Androidアプリケーションの利用情報に基づく消費電力分析手法

古庄 裕貴[1,a) 久住 憲嗣[2,b) 神山 剛[3,c) 稲村 浩[3,d) 中西 恒夫[4,†1,e) 福田 晃[4,f)]

受付日 2013年11月13日, 採録日 2014年5月17日

概要：本論文はAndroid端末上で動作するアプリケーションの省電力化手法を提案する．従来の消費電力分析技術は開発者が与えるテストケースを基にプロファイリングを行うものが多く，多様な利用形態を想定したプロファイリングを行うことが難しかった．提案手法は複数のユーザの利用情報を収集することで様々な利用形態におけるプロファイリングを行う．提案手法は利用情報を取得するコードをアプリケーションへ自動化された手段を用いて埋め込む．そのアプリケーションはユーザに利用されているときに，利用情報のログを収集する．そして収集したログから消費電力を見積もり，チューニングの対象と方針決定の参考になる情報を開発者に提示する．本論文では提案する分析環境を用いてオープンソースのアプリケーションに対して，複数の被験者から収集したログを基に消費電力量の分析できることを示し省電力効果の検討を行った．分析結果から導き出したチューニング方法をアプリケーションに施した結果電力が削減され，分析結果は省電力化の検討に有効な情報だということを確認した．

キーワード：電力消費，Android

Power Consumption Profiling Method Based on Android Application Usage

Hiroki Furusho[1,a) Kenji Hisazumi[2,b) Takeshi Kamiyama[3,c)
Hiroshi Inamura[3,d) Tsuneo Nakanishi[4,†1,e) Akira Fukuda[4,f)]

Received: November 13, 2013, Accepted: May 17, 2014

Abstract: This paper proposes a method for collecting essential data for profiling energy consumption of applications that run on Android OS. The existing power estimating method is unable to take into account all possible usage patterns since developers are able to use the limited number of test cases in the profiling. Our proposed method analyzes power consumption using log collected during the use of the application on users' handset. In our method, the logging code is automatically embedded into applications. The applications get users usage information. Then, our method estimates power consumption using usage information, and gives developers hints that can be helpful in decision making and subject to tuning. In this paper, we analyzed the open source application's power consumption based on the user's log and estimated the application's power-saving effect. Power consumption of the application is reduced by the tuning that derived from analysis results. Therefore, it is conformed that the analysis results is valid information to the study of the power saving analysis.

Keywords: power consumption, Android

[1] 九州大学大学院システム情報科学府情報知能工学専攻
Department of AIT, Graduate School of ISEE, Kyushu University, Fukuoka 819–0395, Japan
[2] 九州大学システムLSI研究センター
System LSI Research Center, Kyushu University, Fukuoka 819–0395, Japan
[3] NTTドコモ先進技術研究所
Research Laboratories, NTT DOCOMO, Inc., Yokosuka, Kanagawa 239–8536, Japan
[4] 九州大学大学院システム情報科学研究院
Faculty of ISEE, Kyushu University, Fukuoka 819–0395, Japan
[†1] 現在，福岡大学工学部電子情報工学科
Presently with Department of Electronics Engineering and Computer Science, Fukuoka University
[a)] furusho@f.ait.kyushu-u.ac.jp
[b)] nel@slrc.kyushu-u.ac.jp
[c)] kamiyamat@nttdocomo.co.jp
[d)] inamura@nttdocomo.com
[e)] tun@fukuoka-u.ac.jp
[f)] fukuda@f.ait.kyushu-u.ac.jp

1. はじめに

スマートフォンと呼ばれる多機能な携帯電話が普及してきた．従来の携帯電話に比べてスマートフォンは多様に活用されるため消費電力が大きく，十分な連続稼働時間が確保できないことがある．スマートフォンは今後も小型化，高性能化され，それにともないスマートフォンに接続される電源への寸法や容量への要求は厳しくなっていき，稼働時間の確保は今後も難しくなっていくと思われる．そのため，アプリケーション実利用時のスマートフォンの消費電力を削減し連続稼働時間を確保する必要がある．

コンピュータの省電力化手法の 1 つにソフトウェアのチューニングがある．ソフトウェアは計算結果が同じでもその実装によって大きく消費電力が異なる場合がある．消費電力増加の要因として以下のような処理があげられる．

- 過剰なインスタンスの生成・破棄
- ループ処理の誤り
- 通信処理
- バグ，その他不具合

その場合は無駄な処理を行わないようソースコードを修正することで消費電力を削減できる．

アプリケーションの開発者が消費電力削減のためのチューニング方針を検討するためには，現時点におけるアプリケーションの消費電力量を把握することが必要である．そのため，アプリケーションの電力消費傾向を分析し，過大に電力を消費しているソースコード中の箇所を特定する電力プロファイリング技術が必要となる．

既存の分析手法の多くは CPU や無線 LAN，補助記憶装置などコンピュータに搭載される部品（以後，デバイスと呼ぶ）ごとの消費電力量を分析し，結果を基にソフトウェアをチューニングして省電力化を行った．しかしその多くは，開発者が与えた入力に対するプロファイリングを想定しているため，数人で構成される開発者グループがあらゆる利用方法を網羅するようなデータを準備することは困難である．有用な分析結果を得るために，様々な利用方法に基づくデータを収集できるプロファイリング技術が必要である．

本研究は，ユーザがアプリケーションを実際に使用しているときに情報を収集し，その情報に基づいて消費電力を推定し可視化する手法を提案する．実環境下におけるアプリケーションの利用データを用いることで，消費電力に加えユーザごとの利用形態や環境の違いによる電力消費傾向の差を分析する．本手法では，消費電力を推定するために必要な情報（デバイスの利用量など）を取得するコードを，開発者がアプリケーションのビルド時に自動化された手段を用いて埋め込む．複数のスマートフォン上でアプリケーションが使用された際のログを集計して，アプリケーションが有する機能ごとの累計電力などの情報を出力し，省電力化が可能な箇所を発見するヒントを開発者に与える．本手法の評価としてオープンソースのアプリケーションへ本手法を適用して，実使用時の電力を推定し，プロファイリングを行う．

本論文の構成を以下に示す．2 章で既存の消費電力分析技術を紹介する．3 章では提案手法である，実環境における運用時消費電力分析手法の概要と消費電力推定に必要なパラメータ収集方法について説明する．4 章では提案手法を用いた際の観測対象となるアクティビティ，サービスをはじめとした分析に必要な情報の取得方法について説明する．5 章では，提案手法を用いた分析の手順について説明する．6 章では，評価実験として提案手法を用いたアプリケーションの画面ごとの消費電流量分析，チューニング方針の検討とチューニング後の消費電力量シミュレーションならびに測定を行ったので報告する．7 章で本論文のまとめと今後の課題を述べる．

2. 関連研究

スマートフォンに限らずコンピュータの消費電力分析手法の 1 つに端末に搭載される各デバイスの利用量（以後，リソース消費量と呼ぶ）をパラメータとする電力モデルを生成して電力量を推定する方法がある．電力モデルを利用することで計測機器を接続せずに電力を測れるという利点がある．電力モデル式の基本形を式 (1) に示す．

$$P_{estimate} = \sum_{m \in M} C_m \cdot V_m \tag{1}$$

$P_{estimate}$ は見積もられる電力，M は電力消費と関連が深い要素の集合，V_m は電力消費と関連が深いパラメータ，C_m はパラメータの係数を表す．C_m は電力値とパラメータとなるリソース消費量の両方を計測して，回帰分析を行い求める．

携帯端末の消費電力量分析手法をあげる．Palm というオペレーティングシステム（OS）を搭載する端末の CPU，無線通信デバイスの利用に加え，ボタン押下やペン入力による電力消費を考慮した電力モデル式を用いて電力消費量推定する方法がある [1]．エミュレータの実行トレースから，デバイスを操作するシステムコールを探索して消費電力を推定している．文献 [2] は Android 搭載端末の電力を推定する．デバイスの稼働状態をパラメータとする電力モデルを作成し，端末から 1 秒ごとに各種デバイスの稼働状態を保存するロガーアプリケーションを開発しており，端末にインストールすることで複数のユーザからの情報収集，分析を実現している．これらの手法は端末全体の消費電力の推定が可能であるが，Android はバックグラウンド処理が行えるため，全体から分析対象アプリケーションの消費電力を分離することが難しい．

アプリケーション自身のリソース消費量から消費電力量を推定する方法もある．Linux のプロセスファイルシステ

ム [3] からリソース消費量を取得し，それらをパラメータとして消費電力を見積もる手法 [4] がある．プロセスファイルシステムとは OS のリソース情報に紐づけられているファイルシステムであり，CPU 使用時間，ファイルシステムへの読み書き量，ネットワーク送受信量が取得できる．この手法によって OS から得た情報だけでプログラム実行時の端末の消費電力を見積もることができる．また，ディスプレイの消費電力推定手法として，エミュレータのスクリーンショットを取得して表示されている色から電力を計算する方法がある [5]．一般的にスマートフォンの画面は 1 つのアプリケーションが占有するため分析対象アプリケーションの消費電力量を推定できる．これらの手法を用いてアプリケーションの消費電力量を推定することで，チューニングを行うかどうか判断したり，チューニング効果を確認したりできる．

さらに有益な情報を引き出すためには，さらに詳細な分析が必要となる．ソースコード中のチューニング箇所を検討するための情報を開発者に提供する分析手法として，メソッドごとのリソース消費量を取得する手法 [6] がある．この手法はトレースログとリソースログと呼ばれる 2 種類のログを収集する．トレースログとは Dalvik VM 組込みのプロファイラを用いて取得したメソッド呼び出し時のメソッド名と実行時刻などが含まれるログで，リソースログとはシステムから取得したリソース消費量が含まれるログを指す．2 つのログ取得時間を付き合わせて，メソッドごとのリソース消費量を取得する．これによりソースコードのどの部分が過剰に電力を消費しているかを知ることができる．

この手法には 2 種類のリソースログ収集方法が提案されているが分析の精度とアプリケーションの処理速度のどちらを優先するかでリソースログ収集手法を使い分ける必要がある．1 つ目の方法は一定間隔で収集する方法で 2 つ目は低レベル API を実行するごとに取得する方法である．一定間隔で取得する場合，処理速度の低下を抑えることができるがトレースログとの対応がとれなくなるために分析精度が低下する．反対に低レベル API の実行ごとに取得する場合，トレースログと 1 対 1 で対応づけることが可能となるため分析精度は高くなるが実行時間は約 36 倍と膨大に増加する．このように処理を行うごとに付随するリソース消費や処理時間のことをオーバヘッドと呼ぶ．

メソッド単位などの細粒度での消費電力を見積もることができるため，開発時の局所的な評価を行うためのアプリケーションプロファイリング手法としては向いているが，オーバヘッドが大きく，長時間にわたるユーザ使用時の消費電力プロファイリングへの適用が難しい．また仮想マシンを改造するこの手法にはシステムの管理者権限が必要となるため，多数のスマートフォンへ導入する手間やセキュリティの低下が発生する．

3. 運用時消費電力分析手法

3.1 分析手法概要

既存手法 [6] には分析にかかるオーバヘッドと Android システムの変更という 2 つの問題があった．本研究ではこれらを解決するために，通常の Android スマートフォンを対象とする従来手法と比較して小さなオーバヘッドを持つログ収集方法と，収集した情報を元に消費電力プロファイリングを行う手法を提案する．本手法は Android アプリケーションの消費電力をチューニングするうえで，アプリケーション開発者にとって必要となる情報を提示する．また，本手法は関連研究で紹介した手法 [4], [5], [6] と同様に，プロセスファイルシステムより取得したリソース消費量を消費電力見積りのパラメータとしている．プロセスファイルシステムにはリソース消費量がアプリケーションごとに格納されているため，スマートフォン全体の消費電力量から計測対象となるアプリケーション自体の消費電力量を分離して推定できる．

本手法の分析手順を説明する．まず，開発者は開発するアプリケーションにログ収集のための仕組みを導入する．後述するようにこのログ収集機能は実利用時において低いオーバヘッド，かつ，開発時においては自動的に導入できる必要がある．次に，このログ収集機能付きの計測対象ターゲットアプリケーションをユーザに配布し，利用させる．ユーザが配布されたアプリケーションを利用しているときに，リソース消費量を含む消費電力プロファイリングに必要な情報を収集し，ログを生成する．生成されたログはログ収集サーバに集約される．集約されたログからリソース消費量を抽出し，電力モデルを用いて消費電力を推定する．集約したデータと推定した消費電力を分析，可視化して開発者に提示する．

提案手法を使用することで以下のような分析が可能になる．まず，不特定多数の被験者に使ってもらうことで試験したことがない利用状況や環境下での消費電力を可視化できる．また，各ユーザの使い方の違いに着目したソフトウェアのチューニングが可能になる．そして，想定外の利用状況における異常な電力消費を検知できる．

3.2 選択的リソース消費ログ収集手法

本節では，運用時消費電力解析を実施するために必要となる，リソース消費量に関するログを収集する方法を説明する．まず，リソース消費量を収集する手法が満たすべき性質について述べ，次に，それらの性質を満たす選択的リソース消費ログ収集手法を提案する．

3.2.1 ログ収集方法の要件

消費電力解析のために，既存のアプリケーションにリソース消費量を取得するためのコードを埋め込み，取得したリソース消費量のログをスマートフォン内に蓄積する．

その後，取得したリソース消費量データをサーバに集約する．そのため，アプリケーション単体で実行する場合と比べ，CPU，通信デバイス，ファイルシステムをより多く使用することになり，アプリケーションの実行時間が延び，電力消費を増やし，スマートフォンの利用可能時間を減少させる．運用時消費電力解析ではユーザが実際に利用するアプリケーションに，リソース消費ログを収集するコードを追加することになるため，できる限りアプリケーション本来の動作とは関係のないこれらのオーバヘッドを減らす必要がある．

リソース消費ログ収集を遂行するために必要なオーバヘッド要因としては，以下の 4 つがある．

(1) リソース消費量を取得するコードを適切なタイミングで実行するための実行管理のために必要な CPU 時間
(2) リソース消費量をプロセスファイルシステムから取得するために必要な CPU 時間
(3) 取得したリソース消費量をログとして保存するために必要な CPU 時間，および，ファイルシステムへの書き込み
(4) リソース消費ログをサーバに集約する CPU 時間，ファイルシステムからの読み込み，および，通信デバイスによる送受信

従来手法 [6] のような Dalvik VM 付属のプロファイラを改造して用いる場合のように，すべてのメソッド呼び出し時にリソース消費量を取得してログ出力する場合には，リソース消費量を取得する際に判断が必要ないため (1) のオーバヘッドは比較的小さい．しかし，リソース消費量の取得回数が膨大になるため (2)，(3) のオーバヘッドによりアプリケーションの動作が遅くなる．また，リソース消費ログが大きくなるため (4) のオーバヘッドが大きくなり，スマートフォンの利用可能時間を著しく減少させる．そこで，運用時消費電力量解析のためのリソース消費ログ収集には，(1) のオーバヘッドがなるべく軽量で，かつ，必要な粒度，必要な場所で選択的にログを取得しログの取得回数を削減することでログの取得回数，および，ログの量を減少させて (2)，(3)，(4) のオーバヘッドを軽減する手法であることが望ましい．

提案手法では上記のオーバヘッドを低減し電力消費を減らすため，Android アプリケーション画面ごとのリソース消費を取得する．Android アプリケーションは複数の画面を持ち，それぞれが特定の機能をになっている．画面の実行順序から，既存手法より大まかではあるがアプリケーションの使われ方を知ることができる．この場合，画面が切り替わるタイミングでログ取得を行うので，数秒から数十秒の間隔でログ取得が行われるとする．既存手法 [6] はメソッド実行ごとにログ取得を行うので，取得間隔は数ミリ秒であるとすると，取得機会をおよそ 1,000 分の 1 から 10,000 分の 1 に削減できる．さらに，詳細な分析情報が必要となりメソッドが消費したリソース消費量を取得する場合には，既存手法を用いて分析を行う．あるいは，独自に定義したメソッドや HTTP 通信など大きな電力消費をともなうメソッドに絞ってログ取得を行い，取得回数を抑える．

3.2.2 AspectJ を利用したログ収集

3.2.1 項で述べた性質を満たすために，消費電力プロファイリングに必要なリソース消費ログ取得のためのコードを AspectJ [7] を用いて計測対象アプリケーションに埋め込む方法を採用する．AspectJ とは Java のアスペクト指向プログラミング [8] 向けの拡張である．アスペクト指向プログラミングとはログ取得処理のようにプログラムの様々な場所で呼ばれる処理を 1 カ所にまとめて記述することを可能にするプログラミング技法である．

AspectJ を援用することにより，すべてのメソッドを対象としてログを取得するのではなく，選択したメソッドだけを対象にログを取得できるようになる．(1) のオーバヘッドは Dalvik VM 付属のプロファイラを改造する場合と比べると若干増加すると思われるが (2)，(3)，(4) のオーバヘッドの大きな削減が期待できる．通常の Java でもログ取得を行うことはできるが，ログ取得コードを挿入する量がアプリケーションの規模に応じて増加するためコード挿入，削除の管理が難しい．また，実装次第で管理を簡単にすることもできるが，アプリケーション開発プロセスの設計段階から計画していなければそのような実装を行うことが困難である．AspectJ によるコーディングはソースコードを変更せずにログ取得処理を実装できるため管理の手間を省くことができ，開発プロセスのコーディング段階においても容易にログ取得機能を実装できる．

4. モジュール利用ログの定義

本手法ではアプリケーションが持つ機能ごとに電力消費を分析する．アプリケーションのある機能をになうプログラムをモジュールと呼ぶ．本章では Android アプリケーション中の処理の分類と，分類に必要なログについて解説する．

4.1 アクティビティのログ

Android アプリケーションはある機能を実現する画面を複数持つのが一般的な構成となっている．Android のアプリケーションフレームワークにはアクティビティという画面を記述するコンポーネント（アプリケーションの主な構成要素となるクラス）が用意されており，起動時や終了時など状態遷移にあわせてコールバックメソッドを実行する．プロセスファイルシステムに格納されているリソース消費量はスマートフォンを起動してからの累計の数値となっているので，画面が表示されている間のリソース消費量を計算するために，画面が表示される直前に実行される onResume メソッドと画面が非表示になった直後に実行さ

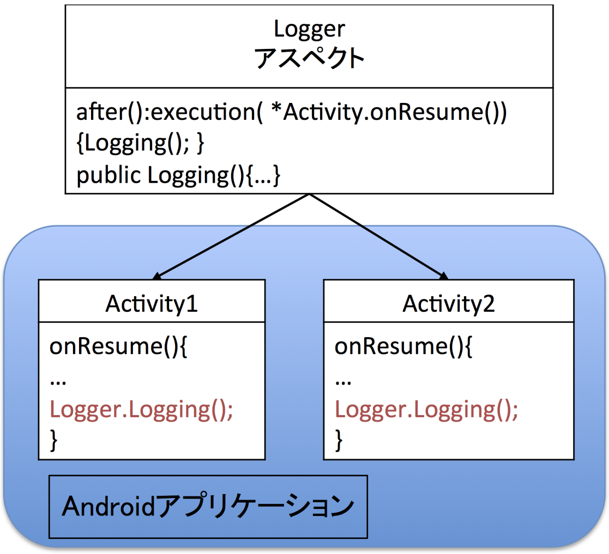


図 1 AspectJ を用いたコードの埋め込み
Fig. 1 Code embedding using AspectJ.

れる onPause メソッドがそれぞれ実行された時点でログ取得処理を行う．

図 1 に AspectJ によるコード埋め込みのイメージ図を示す．AspectJ には通常の Java にはないアスペクトというオブジェクトが追加されている．アスペクトの中にはコードを埋め込む箇所を指定する条件と埋め込むコードを記述する．条件のことをポイントカット，コードのことをアドバイスと呼ぶ．コンパイル時，ポイントカットに符合するソースコード上のすべての箇所にアドバイスに記述された処理のバイトコードが埋め込まれる．提案手法ではソースコード内で定義したメソッド呼び出し，Android ライブラリやその他アプリケーションに含まれるライブラリのメソッド呼び出し，ソースコード内の変数への代入のポイントカットを利用している．

4.2 サービスのログ

アプリケーションの画面が切り替わるとアクティビティの onPause が呼び出され，その後ある程度時間が経過したりメモリが足りなくなったりするとアクティビティは破棄される．そのため長時間にわたる定期的な処理が苦手だが，Android にはそれを補うことができるサービスというコンポーネントがある．サービスはアクティビティより生存期間が長く，バックグラウンド処理や，他のアプリケーションからメソッドを呼び出したりアドレス空間を共有したりする RPC（Remote Procedure Call）を実装できる．サービスは各種コールバックメソッドが呼び出される前後にログ取得をし，リソース消費量を計算する．

4.3 その他分析に必要な情報

4.1 節と 4.2 節ではメソッドの実行を契機としてコンポーネントの挙動を把握するためにログを取得したが，この 2 つ以外にも，スレッドやアラームマネージャを用いた定期的な呼び出しの実装が不適切であるとリソース消費が増大する要因となるため，これらが利用されたという情報も必要である．また，スマートフォンに搭載されているデバイスがどれだけ利用されたかを分析するには，

- デバイスの利用開始時に呼び出すメソッド
- デバイスを操作するメソッド
- 利用終了時に呼び出すメソッド

といったメソッドのログを取得する．その他，アプリケーションの設定状況や無線電波強度なども正常な動作を検証するのに必要な情報である．設定値を保存する仕組みは Android のライブラリで提供されている．設定値を変更するメソッドにフックして設定値のログを保存することで設定状況を把握できる．

5. 分析

本章では提案手法の典型的な適用手順について説明する．提案手法を用いた分析は

(1) モジュール単位の疎粒度な分析
(2) チューニング候補モジュールの詳細な分析
(3) 省電力化方針の検討・調査
(4) 省電力効果のシミュレーション

といった順番で行う．

まず，アクティビティやサービスなどの比較的疎粒度なモジュールごとのログを収集，サーバに集計し，アプリケーション開発者に向けてモジュールごとの積算の消費電力量のグラフを提示する．積算の消費電力量が大きいモジュールは，合計の利用時間が長く，また，画面表示 1 回あたりの消費電力量も大きいので省電力効果のインパクトも大きい．アプリケーション開発者はモジュールに対し積算の消費電力量の大きい順にチューニングの優先順位を割り当てる．

次に，優先順位の高いモジュールに対し詳細な分析を実施する．まず，モジュールの消費電力量におけるデバイスごとの内訳を開発者に提示する．開発者は消費電力量の内訳からチューニング対象となるデバイスを選ぶ．チューニングの優先順位はモジュールと同様で，全体に占める割合が大きい順に優先順位が高い．

そして，選んだデバイスをチューニングするという観点からソースコードを調査し，チューニングの可能性について検討する．デバイスの中でもディスプレイなどユーザの利用時間と消費電力量の相関が高いデバイスはチューニングによる省電力化が難しい．なぜならアプリケーションの利用時間は開発者側からコントロールが難しく，ユーザに短時間で目的を達成してもらうためにはアプリケーションデザインの変更など大規模な修正が必要となるためである．一方 CPU や無線通信デバイスといったものは，処理の中での使用量を開発者側からある程度コントロールでき

るため，チューニングが比較的容易である．

また，ユーザがアプリケーションを利用した曜日や時間帯を分析して，ユーザが利用する状況に応じたチューニングを行う方法もある．統計上多くのユーザが利用する時間帯の直前でデータの先読みをしたり，時間帯ごとの利用回数に応じた処理するデータ量の変更などを行ったりすることで，待機中のディスプレイや，余分なデータ処理によるCPU，無線通信デバイスの消費電力量を削減できる．他にも，アプリケーションに固有なパラメータを収集しておいて比較するなどといった方法もある．

チューニング方法に見通しが立ったら，収集したリソース消費量を基にチューニング後の推定消費電力量を算出し，チューニング前の消費電力量と比較して省電力効果を見積もる．このようにソフトウェアの改変前に事前に見積もることで，チューニング方針の誤りによる手戻りを削減する．

6. 評価

提案手法のログ取得環境の評価を行った．ログ取得処理のオーバヘッドを計測し，オープンソースで公開されているアプリケーションに対して実際に分析を行った．

6.1 実験環境

構築したログ取得環境を用いて実際のアプリケーションに対して消費電力の分析を行った．今回分析したのはCrowdroidというオープンソースのTwitterクライアントである．Twitterとは最大140文字以内の短い投稿を世界中のユーザと共有するサービスである．1つの投稿のことをTweetと呼ぶ．Twitterクライアントは自分が登録しているユーザからのTweetをまとめて表示するためのソフトウェアである．

ログ取得処理を埋め込んだアプリケーションを13人の被験者にのべ51時間利用してもらいログを収集した．端末はGalaxy Nexusを用い無線LAN通信のみを使うという条件で利用してもらった．ログに含まれる情報は，画面表示時刻，メソッド名，CPU使用時間，無線通信時間である．またCrowdroid固有のパラメータとしてTweet取得数もログに含む．今回の評価においてのログの粒度をアプリケーションの画面ごととする．

提案手法では，スマートフォンに搭載された各デバイスが消費する電力量ではなく，各デバイスが稼働する際にバッテリからスマートフォンへ供給される電力量を推定している．また，バッテリから供給される電源の電圧を一定と仮定したうえで，消費電力量の指標としてバッテリの消費電流量を用いる．Galaxy Nexusの消費電流量モデル係数は，文献[4]と同様にベンチマーク実行時の各種デバイスの利用量と端末の消費電流量の重回帰分析より求めた．無線LANの送信と受信に関する電力については，パラメータを各種スループットからほとんど等価な値である通信時間へ変更した．

表1 消費電流量見積りパラメータ
Table 1 Parameters of power model.

V_{offset}（ディスプレイ他）	表示時間 [sec]
V_{CPU}（CPU）	周波数 [KHz] × 稼働率 [%]
V_{WLAN}（通信）	通信時間 [sec]

表2 消費電流量見積りパラメータの係数
Table 2 Coefficients of power model.

C_{offset}	$1.648e^{-1}$
C_{CPU}	$1.910e^{-7}$
C_{WLAN}	$2.005e^{-1}$

今回のモデル式と実測の電流量の比較は行っていないが，文献[4]の評価実験と同様に約10%の誤差で推定できるモデルであると仮定する．提案手法を用いることで文献[4]で示されているリソース消費量をすべて収集可能であるが，ストレージの読み込みと書き込みに関する電力は影響が小さいため計測を行わなかった．もう1つ無線LANの有効化に関する電力に関してはOSや他のアプリケーションと共有しており特定のアプリケーションの消費分を分離できないため同じく計測を行わなかった．今回の評価実験ではCrowdroidが主に利用するCPU，無線LAN，その他の消費電力量を推定した．

式(1)のモデル式を基に消費電流量[Asec]を計算する．パラメータVを**表1**に，係数Cを**表2**に示す．消費電流量見積りパラメータである表示時間は画面起動時と画面終了時の時刻の差で計算する．CPU使用率はプロセスファイルシステム（/proc/stat）に記録されているCPUの動作モード別利用時間を用いた，全体に占めるuserモード（アプリが使用した時間）の割合と周波数から計算する．無線通信時間はHTTP通信を開始するメソッドとデータを受信したときのコールバックメソッドのそれぞれの実行された時刻の差で計算する．

6.2 ログ取得処理のオーバヘッド

提案手法のログ取得処理にかかる時間を測定して，ユーザの操作への影響を検証する．検証のためにログ取得処理の前後それぞれで時刻を取得し，Androidのデバッグ用ツールであるLogCatに時刻データを送信するよう変更を加えた．決まった文字列を画面に表示する単純なアプリケーションを用意し，変更されたログ取得機能を実装した．時間を測定した処理はアクティビティのコールバック呼び出し時のプロセスファイルの読み込み処理，HTTP通信開始時の時刻取得処理，HTTP通信終了時のファイル保存処理の3つである．それぞれの処理を10回実行し平均処理時間を算出した．端末はXperia Z Ultraを使用した．

表 3　ログ取得処理時間の平均
Table 3　Average runtime of logging processing.

処理内容	平均処理時間 [ms]
プロセスファイル読み込み	27.7
時刻取得	1 未満
ファイル保存	5.1

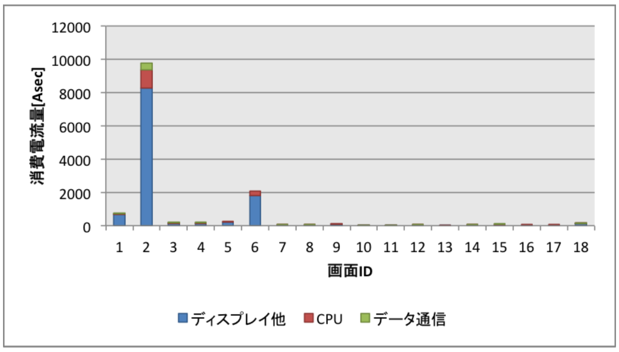


図 2　画面ごとの消費電流量［被験者全員分］
Fig. 2　Aggregate current consumption each activity classes.

実験結果を**表 3** に示す．ここでは，先行報告 [9] で示した AspectJ による実行時の処理遅延の増分 3.9 [ms] に加わる，デバイスの各部の稼働状況の取得における処理時間の増分を示す．表 3 で最大であるプロセスファイルの読み込み処理で約 30 [ms] という結果になった．その他を足し合わせても 35 [ms] に満たない．消費電力の推定に必要なこの稼働状況は 4.1 節で議論したように，onResume，onPause メソッドで実行されるため，アプリケーションの画面が切り替わる度に 2 回実行される．ユーザの画面切り替えによるこの操作間隔はばらつきが大きいが，これを数秒から数十秒と仮定すると，ログ取得の占める割合は 1%から 0.1%程度となり，ユーザの利用への影響は小さいといえる．

6.3　アクティビティのチューニング

被験者のログから消費電流量を分析した．**図 2** は被験者全員分のログから分析したアクティビティごとの消費電流量のグラフである．一番消費電流量の大きい画面は，画面 ID 2 であり，自分がフォローしているユーザの Tweet を表示するという基本的な機能を果たす画面である．画面 ID 2 はアプリケーションの起動時に必ず表示される画面でもあるため使用頻度が高く，消費電流量が大きくなる．画面 ID 2 の電力消費はディスプレイその他のデバイスが消費する電流量が大部分を占めていることが分かったが，ディスプレイの表示時間はアプリケーション開発者側からのコントロールが難しいため省電力化の対象から除外する．CPU に関しても検討を行ったが改善は難しく，3 つ目のデバイスである通信に着目して分析することにした．当該画面は Tweet を表示するための画面であり，主な通信内容はサーバからの Tweet 取得である．1 回の画面表示あたりの Tweet 取得数を通信を評価する指標として分析する．

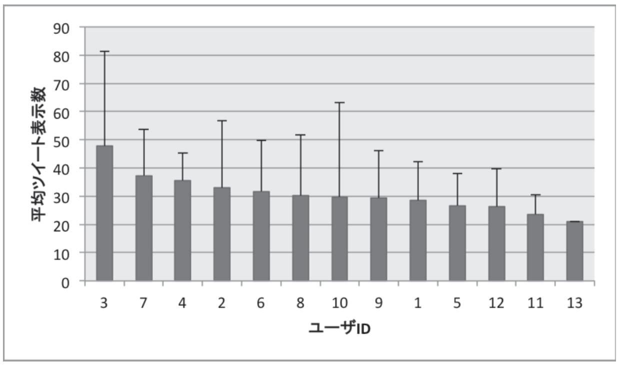


図 3　画面 ID 2 の平均 Tweet 取得数
Fig. 3　Average tweet count of activity id:2.

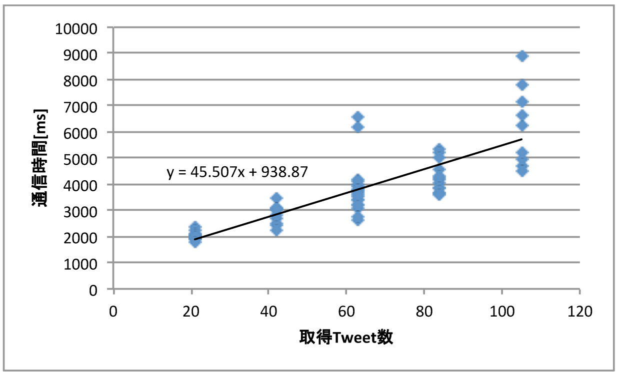


図 4　最適ロード時間の測定
Fig. 4　Measurement of data communication time.

図 3 に画面 ID 2 についてのユーザごとの 1 回の画面表示あたりの Tweet 取得数の平均と分散のグラフを示す．ユーザごとに平均の Tweet 取得数が異なるが，多くのユーザは Crowdroid の標準設定である 21 件の Tweet よりも多くの Tweet を 1 回に読み込んでいることが分かる．そのため，ほとんどのユーザは 2 回以上更新していることになるため，更新回数を減らせばダウンロードの待ち時間で消費する電流量，サーバへの接続で消費する電流量を削減できる．

チューニング方針が決まったら，電力消費の削減効果について事前評価を行う．Crowdroid は 1 度の更新で 21 件の Tweet をダウンロードするようにサーバに要求する仕様になっているが多くのユーザには追加読み込みが必要なことが分かった．そこで，今回の実験では Tweet 取得処理回数を抑えた場合の消費電力削減効果を見積もることとした．

このシミュレーションにおいて，ユーザが必要とする Tweet の数 x が収集したログからあらかじめ分かっていると仮定する．**図 4** は 1 度の更新でダウンロードする Tweet の件数を変化させて再度ビルドした Crowdroid から通信を行い，通信に要した時間をプロットしたグラフである．実線は線形近似曲線を表している．x 件の Tweet を 1 度の更新でダウンロードするのにかかる最適化後の通信時間 T_{opt} を $T_{opt} = 45.507x + 938.87$ [ms] で表す．この式からユーザが必要とする Tweet の数を最適化された方式で取得した

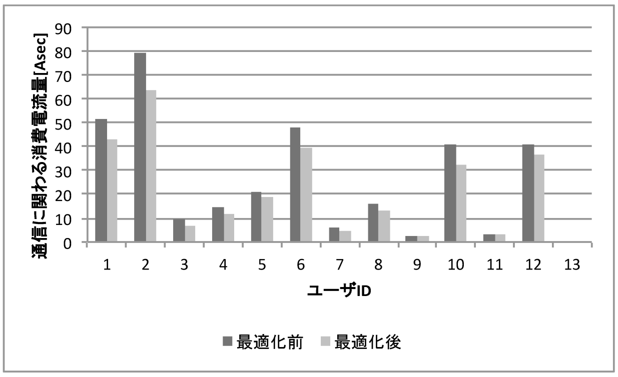


図 5 Tweet 表示数の最適化シミュレーション
Fig. 5 Optimization simulation of tweet number.

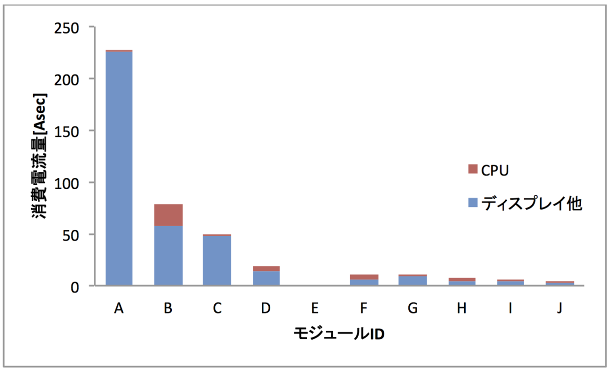


図 6 アクティビティとサービスの消費電力分析
Fig. 6 Current consumption analysis of activities and services.

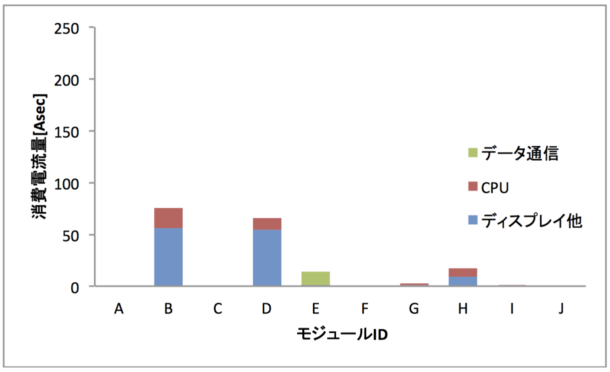


図 7 チューニング後の消費電流量
Fig. 7 Current consumption after optimization.

場合の通信時間が推定できる．最適化された通信に関する消費電流量 P_{optW} は推定消費電流量 P_{WLAN}，端末から収集した実際の通信時間 T と T_{opt} により

$$P_{optW} = P_{WLAN}\frac{T_{opt}}{T} \tag{2}$$

という式で計算できる．

図 5 に実際の消費電流量とチューニングした消費電流量の比較を示す．シミュレーションの結果，通信回数の最適化により通信に関する消費電流量をユーザごとの平均で 2 割程度削減できるという結果が得られた．これはスマートフォンの消費電力全体で見ると 2%程度の改善ではあるが，ここで示した最適化は利用者から見たアプリの挙動を変えないためユーザビリティに影響を与えずに実現されている．実環境における利用者の平均的な利用状況とその消費電力の関係を提案手法により可視化することで，開発者は課題を発見し消費電力の点からの寄与を考えながら修正を実施する判断が容易になった．本節ではこのチューニングの過程を具体的に述べた．

6.4 サービスのチューニング

次にサービスを計測対象に含めて分析を行った．2 名の被験者に 1 カ月間，Crowdroid を利用してもらった．消費電流量の上位 10 個のモジュールを**図 6** に示す．なお，このチューニング前の実験において取得したデータ通信の消費電流量の計測データに不備があったため記載から外した．また，モジュール ID E は B の通信部を担うモジュールであり**図 7** と同様の結果になると思われる．よって比較のために図 6 に記載する．

B のモジュールは図 2 における ID 2 の画面であるがそれよりも消費電流量の大きいモジュール A が存在する．モジュール A は Twitter の新着メッセージを通知する機能の中で呼び出されるものである．ソースコードを調査するとモジュール A は Crowdroid の起動時に一定周期ごとに呼び出されるように設計されていたことが判明した．通知機能は Crowdroid の設定画面から無効化できる設計になっている．被験者に新着通知機能を利用していたか尋ねたところ，2 人とも利用していないということだった．つまり，Crowdroid は設定情報を参照せずにモジュールを呼び出している．これではユーザに必要のない処理が行われることになるので，通知機能モジュールを呼び出す際に設定情報を参照して条件分岐処理を行うようチューニングを施した．チューニングしたアプリケーションを更に 1 カ月間利用させモジュールごとの分析を行った（図 7）．新着通知のための処理が抑制されたことによりモジュール A と，同じく新着通知処理の中で実行されるモジュール C，F の電力消費を削減できた．提案手法を用いて分析を行ったことで，開発者が想定できなかった電力消費を検知できた．

7. おわりに

Android アプリケーションの消費電力に関するチューニングを支援するために，ユーザが実際に利用している状況を考慮した運用時消費電力分析手法を提案した．本提案手法では，多数のユーザがアプリケーションを実際に利用しているときに消費電力推定に必要なリソース消費量を取得し，リソース消費ログを作成する．このログを最小限のオーバヘッドで収集するために AspectJ を用いた選択的リソース消費ログ収集方法を提案した．さらに，収集できたログを用いて消費電力量分析を支援するための手順を提案した．提案手法を評価すべく，Crowdroid という Twitter

クライアントを対象に消費電力量を分析し，チューニングするケーススタディを実施した．その結果，電力消費の多い画面を特定でき，さらにその画面について通信量に関する分析を実施することで，チューニング方針を立てることができた．その方針の効果を，収集したデータを利用して評価し，チューニング方針の効果を確認できた．また，サービスに関する分析を行い，バックグラウンド処理を行うモジュールの巨大な電力消費を発見できた．チューニングを施して分析を行い，無駄なモジュールの起動を抑制できたことを確認した．

今後の課題として，実際の電力見積精度の検証や，省電力チューニングのシミュレーション結果と実際にチューニングを施したアプリケーションの実環境における消費電力分析結果との比較を行いシミュレーションの妥当性を検証することがあげられる．

参考文献

[1] Cignetti, L.T., Komarov, K. and Ellis, C.S.: Energy estimation tools for the Palm, *Proc. 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems* (*MSWIM '00*), pp.96–103, ACM (2000).

[2] Zhang, L., Tiwana, B., Qian, Z., et al.: Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones, *Proc. 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (*CODES/ISSS '10*), pp.105–114, ACM (2010).

[3] Killian, J.T.: Processes as Files, *USENIX Summer Conf.*, Salt Lake City (1984).

[4] Kaneda, Y., Okuhira, T., Ishihara, T., Hisazumi, K., Kamiyama, T. and Katagiri, M.: A Run-Time Power Analysis Method using OS-Observable Parameters for Mobile Terminals, *2010 International Conference on Embedded Systems and Intelligent Technology* (*ICESIT 2010*), Vol.1, pp.39–44 (2010).

[5] Mittal, R., Kansal, A. and Chandra, R.: Empowering Developers to Estimate App Energy Consumption, *Proc. 18th Annual International Conference on Mobile Computing and Networking* (*Mobicom '12*), pp.317–328, ACM (2012).

[6] 部谷修平，久住憲嗣，石原 亨，神山 剛，中西恒夫，福田晃：Android 携帯端末アプリケーション向け消費電力プロファイリング手法，情報処理学会研究報告，EMB, 組込みシステム，Vol.2011-EMB-20, No.2, pp.1–6 (2011).

[7] The Eclipse Foundation, The AspectJ Project, available from ⟨http://www.eclipse.org/aspectj/⟩ (accessed 2013-05-04).

[8] Kiczales, G., Lamping, J., Mendhekar, A., et al.: Aspect-Oriented Programming, *Proc. European Conference on Object-Oriented Programming* (*ECOOP*), *LNCS*, Vol.1241, Springer Verlag (1997).

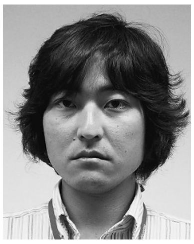
[9] 古庄裕貴，久住憲嗣，神山 剛，稲村 浩，中西恒夫，福田晃：Android アプリケーションのための運用時消費電力プロファイリング環境構築に向けて，情報処理学会シンポジウムシリーズ，マルチメディア，分散，協調とモバイルシンポジウム（DICOMO '12），Vol.2012, No.1, pp.2090–2095 (2012).

古庄 裕貴

2011 年久留米工業高等専門学校機械・電気システム工学専攻卒業．2013 年九州大学大学院システム情報科学府修士課程修了．現在，九州大学大学院システム情報科学府博士後期課程在学中．ソフトウェアの省電力化開発手法の研究に従事．

久住 憲嗣 （正会員）

2004 年九州大学大学院博士課程修了．2008 年から九州大学システム LSI 研究センター准教授．組込みソフトウェア開発方法論の教育，研究に携わる．特にモデル駆動開発や低消費電力化技術に関する研究に従事．ACM 会員．



神山 剛 （正会員）

株式会社 NTT ドコモ先進技術研究所勤務．2006 年東京大学新領域創成科学研究科修士課程修了．同年（株）NTT ドコモ入社．モバイルコンピューティング，ソフトウェア省電力化，分散システムに関する研究に従事．

稲村 浩 （正会員）

NTT ドコモ先進技術研究所勤務．1990 年慶應義塾大学大学院理工学研究科修士課程修了．同年日本電信電話（株）入社．1994～1995 年にカーネギーメロン大学計算機科学科にて訪問研究員．1998 年より NTT ドコモ．2010 年慶應義塾大学大学院開放環境科学専攻後期博士課程単位取得退学．博士（工学）．モバイル環境におけるシステムソフトウェア，トランスポートプロトコルに関する研究開発に従事．電子情報通信学会，ACM，IEEE 各会員．

中西 恒夫　（正会員）

1993年大阪大学工学部通信工学科卒業．1998年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．博士（工学）．1998年より奈良先端科学技術大学院大学情報科学研究科助手．2002年より九州大学大学院システム情報科学研究院准教授．その間，2011年より11カ月，独ホーエンハイム大学農業工学研究所客員研究員．2014年より福岡大学工学部電子情報工学科教授．通信機器，自動車，農業等，各種産業分野へのソフトウェア/システム工学の適用に研究的関心．ACM，IEEE，自動車技術会，電子情報通信学会，農業情報学会，農業食料工学会各会員．

福田 晃　（フェロー）

1977年九州大学工学部情報工学科卒業．1979年同大学大学院工学研究科修士課程情報工学専攻修了．同年日本電信電話公社（現NTT）武蔵野電気通信研究所入所．1983年九州大学助手．1989年同大学助教授．1994年奈良先端科学技術大学院大学教授．2001年九州大学大学院システム情報科学研究院教授，2008年九州大学システムLSI研究センター長（兼任），現在に至る．工学博士．組込みソフトウェア，ユビキタスコンピューティングに関する研究に従事．情報処理学会研究賞（1990年），Best Author賞（1993年）等を受賞．電子情報通信学会，ACM，IEEE Computer Society，日本OR学会各会員，「NPO法人九州組込みソフトウェアコンソーシアム（QUEST）」理事長．

