Vol.2011-SLDM-149 No.2
Vol.2011-EMB-20 No.2
2011/3/18

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

部谷修平[†1] 久住憲嗣[†2] 石原亨[†2]
神山剛[†4] 中西恒夫[†3] 福田晃[†3]

本論文では AndroidOS 上で動作するアプリケーションの省電力化のためのプロファイリング手法を提案する．従来の消費電力分析技術は分析のために対象システムを稼働させなければならなかったり，計算負荷が大きく手軽ではなかった．またシステム全体の電力しか分析できず，ソフトウェアのクラスやメソッドレベルでのボトルネックの発見には役に立たなかった．本手法はリソース消費ログから消費エネルギーを見積もる軽量な線形モデル式をもとにしているためシミュレーションで手軽にできる．またメソッド単位で電力を分析できるという特徴を持つ．本手法の中で電力分析のための侵襲性の異なる 2 種類のログの取り方を示す．本論文の最後で提案手法についてログの取得方法，誤差と侵襲性について評価した結果，それぞれのログの取得方法について精度と侵襲性の関係を明確化できた．

An Enegy Profiling Method for Android Application

Syuhei Hiya,[†1] Kenji Hisazumi,[†1] Toru Ishihara,[†1]
Takeshi Kamiyama,[†4] Tsuneo Nakanishi[†1]
and Fukuda Akirra[†1]

This paper proposes a method for profiling power consumption of applications that run on an Android OS. Many power analysis techniques require running actual system and/or large computational load to analyze the power consumption. Furthermore these approaches cannot figure out bottlenecks at the level of classes and methods since they can only analyze the system-wide power consumption. Our method has features that can calculate the power consumption based on lightweight linear model from resource consumption logs, and can analyze the power consumption more fine grain. This paper describes two ways to obtain the logs that are different types of invasive techniques, and clarifies trade-offs between accuracy and invasive about method for logging.

1. はじめに

スマートフォンが普及し始め，携帯端末の使われ方が変化している．今まではメールや電話など比較的短時間の使用であったのに対して，スマートフォンでは WEB ブラウザや，動画再生など，長時間使用されるアプリケーションが増えている．これにより携帯端末の消費電力量が問題になってきている．アプリケーションの使用する消費電力を削減するためには消費電力のプロファイリング技術が重要である．

消費電力の見積もり技術はハードウェアシミュレーション[1)]，ソフトウェアシミュレーション[2)3)]，電力測定器を用いた方法[4)]，パフォーマンスカウンタを用いた方法[5)] の 4 つに分類できる．従来の電力解析手法では，準備することが困難な計測機器を必要としたり，電力測定工数が必要である．さらにシステム全体の消費電力を測定するだけで，ソフトウェア機能ごとに消費電力を分割できないために消費電力のボトルネックの特定には不十分である．

本研究は特に Android アプリケーションにおいて消費電力プロファイリング技術を確立することで開発者がアプリケーションの消費電力に関するボトルネックを発見するのを手助けし，省電力化を補助することを目的としている．アプリケーションを省電力化するためには，まず電力を開発者がチューニングしやすい粒度に分ける必要がある．また分析はアプリケーション開発者が手軽にできるようにシミュレーションで行えるようにする．

提案手法は Android OS を搭載した無線通信携帯端末のエミュレータで使用できる手法で，汎用的な開発環境である Eclipse のプラグインとして開発する．アプリケーション実行時に生成されるログと，第 2 節で説明する消費電力モデル化技術[6)] を用いて消費電力を細かい粒度で分析する．その際システム対する侵襲度の異なる 2 種類のログの取得方法を提案する．

本論文の構成は以下である．第 2 節で既存の消費電力分析技術，消費電力モデル化技術に

†1 九州大学工学部電気情報工学科
Dept. of EE and CS, School of Eng.,Kyushu Univ.
†2 九州大学システム LSI 研究センター
System LSI Research Center, Kyushu Univ.
†3 九州大学大学院システム情報科学研究院
Graduate School of Information Science and Electrical Engineering, Kyushu Univ.
†4 株式会社 NTT ドコモ先進技術研究所
Research Laboratories, NTT Docomo

Vol.2011-SLDM-149 No.2
Vol.2011-EMB-20 No.2
2011/3/18

ついて説明し，第 3 節で提案手法である消費エネルギー分析手法，プロファイラの概要について説明する．特にログの取得方法とその分析方法を 2 種類提案する．第 4 節ではそれらの手法の精度と侵襲性を評価する．第 5 節で今回の結論を示し，第 6 節で今後の課題を述べる．

2. 関 連 技 術

本節では既存の消費電力分析技術について説明する．

2.1 消費電力分析技術

電力解析の既存手法として PowerScope と呼ばれる電力解析器を使った手法[4] がある．これはモバイルアプリケーションの電力解析についての技術である．SystemMoniter，EnergyMonitor，EnergyAnalyzer の 3 つのシステムで，プロセスの消費電力を分析する．この解析器を用いてムービープレーヤーの消費電力を 46 %削減している．しかし，この方法は電力を測定するために大掛かりな機械を使用するため，一般のアプリケーション開発者が手軽に行うことができない．よって Android アプリケーションの開発に適用するのは困難である．またこの方法ではプロセスごとの電力しか見積もることができない．

2.2 消費電力モデル化技術

研究 6) では CPU 使用率や通信量などの OS から取得できるパラメータのみを利用して実機での電力測定をせずに消費電力を見積もる手法を提案している．既存手法の多くが CPU のみを消費電力解析の対象にしていたのに対し，本手法は無線通信端末全体を対象としている．さらに消費電力を CPU 利用率，無線通信量，ストレージ転送量から実測値との誤差 6%程度で低負荷で見積もることができる．以下の線形モデル式で電力を見積もる．

$$P_{estimate} = C_0 + \sum_{i=1}^{n} C_i P_i \quad (1)$$

$P_{estimate}$ は消費電力の見積もり値，P_i は線形モデルのパラメータ，C_i は各パラメータの係数，C_0 は定数項を表す．パラメータの例として CPU なら CPU の負荷率（CPU 時間），WiFi デバイスならデータ通信量があげられる．この式の C_i パラメータの係数は重回帰分析によってあらかじめ求めておく必要がある，デバイス固有の値である．

3. 消費エネルギー分析手法

本節では，提案手法である消費エネルギー分析手法について説明する．

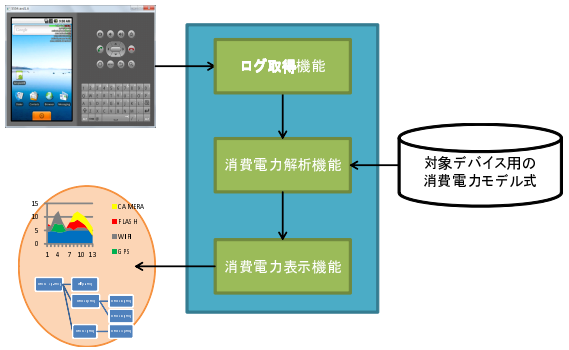

図 1 プロファイライメージ図
Fig. 1 Profiler Image

3.1 消費エネルギープロファイラ概要

消費エネルギープロファイラのシステムの概要について説明する．**図 1** はプロファイラのイメージ図である．このシステムは仮想端末からログを取得し，消費電力を解析，表示する．ログ取得機能では仮想端末からデバイスのスループットやメソッド実行情報などのログを取得する．消費電力解析機能では，それらのログをもとに対象デバイス用の消費電力モデル式を用いて電力を解析する．消費電力表示機能では解析結果をもとに消費電力を可視化する．

3.2 電力分析方法

提案手法は式 (1) の消費電力の線形モデル式を前提としている．メソッドごとなどの消費エネルギーを分析したいプログラムの断片ごとに P_i を計測すると，その断片における消費エネルギー量を求めることができる．またそれらのプログラムの断片における消費エネルギー量をすべて加算するとシステム全体の消費エネルギー量となる．そのためプログラム断片ごとのリソース消費量の取得が正確にできればそれらで消費される消費エネルギーを分析することができる．

消費エネルギーの分析に必要な情報は消費エネルギーを解析したいデバイスの分析対象時間のリソース消費量である．さらにメソッドごとのリソース消費量の情報が取得できれば式 1 を用いて，メソッドごとの消費エネルギーを分析することができる．

コールツリー作成について説明する．コールツリーを作成するためにはメソッドの親子関係が必要である．親子関係はメソッドの実行順序関係を明らかにすることで構築できる．メソッド実行順序の関係はメソッドの呼び出し時刻と終了時刻から構築可能なので，それらが

Vol.2011-SLDM-149 No.2
Vol.2011-EMB-20 No.2
2011/3/18

あればコールツリー作成に必要な情報はとしては十分である．

消費電力の時間軸グラフ作成について説明する．消費電力の時間軸グラフはアプリケーション起動中に呼び出されたメソッドのスループットと呼び出し時刻，終了時刻により作成できる．

3.3 可視化方法

可視化手法について説明する．デバイスごとの消費電力表示，消費電力の時間軸表示，消費エネルギーコールツリーの表示の 3 つの可視化手法を実現する．

図 2 はトップビューである．トップビューではアプリケーションの開始から終了までのデバイスごとの消費電力の割合を表示する．これにより開発者はどのデバイスが消費電力のボトルネックになっているのかを知ることができる．

図 3 は時間軸グラフである．消費電力の時間軸グラフはアプリケーションの開始から終了までの各時刻におけるデバイスごとの消費電力を表示する．GPS やカメラなどメソッドと非同期に起動するデバイスの消費電力を可視化することに向いている．これにより開発者はどの時刻に消費電力が大きくなっているかを知ることができ，また不要なデバイスが使用されていないかを確認することができる．

図 4 の消費エネルギーのコールツリーでは，アプリケーションのコールツリーに消費エネルギーを付加して表示する．主に無線通信デバイスやストレージなどのメソッドと同期して消費電力の発生するデバイスの可視化に向いている．これにより開発者はどのメソッドが消費エネルギー的なボトルネックになっているかを知ることができる．

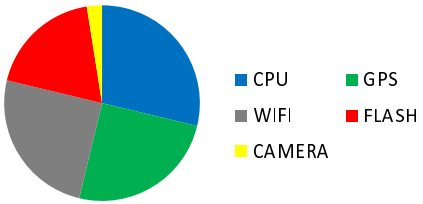

図 2 トップビュー
Fig. 2 TopView

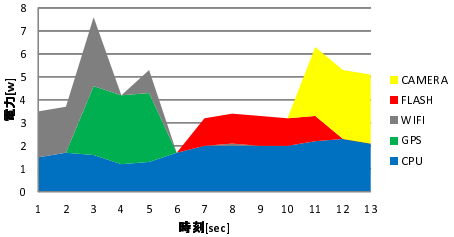

図 3 時間軸
Fig. 3 Time Line

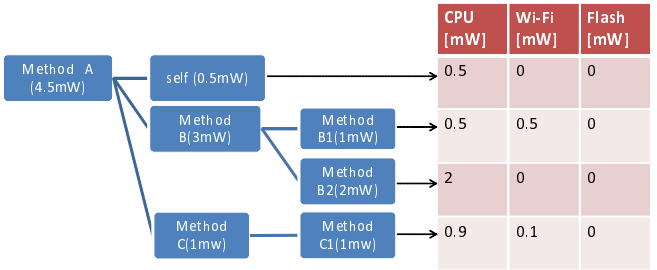

図 4 コールツリー
Fig. 4 Call Tree

3.4 ログ取得方法

消費エネルギーの分析に必要な情報として，メソッドが呼び出された時刻，メソッド名，メソッドが終了した時刻，メソッドが使用したリソース量の 4 つがある．メソッド名は開発者がメソッドを識別するために必要である．このうちメソッド名，メソッドが呼び出された時刻，メソッドが終了した時刻は既存のプロファイラが必要とする情報と同じである．メソッドが使用したリソース量は OS から取得したデータをもとに解析する，デバイスを操作するメソッドにフックをするなど，プラットフォームによって手法を検討する必要がある．

4. Android における実装

提案手法を Android において実装した．実装内容について説明する．

4.1 システム概要

消費エネルギープロファイラの Android において実装した．実装内容について概要を説明する．これらの機能は基本的に Eclipse のプラグインとして動作することを想定している．論文 6) では実験対象の端末として Nokia 社の N810 を用いているが，提案されているモデル生成フローを用いて他のデバイスの消費電力モデル式を生成することが可能である．本稿では Sony Ericsson の Xperia 用の消費電力モデル式を用いる．プロファイリングは Android のエミュレータ上で動くアプリケーションに対して行う．

システムはリソースログ取得部，メソッド毎のリソース消費割合算出部，メソッド毎の消費電力算出部からなる．リソースログ取得部ではリソースログの不要な情報を排除し，必要なログを抽出する．次にメソッド毎のリソース消費割合算出部では，トレースログとリソースログからメソッド毎の消費割合を算出する．算出の方法は 4.3 項と 4.4 項で詳しく説明す

る．メソッド毎の消費電力算出部では，メソッド毎の消費割合と消費電力モデル式用パラメータ係数を用いてメソッド毎の消費電力を算出する．最後に表示部で消費電力の可視化を行う．

今回消費電力解析の対象としているデバイスは CPU，Wi-Fi デバイス，Flash デバイス，GPS デバイス，カメラデバイスの 5 つである．Wi-Fi，Flash などは消費電力については読み書きを行ったメソッドに責任を割り当てる．GPS やカメラなどはそれを起動したメソッドに責任を割り当てる．

4.2 ログ取得概要

3.4 項で説明したように，消費エネルギーの解析には，メソッドが呼び出された時刻，メソッド名，メソッドが終了した時刻，メソッドが使用したリソース量の情報が必要である．

このうちメソッドが呼び出された時刻，メソッド名，メソッドが使用した CPU リソース量メソッドが終了した時刻はトレースログに含まれる．トレースログは電力解析対象アプリケーションのメソッド情報やメソッド呼び出し情報であり，Android に付属している既存のプロファイラ hprof によって出力される．hprof ではメソッド呼び出し時刻は gettimeofday() で取得している．トレースログはアプリケーションから android.os.Debug クラスを用いて取得することができる．

その他のデバイスのリソース情報はリソースログに含まれる．リソースログについては 2 種類の取得方法を提案する．

- /proc ファイルシステムから取得する方法
- 低レベル API にフックして取得する方法

それぞれの取得方法で分析方法が異なる．1 つ目の方法は Linux カーネルの/proc ファイルシステムからリソース情報を取得する方法である．この方法で取得できるログは時刻とシステム全体のリソース消費量なので，メソッドが使用したリソース量を算出する必要がある．具体的な方法については 4.3 項で説明する．以降このログを OS リソースログと呼ぶ．

2 つ目の方法は低レベル API にフックして，直接スループットの情報を得る方法である．この方法ではメソッドが使用したリソース量は容易に分析できる．具体的な方法については 4.4 項で説明する．以降このログをメソッドリソースログと呼ぶ．

これらの取得方法では Android の Logcat という機能を用いて出力する．Logcat は android.util パッケージの Log クラスを用いて使用することができる．この機能を用いて出力されるログは Android SDK に付属する ADB(Android Debug Bridge) ツールで取得できる．これは Android エミュレータと開発用 PC でのデータのやり取りをするためのものである．

4.3 /proc から取得する方法

OS リソースログからメソッドごとのデバイスの消費エネルギーを分析する方法について説明する．OS リソースログは/proc ファイルシステムから取得する．/proc とは Linux システム上のリソース情報にひもづけられている仮想的なファイルシステムである．Wi-Fi デバイスは/proc/net/dev，Flash デバイスは/proc/diskstats から取得する．

シミュレーション時に，このリソース情報をアプリケーションの開始から終了まで一定時間間隔ごとに取得する．OS リソースログにはリソース量とともにデータの取得時刻を持たせる．そしてトレースログのメソッド呼び出し時刻と照らし合わせる．OS リソースログのある取得間隔でのリソースの増分をその間に発行されたメソッドに割り当てる方法を提案する．

対象区間のうち，ある 1 つのメソッドに割り当てるリソースの割合をリソース消費割合と呼ぶ．この方法は対象区間の低レベル API の実行時間の比率で OS リソースログをメソッドに割り当てる．対象区間で呼ばれた低レベルメソッドの集合を $Methods$ とする．メソッド $A(Methods \ni A)$ のリソース消費割合 A' は

$$A' = \frac{time(A)}{\sum_{x \in Methods} time(x)} \tag{2}$$

となる．これに対象時間の OS リソースログの増分をかけたものをメソッドに割り当てる． ただし time(メソッド) はメソッド実行時間を表す．

4.4 低レベル API にフックする方法

メソッドリソースログからメソッドごとの消費電力を分析する方法と，メソッドリソースログの取得方法について説明する．出力するログの内容はタイムスタンプ，コールスタック，スループットの 3 つである．タイムスタンプは gettimeofday API で取得する．

無線通信量について，この方法で取得したデータは 4.3 項の方法でとったデータ（OS リソースログ）に比べて少なめな値を示す．**図 5** に OS リソースログとメソッドリソースログのグラフを示す．縦軸が無線受信量で，横軸が時刻を表す．実際にメソッドリソースログの方が無線受信量が少なくなっているのがわかる．これは OS リソースログには IP ヘッダなどが含まれるためと考えられる．そこでメソッドリソースログを補正する必要がある．1 つの方法として OS リソースログを用いて保管したデータとメソッドリソースログのデータに対して最小二乗法を用いて，一次近似する方法がある．しかしアプリケーションの種類によって最小二乗法で算出したパラメータが変化するので，実用にはいたっていない．

Vol.2011-SLDM-149 No.2
Vol.2011-EMB-20 No.2
2011/3/18

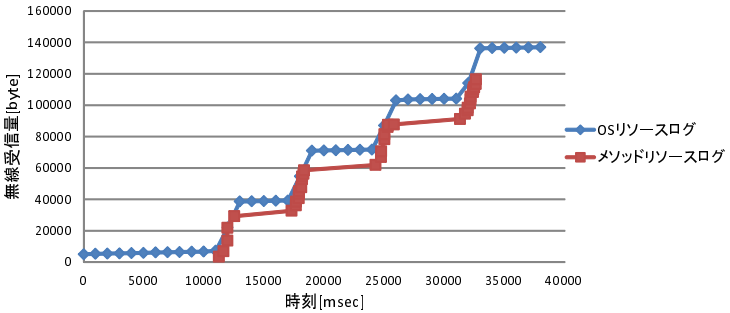

図 5 無線受信量の積算
Fig. 5 amount of radio reception

5. 評　　価

5.1 ログ取得方法と精度評価

実験環境と実験時のログの取得方法について説明する．/proc ファイルシステムからのデータは試験アプリケーションの開始と同時に別スレッドを作り，一定時間ごとにログとして記録する，また同時に gettimeofday API で取得した時刻情報も同時に付与する．低レベル API にフックして取るデータは，API が呼ばれてすぐ時刻を取得し，通信，IO 処理などが終わったところでスループットなどとともにログとして記録する．両者とも adb logcat コマンドを用いて取得する．

/proc からの情報とコールツリーを対応づける方法と低レベル API にフックする方法の比較をする．低レベル API にフックして取得した情報は最小粒度の情報で，メソッドの消費した正確な電力を見積もることができる．しかしデバイスが使用したトータルの消費エネルギーを見積もることには向いていない．/proc からの情報はシステム全体のものであるため，粒度は大きいがシステム全体の正確な消費エネルギーを見積もることができる．

/proc からログを取得する間隔と精度の関係を示すために 2 種類の誤差指標を定義する．誤差指標 1 はアプリケーション開始時から終了時までの/proc の増分と，見積もられた API ごとのスループットの積算の差である．デバイスの消費エネルギーはスループットに係数をかけるだけで求められるので，これはアプリケーションを通しての 1 つのデバイスの消費エネルギーの誤差と等しい．この誤差をトータル誤差とする．誤差指標 2 は「API にフックした場合」と「/proc からの情報とコールツリーを対応づけ補間をした場合」の同時刻に発行された API ごとのスループットの差の集合を X とすると，X の分散である．これは/proc からの情報とコールツリーを対応づける方法を実際に行った場合に適切にメソッドにリソースが割り当てられているかの指標である．この誤差を振り分け誤差とする．

表 1 **表 2** は無線通信デバイスについての実験結果である．表 1 は/proc からの情報とコールツリーを対応づける方法に関して，/proc をから取得する間隔とトータル誤差の関係である．実験用のアプリケーションは 3000msec 間隔で 5 回 www.google.com にアクセスするプログラムである．実験の結果，/proc を読む間隔が長いほど誤差が少ないことがわかった．また誤差は/proc 取得間隔が 500msec でも 5%以内に収まっている．

表 2 は/proc からの情報とコールツリーを対応づける方法に関して，/proc を読む間隔と振り分け誤差の関係である．実験の結果は/proc を読む間隔が短いほど誤差が少ないことがわかった．この実験で以下のことがわかった．/proc 間隔が小さい場合はトータル誤差が大きくなり，振り分け誤差は小さくなる．/proc 間隔が大きい場合は総転送量の誤差が小さくなり，振り分け誤差は大きくなる．

表 1 /proc 取得間隔とトータル誤差
Table 1 /proc acquisition interval and the Total Error

間隔 [ms]	試行 1[%]	試行 2[%]
500	3.51	3.66
1000	3.66	2.25
2000	0.65	1.53
3000	0.69	0.43

表 2 /proc 取得間隔と振り分け誤差
Table 2 /proc acquisition interval and the Allocation Error

間隔 [ms]	分散
500	2099537
1000	2322013
2000	3530099
3000	3948501

5.2 ログ取得方法と侵襲性評価

5.2.1 /proc の取得の侵襲性

/proc からの情報とコールツリーを対応づける方法において，/proc の取得間隔とオーバーヘッドについて評価する．実験方法について説明する．サンプルのアプリケーションはなるべく実行時間が外部要因に影響されないようにした．そして/proc 取得間隔を 1000ms，500ms，400ms，300ms，200ms，100ms と変化させて，それぞれ 2 回ずつアプリケーションの実行時間を測定した．結果を**図 6** に示す．取得間隔が 100ms のときの実行時間が，取得間隔が 1000ms のときに比べて約 2.5 倍になっている．200[ms] より取得間隔が小さくな

Vol.2011-SLDM-149 No.2
Vol.2011-EMB-20 No.2
2011/3/18

表 3 アプリケーション実行時間の API フックによる影響
Table 3 The effect of API hooks on the application execution time

回数	フックなし [ms]	フックあり [ms]
平均	386	13581

ると急激にオーバーヘッドが大きくなるのがわかる．

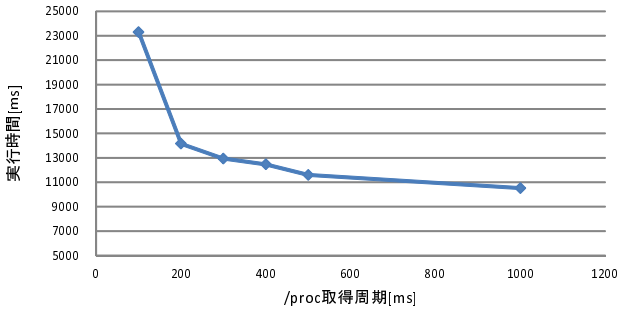

図 6 /proc 取得間隔とアプリケーション実行時間の関係
Fig. 6 /proc acquisition interval and application execution time

5.2.2 低レベル API へのフックの侵襲性

次に API フックによるオーバーヘッドの評価を行う．**表 5.2.2** で示しているのは，アプリケーション実行時間の API フックによる影響についての試行 5 回の平均である．アプリケーションは java.net パッケージの HttpURLConnection クラスを使用して”http://www.google.co.jp” にアクセスするものである．この表によると API フックによりアプリケーション実行時間は約 35 倍になっている．上記の/proc の取得によるオーバーヘッドに比べて API フックによるオーバーヘッドは非常に大きいことがわかる．このアプリケーションの実行開始から終了までの該当 API の発行回数は 616 回であった．フックありの場合の平均アプリケーション実行時間からフックなしの場合を引いたものが 13188[ms] であった．よって一回の API 発行時のオーバーヘッドは約 21.4[ms] である．

5.3 評価のまとめ

トータル誤差を重視するためには/proc から取得する方法を取得間隔を大きくして用いればよい．振り分け精度を重視するためには低レベル API にフックする方法を用いるか，/proc から取得する方法において，小さくすればいい．

6. おわりに

本論文では Android アプリケーションの消費電力プロファイリング手法を提案した．またプロファイリングに使用するログの取得方法として/proc から取得する方法と，低レベル API にフックして出する方法の 2 種類を提案した．/proc からの情報とコールツリーを対応づける方法において，/proc 取得間隔を小さくしていけばメソッドの消費電力がより正確に見積もれる．また API フックによるオーバーヘッドが/proc 取得によるオーバーヘッドよりもかなり大きいことがわかった．つまりログの取得方法によってデバイスごとのトータルの消費電力の精度，時間軸の精度，消費電力コールツリーの精度は変わってくる．トータル誤差を重視する時には/proc から取得する方法を取得間隔を大きくして用い，振り分け精度を重視するためには低レベル API にフックする方法を用いるか，/proc から取得する方法において取得間隔を小さくして用いるなど，ユーザがその時どれを重視するかによって，ログの取得方法を変えられるようにするべきである．

今後の課題として実用的な規模のアプリケーションに提案手法を適用し，消費電力の削減に効果があることを評価しなければならない．

参 考 文 献

1) Ye, W., Vijaykrishnan, N., Kandemir, M. and M.J.Irwin: The Design and Use of SimplePower: A Cycle-Accurate Enegy Estimation Tool, *Proc. Design Automation Conference*, pp.340–345 (2000).
2) L., T., Komarov, K. and Ellis: Energy estimation tools for the Palm, *Modeling, Analysis and Simulation of Wireless and Mobile Systems (Boston, MA)* (2000).
3) Sinha, A.: Jouletrack: A web based tool for soft - ware energy profiling, *Design Automation Conf*, pp.220–225 (2001).
4) Flinn, J. and Satyanarayanan: PowerScope: a tool for profiling the energy usage of mobile applications, *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications*, pp.2–10 (1999).
5) Joseph, R. and Martonosi, M.: Run-time Power Estimation in High-Performance Microprocessors, *ISLPED*, pp.135–140 (2001).
6) 石原 亨，奥平 拓見，久住 憲嗣，神山 剛，関根 和寿，片桐 雅二：OS から解析可能な無線通信端末の消費電力モデルとその生成手法, **情報処理学会研究報告**.*EMB*, **組み込みシステム**, pp.5-30 (2009).