Serveis permanents en Android

Pol Magre Moya

Resum—Aquest document conté informació sobre l'actualitat de la programació background en Android. Actualment, es troben limitacions a l'hora d'executar tasques en segon pla imposades pel sistema operatiu o pels propis fabricants de dispositius, per a que no consumeixin ràpidament els recursos limitats com ara bateria o memòria. Aquestes restriccions han provocat que programar serveis en background no sigui una opció viable quan es vulgui executar tasques de llarga duració en segon pla. Aquest estudi es basa en entendre les eines que hi ha actualment per programar en background en Android i aconseguir la màxima durabilitat temporal d'aquestes sense ser aturades pel sistema. En aquest treball s'han programat múltiples aplicacions amb diferents eines per treballar en background i s'han obtingut resultats per comprovar si funcionen amb la periodicitat i estabilitat previstes.

Abstract— This document contains information about current Android background programming. Nowadays, there are limitations to run background tasks imposed by the operating system or by the device manufacturers themselves, to avoid a quickly consume of limited resources such as battery or memory. These restrictions have made background services not a viable option when wanting to schedule long-running tasks in the background. This study is based on understanding the currently available tools for background programming on Android and achieving maximum temporal durability without being stopped by the system. In this study, multiple applications have been programmed with different tools for working in the background and results have been obtained to check if they work with the expected periodicity and stability.

1 Introducció

Aquest treball neix a causa de la necessitat de programar en Android tasques que s'executin en background de forma permanent, és a dir, que funcionin sense ser aturades pel sistema mentre l'usuari no utilitza directament l'aplicació o aquesta està apagada. Per exemple, ens pot interessar comunicar-nos amb un servidor, llegir la localització mentre no s'utilitzi l'aplicació o, enviar una notificació a l'usuari quan es compleixin determinades condicions.

En les version antigues d'Android, els serveis eren la millor manera de fer aplicacions permanents en background. Però Android està en constant evolució, degut a la millora de les prestacions de les noves generacions de telèfons mòbils i l'oferiment de millors serveis, provo-cant classes obsoletes, noves restriccions que afecten a la programació background, limitacions depenent dels fabricants de dispositius mòbils, etc.

Els serveis són components d'aplicacions que permeten executar tasques mentre l'usuari no està utilitzant l'aplicació i, per tant, no proporcionen una interficie d'usuari. La diferència amb un subprocés és que aquest només s'ha d'utilitzar per realitzar tasques mentre l'usuari estigui utilitzant l'aplicació.

En aquest estudi es mostraràque executar tasques en background de forma permanent va més enllà dels serveis i que, a causa de certes restriccions per part del sistema operatiu o pel fabricant de l'arquitectura mòbil, l'ús d'aquests per a executar un treball de forma permanent no és l'opció més adeqüada amb les versions actuals del sistema operatiu. Avui en dia es té a l'abast diverses eines que permeten l'execució de tasques en segon pla i escollir una o altra dependrà del treball que es desitgi fer.

- *Email de contacte: pol.magre@e-campus.uab.cat.*
- *Menció realitzada: Enginyeria de Computadors*
- *Treball tutoritzat per: Vicenç Soler Ruiz*
- *Curs 2019/20*

2 Objectius

L'objectiu principal d'aquest estudi és el trobar la forma de mantenir una aplicació en execució en background de manera permanent.

Per a fer-ho, aquest treball estudia les eines de les que es disposa actualment per a treballar en background, i observar sota quines condicions aquestes funcionen sense ser aturades pel sistema.

3 Estat de l'art

El món de la programació background en Android està en constant evolució, les noves versions de sistemes operatius afecten a la manera de treballar dels programadors, noves eines, etc. És per això que si es vol obtenir informació del tema, es pot trobar informació obsoleta que avui en dia no ens serviria per res,i és aquest el motiu pel qual és important revisar actualitzacions.

Avui en dia les empreses s'orienten cap a una millor experiència d'usuari. Amb l'actual auge de les xarxes socials i que actualment tot es fa des del mòbil és important que la duració de la bateria sigui elevada i ens permeti l'ús del dispositiu el major temps possible. És per això, que a mida que Android treu noves versions del sistema operatiu aplica restriccions a les aplicacions que el sistema considera que no afavoreixen a l'experiència d'usuari i, a la vegada, estan consumint recursos limitats del sistema, com bateria, memòria, etc. Un servei executant-se en background és un dels components d'Android que més afectat es veu quan s'apliquen aquestes restriccions.

A part d'això, es pot veure que els propis fabricants de dispositius també busquen la manera de que el seu producte sigui econòmic i mantingui la bateria funcionant la major part del temps possible. Grans companyies conegudes com Huawei o Xiaomi introdueixen tècniques molt agressives per a estalviar bateria, que poden aturar processos executant-se si ho consideren necessari, i així d'aquesta manera, fer que el dispositiu no consumeixi tants recursos mentre s'estigui utilitzant. A ulls del consumidor el telèfon guanya en prestacions però per al desenvolupador pot arribar a ser tot un repte de programació.

Actualment hi ha guies de programació en background en Android, explicant les eines que existeixen per a executar aplicacions en segon pla. A més, es pot trobar informació sobre les restriccions que apliquen els fabricants de dispositius a l'hora d'executar aplicacions per a estalviar bateria. El que no s'ha trobat, són estudis on es mostrin els resultats obtinguts al treballar amb les eines actuals per programar en background en diferents dispositius mòbils.

En aquest treball s'ha estudiat el comportament d'aquestes eines en diferents dispositius i en el punt 7 es mostra com les restriccions aplicades per diversos fabricants afecten a la durabilitat temporal de les execucions en background.

En el següent punt es procedirà a explicar com s'ha fet l'estudi sobre les eines per a executar tasques en background, i en el punt 5 es veurà més concretament quines limitacions tenen actualment les aplicacions que executen feina en background. Més endavant als punts 6 i 7 s'explicaran les opcions més efectives que té un programador a l'hora de treballar amb aquestes, contrastant arguments amb els resultats que hem obtingut. Finalment, el punt 7 mostrarà els resultats de proves fetes amb diversos dispositius mòbils i el punt 8 les conclusions finals del treball.

4 METODOLOGIA

En primer lloc s'ha invertit el temps necessari en fer una recerca en diferents fonts d'informació sobre quines eines es poden utilitzar a l'hora de crear una aplicació que executi tasques en background.

Una vegada es coneixen aquestes eines i s'entén el seu funcionament, el que s'ha fet és programar varies aplicacions que treballen amb cadascuna d'aquestes eines i s'han instal lat a diversos dispositius Android per a provar el seu funcionament. Per a comprovar que les proves implementades funcionen amb la periodicitat, durabilitat temporal i estabilitat previstes, encara que l'usuari les hagi tancat al seu dispositiu, aquestes aplicacions estan connectades amb una base de dades de Firebase[1] a la qual envien constantment informació on s'especifica la data i l'hora exactes en la que s'ha realitzat l'operació, el dispositiu des d'on s'ha rebut la informació, la versió del sistema operatiu i el nom de l'apk. Els resultats extrets d'aquestes proves s'han utilitzat per a complementar i verificar la informació obtinguda sobre les eines actuals que ens permeten treballar en background en Android.

5 LIMITACIONS BACKGROUND

En versions de sistema operatiu anteriors a Android Marshmallow 6.0 o Android Oreo 8.0 no existien tantes restriccions a l'hora d'executar feina en background. Les tasques en segon pla de llarga duració es podien executar utilitzant serveis background sense restriccions, per exemple, per comunicar-se amb un servidor. Ara bé, si múltiples aplicacions fan feina en background a la vegada, el rendiment del telèfon quan l'usuari l'estigui utilitzant no serà el mateix degut als recursos limitats d'aquest. És per això que Android ha pres mesures al respecte.

En els següents punts d'aquesta secció es veuran quines són les limitacions que aplica Android a les aplicacions executant-se en background.

5.1 Doze Mode

S'introdueix amb Android Marshmallow 6.0, i és com s' anomena a l'estat al que passa el dispositiu quan està uns minuts inactiu. Dins d'aquest mode s'impedeix que algunes tasques es puguin executar en background. El seu propòsit és estalviar la vida de la bateria reduint el consum d'energia mentre no s'utilitza el telèfon. En aquest mode, el sistema intenta conservar la càrrega de la bateria restringint l'accés de les apps a serveis d'ús intensiu de CPU i de xarxa, ajornant les seves tasques, sincronitzacions i alarmes estàndards.

Cada un cert interval de temps, el mode "Doze" es desactiva i es creen unes petites finestres de temps on s'executen totes les tasques aplaçades que no s'han pogut executar durant el mode de descans. La Fig.1 mostra un exemple de funcionament de "Doze Mode", on les ratlles de color taronja són les finestres de temps que s'acaben de comentar:

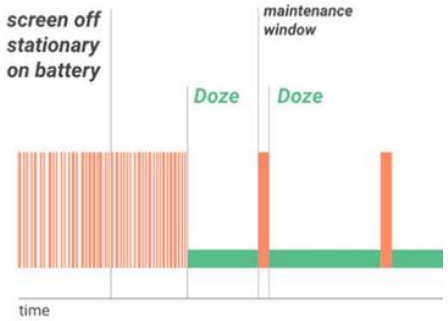

Fig.1.Funcionament "Doze mode". Font: https://developer.android.com/training/monitoring-device-state/doze-standby

Algunes de les restriccions que aplica "Doze Mode" són les següents:

- Impedeix accés a la xarxa.
- Treballs programats amb JobScheduler[2] no

[1] Firebase és una plataforma que ofereix Google per al desenvolupament d'aplicacions. Dins dels serveis que ofereix hi ha una base de dades anomenada Cloud Firestore que és la que s'ha utilitzat per fer aquest estudi.

[2] "JobScheduler és una eina per programar treballs en background, per

s'executen fins que arriba una finestra de manteniment.
- Alarmes que no estan programades amb els mètodes *AllowIdle*[3] són aplaçades.
- Ignora l'existència de WakeLocks[4].

Doze pot afectar les aplicacions de manera diferent, segons les capacitats que ofereixin i els serveis que utilitzin. Moltes aplicacions funcionen normalment en cicles de Doze sense cap modificació. En alguns casos, s'haurà d'optimitzar la manera com l'aplicació gestiona la xarxa, les alarmes, els treballs i les sincronitzacions. Les aplicacions haurien de poder gestionar de manera eficient les activitats durant cada finestra de manteniment.

5.2 Restriccions en l'execució background a partir d'Android Oreo 8.0

Amb versions més actuals de sistemes operatius d'Android, més concretament des d'Android 8.0 (API 26), s'han aplicat fortes restriccions a les aplicacions que executen tasques en segon pla.

Des d'Android Oreo, els serveis disposen d'una finestra de pocs minuts per a executar la tasca que estaven fent mentre l'aplicació està en segon pla. Una vegada esgotat aquest temps, el sistema els aturarà o llançarà una *IllegalStateException* si s'intenta cridar a *startService*[5] en background.

Per tant, si es vol programar l'execució de tasques de llarga duració mentre l'usuari no utilitza l'aplicació, utilitzar un servei en background no sembla l'eina adequada.

Per exemple, per a serveis que utilitzen la xarxa, si l'usuari té una connexió lenta i la tasca a realitzar triga a completar-se, en cas que l'usuari la desplaci a segon pla, es podria destruir el servei abans de finalitzar la tasca sense donar informació a l'usuari sobre si la tasca ha finalitzat correctament o no. Per això, no s'ha de confiar en que la tasca s'acabi de completar en aquest petit període de temps del que disposa el servei un cop l'aplicació treballa en background.

Aquestes limitacions d'Android Oreo, no s'apliquen a serveis en primer pla(foreground) ni als serveis enllaçats(bound)[1].

A causa d'aquestes limitacions Android posa a l'abast del desenvolupador noves alternatives per treballar en background. Per tant, el repte d'un programador en aquest cas és buscar l'eina que més s'adeqüi al que pretén fer.

En el punt 6 d'aquest document es donen a conèixer les alternatives que hi ha per a treballar en background i es guia al desenvolupador en situacions on ha d'escollir quina utilitzar, contrastant amb resultats experimentals que donaran validesa als arguments exposats al llarg del treball.

5.3 Optimitzador de bateria

En la majoria de fabricants de telèfons mòbils s'apliquen tècniques d'estalvi de bateria que poden aturar els nostres processos treballant en background. Desde la configuració del telèfon, un usuari pot desactivar l'optimitzador de bateria per a una aplicació en concret (que per defecte està activat), permentent que el sistema no la tingui en compte a l'hora d'aturar processos per a estalviar bateria. Segons proves realitzades en aquest estudi, això permet que serveis en background que abans s'aturaven al cap d'un minut després de passar l'aplicació en segon pla, estiguin setmanes funcionant. Actualment, a la PlayStore[6] trobem casos d'aplicacions molt ben valorades, on el número de descàrregues supera els cinc milions, on es proposa directament a l'usuari desactivar aquesta opció a l'iniciar l'App[2].

Encara que l'acció de desactivar l'optimitzador de la bateria es troba a la configuració del telèfon i s'ha de fer manualment per l'usuari, una app pot conduir l'usuari directament a la pantalla corresponent i demanar-l'hi, facilitant així que un usuari inexpert pugui fer-ho fàcilment.

Aquesta no és l'opció més recomanda i només s'aconsella l'ús en cas de que sigui estrictament necessari per al funcionament de l'app. Abans de recórrer a aquesta solució, s'hauria de donar un cop d'ull a les alternatives que proposa Android.

5.4 Arquitectures mòbils

A mida que passa el temps augmenten les prestacions del nostre dispositiu mòbil, introduint cada vegada més i més funcionalitats noves. És difícil mantenir la bateria viva durant un ampli espai de temps, i és per això que Android ha introduix eines com el Doze mode que intenten regular el consum de bateria.

A part d'això, sembla que alguns fabricants afegeixen afegeixen estalviadors de bateria desenvolupats per ells mateixos als seus dispositius, per a fer que aquests puguin romandre el major temps amb bateria disponible.

Moltes d'aquestes eines per a estalviar bateria que desenvolupen els propis fabricants han portat mala fama a les companyies que les utilitzen, ja que limiten la feina que es pot fer en background i provoquen que les eines que ofereix Android per a treballar en segon pla puguin arribar a no funcionar com està previst.

El ranking establert per la web *dontkillmyapp.com*, pàgina bastant reconeguda en el món de la programació background en Android, posa a grans empreses com Huawei o Samsung com a les pitjors a l'hora de tractar amb aplicacions que treballen en segon pla. És tant el descontent amb aquests fabricants que grans empreses com "VLC media Player" han tret de la Play Store la seva aplicació i no es pot obtenir per aquesta via desde dispositius Huawei[4]. A continuació es mostren alguns d'aquests casos en detall.

5.4.1 Huawei

Huawei, sobretot a partir de la versió de sistema operatiu EMUI 9+, ha introduït un software anomenat "Power Genie" que s'encarrega d'aturar la majora de

a que siguin executats en un moment determinat en el temps o de forma periòdica, en el punt 7.4 d'aquest document s'explica com funciona.

[3] Mètode que permet a una alarma executar-se quan el dispositiu es trobi en Doze mode.

[4] Un WakeLock és un component de programació que utilitzes per mantenir activa la pantalla o la CPU del dispositiu a l'executar una tasca.

[5] Mètode per iniciar l'execució dun servei en segon pla.

[6] Plataforma oficial de distribució d'aplicacions mòbils d'Android.

processos que s'executen en segon pla en el nostre dispositiu. PowerGenie atura les aplicacions que no es troben a la seva whitelist, que és una llista interna d'aquelles aplicacions que el sistema permet mantenir-se executant sense cap restricció. Com no es poden afegir aplicacions personalitzades a la whitelist predefinida per Huawei, significa que no hi ha cap altra manera de garantir el correcte funcionament d'una aplicació a Huawei EMUI+9 que la desinstal lació de PowerGenie. No s'ha trobat documentació de PowerGenie, però es pot desinstal lar[5].

En l'apartat 7, corresponent a l'exposició de resultats, es comprova l'efecte de PowerGenie amb un dispositiu Huawei JSN-L21 i es compara el funcionament d'aplicacions en background amb o sense PowerGenie activat.

En versions anteriors a EMUI+9, en concret a EMUI 4, quan el sistema procedia a aturar una aplicació, abans de fer-ho comprovava el tag del Wakelock (veure secció 5.1 DozeMode) que utilitzava el servei i, si aquest es corresponia amb uns noms concrets dins d'una whitelist definida per Huawei, el sistema no aturava l'app.

5.4.2 Altres fabricants

En el cas de Samsung, per exemple, porta incorporat el software anomentat "Sleeping apps" que pot aturar tasques programades en background si l'aplicació porta un temps sense ser utilitzada, encara que per defecte les atura als tres dies. A part, té altres maneres d'optimitzar bateria, però en cap cas tan restrictiu com Huawei, ja que aquestes opcions d'optimització es poden desactivar manualment per l'usuari, com s'ha comentat abans.

Altres marques com Xiaomi o Oneplus també tenen mala reputació en aquest aspecte, tots els casos es troben analitzats en la web referenciada[6].

En la secció 7 d'exposició de resultats es comparen les diferents solucions treballant en diversos dispositius de diferents marques per a veure com es comporten.

6 Ús d'eines background

En aquesta secció s'explicarà l'ús de les diferents alternatives que ofereix Android en l'actualitat.

6.1 Servei background

Android prioritza aturar processos que no són visibles per a l'usuari abans que altres que l'usuari està veient en primer pla, i és per això que els serveis background són molt propensos a ser aturats pel sistema.

Si el que es desitja és fer que una tasca s'executi permanentment en background utilitzant Android, és a dir, que estigui de manera indefinida funcionant, utilitzar serveis en background no és l'opció més recomanable ja que el sistema els aturarà.

6.1.1 Quan utilitzar un servei Background?

Quan es vulgui realitzar qualsevol tasca que no té perquè ser visible a l'usuari i, per tant, no necessita IU, però que té definides data d'inici i de finalització. Si l'objectiu és que la tasca hi ha de ser per sempre, és millor que s'esculli un altra alternativa.

6.2 Servei foreground

Un servei foreground no és més que un servei que es promou al primer pla, mostrant una notificació constant a l'usuari indicant-li que l'aplicació està executant alguna tasca.

Com que hi ha un element visible per a l'usuari que l'està avisant permanentment de que aquella aplicació està funcionant, té menys probabilitats de ser aturat que un servei background, ja que el sistema considera que és important per a l'usuari, però si detecta que necessita aturar-lo per recuperar recursos ho farà.

Els requisits que demana Android a l'utilitzar un servei de primer pla són que l'aplicació estigui executant una tasca que sigui immediata, remarca que aquesta tasca s'ha de completar (donant a entendre que no serà permanent), sigui perceptible per a l'usuari mostrant una notificació permanent i ha de tenir un inici ben definit i acabar[7].

Alguns exemples d'ús de serveis de primer pla són la reproducció de música, la realització d'una transacció de compra, el seguiment d'ubicacions en una ruta quan es fa exercici o les dades del sensor de registre per dormir. L'usuari iniciarà totes aquestes activitats, han de succeir immediatament, tenir un inici i final explícit i totes poden ser cancel lades per l'usuari en qualsevol moment.

Un altre bon cas d'ús d'un servei de primer pla és assegurar-se que les tasques crítiques i immediates (per exemple, guardar una foto, enviar un missatge, processar una compra) es completen si l'usuari tanca l'aplicació i obre una de nova. Si el dispositiu està sota una pressió d'alta memòria, podria matar l'aplicació anterior mentre continua processant, provocant pèrdues de dades o comportaments inesperats.

Si la tasca que necessita fer l'aplicació és permanent, la recomanació és que s'esculli una de les alternatives que es mostraran a continuació.

6.3 Alarm Manager

AlarmManager serveix per a programar alarmes en un moment precís, ja siguin repetitives o no. Quan aquesta s'activi, l'aplicació tindrà una finestra de varis segons on haurà de realitzar la feina que té programada, i és responsabilitat del programador que la feina pugui ser realitzada dins d'aquest període. L'avantatge d'aquesta eina en front les restriccions d'Android és que, a diferència dels serveis, una alarma no s'està executant de manera permanent, sinó que s'activa cada certs períodes de temps. Al no estar consumint recursos de manera continua i permanent és molt més difícil que Android l'aturi.

Tot i així, utiltizar AlarmManager de manera abusiva pot consumir bastanta bateria i s'ha d'utilitzar només quan volem realitzar tasques programades que s'executin moments precisios del dia. Si les tasques a realitzar poden ser posposades en el temps i són tasques que requeriran bastant temps en ser executades, és més recomanable utilitzar JobScheduler o WorkManager.

Dins d'Alarm Manager trobem mètodes com *setExact*,o set. El primer serveix per a tasques que han de ser executades en un moment exacte del dia, mentre que el segon mètode pot tenir una variabilitat d'uns quants minuts en la seva execució. Alternativament, també existei-

xen *setRepeating* i *setInexactRepeating* com a alternatives que fan tasques repetitives, on la segona consumeix menys bateria però a la vegada és menys precisa en el temps.

Com s'ha vist en el punt 5.1, en general les alarmes no s'executen en Doze Mode. Si es vol que l'alarma s'executi mentre el mòbil estigui en aquest estat, s'ha d'especificar fent servir els mètodes *setExactAndAllowIdle* o *setAndAllowIdle*.

Independentment del Doze Mode, desde l'API 19 (Android 4.4), totes les alarmes programades amb mètodes que no siguin *setExact* poden ser posposades en el temps si el sistema ho considera necessari per a estalviar recursos.

Per últim, no tots els mètodes mencionats tenen la mateixa repercussió a l'hora de gastar bateria del telèfon. Per tant, Android posa restriccions diferents en funció dels mètodes que s'utilitzin. Pel que fa a *setExactAndAllowIdle* o *setAndAllowIdle* se'ls permet executar una alarma, com a molt, cada 9 minuts[10], i *setExact* o *set*, cada minut. S'ha d'anar en compte amb l'arquitectura del mòbil, ja que certs optimitzadors de bateria d'alguns fabricants com Huawei o Samsung poden cancel lar les nostres alarmes si no prenem precaucions. Com s'ha vist a la secció 5.3, una solució és demanar a l'usuari que desactivi aquests optimitzadors per a l'aplicació concreta.

6.4 Job Scheduler

JobScheduler permet executar tasques en background amb menys impacte en els recursos del sistema com la bateria o la memòria. A diferència de l'AlarmManager, JobScheduler no es tant precís en el temps, serveix per a executar tasques que poden ser posposades i Android garanteix que s'executaran.

Amb JobScheduler es pot programar una sola tasca per a executar-se en un moment determinat o es pot programar execucions periòdiques, amb un interval mínim de quinze minuts[18]. És important tenir en compte que JobScheduler ,a diferencia de l'AlarmManager, no té un mètode "setAndAllowIdle", és a dir, JobScheduler no està disponible mentre el mòbil es troba en Doze mode, sinó que només s'executaran els treballs en finestres de manteniment o quan es torni a sortir d'aquest estat.

Des de JobScheduler es pot programar sota quines condicions es vol que es realitzi la tasca en segon pla, per exemple, es pot especificar que només s'executi quan el dispositiu s'estigui carregant, o que només funcioni quan el dispositiu tingui connexió a internet.

6.5 Work Manager

WorkManager és una eina que, a l'igual que JobScheduler, permet programar la realització de tasques (que es poden posposar) en segon pla ja sigui per a executar una tasca un sol cop o repetidament de forma periòdica. Al cap i a la fi no deixa de ser una barreja de les dues eines comentades anteriorment, JobScheduler i AlarmManager. JobScheduler no funciona en versions anteriors a Android 6.0 (API 23) i és per això que, a partir de l'API 23 en amunt, WorkManager utilitzarà JobScheduler internament per a programar tasques, mentre que per sota de l'API 23, utilitzarà AlarmManager[8]**¡Error! No se encuentra el origen de la referencia.**. Estalvia feina als programadors ja que no els fa fer dues versions de codi diferents en funció de la versió d'Android que té cada usuari, ja que, com s'ha comentat, WorkManager ja ho gestiona sol internament. A l'igual que JobScheduler, des de WorkManager es pot programar sota quines condicions es vol realitzar la tasca en segon pla. Per exemple, es pot programar que només s'executi quan el dispositiu s'estigui carregant, o que només funcioni quan el dispositiu tingui connexió a internet.

WorkManager gestiona el treball background que cal executar quan es compleixen les restriccions implementades pel programador, independentment de si l'aplicació està apagada o no. El treball en segon pla es pot iniciar quan l'aplicació es troba en segon pla, quan l'aplicació es troba en primer pla o quan l'aplicació s'inicia en primer pla i després passa a segon pla. Independentment del que faci l'aplicació, el treball background s'hauria de continuar executant o reiniciar si Android atura aquest procés[3].

WorkManager té algunes característiques de les quals no disposa JobScheduler. Per exemple, permet fer treballs encadenats, és a dir, programar que un treball s'executi quan hagi acabat un altre. Això pot ser molt útil, per exemple, per a comprimir una imatge i, en acabar, pujarla a un servidor. Tot i això no s'entrarà en detall amb aquestes característiques ja que s'allunyen de les intencions d'estudi d'aquest treball.

6.6 Push notifications

És possible que, si el programador no és un expert en Android, es pensi que una aplicació de missatgeria instantànea es tracta d'un servei background que no s'atura mai i que escolta constantment fins que arriba un missatge nou a l'usuari. A l'hora de programar això, actualment no funciona així i és per això que en aquest treball s'ha presentat com funcionen les eines de "Firebase Cloud Messaging" i les "Push notifications" fent una aplicació que les utilitza.

En primer lloc les Push notifications no són més que les notificacions habitualts que tothom coneix però amb la característica de que són "push", i es defineixen així a les notificacions enviades des d'un servidor cap a un client. D'aquesta manera es poden enviar notificacions als usuaris quan se'ls vol informar de qualsevol novetat, com per exemple que han rebut un missatge nou o que tenen un nou e-mail a la bústia d'entrada. Estan creades a través de varies APIs, en primer lloc la "Notifications API"[9], la "Push API"[10] i la "ServiceWorkerAPI"[11]. La primera s'encarrega de que l'aplicació pugui mostrar notifications a l'usuari, mentre que la segona permet gestionar els missatges que arriben des del servidor, encara que l'app estigui apagada, tot això s'executa gràcies a la tercera API, ServiceWorker.

En aquest estudi s'ha treballat amb Firebase Cloud Messaging per fer ús de les Push notifications ja que és una eina gratuïta, però hi ha diferents plataformes com Amazon SNS[14], OneSignal[15] o Airship[16] amb les que també es poden implementar. A l'Apèndix III es pot consultar una explicació més àmplia del FCM.

7 Resultats

Totes les aplicacions amb les que s'han obtingut aquests resultats es troben al final d'aquest informe. A no ser que s'especifiqui el contrari, els resultats de les proves que es mostren a continuació han sigut obtinguts sense demanar a l'usuari que desactivi les optimitzacions de bateria per a aquella aplicació en concret.

Els mòbils fets servir per a les proves han estat 4: un Sony XA1 (amb API 26), un Samsung Galaxy J7 (API 27), un Huawei JSN-L21 (API 29 amb PowerGenie) i un Huawei P8 (API 24 i no porta el PowerGenie). Aquests 4 dispositius mostren resultats per a 3 fabricants diferents, amb diferents versions d'API.

7.1 Resultats servei background

Cal comentar que les proves realitzades amb aquesta aplicació s'han fet amb un dispositiu Huawei JSN-L21 (Honor 8X).

L'aplicació desenvolupada en aquest estudi treballa amb la classe Timer[13], i escriu logs en un fitxer extern cada trenta minuts per a fer un seguiment de funcionament del servei.

Sense donar permisos especials a l'aplicació com els que s'han comentat en el punt 5.3, la duració ha estat d'un minut aproximadament mentre que demanant a l'usuari que desactivi les optimitzacions de bateria per a aquesta aplicació en concret, el servei ha estat funcionant més de 7 dies.En els documents entregats en el dossier del treball es troben els logs que ha generat l'aplicació.

Actualment, moltes aplicacions opten per demanar a l'usuari que doni aquests permisos manualment per al correcte ús de l'aplicació. Per a no gastar recursos de manera innecessària, és important assegurar-se a l'hora de dissenyar l'aplicació si la feina que es vol realitzar es podria dur a terme amb una de les altres eines anomenades al llarg del document.

7.2 Resultats servei foreground

En el dispositiu Huawei amb PowerGenie s'han fet varies proves amb aquest tipus de servei. En la Taula 1 es poden veure els resultats d'aquestes proves. La primera columna fa referència a si PowerGenie està activat o no en el dispositiu. Recordar que per defecte, en els dispositius Huawei EMUI+9 ve sempre activat. La segona columna fa referència a si s'ha desactivat manualment l'optimitzador de bateria (punt 5.3 del document) per a aquesta aplicació, que, per defecte, està sempre desactivat a no ser que l'usuari l'activi. Finalment, es mostra la duració de la prova. Als documents entregats al dossier es troben els logs que ha generat l'aplicació.

TAULA 1
SERVEI FOREGROUND EN UN DIPOSITIU HUAWEI EMUI+9

Power Genie	Optimització manual de bateria desactivat	Duració
Sí	No	≈5 hores
Sí	Sí	=7 dies
No	Sí	>2 setmanes

Tot i mantenir el servei actiu molt més temps desactivant les opcions d'optimització de bateria manualment, no es garantitza que el sistema no acabi aturant mai el servei si PowerGenie està actiu. En versions posteriors a Huawei EMUI+9 l'única manera de garantitzar un funcionament permanent del servei és desinstal lant PowerGenie.

En dispositius que no són Huawei, el servei ha durat setmanes actiu.

7.3 Resultats AlarmManager

En aquest estudi s'han fet múltiples proves amb AlarmManager provant tots els seus mètodes. S'ha programat una aplicació que puja informació a una base de dades cada cert període de temps, amb l'objectiu d'observar la durabilitat temporal de l'eina i la precisió en el temps a l'hora d'executar les tasques.

A la Taula 2 es mostra el temps que han estat executant-se les diferents proves, la mitja ($\bar{\mathbf{x}}$) i desviació estandar ($\boldsymbol{\sigma}$) de cada interval (en minuts) durant el temps que ha estat en funcionament l'aplicació, el % de fiabilitat corresponent al percentatge d'enviaments que s'han fet dins d'un temps prudencial inferior al 50 minuts (%fiab) i el % d'enviaments que s'han fet dins de l'interval de temps programat (% on time). No es mostren experiments amb el dipositiu Huawei JSN-L21, ja que amb el poc temps que ha durat l'eina en funcionament no s'han pogut treure dades fiables. Les proves fetes corresponen a les diferents funcions disponibles d'AlarmManager, especificades en la secció anterior 6.3.

Els nombres que aparaeixen en la prova corresponen a l'interval en minuts de l'alarma: 5 minuts en general, i 10 minuts en els casos de *SetWhileIdle*, ja que és el temps mínim requerit per Android a l'utilitzar aquest mètode. "AF" fa referència a aturada forçada ja que s'ha tingut un temps limitat per fer les proves i s'ha considerat que 3 dies era suficient temps per comprovar la seva duració.

TAULA 2
EXPERIMENTS ALARMMANAGER

Disp.	Prova	Duració	$\bar{\mathbf{x}}$	σ	%fiab.	%on-time
Sony XA1	SetExact5	4 dies AF	5,2	3,1	99%	97%
Sony XA1	Set5	>7 dies	5,4	3,3	99%	92%
Sony XA1	SetRep5	3 dies AF	5,2	3,1	99%	87%
Sony XA1	SetInexactRep5	3 dies AF	5,1	1,8	99%	89%
Sony XA1	SetWhileIdle10	>7 dies	10,2	1,1	100%	94%
Huawei P8	SetExact5	4 dies AF	5,7	5,3	85%	83%
Huawei P8	Set5	>7 dies	5,4	2,2	95%	80%
Huawei P8	SetRep5	3 dies AF	5,2	2,0	87%	75%
Huawei P8	SetInexactRep5	3 dies AF	5,8	6,0	89%	69%
Huawei P8	SetWhileIdle10	>7 dies	10,6	5,3	92%	76%
Galaxy J7	SetExact5	3 dies	4,9	0,3	100%	100%

Galaxy J7	Set5	3 dies	5,6	1,0	100%	84%
Galaxy J7	SetInexactRep5	3 dies AF	5,0	1,3	100%	90%
Galaxy J7	SetWhileIdle10	3 dies	10,6	1,0	100%	83%

En primer lloc, es pot observar com les mateixes proves que en el Sony i en el Huawei duren més de 7 dies funcionant, en el dispositiu Galaxy J7 cap experiment dura més de tres dies. Aquest fet és degut a la característica "Sleeping Apps" de Samsung que s'ha comentat en el punt 5.4.2 d'aquest document. Això s'ha de tenir en compte a l'hora de dissenyar l'aplicació i, si ens interessa tenir un AlarmManager funcionant més de 3 dies, demanar als usuaris de Samsung que desactivin l'opció "Sleeping Apps".

A partir de l'API 19, Android diu que les entregues d'alarmes no seran exactes a no ser que s'utilitzi el mètode *setExact* per programar-les[17]. En cas contrari, poden ser posposades pel sistema per a minimitzar l'ús de bateria(tot i ser més exacte, les alarmes programades amb *setExact* també seran posposades si el mòbil es troba en Doze Mode).

Si donem un cop d'ull als resultats podem verificar el que diu Android, veient com les proves amb *setExact* obtenen els millors resultats respecte les altres a l'hora d'executar-se en el moment que toca (columna %on-time). En el cas de Huawei, el %on-time és entre un 14%-17% menys precís. Això és degut a que el mòbil no s'ha fet servir per cap usuari mentre s'estaven fent les proves i, per tant, el temps que pot haver passat en DozeMode és molt superior als altres.

SetWhileIdle pot executar-se en DozeMode però no és exacte, seria l'equivalent al mètode *set*, i el sistema deu haver considerat que no era adequat executar-lo en el moment que tocava i per això els resultats en %fiab i %on-time no són els més elevats.Com a últim tret a destacar de la taula 3, podem veure en el cas de Sony com el mètode *setRepeating* és menys exacte que el *setInexactRepeating*. Tot i així ja ens avisa Android que les alarmes amb aquests dos mètodes poden ser aplaçades per estalviar recursos. Com que es van executar en moments diferents és possible que les condicions del telèfon afectessin al rendiment de l'aplicació.

Com a conclusions generals d'aquests resultats podem extreure que el tipus de mètode no influeix en la duració d'AlarmManager si es vol deixar funcionant diversos dies. El que sí pot influir en la durabilitat temporal d'aquestes proves és l'arquitectura de mòbil on s'executin. En el cas de Samsung caldria treure l'aplicació de la llista "Sleeping Apps" i en el de Huawei (a partir d'EMUI+9) demanar a l'usuari que desactivés les opcions d'optimització de bateria manualment o desinstal lar PowerGenie.

Si es vol la màxima precisió possible a l'hora d'executar treballs, s'haurà d'utilitzar *setExact* com a mètode per a programar AlarmManager.

7.4 Resultats JobScheduler

S'ha programat una aplicació perquè executi un treball de manera periòdica cada cert interval de temps. En total s'han fet tres proves amb intervals de 10, 15 i 20 minuts.

Per al correcte ús de JobScheduler, Android demana un temps mínim de quinze minuts entre execucions del treball[18], en aquest estudi s'ha inclòs un interval de 10 minuts per a comprovar com es comporta l'eina.

És oportú recordar que els treballs programats tant en JobScheduler com en WorkManager s'executaran quan el dipositiu tingui recursos disponibles per executar-los i certes condicions establides pel programador es compleixin, tampoc s'executaran quan el dispositiu es trobi en Doze Mode. En el cas d'aquesta aplicació l'única condició és disposar de connexió a internet per actualitzar la base de dades. Per tant, els resultats de les proves mostraran variabilitat en els intervals de temps entre execucions ja que la majoria de vegades els intervals no seran exactes, sinó aproximats. Si ens interesa executar tasques en un moment precís és millor utilitzar AlarmManager amb mètodes com *setExact*.

Cada vegada que s'executa un treball, aquest envia informació a una base de dades, com ara la data exacta de quan s'ha executat, la marca i model del dipositiu en el que està instal lada l'aplicació, etc. D'aquesta manera es té controlat el funcionament de l'eina.

Com s'ha vist en el cas de les alarmes per al dispositiu Samsung en concret, l'aplicació no ha funcionat més de 3 dies degut a "Sleeping Apps". Com s'ha comentat en el punt 6.5, els treballs es tornen a reprogramar si el sistema els atura quan l'usuari torna a obrir l'aplicació. És per això que en els resultats que es veuran a continuació, referents a les proves per al dispositiu Samsung, han durat més de 3 dies.

A la Taula 3 es mostren les diferents proves amb l'objectiu d'observar la durabilitat i freqüència temporal de JobScheduler, utilitzant diferents intervals de temps en diversos mòbils. Tant a les taules com a les gràfiques, "PG" fa referència a que PowerGenie està activat.

TAULA 3
EXPERIMENTS JOBSCHEDULER

Disp.	Interval	Duració	$\bar{x}$	σ	%fiab.	%on-time
Sony XA1	20 min	>7 dies	20,4	5,4	82%	63%
Sony XA1	15 min	>7 dies	15,8	6,2	87%	61%
Sony XA1	10 min	>7 dies	15,2	6,9	95%	26%
Huawei JSN-L21	20 min (PG ON)	≈4 hores	-	-	-	-
Huawei JSN-L21	15 min (PG ON)	≈6 hores	-	-	-	-
Huawei JSN-L21	15 min (PG OFF)	>7 dies	14,2	4,8	88%	69%
Huawei JSN-L21	10 min (PG ON)	≈ 1 hora	-	-	-	-
Huawei P8	20 min	>7 dies	20,3	3,3	56%	44%
Huawei P8	15 min	>7 dies	15,1	1,7	62%	57%
Huawei P8	10 min	>7 dies	14,5	2,8	81%	8%
Galaxy J7	20 min	>7 dies	19,8	5,5	90%	67%
Galaxy J7	15 min	>7 dies	15,2	3,9	89%	70%

Galaxy J7	10 min	>7 dies	14,6	4,9	96%	20%

El fet més destacable de la taula anterior és veure com PowerGenie de Huawei atura les feines executant-se en background i després d'escasses hores d'haver-se posat en marxa deixen de funcionar. En aquest estudi també s'ha provat amb PowerGenie desactivat (PowerGenie es pot desactivar a través de comandes especials des d'un ordinador) i s'ha pogut observar clarament la diferència pel que fa a la duració quan PowerGenie està desactivat (PowerGenie OFF). Les altres proves van estar més d'una setmana funcionant i van ser aturades expressament.

En segon lloc, s'observa que en intervals de 20 i 15 minuts les mitges coincideixen amb el temps programat. En canvi, quan reduïm l'interval per sota dels 15 minuts, l'eina no acaba de funcionar segons el previst. En totes les proves on s'ha programat un interval de 10 minuts la mitja ronda els 15, resultat normal ja que Android especifica que JobScheduler pot no funcionar com s'espera si es programa amb intervals inferiors a 15 minuts.

Com s'ha comentat anteriorment en aquest punt, JobScheduler i WorkManager no estan programats per a ser precisos a l'hora d'executar els treballs. En aquest sentit, a la columna "on-time" es pot observar les vegades que un treball s'executa en l'interval de temps esperat. Una vegada més, es veu que programant intervals per sota dels quinze minuts, només entre un 8% i un 26% de les vegades s'envien les dades dintre dels 10 minuts demanats contra el 60%-70% que es fa en intervals superiors als 15 minuts. Respecte a les marques, Huawei P8 és la que obté les pitjors dades.

Respecte a la fiabilitat (% de vegades que s'envien les dades en un interval inferor als 50 minuts) també es veu que el Huawei P8 té pitjor rendiment respecte a la resta però, dintre de cada fabricant, la prova amb 10 minuts és la que obté millors dades. A aquest fet no se li ha de donar molta importància, ja que les proves de 10 minuts es van realitzar en moments diferents que les altres i hi ha molts factors que influeixen a l'hora d'executar JobScheduler: Doze Mode, connexió a internet, optimització de bateria, etc.

La Figura 3 mostra gràficament la freqüència amb la que s'executen aquests treballs en un dels casos de la taula 3 (interval de 15 minuts) i ens premet observar si s'acosten als intervals de temps prèviament definits i les possibles diferències en la seva variabilitat.

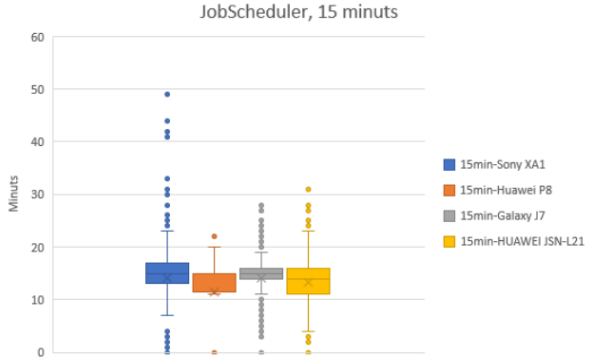

Fig.3.Gràfica de freqüència, experiment JobScheduler de 15min.

En aquesta prova, s'observa com la majoria dels treballs estan executats al voltant dels quinze minuts tal i com estava previst. En tots els casos la distribució dels quartils és homogènia a excepció del huawei P8, potser degut a que era un mòbil que no es tocava mai i per tant estava en DozeMode més temps que els altres. Que hi hagi outliers pot tenir varies explicacions. Pot ser degut a una falta de bateria o RAM i per tant el sistema no ha considerat adequat executar-los i els ha posposat. Una altra explicació pot ser perquè quan s'havien d'executar no tenien connexió i/o el mòbil estava apagat.

7.5 Resultats WorkManager

Cal recordar que WorkManager utilitza JobScheduler en versions posteriors a l'API 23 i AlarmManager a les versions anteriors. Tots els dispositius en aquest experiment tenien una versió de sistema operatiu superior a l'API 23. A la Taula 4 es mostren els resultats d'una aplicació amb la mateixa funcionalitat que la que s'ha provat amb JobScheduler.

Les proves d'aquesta aplicació s'han executat a la vegada que les de JobScheduler, per tant, és interessant comparar els valors de les dues taules (3 i 4).

TAULA 4
EXPERIMENTS WORKMANAGER

Disp.	Prova	Duració	$\bar{x}$	σ	%fiab.	%on-time
Sony XA1	20 min	>7 dies	23,3	4,9	92%	52%
Sony XA1	15 min	>7 dies	16,5	3,4	94%	64%
Sony XA1	10 min	>7 dies	17,9	5,1	96%	1%
Huawei JSN-L21	20 min (PG ON)	≈3 hores	-	-	-	-
Huawei JSN-L21	15 min (PG ON)	≈1 hora	-	-	-	-
Huawei JSN-L21	15 min (PG OFF)	>7 dies	17,0	3,7	94%	63%
Huawei JSN-L21	10 min (PG ON)	≈ 1 hora	-	-	-	-
Huawei P8	20 min	>7 dies	24,1	5,4	53%	28%
Huawei P8	15 min	>7 dies	15,4	1,6	81%	75%
Huawei P8	10 min	>7 dies	15,8	2,5	80%	1%
Galaxy J7	20 min	>7 dies	24,5	5,3	95%	42%
Galaxy J7	15 min	>7 dies	17,0	3,8	95%	66%
Galaxy J7	10 min	>7 dies	17,2	4,3	95%	0%

Pel que fa a la durabilitat temporal, a l'igual que JobScheduler es veuen perjudicats els treballs executant-se en un dispositiu amb PowerGenie.

En la columna $\bar{x}$, la mitja de temps que duren els intervals entre execucions, coincideix o s'apropa molt al que estava programat en les versions de 15 i 20 minuts, però en la de 10 no. De la mateixa manera, en les versions de 10 minuts pràcticament cap treball s'executa dins de l'interval de temps previst (% on-time), corroborant un altre cop que utilitzar intervals inferiors a quinze minuts influeix en que el treball no s'executi dins l'interval esperat. El que sí que destaca en la columna "% on-time" és

que les proves de 20 minuts han tingut pitjors valors que les de 15 minuts en general, resultats que a simple vista no tenen una explicació clara.

Respecte a la fiabilitat, els números d'aquesta columna no s'allunyen molt dels de la taula 3 (JobScheduler), però aquest cop les proves de 10 minuts tenen un %fiabilitat molt similar entre elles, a diferència del JobScheduler, a més de ser ara quasi 0. Donat que els mòbils estan en les mateixes condicions quan s'ha executat WorkManager que quan ho ha fet JobScheduler (les dues aplicacions s'han executat en paral lel en tots els mòbils), aquesta diferència en els % de fiabilitat, no té a veure amb l'estat en el que es trobava el dispositiu quan tocava executar-se un treball. Caldria entrar en detall en les diferències internes entre WorkManager i JobScheduler.

A la figura 4 es mostra, a mode d'exemple, el gràfic de la variabilitat dels intervals en el cas de la prova on cada treball s'executa periòdicament cada 15 minuts. En aquesta prova PowerGenie està desactivat per al dispositiu Huawei JSN-L21 per a comparar el seu funcionament amb els altres dispositius.

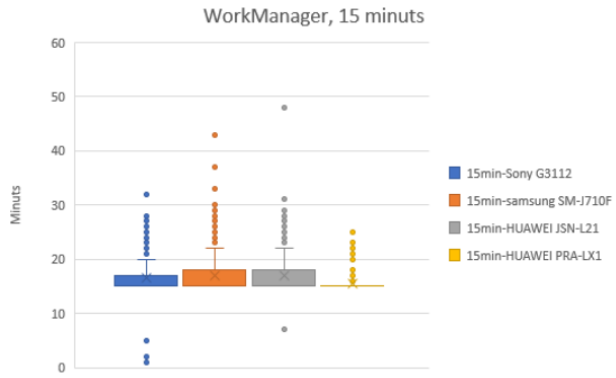

Fig.4 Gràfica de freqüència, experiment WorkManager15.

En la prova anterior es veu com la majoria de treballs s'executen al voltant dels 15 minuts, tal i com estava previst, i no es veu diferència significativa en la variabilitat dels intervals, a expeció del Huawei P8. Els outliers que hi ha són completament normals ja que WorkManager no executa treballs en DozeMode i per tant, al no ser una eina precisa, pot ser que s'hagin posposat.

8 Conclusió

Actualment, Android propsa diverses eines que substitueixen als serveis a l'hora d'executar tasques de llarga durabilitat temporal quan passem l'aplicació a segon pla. L'ús d'una o altra dependrà del que desitgem fer amb la nostra aplicació.

Per exemple, si volem fer un pagament online des de la nostra aplicació o tasques similars on és important assegurar-ne l'execució, podem utilitzar un servei Foreground. D'aquesta manera, encara que l'usuari tanqui l'aplicació o la passi a segon pla, aquella feina s'acabarà realitzant. Un altre exemple on es podria utilitzar un servei Foreground és a l'hora de guiar a un usuari via GPS. Encara que l'usuari tanqui l'aplicació se'l seguirà informant de la seva localització i sobre quant falta per a finalitzar la ruta.

Si es desitga executar una tasca en background en un moment precís, s'ha vist amb els resultats obtinguts al treball que la millor manera és utilitzant AlarmManager amb el mètode *setExact*. Amb els altres mètodes les tasques poden ser posposades pel sistema per estalviar recursos.

Si volem sincronitzar dades amb un servidor o pujar informació a una base de dades, i aquesta feina pot ser diferible, podem utilitzar AlarmManager amb mètodes que no siguin *setExact*, JobScheduler o WorkManager. L'avantatge d'utilitzar una eina com WorkManager és que ella ja gestiona si utilitzar AlarmManager o JobScheduler en funció del dispositiu on s'està executant.

Segons resultats obtinguts en aquest estudi utilitzar AlarmManager abans que WorkManager proporciona una fiabilitat més gran a l'hora de garantitzar que les tasques s'executaran en un temps determinat, però a la vegada consumeix més bateria.

En quant a la comparació de JobScheduler amb WorkManager no es veu una diferència tant gran en els percentatges de fiabilitat (que la tasca s'executi en el temps demanat), però sí en com WorkManager "castiga" més als treballs que no respecten l'interval mínim de 15 minuts a l'hora d'executar-se dins del període programat, per tal d'estalviar bateria (proves de 10 minuts).

Respecte als resultats obtinguts de WorkManager en què les proves de 20 minuts tenien pitjor precisió que les de 15, es proposa seguir fent més proves en més dispositius per a corroborar aquests resultats.

WorkManager té altres característiques que no té JobScheduler[19] però no influeixen a l'hora de complir l'objectiu d'aquest treball i, per tant, no es mencionen. A la següent figura podem veure com funciona WorkManager per dins.

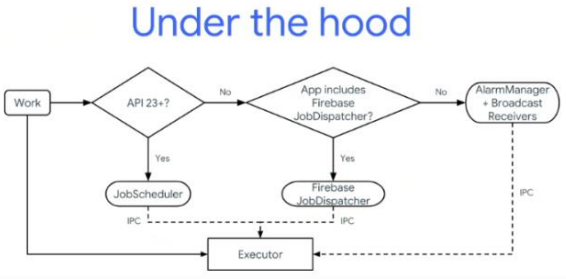

Fig.5.Esquema de funcionament de WorkManager. Font: http://www.douevencode.com/articles/2018-05/android-work-manager-one-scheduler-to-rule-them-all/

En resum, WorkManager substitueix JobScheduler perfectament i, a sobre, aporta funcionailitats noves, fent que el programador no s'hagi de preocupar per l'API on s'executa la seva aplicació. En els resultats obtinguts no es veu una clara situació on seria millor utiltizar JobScheduler abans que WorkManager.

Pel que fa a aplicacions de missatgeria instantània o semblants, on es rep informació d'un servidor, és recomanable utilitzar Push notifications i FCM (Firebase Cloud Messaging) o alguna plataforma semblant que permeti aquest tipus de comunicació. Segons el que s'ha vist a l'hora de construir l'aplicació amb FCM i Push noti-

fications, es poden rebre notificacions en qualsevol moment sense establir cap periodicitat i en continu funcionament.

Android proposa un esquema (Fig. 6) per a decidir quina eina utilitzar si es vol treballar en background.

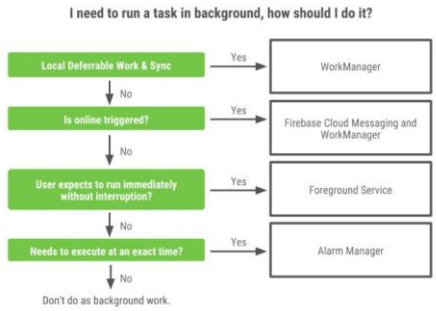

Fig.6.Diagrama de casos per a executar tasques en background. Font: https://android-developers.googleblog.com/2018/10/modern-background-execution-in-android.html

Per últim, s'ha vist com poden arribar a influir diferents arquitectures mòbils a l'hora d'executar tasques en background. S'ha comprovat que alguns optimitzadors de bateria en dispositius com Samsung o Huawei, a partir d'algunes verions de sistema operatiu, afecten directament a la durabilitat temporal de les tasques que s'executen en segon pla.

Com a conclusió final, si volem mantenir un treball en background sense que sigui aturat pel sistema, s'han de tenir en compte les limitacions que imposa cada fabricant i les pròpies restriccions d'Android a l'hora de dissenyar una aplicació. Segons el que s'ha vist als resultats, respectant els límits de cada eina, la durabilitat temporal de les aplicacions executant-se en background és superior a una setmana. En alguns casos com Samsung, s'ha de recordar a l'usuari que desactivi l'opció de "SleepingApps".

En el cas puntual de Huawei amb PowerGenie, s'ha vist que les eines que en altres dispositius es mantenen dies funcionant, en Huawei està poques hores. El màxim que es pot fer com a programador de l'aplicació és guiar a l'usuari perquè desactivi els optimitzadors de bateria per l'aplicació en concret, com fan moltes de les aplicacions disponibles a la PlayStore actualment.

AGRAÏMENTS

Els agraïments d'aquest treball estan dedicats a la meva família, per deixar-me un espai dins de casa on treballar durant el confinament i, al meu tutor Vicenç Soler Ruiz, ja que sense el seu recolzament i supervisió durant el semestre la qualitat del treball no seria la mateixa.

REFERÈNCIES

[1] Documentació de Service. https://developer.android.com/reference/android/app/Service?hl=es-419

[2] Enllaç al PlayStore de Podometer. https://play.google.com/store/apps/details?id=pedometer.stepcounter.calorieburner.pedometerforwalking&hl=es

[3] Reprogramació de treballs WorkManager. https://issuetracker.google.com/issues/113676489#comment2

[4] Noticia VLC deixa de treballar amb Huawei: https://www.theverge.com/2018/7/25/17614014/vlc-blacklisting-recent-huawei-devices-negative-app-reviews

[5] Desinstal lació de PowerGenie. https://forum.xda-developers.com/mate-20-pro/themes/remove-powergenie-to-allow-background-t3890409

[6] Solucions i informació sobre optimitzadors de bateria en diferents fabricants. https://dontkillmyapp.com/

[7] Android sobre l'ús efectiu de serveis en primer pla. https://android-developers.googleblog.com/2018/12/effective-foreground-services-on-android_11.html

[8] Documentació WorkManager. https://developer.android.com/topic/libraries/architecture/workmanager

[9] Documentació oficial de Notifications_API https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API

[10] Documentació oficial de Push_API https://developer.mozilla.org/en-US/docs/Web/API/Push_API

[11] Documentació oficial de Service_Worker_API, https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API

[12] GCM, https://developers.google.com/cloud-messaging

[13] Documentació classe Timer. https://developer.android.com/reference/java/util/Timer

[14] Documentació AmazonSNS. https://aws.amazon.com/es/sns/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc

[15] Documentació OneSignal. https://onesignal.com/

[16] Documentació sobre Push Notifications en Airship. https://docs.airship.com/reference/messages/message-types/push-notifications/

[17] Documentació AlarmManager. https://developer.android.com/reference/android/app/AlarmManager.html#setAlarmClock(android.app.AlarmManager.AlarmClockInfo,%20android.app.PendingIntent)

[18] Mínim interval per programar treballs de manera periòdica. https://developer.android.com/reference/kotlin/androidx/work/PeriodicWorkRequest

[19] WorkManager vs JobScheduler comentaris a stackoverflow. https://stackoverflow.com/questions/50279364/android-workmanager-vs-jobscheduler.

APENDIX I

TAULA 5
PRIORITATS DEL SISTEMA A L'HORA D'ATURAR UN PROCÉS

Tipus	Descripció
Natius	Els processos daemon de Linux natius són els responsables de fer funcionar tot. (Inclòs la pròpia eina que s'encarrega d'aturar processos)
Sistema	El procés *system_server*, que és responsable de mantenir aquesta llista.
Apps persistents	Les aplicacions persistents, com ara el telèfon, Wi-Fi i el Bluetooth, són crucials per mantenir el dispositiu connectat i poder proporcionar les seves funcions més bàsiques.
Apps en primer pla	L'aplicació que actualment està sent visible per l'usuari
Apps perceptibles	Són aplicacions que l'usuari pot percebre que s'estan executant. Per exemple, una aplicació amb un servei de primer pla que reprodueix àudio.
Serveis	Serveis background, és a dir, en segon pla.
Home	L'aplicació Launcher que conté fons de pantalla d'escriptori.
App anterior	L'aplicació anterior que estava utilitzant l'usuari.
Cached apps	Aquestes són les aplicacions que l'usuari ha obert i posteriorment ha deixat en segon pla. Seran les primeres en ser aturades per recuperar la memòria i se'ls aplicarà la majoria de restriccions.

Font: https://android-developers.googleblog.com/2018/12/effective-foreground-services-on-android_11.html

APENDIX II: GRÀFIQUES DE FREQÜÈNCIES DE JOBSCHEDULER I WORKMANAGER

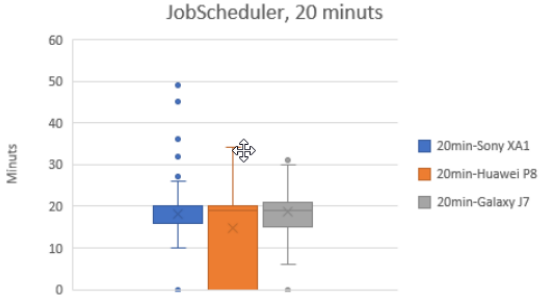

Fig.7.Gràfica freqüència, experiment JobScheduler de 20min.

Fig.8.Gràfica freqüència, experiment JobScheduler de 10min.

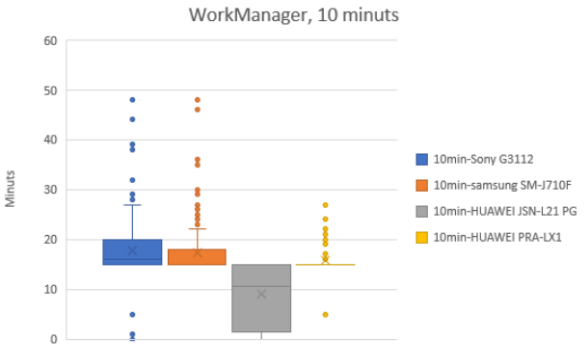

Fig.9.Gràfica freqüència, experiment WorkManager10.

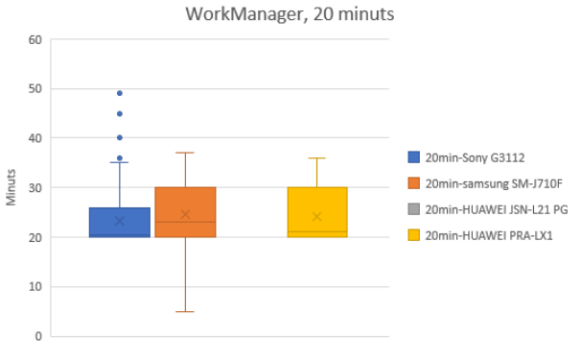

Fig.10.Gràfica freqüència, experiment WorkManager20.

APENDIX III: Firebase Cloud Messaging

Firebase Cloud Messaging és un servei que proporciona Google que permet al desenvolupador enviar "Push Messages" (missatges enviats des d'un servidor) i rebre'ls com a notificacions. Aquest és el successor del conegut GCM, Google Cloud Messaing[12]. El procés per a enviar una "Push notification" des de FCM consta de dos passos principals: en primer lloc cal registrar al client i en segon lloc enviar el missatge des de Firebase.

El pas de registrar al client comença per identicar a cada dispositiu. Aquests rebran el que Firebase anomena com a "token", que no és més que una cadena de caràcters única per a diferenciar els diferents usuaris. Una vegada obtingut aquest "token", podem desar-lo a la nostre base de dades. A la Figura 2 s'il lustra el que s'explica en aquest paràgraf.

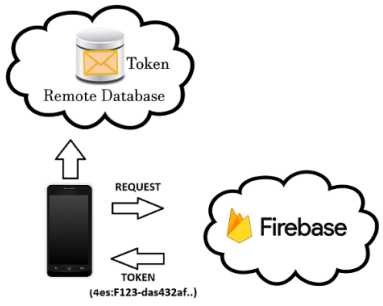

Fig.2.Esquema de la petició al servidor i rebuda de token.

Un cop tenim els tokens dels dispositius als que volem enviar una notificació, el posem dins d'una petició que farem cap a Firebase junt amb el missatge que volem transmetre als usuaris, com es mostra a la Figura 3.

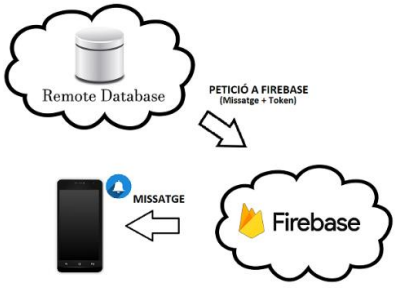

Fig.3.Esquema de la rebuda de la notificació.

Per a veure una demostració completa, en aquest estudi s'ha realitzat una aplicació on es treballa amb les "Push Notifications" i el FCM. Es pot trobar el codi complet en els documents entregats en aquest treball.