução das redes para as coordenadas. Caso o aparelho já tenha feito essa tradução em um momento recente, é possível localizar mesmo sem acesso à internet, pois parte do dicionário de tradução é baixado localmente para o aparelho. Possui uma precisão melhor que o outro provedor em locais urbanos com alta densidade de redes *Wi-Fi*, mas apresenta um resultado inferior em ambientes rurais e com poucas redes.

Além disso, uma implementação básica de um *Android* não possui um serviço para alimentar esse provedor. No caso dos *Androids* fornecidos pela *Google*, esse serviço é provido pela mesma. Esse intermediário se faz necessário para poder prover o serviço, por exemplo, de tradução de roteadores *Wi-Fi* em coordenadas geoespaciais.

3.8 IMPLEMENTAÇÕES DE SERVIÇOS DE LOCALIZAÇÃO

O *Android* permite a implementação de Serviços de Localização [15] por cima da *API* padrão, mas para esse cenário, iremos descrever apenas as soluções suportadas pelo *Android* comum por motivos de escalabilidade.

A *API* chamada *Location Manager* é a interface básica para obtenção de dados de localização, que fornece três tipos de provedores: o *GPS*, *Network*, e *Passive*. Este último não provê dados novos em relação aos outros dois, mas permite que uma aplicação escute os dados de *GPS* e rede passivamente, utilizando a solicitação que outras aplicações

possam ter feito. Além disso, o *Location Manager* permite que uma aplicação inscreva-se para receber alertas de proximidade no momento em que o usuário entre ou saia de determinadas regiões, delimitadas por latitude, longitude e raio. Isso viabiliza que aplicações repassem a responsabilidade de detecções para o sistema operacional, reduzindo a quantidade de recursos que a aplicação precisará consumir.

3.9 PERMISSÕES

Uma aplicação *Android* pode utilizar um conjunto de funcionalidades providas pelo sistema operacional, no entanto, para tal é necessário que cada permissão esteja declarada no *Android Manifest*. Para esse projeto, as permissões abaixo são as mais relevantes:

- ***Internet***: Acesso à internet
- ***Access Network State***: Acesso à internet e conectividade.
- ***Access Wifi State***: Acesso ao estado da conexão *Wi-Fi*.
- ***Change Wifi State***: Permite alterar o estado do serviço de *Wi-Fi*, possibilitando realizar uma varredura dos dados de Wi-Fi nas proximidades.
- ***Access Coarse Location***: Acesso aos dados de localização utilizando o *GPS* assistido. Necessário para acessar os dados de *Wi-Fi* dos roteadores próximos a partir do *Android* M.
- ***Access Fine Location***: Acesso aos dados de localização fornecidos pelo *GPS* e pelo *GPS* assistido.
- ***Wake Lock***: Acesso ao controlador do processador para impedir que ele durma quando a aplicação precisar realizar uma operação enquanto a tela estiver desligada. Sem essa permissão, a frequência de acesso aos dados dos sensores são reduzidos severamente.

3.10 ANÁLISE DE PERFORMANCE

Para mensurar cada cenário, será utilizado o *Battery Historian* [16], que permite visualizar o consumo estimado de cada aplicação, bem como o comportamento de requisições de *GPS*, *Wi-Fi*, solicitações de *Wake Lock*, entre outras informações. Possui limitações para diferenciar o consumo de bateria provocado por vários aplicativos utilizando um mesmo recurso, como vários aplicativos solicitando o *GPS* concorrentemente, mas será o suficiente para analisar o consumo de bateria.

CAPÍTULO 4

ESTRATÉGIAS

4.1 ESTRATÉGIAS

Este capítulo descreve as estratégias para obtenção dos dados de localização. Será descrito como o experimento foi realizado em cada iteração que a solução precisou receber. Como levantado no capítulo anterior, para o caso de uso padrão, o *Android* não consegue prover garantias para que serviços em baixo grau de prioridade não seja removido. Por isso, é importante ressaltar que as soluções propostas não conseguirão garantir uma cobertura de cem por cento de coleta. O objetivo das estratégias é obter a maior quantidade de dados úteis com o menor custo de bateria. Para uma solução que precise dos dados, sem tolerância à perda de informações, nenhuma das propostas abaixo será satisfatória.

Todos os cenários enviam seus dados para um serviço *web* para validar a quantidade de dados enviados, mas o conteúdo dos mesmos não foi mantido, com objetivo de quantificar o registro de recebimento. Os testes foram executados em um conjunto reduzido de aparelhos, sendo eles:

- **Nexus 5**
- **Nexus 6P**
- **Moto E (2 Geração)**
- **Moto X (1 Geração)**
- **Moto Maxx**

4.2 PROPOSTA INICIAL

Havia um interesse em utilizar os dados dos sensores mencionados na seção 3.4 para mensurar a movimentação dos aparelhos e auxiliar na detecção de paradas. Tal abordagem foi abandonada numa etapa inicial de estudos a respeito das capacidades dos serviços, pois a permissão de *Wake Lock*, necessária para coletar os dados dos sensores na frequência máxima em background, apresentou capacidade reduzida a partir da *API* 23 do *Android* devido aos estados *Doze* [17] e *Stand By* que foram introduzidos.

4.2.1 Cenário 1

A primeira abordagem utilizará um *Service*, em *background*, que registra um *Location Manager*, para o qual a aplicação solicitará atualizações de localização com intervalos regulares para os provedores *Fine* e *Coarse*. Também irá registrar o provedor passivo para receber os dados que outras aplicações podem ter solicitado. O *Service* será *Sticky*, o que fará com que ele seja revivido caso o sistema operacional precise de recursos.

4.2.2 Cenário 2

Essa abordagem utilizará um *Alarm Manager* para registrar alarmes de intervalos periódicos, de 30 minutos, para acordar um *Broadcast Receiver*. Este, por sua vez, irá solicitar ao *Location Manager* uma nova localização e, ao recebê-la, irá enviar para o serviço *web* e encerrar o processo.

4.2.3 Cenário 3

Esse cenário será um pouco diferente dos anteriores, que tentam coletar os dados em todos os locais. Nesse caso, apenas coletará dados de locais específicos, definidos inicialmente dentro da aplicação. Para isso estaremos usando o serviço de alertas de proximidade do *Location Manager*, para alertar-nos quando o usuário entrar em um local desejado. Foi usado um *Broadcast Receiver* para receber os eventos de alerta, e nesse momento a

localização foi recebida e enviada para o serviço externo.

Dado que existe uma limitação de 100 regiões a serem monitoradas, uma lógica adicional foi criada com zonas de 5 quilômetros ao redor da posição do usuário e todas as regiões dentro desse círculo são adicionadas. Quando o usuário sair da região, uma nova área será criada com todas as regiões dentro do novo espaço. Serão registradas as 100 regiões mais próximas dentro de um raio de 5 quilômetros e, caso as regiões se encontrarem todas em um raio menor que 5 quilômetros, iremos reduzir até o mínimo possível.

4.3 PRIMEIRA ITERAÇÃO

Para esse primeiro ciclo, percebemos que o consumo de bateria estava bastante elevado devido às requisições externas utilizando a conectividade. Um dos principais motivos era que, durante o primeiro cenário, o recebimento de novas posições de localização passivas fazia com que a internet fosse acessada frequentemente, mantendo o *hardware* responsável por esse acesso no estado ativo, não permitindo que ele entrasse em estado de hibernação. Foi verificado também que a precisão do *GPS* é bastante instável em ambientes fechados, com acurácia média mantendo-se em 100 metros dependendo da residência em questão. Um maior obstáculo foram as oscilações periódicas da precisão das amostras, que incluíam picos frequentes de 1 a 2 quilômetros que prejudicaram o terceiro cenário.

Um problema detectado ocorreu quando o usuário não possuía localização disponível e a aplicação era acordada desnecessariamente, tanto nos cenários um e dois.

4.3.1 Cenário 1

Esse cenário foi o que se apresentou melhor no quesito volume de dados coletados. Entretanto, a quantidade de localização recebida junto ao envio constante causou um gasto de bateria extremamente alto ultrapassando 7% da bateria do aparelho naquele dia. Esse cenário obteve a maior quantidade de dados passivos, com picos quando o usuário utilizou aplicativos de navegação, como o *Google Maps*.

4.3.2 Cenário 2

Esse modo mostrou-se bem instável para aparelhos de baixo poder computacional. Como a requisição de localização necessitava ser executada em uma *thread* à parte para receber o retorno da consulta de localização, a aplicação encontrava-se na quarta categoria de prioridade, pois não havia componentes que davam prioridade para o nosso serviço.

4.3.3 Cenário 3

Esse cenário apresentou-se bastante inconsistente, necessitando de um raio mínimo de 300 metros, para ter uma qualidade satisfatória de coleta, com vários eventos de entrada e saída sendo perdidos. Por outro lado, o consumo de bateria do aplicativo para o melhor caso ficou em números bastante satisfatórios, abaixo de meio por cento. Entretanto, outros valores do sistema operacional pareceram subir, levando a conclusão que o consumo de bateria não estava sendo atribuído ao aplicativo que fez a solicitação. Embora isso seja útil para esconder o consumo real da aplicação, não é uma estratégia apropriada para a proposta.

Pela noite, a imprecisão de *GPS* causou um excesso de entradas e saídas incorretas e como nesse momento o usuário não está usando o aparelho, a porcentagem total de bateria consumida, se comparada com os outros aplicativos que estavam inativos, ficou bastante aparente, exibindo acima de 5% em vários cenários. Essa abordagem possui um número significantemente menor de dados redundantes, porém ao se locomover no raio de 300 metros, nenhum dado foi coletado, tornando essas perdas relevantes.

4.3.4 Considerações Parciais

Para resolver o problema de comunicação, todos os dados passarão a ser armazenados localmente e só enviados uma vez por dia, no período da noite. Caso o volume de dados for grande, acima de 50 kb, serão enviados apenas pela *Wi-Fi*, para não gerar custos adicionais para a banda móvel.

Uma análise na origem dos dados mostrou que a localização utilizada era proveniente

do *Network Provider*, que por fornecer uma localização mais rápida, acabava sendo utilizada e a aplicação não esperava o provedor de *GPS*. Com isso estaremos removendo a utilização do provedor de *GPS*, mantendo seu acesso apenas passivamente, coletando a última posição disponível por esse provedor, em vista que ele também é considerado mais caro a nível de bateria.

Em relação ao problema do aparelho estar com localização indisponível, será adicionada uma lógica de alarmes com crescimento de intervalo em caso de falha. No momento em que a aplicação for acordada ao disparar um alarme, um novo registro no *Alarm Manager* será realizado, com um intervalo de tempo maior que o anterior. Nesse momento, um outro registro será persistido indicando que é uma segunda tentativa. Ao concluir a operação, o alarme será cancelado e um novo alerta, indicando que é a primeira tentativa, será registrado. Dessa forma, se a operação não concluir, o próximo alarme já estará pronto para lidar com isso. O alarme é registrado antes de confirmar a falha da solicitação, para que seja possível lidar não apenas com perdas inerentes ao recurso, como em ausência da permissão de localização, como por qualquer encerramento prematuro que o processo possa sofrer. O primeiro cenário também receberá uma alteração adicional no qual irá, ao receber um evento de localização passivo, adiar o próximo alarme para tentar aproveitar melhor o volume que ele coleta por estar aberto constantemente.

Para mitigar as perdas do segundo cenário, será alterado o componente utilizado. Ao invés de um *Broadcast Receiver*, será registrado um *Service* que não utilizará a configuração *Sticky*, fazendo-o ficar ativo durante o momento que a localização seja coletada, sendo encerrado após a conclusão do fluxo.

4.4 SEGUNDA ITERAÇÃO

As alterações de internet e de não utilizar o provedor que utiliza satélites apresentaram resultados consideráveis na redução de consumo de bateria. Por não necessitar esperar o recebimento desse provedor, o qual atinge até um minuto de latência, as sessões de coleta ficaram reduzidas, favorecendo bastante o segundo cenário, no qual o tempo de *CPU* utilizado pela aplicação tornava-se o melhor dos três casos. Em relação à qualidade dos

dados, o terceiro cenário foi eficiente para pegar dados de navegação, pois as zonas que delimitavam as regiões inseridas eram ultrapassadas durante a navegação em automóveis, fazendo com que a rota fosse facilmente deduzida. No cenário 2, em traslados de alta velocidade, conseguiu-se detectar apenas o ponto de partida e o de chegada.

4.4.1 Cenário 1

Os resultados foram satisfatórios para esse cenário, com consumo de bateria perto de 1% nos cenários que a aplicação precisou realizar as coletas, e abaixo de 0.5% quando conseguiu aproveitar-se dos dados passivamente.

4.4.2 Cenário 2

A estratégia de usar um *Service* reduziu o número de remoções para os aparelhos medianos. No caso do *Moto E*, considerado *low-end*, o resultado ainda acarretou diversas perdas. A única opção capaz de resolver essa perda seria colocando uma notificação na tela, indicando que a operação estava ocorrendo, dessa forma subindo a prioridade do nosso processo para a segunda categoria. Essa abordagem envolve notificar a interface do usuário, portanto, não sendo uma solução completamente em *background*, tornando-a inviável para esse teste.

4.4.3 Cenário 3

Esse cenário como uma solução independente não apresentou resultados satisfatórios para a coleta comparado aos outros dois, pois o serviço constantemente perdeu eventos de transição, tanto ao entrar em uma região, como ao sair dela. Além disso, como utilizamos uma região de controle para delimitar quais regiões seriam monitoradas, houve casos em que a aplicação foi encerrada prematuramente pelo sistema operacional no momento de transição. Essa falha resultou no não cadastramento das novas regiões, inviabilizando o monitoramento no aparelho.

4.4.4 Considerações Parciais

Dados os resultados dos cenários anteriores, será proposto um novo cenário, 4, que será a combinação dos cenários 2 e 3. O modelo usará um *Service* para receber todos os seus eventos. O fluxo começará quando o usuário abrir a aplicação e o serviço entrar no estado estacionário. Nessa configuração iniciaremos um monitoramento de 500 metros dentro do qual faremos coletas periódicas similares ao cenário 2. A coleta terá um intervalo que aumentará caso o usuário não saia da região monitorada ou caso a localização não possa ser obtida. Quando o usuário sair dessa região delimitada, ele entrará no estado em movimento, criando uma nova região de monitoramento. Para determinar o novo raio, iremos verificar a velocidade do usuário, que será determinada de acordo com a distância em relação à última coleta com um teto de 2 quilômetros, para evitar que a aplicação acorde muito recorrentemente. Quando o alarme periódico for ativado, caso o raio da última região seja acima de 500 metros, iremos agendar um outro alarme, com intervalo menor, para garantir que o usuário tenha parado de se movimentar. Quando esse alarme for disparado, caso o usuário ainda esteja na região, um monitoramento padrão de 500 metros será criado, e o serviço entrará no estado estacionário.

Será criado também um outro alarme, de recuperação. No momento que a aplicação iniciar qualquer operação em baixa prioridade, será ativado um alarme pequeno, do tempo esperado para executar a operação, o qual será responsável por recuperar a processo caso ele seja terminada prematuramente. Esse alarme será responsável por recuperar a região monitorada, caso durante uma transição o processo seja fechado e também quando o alarme de coleta não conseguir concluir sua operação.

Com esse alarme adicional será possível manter a aplicação em modo ocioso durante um minuto quando o alarme de coleta for disparado, aguardando uma posição de localização passiva do sistema. Para impedir riscos que possam ocorrer com vários alarmes caso a aplicação esteja em um momento crítico, será adicionado um limite no número de alarmes em um intervalo de tempo. Essa última alteração será feita para reduzir os impactos de cenários de alta imprecisão ou em estados críticos de recursos. Esse cenário foi pensado para resolver os obstáculos abaixo:

- **Coleta em movimento**: As zonas de monitoramento vão garantir que teremos uma quantidade razoável de pontos de localização mesmo com o usuário em movimento.

- **Coleta em raios pequenos**: Como não é possível ter zonas com raios pequenos, a coleta periódica ainda é necessária.

- **Imprecisão de localização**: Com o limite de 2 quilômetros não teremos o risco que oscilações grandes de *GPS* façam com que a aplicação crie um raio muito grande acima do fluxo diário de movimentação do usuário.

- **Processo sendo removido prematuramente**: O alarme de recuperação foca em tratar esse cenários.

- **Coleta redundante em casa**: O crescimento dos alarmes quando o usuário encontra-se parado resolve esse problema sem perder em performance com a utilização das regiões de monitoramento.

O modelo de alarmes de recuperação serão implementados tanto para o cenário 1 como para o novo cenário 4.

4.5 TERCEIRA ITERAÇÃO

Essa iteração apresentou o resultado mais satisfatório, com duas abordagens que são capazes de obter os dados necessários.

4.5.1 Cenário 1

Esse cenário não apresentou nenhum problema persistente com a coleta de dados se mantendo de 0.5% a 1% por dia de uso.

4.5.2 Cenário 4

Esse cenário também conseguiu se manter perto de 1% de consumo de bateria por dia, possuindo dados satisfatórios de navegação e de movimentação em regiões próximas.

4.5.3 Considerações finais

Pelos números absolutos, o cenário 1 seria a melhor abordagem. Entretanto, ter uma aplicação aberta o tempo todo não é ideal, pois se vários aplicativos usarem a mesma estratégia, o número de aplicações na memória a cada momento será muito alta, o que pode causar uma escassez de recursos, fazendo com que todas as aplicações sejam removidas. Além disso, existe um risco de rejeição por parte de usuários que saibam verificar esse comportamento, podendo gerar críticas ao aplicativo. Esse cenário seria o mais eficiente caso fosse possível ter uma aplicação embarcada no dispositivo, onde outras aplicações que precisassem desse recurso pudessem apenas solicitar as notificações dos eventos.

A solução factível para um desenvolvedor sem acesso a aplicativos embarcados seria o último cenário abordado, possuindo um bom custo benefício, com medições em torno de 1% de bateria por dia. O principal ponto negativo dessa solução, ainda não resolvido, foi a perda de eventos de transição, devido às instabilidades providas pelo serviço de monitoramento. No entanto, esse é um problema que espera-se ser resolvido com a evolução das versões do próprio serviço de localização. Adicionalmente, essa solução é mais complexa de ser implementada devido ao número de cenários alternativos que podem ocorrer nas diversas versões do *Android*.

A estratégia de utilizar sensores para detectar a parada e evitar requisições redundantes seria ideal caso o aparelho possuísse um *hardware* de baixa energia que processasse os dados dos sensores para indicar paradas, cabendo à aplicação apenas consultar essa informação. É possível que essa solução seja disponível no futuro, pois já existem *hardwares* específicos, por exemplo, para o monitoramento de regiões. O maior obstáculo é a difusão de tais equipamentos, visto que os únicos aparelhos que estão sendo lançados com esse tipo de recurso são os disponibilizados pela *Google*, a antiga linha do *Nexus* e atual

Pixel, o que reduz a eficácia, pois o foco dessa abordagem é tentar obter dados do maior número de aparelhos possível.

CAPÍTULO 5

CONCLUSÃO E TRABALHOS FUTUROS

Dados de localização se tornarão cada vez mais valiosos, seja para abordar desafios da vida urbana ou para que grandes indústrias consigam tomar decisões mais eficientes. Neste trabalho foi possível mostrar que é factível coletar dados de localização sem um peso muito grande para a bateria de *smartphones* e considerando que existe o interesse da própria *Google* em prover esses serviços com *hardwares* especializados, podemos ter uma expectativa de que no futuro será possível reduzir esse consumo ainda mais. Com os resultados obtidos nesses experimentos, existem três desafios:

O primeiro seria para integrar na solução dados de *Wi-Fi* e *Bluetooth* diretamente, sem depender dos serviços de *A-GPS* providos pela *Google*, com o objetivo de conseguir reduzir as imprecisões causadas pelo *GPS*. As localizações obtidas com tais serviços têm um potencial de precisão maior para ambientes fechados, como mostrado no trabalho de U. Shala [18]. Não seria recomendado, para esse propósito, utilizar Campo Magnético [19] pois essa solução necessita mensurar os sensores por um período maior até que o posicionamento seja determinado e o objetivo deste trabalho busca reduzir ao mínimo o tempo em que o processador é utilizado.

O segundo seria a categorização dos dados recebidos. Para tal, será necessário obter uma base de estabelecimentos e uma lógica de classificação para atribuir cada coordenada geo espacial ao seu respectivo estabelecimento, tendo como grande dificuldade a precisão das coordenadas em áreas como Shoppings e Aeroportos, cuja concentração de estabelecimentos internos é grande.

E o terceiro, é em relação a privacidade do usuário. Durante todo o documento, tratamos de garantir a coleta das informações que possam levar exatamente ao usuário em questão. Nos últimos anos, vários vazamentos de informações ocorreram [20] com graves

consequências, e como informações de localização são críticas, o desafio será conseguir manter os dados de forma que eles consigam agregar valor para as aplicações de forma que o usuário esteja o mais protegido possível.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] "Google has 2 billion users on android, 500m on google photos — techcrunch," https://techcrunch.com/2017/05/17/google-has-2-billion-users-on-android-500m-on-google-photos, (Accessed on 2017-07-07).

[2] "Report: E-commerce accounted for 11.7% of total retail sales in 2016, up 15.6% over 2015," http://marketingland.com/report-e-commerce-accounted-11-7-total-retail-sales-2016-15-6-2015-207088, (Accessed on 2017-07-07).

[3] R. D. B. Gozick, K. P. Subbu and T. Maeshiro, *Selectively Taming Background Android Apps to Improve Battery Lifetime*, 2015, (Accessed on 2017-07-07).

[4] "Background positioning for mobile devices - android vs. iphone," *School of Electrical and Computer Engineering Technical University Berlin*, 2012.

[5] "A new approach for location based tracking," *IJCSI International Journal of Computer Science Issues*, vol. 10, no. 1, 2013.

[6] "Utilizing gps and sms for tracking and security lock application on android based phone," *Procedia - Social and Behavioral Sciences*, no. 57, pp. 299–305, 2012.

[7] R. F. Marcelo Martins, Justin Cappos, "Selectively taming background android apps to improve battery lifetime," *USENIX Annual Technical Conference*, 2015, (Accessed on 2017-07-07).

[8] "Announcing the android 1.0 sdk, release 1," https://android-developers.googleblog.com/2008/09/announcing-android-10-sdk-release-1.html, accessed: 2017-07-07.

[9] "Android compatibility definition document," https://source.android.com/compatibility/cdd, accessed: 2017-07-07.

[10] "In loco media - a maior rede de anúncios mobile do brasil," https://www.inlocomedia.com, (Accessed on 2017-07-07).

[11] "Application fundamentals," https://developer.android.com/guide/components/fundamentals.html, accessed: 2017-07-07.

[12] "Processes and application life cycle," https://developer.android.com/guide/topics/processes/process-lifecycle.html, accessed: 2017-07-07.

[13] "Alarm manager - android developers," https://developer.android.com/reference/android/app/AlarmManager.html, (Accessed on 2017-07-07).

[14] "Implementation of location based services in android using gps and web services," *IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 1, No 2*, vol. 9, no. 2, 2012.

[15] "Location manager," https://developer.android.com/reference/android/location/LocationManager.html, accessed: 2017-07-07.

[16] "Batterystats and battery historian walkthrough - android studio," https://developer.android.com/studio/profile/battery-historian.html, (Accessed on 2017-07-07).

[17] "Optimizing for doze and app standby — android developers," https://developer.android.com/training/monitoring-device-state/doze-standby.html, (Accessed on 2017-07-07).

[18] U. Shala and A. Rodriguez, "Indoor positioning using sensor-fusion in android devices," *MSc Thesis School of Health and Society Department Computer Science Embedded Systems*, 2011.

[19] R. D. B. Gozick, K. P. Subbu and T. Maeshiro, "Magnetic maps for indoor navigation," *IEEE Transactions on instrumentation and measurement*, 2011.

[20] "The biggest cybersecurity disasters of 2017 so far — wired," https://www.wired.com/story/2017-biggest-hacks-so-far, (Accessed on 2017-07-07).