B-004

Android OS における状態変化通知による同期の回避に関する検討

Reducing synchronization on the notification behavior in the Android OS

川崎 仁嗣† 神山 剛† 小西 哲平† 大久保 信三† 稲村 浩†
Satoshi Kawasaki Takeshi Kamiyama Teppei Konishi Shinzo Ohkubo Hiroshi Inamura

1. まえがき

昨今のモバイルインターネット利用環境は今までのフィーチャーフォンから Android OS[1] や iOS[2]，Windows Phone[3] を搭載したスマートフォンに移行しつつある．既存のフィーチャーフォンは通信事業者がサービス仕様を策定していたため，通信の発生頻度や時間帯を管理する事ができ，通信事業者は想定される通信量に応じて設備投資を行ってきた．

スマートフォンでは様々な開発者が作成したアプリケーション（以降ではアプリと呼ぶ）をインストールして利用する事ができるため，通信の発生するタイミングや頻度はそれぞれのアプリで様々であることが多い．また，スマートフォン向けの OS ではユーザーが操作を行っていない画面オフの状態においても，データをサーバーと同期するために定期的にアプリが動作して通信などの処理を行っている．バックグラウンドでのアプリ動作タイミングを OS 側で制御している iOS や Windows Phone と比べると，Android OS は動作タイミングをアプリ開発者が自由に指定できるため，注意深く実装を行わないと特定のタイミングに偏ってアプリが動作する場合があり，これによってプロセッサへの処理が集中したり，限られた通信リソースへバースト的なアクセスが発生したりしかねない．このようなバックグラウンドで定期的な処理を行うアプリが発生させるトラフィックについての分析[4]が最近になって行われている．

バックグラウンドでのアプリ動作タイミングが偏ってしまう要因の一つとして，Android OS では端末状態の変化が起こった際に，その通知を受けた他のアプリが動作を開始するような動作（以降ではカスケード起動と呼ぶ）が頻繁に発生しており，開発者の意図していないカスケード起動が発生しうるため，アプリの動作タイミングが集中しないよう考慮した実装を行うことが難しい点が挙げられる．このため，アプリ開発者が本来意図したタイミングではないにもかかわらず動作することになり，想定していなかったタイミングに偏って動作の集中を発生させてしまう可能性がある．例えば，目覚ましアプリが朝の 7 時 に鳴動しディスプレイが画面オン状態へと遷移する挙動を想定してみると，画面オン状態へと遷移したことで端末状態の変化を示す通知が配送され，この通知を受取ったアプリがカスケード起動することにより動作タイミングが集中するような事例が考えられる．その他にも，電車がトンネル内などの圏外エリアから圏内エリアへと移動することにより，乗客の持つ端末が圏内状態へと復帰し，この状態変化による通知を受けたアプリが複数の端末で一斉にカスケード起動する事例も考えられる．

†（株）エヌ・ティ・ティ・ドコモ 先進技術研究所，
Research Laboratories, NTT DOCOMO, INC.

本稿では，端末状態変化の通知によって発生するカスケード起動において動作の集中が発生する要因の分析を行い，アプリ開発者が端末状態変化の通知による動作の集中を回避するための処理を行わずとも，動作の集中が発生しないよう OS 側で動作タイミングを制御する手法について検討する．

2 ． 状態変化通知の概要

端末状態変化の通知による動作の集中が発生する過程を理解するにあたって，まずは状態変化通知の概要を述べる．

スマートフォンにおいては，端末の状態などが変化したタイミングで動作するアプリが多い．例えば，電話アプリは音声通話の着信が発生した際に，画面を点灯させて着信中の画面を表示するとともに着信音を再生する．このようなアプリを実装するために，端末状態変化が発生したことをアプリ側に通知する機能が OS 側で提供されていることが多い．この通知機能を実現する仕組みを状態変化通知機構と呼ぶ．

Android OS においては，Broadcast Intent[5]を配送することで状態変化の通知が実現されており，アプリは受けとりたい通知を事前にシステム側へ登録する必要がある．このように，システム側に登録を行ったアプリのみが通知対象アプリとなる．例えば，電話アプリは電話の着信に関する通知は受取るがメー ルの着信については扱わないため，電話着信の通知のみを受取るよう指定する．

2.1 状態変化の種類

状態変化は，発生要因に応じて以下の 2 種類に大別できる．

(1) 内部要因状態変化
バッテリー残量の変化や，他アプリの実行による画面の点灯，および音楽再生の開始など，端末内での要因により発生する状態変化

(2) 外部要因状態変化
ユーザー操作による画面の点灯や，ネットワーク側からの電話やメール着信，および電波状態の変化など，外部からの要因により発生する状態変化

いずれの状態変化もカスケード起動を引き起こす要因となるが，外部要因状態変化のうちユーザー操作によるものはアプリ開発者が意図したタイミングである可能性が高い．一方で，内部要因状態変化のようにシステムや他のアプリが契機となるものや，外部要因状態変化でもネットワークが契機となるものは，アプリ開発者が意図したタイミングではないことも多い．

2.2 通知方法の種類

Android OS においては，システム側からの通知方法として以下の 2 種類があり，通知の種類に応じて通知方法が決定される．逐次通知と比べ，一斉通知は同時に複数アプリ

が動作することとなり，プロセッサやネットワークへの負荷は大きくなりやすい．

(i) 一斉通知
全ての通知対象アプリに対して同時に通知を行う方法

(ii) 逐次通知
通知対象アプリに対し優先順位の高いものから順に通知を行う方法

3. カスケード起動による問題点

3.1 同期干渉

あるタイミングに同期した動作を意図してはいないアプリであっても，同一端末上で動作している他のアプリからカスケード起動されることにより，双方のアプリの動作タイミングが同期する事象を，我々は同期干渉と呼ぶ．

同一の端末内において，アプリ自身が設定した特定のタイミングで動作するアプリ（自律型アプリ）と，特定のタイミングで動作するのではなく，状態変化通知などのアプリ外で発生した契機で動作するアプリ（他律型アプリ）の双方が動作している場合，カスケード起動により自律型アプリの動作するタイミングにおいて他律型アプリも動作する．つまり，自律型アプリの動作タイミングに他律型アプリが同期する．

3.2 大域同期

複数の端末間でアプリの動作タイミングが揃っている状態を，大域同期していると呼ぶ．大域同期が起こる要因として，以下の 3 種類の状態変化を契機とした同期干渉の発生が想定される．これらの状態変化を契機として動作するアプリが端末にインストールされている場合には大域同期が発生する．

(a) ネットワーク側における状態変化
圏外状態から圏内状態へと遷移するなど，エリア内の複数の端末に影響を及ぼす状態変化
例）ラッシュ時の満員電車がトンネル内などの圏外エリアから圏内エリアへと移動することによる，乗車している人が所有する端末の圏内状態への変化

(b) 自律型アプリが動作時に引き起こす状態変化
異なる複数の端末においても同一の時刻に動作する自律型アプリが，設定された時刻に動作して発生させる状態変化
例）目覚しアプリの鳴動時刻を毎朝 7 時ちょうどに設定された複数の端末が一斉に画面オンおよび鳴動状態へと遷移させることによる状態変化

(c) サーバーからの通信を受信するアプリが引き起こす状態変化
複数の端末に対し同報を行うサーバーからの通信を契機として動作するアプリが発生させる状態変化
例）複数の端末に対してメッセージを配信するサービスのクライアントアプリが画面オン状態へと遷移させることによる状態変化

なお，(b)の例で示した目覚しアプリは通信しないためそれ自体が問題にはならないが，自律型アプリであるという性質上，同期干渉を発生させることは問題である．何故ならば，大域同期した自律型アプリによる同期干渉が発生することにより，アプリ開発者の意図に反して，他律型アプリが大域同期したタイミングで動作するからである．目覚しアプリの例でいえば，図 1に示すように，画面点灯や鳴動状態への変化を通知された他律型アプリ（App3）も毎朝 7 時ちょうどに動作することとなる．

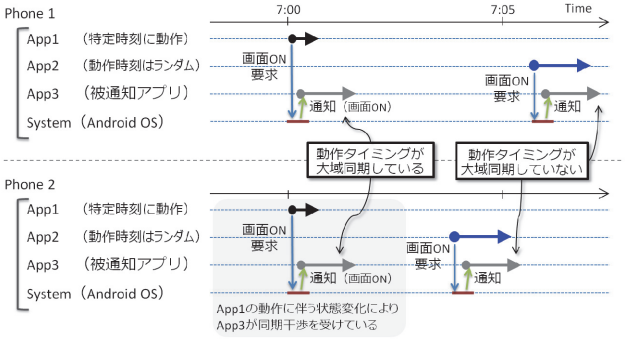

図 1 大域同期の発生する様子

アプリ開発者は自律型アプリの実装において通信集中を避けるよう意識するため，特定時刻に起動した際は通信を発生させないような処理とするが，他律型アプリは動作タイミングが端末間で同期している状態になることに気付かない開発者が多く，動作を開始した時点で通信を行う処理とすることも多い．このように，同期干渉を受けるアプリの開発者が本事象を予見することは難しく，OS 側で本事象を回避するような手法が必要である．

4. 提案手法

本提案手法では状態変化の発生要因を取得して，ネットワーク側や他のアプリを契機とした状態変化である場合には，同期干渉が発生しないように通知を行うタイミングを制御する手法をとる．

4.1 状態変化発生要因の取得

状態変化が発生し状態変化通知機構が通知を行う際に，状態変化を発生させた要因を取得し，3.2 節で述べた様な大域同期を発生させる可能性のある状態変化であるかどうかを判定する．

大域同期を発生させる可能性がある場合には大域同期型，発生させない場合には大域非同期型として，状態変化を分類する．

4.2 発生要因に基づく通知の制御

大域同期型の状態変化である場合は，通知を行うと同期干渉が発生するため，同期干渉が発生しないように通知を制御する．通知の制御方法としては以下の 2 パターンを検討した．

(A) 通知の遮断
通知対象アプリへの通知を破棄する．通知対象アプリがカスケード起動しないため，同期干渉は発生しない．

(B) 通知の遅延
通知対象アプリへの通知を遅延させる．状態変化が発生してから遅延時間分だけ間を空けてカスケード起動するため，同期干渉は発生しない．また，端末間で遅延時間を分散させることで，大域同期の発生を防ぐことができる．

それぞれの制御方法について，アプリ挙動へ与える影響，および実装コストを比較した結果を表 1に示す．

表 1

通知の制御方法	アプリ挙動への影響	実装コスト
通知の遮断	×（状態変化に応じた動作が阻害される）	○（状態変化通知機構の改変のみ）
通知の遅延	△（状態変化に応じた動作が遅延する）	○（状態変化通知機構の改変のみ）

通知の遮断を行った場合，アプリは通知を受け取れることを前提に実装されているため，挙動に影響を与える可能性が高い．例えば，電話の着信通知を受け取り，指定時間経過後に留守録機能を起動するようなアプリを考えた場合，そもそも通知を受け取れないため正常に動作しない．

通知の遅延を行った場合，遅延は発生するがアプリは通知を受け取れるため全く動作しなくなる状況は回避できる．また，Android OS の場合，そもそも逐次通知はより優先度の高いアプリが処理を完了しないと通知が行われないため，既存の機構においても通知が遅延する可能性がある．しかし，低遅延で通知が受け取れることを前提にしたアプリが存在する可能性はあり，アプリ挙動への影響もあると考えられる．

以上より，我々は，実装コスト，およびアプリ挙動への影響を抑えることが可能な，通知の遅延による制御を用いることとした．

5. 実装

提案手法の実現性を検証するため，状態変化の発生要因を取得して，状態変化が大域同期型か大域非同期型かを判別するとともに， Broadcast Intent を用いた状態変化の通知動作に対して，通知の遅延制御を実施する機能を Android OS 上へ実装した．

なお，ベースとして Android プロジェクトによりオープンソースとして開示されている AOSP[6] 4.0.3 r1 を用い，フレームワーク層の JAVA 実装箇所に対し修正を行っている．

5.1 全体構成

実装の構成を図 2に示す．なお，図中では通知されるデータ，つまり Broadcast Intent を BI と省略して記載している．

状態変化発生要因の取得や通知の制御機能を実現するために，フレームワークに含まれるいくつかの Manager サービスに修正を加えている．具体的には，Broadcast Intent の配送を司る Activity Manager Service，画面点灯状態の切り替えを制御する Power Manager Service および Window Manager Service，自律型アプリの動作タイミング契機となり得る Alarm を管理する Alarm Manager Service，そしてネットワーク接続状態を管理する Connectivity Service である．また，取得した発生要因に基づき，Broadcast Intent 配信の遅延が必要か判断する処理を，新たに設けた Broadcast Manager Service 内に実装した．

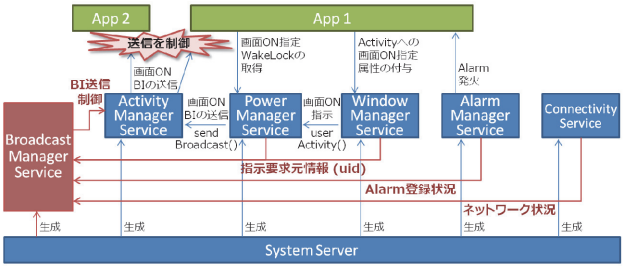

図 2　実装構成

5.2 状態変化発生要因の取得

大域同期の原因となり得る 3 種類の状態変化について，発生要因を取得することで大域同期型か大域非同期型のどちらかに分類する．

(a) ネットワーク側における状態変化

Connectivity Service において，状態変化がネットワーク側を契機としたものか，ユーザー操作を契機としたもののいずれであるかを取得する．具体的には，状態変化の発生前後における Wi-Fi や Bluetooth，モバイルデータ通信，機内モードなどのそれぞれの設定状態を比較し，設定状態が変化していればユーザー操作によるものと判定し，設定状態に変化が無い場合はネットワーク側での状態変化と判定することとした．この場合，ユーザー操作による状態変化は大域非同期型であり，ネットワーク側での状態変化は大域同期型である．

(b) 自律型アプリが動作時に引き起こす状態変化

自律型アプリが動作時刻を任意に設定して動作させたい場合は，Alarm Manager[7]に対して Alarm を設定する方法が用いられることが多い．

Alarm を設定する際には，厳密に指定した時刻で動作させる指定方法（以降では Exact 指定と呼ぶ）と，指定した時刻をベースにシステム側で決定される時刻で動作させる指定方法（以降では Inexact 指定と呼ぶ）とがある．また，指定時刻において端末のプロセッサがスリープ状態の場合，プロセッサをスリープ状態から解除して直ちに動作させる設定（以降では wakeup 設定と呼ぶ）と，次に何らかの要因でスリープ状態から解除された時点で動作させる設定（以降では non-wakeup 設定と呼ぶ）が存在する．

Alarm Manager Service で管理されている Alarm の登録状況を取得し，Exact 指定かつ wakeup 設定である Alarm により動作したアプリを識別する．この種別の Alarm で動作するアプリは大域同期しやすいと考えられるためである．識別したアプリの uid をリストとして保持しておき，次に状態変化が発生した際の指示要求元情報の uid と比較し，リストに含まれていれば大域同期型の状態変化，含まれていなければ大域非同期型の状態変化と判定する．なお，指示要求元情報とは，Window Manager Service や Power Manager Service に対して画面点灯などの指示を要求したアプリの uid のことである．

(c) サーバーからの通信を受信するアプリが引き起こす状態変化

サーバーからの通信には，Google 社の提供する Google Cloud Messaging[8]（GCM）や WAP-Push[9]がある．

GCM によるアプリの起動は，起動対象となるアプリに対する特定の Intent を送信することで実現されている．し

たがって，Activity Manager Service において，送信される Intent を監視することで起動されるアプリを識別することが可能である．WAP-Push についても同様に，WAP Push Manager から対象のアプリへ特定の Intent が送信されるため，Intent を監視することで起動されるアプリを識別することが可能である．

(b)の場合と同様に，識別したアプリの uid をリストとして保持しておき，次に状態変化が発生した際の指示要求元情報の uid と比較し，リストに含まれていれば大域同期型の状態変化，含まれていなければ大域非同期型の状態変化と判定する．

5.3 発生要因に基づく通知の制御

取得した状態変化発生要因から，状態変化が大域同期型と判定された場合において，Activity Manager Service に対して該当の通知を行うタイミングを遅延するよう指示を行う．具体的には，該当する Broadcast Intent の送信処理をキャンセルし，その情報を Broadcast Manager Service 内に保持する．所定の遅延時間経過後に，改めて Activity Manager Service へ Broadcast Intent の配送を指示する．

遅延時間は 30 秒～60 秒の範囲で，端末起動時刻を基に算出することで，端末毎に異なる値としている．

6. 今後の課題

端末状態変化の通知によって発生するカスケード起動において動作の集中が発生する要因の分析を行い， OS 側で同期干渉の発生を回避するための手法について検討した．また，検討した手法を AOSP 上へ実装した．

今後は，実装を用いて動作集中の回避が図れていることを評価により確認したい．また，提案手法については大域同期型であるか否かの判定精度を向上させたい．具体的には，状態変化発生要因の(c)サーバーからの通信を受信するアプリが引き起こす状態変化を判定するにあたって，現在は GCM や WAP-Push で起動されたアプリを対象としている．しかし，実際にはアプリ開発者が独自の Push 通知手段を実装している場合もあり，これを現在の手法で検知することはできない．サーバーとの通信履歴を参照することで，サーバーからの通信を契機とした動作であるか判別することが可能となり，大域同期型かどうかの判別精度も向上すると考えられる．

7. あとがき

Android OS における動作タイミングの集中が発生する要因として，大域同期型である状態変化を通知する動作に伴って同期干渉が発生することが挙げられる．これにより，端末内処理リソースの一時的な枯渇や，大域同期したアプリによる通信に伴って通信集中の発生が見受けられる．

同期干渉により大域同期している状態になることはアプリ開発者の意図していない動作であり，この挙動を意識した実装を全ての開発者に求めることは難しい．

本研究では，意図しない大域同期を引き起こす同期干渉を抑制するために，大域同期型である状態変化であるかどうかを判定し，大域同期型の可能性がある場合には，状態変化通知の配送を遅延させる制御方法を提案した．

提案手法の実現性検証を目的として，AOSP により公開されている Android OS のフレームワーク部分を改変することで実装を行った．

参考文献

[1] Google Inc. : Android - Discover Android,
http://www.android.com/about/,
(last access : 2012.12.5)

[2] Apple Inc. : アップル - iOS 6,
http://www.apple.com/jp/ios/,
(last access : 2012.12.5).

[3] Microsoft Corporation : Windows Phone,
http://www.microsoft.com/ja-jp/windowsphone/,
(last access : 2012.12.5).

[4] F Qian, et al. : Periodic Transfers in Mobile Applications: Network-wide Origin, Impact, and Optimization, International World Wide Web Conference, 2012.

[5] Google Inc. : Broadcast Intent,
http://developer.android.com/reference/android/content/Context.html#sendBroadcast(android.content.Intent),
(last access : 2013.6.26).

[6] Google Inc. : Android Open Source Project,
http://source.android.com/,
(last access : 2013.6.26)

[7] Google Inc. : Alarm Manager,
http://developer.android.com/reference/android/app/AlarmManager.html,
(last access : 2013.6.26).

[8] Google Inc. : Google Cloud Messaging for Android,
http://developer.android.com/google/gcm/index.html,
(last access : 2013.6.26).

[9] Open Mobile Alliance Ltd. : Push Architectural Overview,
http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-250-pusharchoverview-20010703-a.pdf,
(last access : 2013.6.26).