端末状態の監視により端末利用状況に合わせたCPU周波数制御を行う携帯端末向け省電力手法

野呂 正明[1,a)]　長谷川 英司[1]　村上 岳生[1]　上和田 徹[1]　松本 達郎[1]

概要：近年急速に普及しているスマートフォンでは，消費電力の抑制が急務である．しかし，スマートフォンでは，自由にマーケットからアプリをダウンロードでき，アプリ開発者はユーザがどのような機種で実行するか予測も困難であるため，CPU の周波数をアプリから制御して電力を最適化する従来の手法が利用できないため，消費電力の抑制が困難である．しかし，携帯端末において，消費電力とユーザの体感性能を両立させるためには，端末の利用されている状況を認識し，その状況にあった制御方式を用いる必要がある．これに対して現在のスマートフォンでは，CPU 負荷に応じて周波数と電圧を制御してきたが，対話的に利用する場合の体感性能確保のための設定となっているため，消費電力の無駄が発生する．この問題に対して複数の改善手法が存在するが，これらの手法では，端末の状態を正しく認識できない場合がある上，個々のアプリに対する周波数設定をユーザが行う必要があるなどの問題がある．本研究では，端末の周辺機器やアプリの状態変化に加え，スマートフォンのシステムサービスの活動状況を観測し，それら観測結果の組み合わせと周波数制御の設定を対応付ける DB を用いることで，端末の状態を正しく把握し，設定変更を行う．また Android 向けに試作を行い，実機による評価で電力の削減効果を確認した．

キーワード：CPUfreq，スマートフォン，Android，省電力

1. 背景

現在，スマートフォンは非常に多数の機種が市場に投入されており，シェアも急速に拡大している．しかし，スマートフォンはフィーチャーフォンと比較して消費電力大きいことが知られている．その理由として，スマートフォンは，任意のアプリをダウンロードしてインストール可能であることと，非常に多くの機種が存在するため，アプリケーション開発者も端末開発者の双方共に実行される端末やアプリを限定することができず，どのアプリがどの程度の CPU 能力が必要かを事前に測定しておき，CPU の動作周波数を最適化するというフィーチャーフォンの手法を用いることができない．

また，スマートフォンが普及し始めた当初は，ディスプレイが OFF の期間に動作するアプリケーションは，メールの着信チェックなど少数の種類に限られ，CPU が動作する時間も非常に短かったため，この期間の消費電力はあまり着目されていなかった．以上のような理由により，従来のスマートフォンではディスプレイが ON の状態でユーザが対話的に利用している状況での体感性能の確保と消費電力の削減が可能な CPU の動作周波数の設定がなされており，ディスプレイが OFF の期間も同じ設定である．

一方，スケジュールや場所に応じて情報を提供するサービス [1][2] が広がりつつあり，今後もサービスの種類，利用者の増加が予想される．このようなサービスでは，ユーザが携帯する端末は定期的にセンサの値を読み取とり，ユーザの情報を取得してサーバに送信する．このような状況では，ユーザが対話的に利用していないが，端末が活動している期間がより増加するため，消費電力の抑制がより求められることとなる．一般に現在の携帯端末は画面をつけて，対話的に操作している時に適した周波数制御を行なっているため，対話的に操作していない場合の電力の無駄が大きい．そのため，電力の無駄を削減する上で解くべき課題は，実行中の端末の状態を検出し，現在の状態に適した電力設定を判定し，素早く適用することである．

本研究では，ユーザの体感性能を確保しつつ消費電力を削減するため，携帯端末でディスプレイを OFF にしている状態でも動き続けるアプリケーションの種類が限られると共に，アプリケーションの種類毎の共通の特性があることを利用して，ユーザによるアプリの登録無しに端末状態を判定し，状況にあった CPU の周波数設定を行う．

[1] 富士通研究所
FUJITSU LABORATORIES LTD 4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa, Japan
[a)] noro@jp.fujitsu.com

2. 携帯端末の使われ方と CPU の周波数制御

現在の CPU では，動作周波数と電圧をプログラムから制御する (DVFS:Dynamic Voltage and Frequency Scaling) ことが可能である．CPU の消費電力は実際の計算に利用される電力に加えて，漏れ電流によって失われる電力の合計であるが，消費電力は動作周波数および電圧の 2 乗に比例する．そのため，動作周波数が増加するに伴い消費電力は増加するが，ある周波数を超えたところで消費電力が大きく増加する．これは，バス等の周辺回路の周波数が増加するためである．さらに周波数が増加すると，ある周波数で CPU の消費電力が急激に増加する．これは，ある周波数を境に必要となる電圧が上昇するためである．以上のことから，最も電力的に有利となるのは，CPU の動作周波数を動作電圧やバスの周波数を増加させる必要のない範囲だけに限定して利用することである．

一方，CPU の動作周波数の上限値を制限し，電圧やバスの周波数を増加させないようにした状態では，処理に時間的な期限がある場合に問題が発生する．例えば，スマートフォンのタッチパネルをユーザが対話的に操作している場合，CPU の処理能力が不足すると，ポインタの動きが指の動きより遅れる，画面の書き換え処理が画面の更新に間に合わなくなるなどの問題が発生する．その一方で，画面が消えている状態で位置情報等を定期的にサーバに送信するような用途では，短期的な処理の遅延は問題とならないため，極力消費電力が少なくなる周波数範囲で運用することが求められる．

以上のように，ユーザが端末を対話的に操作しているか否かなどによって，CPU の動作周波数抑制の可否，CPU 負荷の変化に対する CPU 動作周波数制御の応答性への要求が変化する．そのため本章では，携帯端末の典型的な使われ方（シナリオ）と，それぞれのシナリオに適した周波数の制御方法について検討する．

2.1 携帯端末の利用シナリオと CPU の周波数制御

携帯端末では，ユーザが対話的に利用している場合，ユーザの体感性能の劣化を防止するため，CPU 負荷に追従して動作周波数を迅速に増加させる必要がある．しかし，CPU 負荷はごく短時間で大きく変化するため，CPU 負荷の変化に対して，CPU の周波数と電圧をすばやく追従させた場合，消費電力は大きくなるため，応答性能が重要でない場合は必要以上の電力を消費する．そのため，ユーザが対話的に利用していない状態を把握し，その期間だけ CPU の動作周波数を低く抑えることで，消費電力を削減することが可能となる．ユーザが対話的に利用していないことを把握するために，一般的に用いられるのはディスプレイの ON/OFF の状態の変化である．

まず，ディスプレイが ON の状態でユーザが対話的に操作していない場合には，Web ブラウジングなどでユーザが画面を凝視している場合と，給電中はディスプレイが消灯しないように設定している場合がある．給電中は消費電力を抑制する必要性が低い．また，ユーザが画面を凝視している期間は，いつユーザが次の操作を行うか予測困難であり，ユーザの操作の再開時のレスポンス低下を避けるためには，消費電力抑制のための特別な設定を行うことが困難である．以上のような理由から，ディスプレイが ON の場合は CPU の処理能力不足にならないよう，CPU の動作周波数を CPU 負荷に対応して迅速に増加させる必要がある．

次に，ディスプレイが OFF の場合の端末の利用シナリオについて検討する．現在市場に出回っている多くの端末では，ディスプレイの ON/OFF にかかわらず共通の方法で CPU の周波数制御が行われてきたため，ディスプレイが OFF の場合にアプリが動作していると，周波数が高めに設定されるため電力の無駄が発生する．しかし，ディスプレイが OFF にしている間，端末の状態に関係なく周波数を抑制すると利用するアプリによっては，問題が発生する．

2.1.1 CPU の処理能力不足が問題となる場合

ディスプレイを消した状態で音楽を聞いている場合やインターネットラジオ等の音声ストリーミングを受信している場合などは CPU の処理性能が不足すると，音飛びの発生や，通信のスループットが十分得られないといった，ユーザの体感性能を著しく損なう可能性がある．ただし，このようなアプリケーションは再生するデータをバッファリングするなどしており，ミリ秒単位の応答の遅れは問題にならない．このような利用シナリオでは，負荷に応じて CPU の動作周波数を制御する必要があるが，ごく短期間の負荷の増加に対応して，急激に CPU の周波数を増加させる必要はなく，ある程度長めの期間の CPU 負荷を観測し，その処理量に応じて CPU の動作周波数を設定すれば良い．

2.1.2 その他の利用シナリオ

ディスプレイが OFF の状態かつ，端末の内蔵センサを観測している場合など，上記シナリオに当てはまらない場合は，CPU の処理能力が不足してもユーザの体感性能には影響しない．そのため，CPU の動作周波数のうち，最も消費電力の少ない周波数範囲に限定して CPU を動作させることが，消費電力を抑制する上で有効となる．

以上のような理由から，CPU の消費電力を削減するためには，ディスプレイの ON/OFF 変化，ディスプレイ OFF 時のユーザシナリオを正確に把握し，そのシナリオに適した周波数設定を行うことが課題となる．**表 1** は，端末利用シナリオの分類と CPU 周波数制御の対応表である．ディスプレイが ON の場合は現在の一般的な設定と同じであり，ディスプレイ OFF の場合は，音楽再生などの CPU の処理性能不足が許されない場合と，それ以外の場合を分けて，それぞれ異なる制御を行うことにより，体感性能の確保と消費電力の削減が可能となる．

表 1　端末利用シナリオと CPU 制御内容の組み合わせ

端末利用シナリオ	CPU の動作周波数の制御内容	
	CPU 処理量の変化に対する周波数制御の追従性	周波数の上限値制限
ユーザが端末を操作している場合（ディスプレイ ON）	短期間の負荷変動にすばやく追従させる	処理能力不足になる危険がある
持ち歩きながら，音楽等を聞いている場合（ディスプレイ OFF，マルチメディア系のアプリ利用）	比較的長期間の負荷の観測結果に基づき，徐々に変化させる	同上
持ち歩いている状態で位置情報等を収集している場合（ディスプレイ OFF，マルチメディア系以外のアプリ利用）	同上	周波数制限により消費電力が削減可能

3. 従来手法とその問題点

スマートフォンの代表的な例である Android[3][*1] のカーネルは Linux[*2] であり，他にも Firefox OS[4][*3]，Ubuntu phone[5][*4]，Tizen[6][*5] などもカーネルに Linux を採用している．Linux における CPU の動作周波数の制御は CPUfreq[7][8] が利用可能である．本章では，Linux の CPU 動作周波数の制御機構である CPUfreq，現在の Android における CPUfreq の利用法および，既存の改善手法について述べる．

3.1 CPUfreq

CPUfreq では，CPU の利用率を観測し，カーネル内部の governor と呼ばれるモジュールで周波数を制御する．カーネル内部では，governor で目標とする動作周波数を決定し，その目標周波数を受け取った CPU 毎に異なるドライバで電源回路やクロック回路を制御して，CPU の動作周波数および電圧を制御する．また，CPUfreq には CPU の動作周波数の上限および下限値，利用する govenor といった共通のパラメータの他に，governor 毎のパラメータが存在する．これらのパラメータの値はカーネルコンパイル時にデフォルト値が決定されるが，**図 1** のように管理者ユーザもしくは，root 権限を持つプログラムから動的に変更することも可能である．

CPUfreq では公式のカーネルでサポートされている governor に加えて，特性の異なる数多くの governor がオープンソースとして公開 [9] されており，カーネルに自由に組み込んで利用することが可能である．ただし，その大部分が以下に紹介する governor と制御アルゴリズムや CPU 負荷の観測の考え方が共通しているため，詳細には説明しない．

3.1.1 ondemand governor

ondemand governor[10] は，CPU 利用率が上昇すると，CPU の動作周波数を最大まで増加させる．その後，徐々に CPU 利用率を観測しながら適切な周波数まで下げていくアルゴリズム（**図 2**）である．調整可能なパラメータは，目標周波数の再計算の時間間隔（sampling_rate）と CPU の動作周波数を最大に上げる基準となる CPU 利用率（up_threshold）の 2 種類である．[*6]

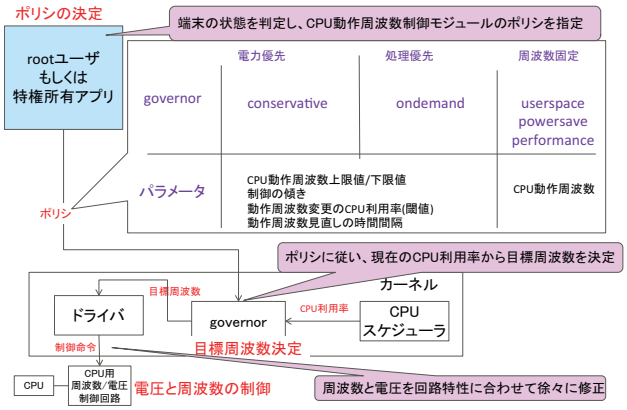

図 1　CPUfreq のアーキテクチャ

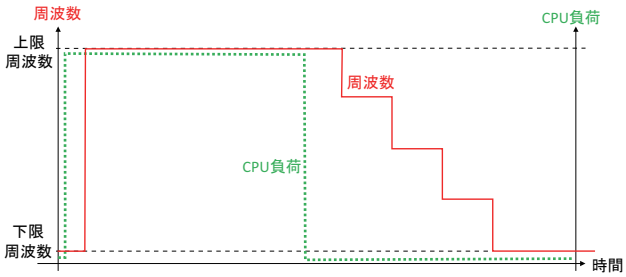

図 2　ondemand governor における CPU 負荷と周波数の関係

ondemand governor では，早期に周波数を引き上げてから徐々に周波数を減少させるため，CPU の応答性は良いものの，消費電力は大きくなる．

3.1.2 conservative governor

conservative governor は，CPU の利用率に応じて動作周波数を増減させるが，増減ともに徐々に行われる（**図 3**）．調整可能なパラメータは，周波数修正の時間間隔（sampling_rate），増減の比率（freq_step：増加/減少共通），増加・減少の閾値（up_threshold, down_threshold），周波数を削減可能か否かのチェックを行う時間間隔を増加の判定より長くするための係数（sampling_down_factor）である．conservative governor では，CPU の動作周波数の増加が緩やかになるため，消費電力は低めになる場合が多いが，

*1　「Android」は、Google Inc の商標または登録商標です．
*2　「Linux」は Linus Torvalds 氏の登録商標です．
*3　「Firefox OS」は Mozilla Foundation の商標です．
*4　「Ubuntu phone」は Canonical LTD の商標です．
*5　「Tizen」は The Linux Foundation の登録商標です．
*6　利用可能なパラメータはカーネルのバージョンによって異なる．

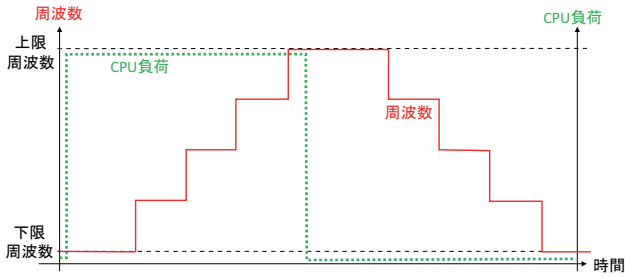

図 3　conservative governor における CPU 負荷と周波数の関係

CPU 負荷が急変した際の追従性が悪い．

3.1.3　interactive governor

ondemand や conservative governor では，数十 ms から数百 ms の負荷観測期間が必要であるため，CPU 負荷量に応じた周波数に遷移する所要時間が長い．これに対して，周波数の再評価の間隔を短るすることで，周波数制御のコストが増加するが，CPU 負荷量に応じた周波数に遷移する時間を短縮し，体感性能の確保と消費電力の削減を目指している governor が interactive governor [11] である．ただし，interactive govenror の周波数制御のアルゴリズムは ondemand governor と同じであるため，CPU 負荷の変動が大きい場合，CPU の動作周波数が高い頻度で最大になるため，消費電力が大きくなる危険がある．

3.2　現在の CPUfreq の利用法

Linux ベースの PC やサーバでは，ユーザが PC と対話的に操作しているか否か，ユーザによる操作の意図の判定が困難であったことから，体感性能を確保することが容易な ondemand governor を利用している．スマートフォンも当初は同じであったが，消費電力削減の必要性が一般的な PC と比較して大きいことから，governor のパラメータチューニングだけでなく，独自に governor を拡張している場合もある．ただし，現在の CPUfreq の利用方法では，ユーザが対話的に操作している場合を想定した設定となっているため，ユーザが対話的に利用していない場合の消費電力は無駄が大きい．

3.3　従来の改善手法とその問題点

前節で説明した CPUfreq の問題を改善するため，端末の状態に合わせて，cpufreq の設定を切り替える機能を持つ手法に，Linux における cpufreqd[12] や Android における SetCPU[13] がある．これらの手法では，外部給電の有無，特定のアプリの起動/終了，装置の内部温度，ディスプレイの ON/OFF 等のイベントを検出し，あるイベントに結び付けられた governor の選択や governor のパラメータ設定を行う．

しかし，これらの手法ではイベントと設定を 1 対 1 で結びつけており，端末状態の誤認識が発生しうる．例えば，ディスプレイ ON の状態で音楽再生を開始し，その後，ディスプレイを OFF にすると，ディスプレイ OFF のイベントに対応させた設定が有効となり，音楽再生独自の設定を行うことができない．また，特定のアプリの起動や終了で設定を変更することはできるが，音楽再生だけをとってみても，数えきれない程の数のアプリが存在するため，すべてのアプリをベンダ側で登録することはできず，ユーザが自分の使うアプリを個々に登録する必要があり，ユーザが的確に登録することも困難である．つまり，現在提案されている改善手法の問題点は以下の 2 つである．

(1) ディスプレイの状態変化等のイベントの発生と，CPU 動作周波数制御の内容を 1 対 1 に結びつけていること．
(2) 音楽再生や音声ストリーミングの受信をアプリの起動と停止で判断していること．

4.　提案手法と CPU 周波数制御の内容

本研究で想定している端末利用シナリオを識別し，CPU 動作周波数の制御内容を変更する上で，従来手法には 2 つの問題点があった．これに対して，本研究の提案方式では，次のような手法でこれらの問題の解決を図る．

1 番目の問題に対しては，イベント発生毎に変化が発生したデバイスやアプリの状態だけでなく，監視対象全ての状態の組み合わせで端末の利用シナリオを判定し，判定したシナリオに対して CPU 動作周波数の制御内容を対応付けることで解決する．

2 番目の問題はアプリの状態変化を監視するのではなく，該当する種類のアプリが共通に利用する API やシステムサービスの状態を監視することで，個々のアプリを登録することなく利用シナリオを判定する．本研究における端末の利用シナリオでは，音楽再生やストリーミング放送の受信開始と停止を検出する必要があるが，これらのアプリは音声を出力するため，一般的に音声データのデコードやミキシングを行うシステムサービス（Android の場合は mediaserver）を利用する．この性質を活用し，システムサービスの活動を監視することで，個別アプリケーションの登録なしに，ディスプレイ OFF 時の利用シナリオを正しく識別する．

また，本研究開発ではプロトタイプの開発と実機での性能評価行った．従って，Android をはじめとした OS や Linux カーネルのバージョンアップに迅速に追従するため，カーネルや Android 等のフレームワークの既存インターフェースを極力活用する．そのため，実際の CPU 周波数制御は CPUfreq を利用し，governor の切り替え，governor のパラメータの変更によって実現する．

図 4 は提案方式のシステム構成図である．本提案方式は，以下の 4 種類のモジュールと 1 つのデータベースから成り立っており，「アプリ状態監視機能」以外は Android フレームワークと Linux カーネルの標準的なインターフェースだけを利用するため，アプリケーションはまったく改造

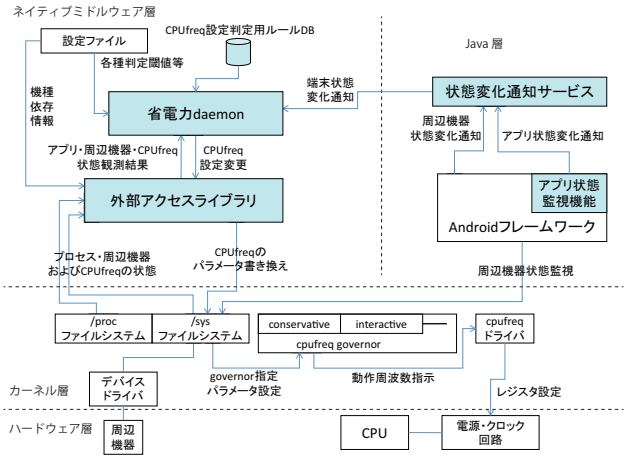

図 4　提案手法のシステム構成

する必要がなく，カーネルや Android のバージョンアップ時も容易に追従できる．

4.1　アプリ状態監視機能

このモジュールは，アプリケーション（Android の場合，Activity および Service）の状態が変化した場合（onCreate から onResume 等）にそれを検出して状態変化通知サービスに検出したイベントを通知（Intent を発行）する．これにより，アプリの状態変化が即時に状態変化通知サービスに通知される．この機能は本研究で試作したプロトタイプでは，Android のフレームワークを拡張して実装を行った．そのため，余分な CPU 負荷なしに，状態変化が検知できる．

4.2　状態変化通知サービス

ディスプレイの ON/OFF の変化や，アプリケーション状態変化といったイベントを集約し，外部の daemon にソケット経由で通知するためのサービス．プロトタイプでは，Android 常駐アプリケーションとして実装した．

4.3　省電力 daemon

上記の状態変化通知サービスからのイベントを待ち受ける他，定期的にバッテリ残量や特定のプロセスの活動状況を監視し，監視結果に変化があった場合に内部的にイベントを発生させ，イベント発生時の端末の状態を判定する．調査結果から，変更すべき CPUfreq の設定内容をルール DB から判定し，ルール中に記述されたアクションで CPUfreq の設定を書き換える．本研究のプロトタイプでは，mediaserver の監視をこのモジュールで行なっている．

4.4　外部アクセスライブラリ

バッテリ残量の情報，CPUfreq 関連ファイルの置き場所や，ファイルのフォーマット（昇順と降順の違い）など，機種やカーネルのバージョンによる I/F の違いを吸収し，省電力 daemon の移植性を高めるためのライブラリ．

4.5　CPUfreq 設定用ルール DB

端末のハードウェアや特定のアプリの利用状況の組み合わせをインデックスとし，CPUfreq の governor および governor のパラメータを書き換えるための実際のアクションの列をデータとして保持するルールを格納する SQLite 形式のデータベース．これにより，任意の観測結果の組み合わせで端末の状態を判定することが可能である．

4.6　プロトタイプ

本研究では実際に動作するプロトタイプを開発し，以下の 6 機種で動作を確認した．ただし，一部の端末では，カーネル CPUfreq にベンダ独自の拡張がなされているため，CPU 動作周波数の上限/下限値の設定ができないといった制限がある．

- Google Android Developer Phone2
- Google Nexus One
- Google Nexus S
- Google Galaxy Nexus
- Samsung Glaxy S
- Motorola Milestone

4.7　周波数制御内容とその決定方法

本提案手法では，端末の状態を検出した後，端末の状態に合わせた CPU の周波数設定を CPUfreq を用いて行う．つまり，あるシナリオにおいて性能の要求と消費電力の削減が可能な governor とパラメータの組み合わせを決定し，それを先のプロタイプを用いて端末のカーネルに適用する．ここでは，想定する利用シナリオに対応した CPUfreq 設定を決定する手法について述べる．

本研究では，以下のような条件を満たす端末として，Nexus One を OS はカスタム ROM（CyanogenMod7）[14] を利用した．また，物理的な消費電力は端末に流れこむ電流を測定した．

- 利用可能な周波数の範囲が広いこと
- 多くの周波数が利用可能であること
- 基本的な governor のすべてが利用可能であること
- シングルコア CPU を搭載していること [*7]
- 端末ベンダによる独自の governor の拡張がないこと

本研究開発では，ディスプレイが OFF の場合において，ユーザが対話的に利用している場合と，センシングなどをバックグラウンドで行なっている場合の 2 通りのシナリオにおいて，CPU の動作周波数の制御内容を切り替えることにより，消費電力を削減することである．**表 3** のように，

[*7] マルチコア CPU におけるコア制御の影響を排除するため

表 2　利用したプラットフォームの情報

モデル名	Nexus One
CPU	QSD8250
カーネル	2.6.35.7
利用可能な governor	interactive conservative userspace powersave ondemandperformance
標準 governor	ondemand
利用可能な周波数（kHz）	245000 384000 422400 460800 499200 537600 576000 614400 652800 691200 729600 768000 806400 844800 883200 921600 960000 998400

表 3　実験の端末利用シナリオ（ディスプレイ OFF 時限定）

シナリオ	アプリ番号	アプリの詳細	無線の状態
音楽再生等	1	音楽再生	機内モード
	2	音楽再生	3G
その他	3	センサ情報取得	機内モード
	4	センサ情報取得	3G
	5	位置とセンサ取得	3G
	6	位置情報取得	3G
	7	待受	機内モード
	8	待受	3G

対話的利用のシナリオは音楽再生アプリを利用し，センシング関係の処理は性能評価用に新規にアプリを開発した．さらに，本研究の対象は画面 OFF の状態で常時バックグラウンドでデータを収集するような利用方法を想定しているため，画面が OFF の状態で CPU が起きているものの，アプリが活発に動作していない待機状態（静止待受状態）もデータを取得した．

4.7.1　governor の選択

まず，各シナリオで利用する governor を選択するため，デフォルトのパラメータを用い，表 3 の各端末利用シナリオにおいて，どの程度の電流が電池から端末に流れこむかを測定した． この測定において，利用した governor は ondemand，conservative，interactive の 3 種類である．**図 5** は各利用シナリオにおいて，ondemand govenror での電流を 1 とした場合の conservative governor と interactive governor の電流値の比を示したものである．これを見てわかるように, interactive governor では, ondemand governor とほとんど差がないのに対して，conservative governor では音楽再生が ondemand governor より電流が大きくなる他は，同じか小さくなっているため，conservative governor を用いることとした．

このような結果となる理由として，interactive governor は，負荷が増加した際に一度最高の周波数にすることと，他の governor より短い間隔で周波数を変更するため，周波数の減少も早いが，同様に頻繁に最高の周波数まで増加するため，このような結果となっている．それに対して，conservative governor は徐々に周波数を増加させるため，CPU の動作周波数は短期間の負荷ではそれほど増加せず，消費電力が低めとなる．

4.7.2　周波数を増加させる閾値

次に，周波数の増減の閾値，周波数を再計算する間隔等を決定する必要がある．まず周波数を増加させる閾値（*up_threshold*）は CPU 利用率の値であり，0 から 100 までの整数にする必要がある．周波数増加の閾値は大きいほうが周波数が上がりにくくなり消費電力の抑制が可能である．しかし，あまり大きい数字にした場合（例えば 100）に，I/O に伴う割り込み待ちで CPU が短時間でも処理がない状態になった場合，CPU 利用率は閾値を超えないため，周波数が増加できなくなる．そのため，I/O 等の割り込み待ち等の時間を見込んで少し低めの値にする必要がある．近年出荷された端末（8 社 17 機種）では，Android のバージョンが上がるにつれて，*up_threshold* の設定値も大きくなっているが，GingerBread 以降のバージョンでは，1 機種を除き 90 もしくは 95 である．そのため，本研究でも *up_threshold* の値は 95 とした．

4.7.3　周波数再評価する時間間隔

次に，周波数再評価する時間間隔（*sampling_rate*）について考える．図 5 では，音楽再生において ondemand governor より conservative governor の方が消費電力が大きい．この場合の負荷状況を確認すると，音楽再生ではほとんどの期間において，CPU 負荷が殆ど無いが，短時間の高負荷状態が周期的に現れている．また，ondemand governor のサンプリング時間が 50ms なのに対して，conservative governor では 200ms となっている．そのため，200ms の平均負荷が governor の *down_threshold* 以上となる状態では，音楽再生以外の処理が同時に発生し，一度周波数が増加してしまうと，以後まったく周波数が下がらなくなる．逆に，ondemand governor では 50ms の周波数再評価のタイミングで複数回周波数を下げることができるため，消費電力の差が発生する．これを防止するためには，*sampling_rate* を短くするか，*down_threshold* を大きくする必要があるが，今回利用したプラットフォームでは，conservative governor の *sampling_rate* は既に最小値であるため，この値は変更しない．

4.7.4　周波数を減少させる閾値

周波数を減少させる閾値（*down_threshold*）であるが，先ほどの音楽再生の例のように，この値はなるべく大きい方が省電力となる．そのため，*down_threshold* の可能な最大値を計算により求める．一般的には，周波数を減少させた場合に，遷移先周波数における周波数増加閾値以上の仕事量になる（以下の式が満たされた場合）と，周波数が低下した次の周波数評価のタイミングで，周波数を増加させる判定がされ，結果として周波数が振動する危険がある．つまり，CPU の動作周波数を低い方から順に $f_0, f_1 ... f_n$ とすると，次の式が満たされることがないように各閾値を設

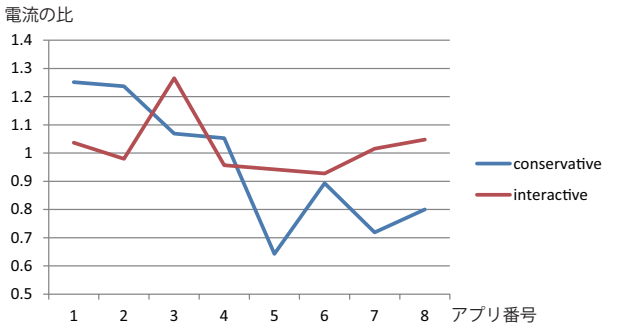

図 5 デフォルトパラメータによる各 governor の消費電流特性

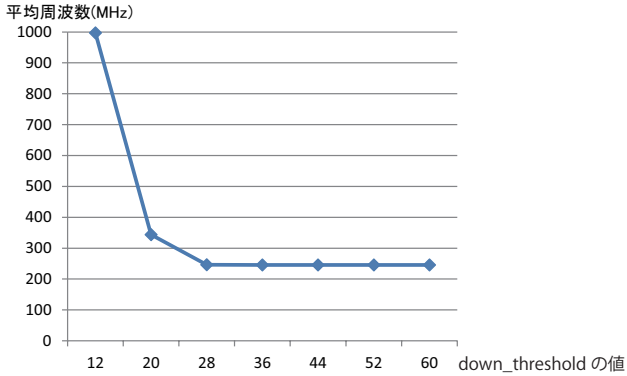

図 6 周波数減少の閾値を変化させた場合の平均周波数の特性

表 4 conservative governor のパラメータ

パラメータ	標準	変更後
周波数増加閾値 (*up_threshold*)	76	95
周波数減少閾値 (*down_threshold*)	12	60
周波数見直し間隔 (*sampling_rate*)	200	200
周波数変更率	5	5
周波数の上限値（kHz） （一部のシナリオのみ）	998400	384000

定する必要がある．

$$f_k \times up\_threshold \leq f_{k+1} \times down\_threshold$$

そのため，端末で利用可能な周波数のリストと先に決定した *up_threshodl* の値を用いて，多少の負荷の測定誤差があっても，上の式が満たれることがない値の上限値として 60 を算出した．次に，確認のため音楽再生の場合の CPU の動作周波数の平均値と *down_threshold* の関係を測定した．**図 6** からわかるように，*down_threshold* の値が 28 以上であれば，CPU の動作周波数は十分抑制することができ，消費電力の低減が期待できる．

他の利用シナリオでは，CPU の処理能力が不足しても，ユーザの利用感には影響しないため，音楽再生と同じく conservative governor を採用した上で，さらに，CPU の動作周波数の最大値を抑制することとし，**表 4** のようにパラメータを確定した．

表 5 電流の測定結果

governor	ondemand		conservative		interactive
周波数制限	なし		なし	あり	なし
アプリ番号	電流	比率	比率	比率	比率
1	70.11	1	1.00	—	1.04
2	74.78	1	0.99	—	0.98
3	7.92	1	—	0.58	1.27
4	14.09	1	—	0.65	0.96
5	20.31	1	—	0.62	0.94
6	18.99	1	—	0.66	0.93
7	7.74	1	—	0.39	1.02
8	12.17	1	—	0.55	1.05

表 6 平均周波数と最低周波数の比

governor	ondemand	conservative		interactive
周波数制限	なし	なし	あり	なし
アプリ番号	比率	比率	比率	比率
1	1.34	1.00	—	1.08
2	1.07	1.00	—	1.06
3	1.64	—	1	1.69
4	1.55	—	1	1.74
5	1.19	—	1	1.39
6	1.20	—	1	1.34
7	1.66	—	1	1.60
8	1.57	—	1	1.86

5. 評価

本章では，ディスプレイを OFF にしている状態で利用される典型的なアプリケーションに対して，最適化した設定を行うことで，どの程度消費電力を削減することが可能であるかを実際の端末の消費電力の測定結果によって示す．

5.1 測定対象のシナリオと測定手法

本評価では，4.7 節と同じアプリ（表 3）を用いて計測した．ただし，電流測定回路を端末に接続しており，屋外における GPS 測位が困難であるため，3G の無線基地局の情報を用いた測位に限定した．また，予期しない無線通信により，消費電力が変化することを防止するため，スケジュールなどの同期といった処理は抑制している．

収集したデータは，「どの周波数でどれだけの時間動作したか」の統計データ，および，「電池から流れた電流の実測値」の 2 種類である．これについても，4.7 節と同じであるが，より精度を上げるため，各シナリオ毎に 2 時間分のデータを 10 回収集し，平均を求めている．

5.2 消費電力

表 5 は電流の測定結果である．音楽再生時以外は governor のパラメータチューニングや上限周波数の抑制により，消費電流を 30〜40%程度削減可能であるのに対して，音楽再生では消費電流の差が殆ど無い．また，待受時は 45%〜

最大 60%程度削減できている．

表 6 は，測定時の平均の動作周波数と最低周波数（245MHz）との比である．この表中の数値において，「1.00」は 1 よりやや大きい値，「1」は常時利用可能な最低の周波数（245MHz）で動作していたことを示している．conservative governor では，governor 設定の利用可能周波数の差と $freq\_step$ の積を用いるため，上限周波数の変更は，周波数の変化の傾きの変化させる働きがあり，長期間 CPU 負荷が高い状態が続かないかぎり周波数が上がらないため，このような結果となる．

この表から，音楽再生以外のすべてのシナリオにおいて，チューニング後の conservative governor で CPU 動作周波数の上限値を抑制した場合は最低の周波数で動作しており，電流の差は CPU 以外の回路における消費電力によるものであることがわかる．このように，上限周波数を抑制した場合に，周波数が最低周波数で固定されるのは以下のような理由である．

conservative governor では，周波数変更時の幅を計算する場合に，カーネルのサポートする最大の周波数ではなく，governor 設定の上限周波数と周波数変更率の積を用いるため，上限周波数の変更は，周波数の変化の傾きの変化させる働きがあり，長期間 CPU 負荷が高い状態が続かないかぎり，周波数が上がらない．以上の governor の性質のため，このような結果となる．

また，音楽再生については，ondemand と conservative governor では，conservative governor は周波数がやや低くなるものの，消費電力は同じとなっている．この理由として，音楽再生で利用される DSP やアナログ回路の消費電力が大きく，CPU の電力の差が埋没してしまうことと，CPU の処理量は比較的定常的な負荷となる．そのため，2 時間の平均を計算すると，ほぼ同一の周波数分布と仕事量となるためである．

本提案方式では，複数の箇所で数多くの端末状態の監視を行なっているが，daemon 本体を除くと，イベントが発生した場合のみ追加の処理が実行されるため，それほど CPU 負荷は増加しない．逆に daemon は，数秒に一回プロセスの動作状況や CPU の利用率のチェックするため，負荷の増加に伴い電力消費は増加する．しかし，消費電流の測定結果からわかるように，提案手法の動作によって増加する電力よりシステムの電力設定を制御することによる電力削減が上回るという結果が得られた．

6. まとめ

本研究では，スマートフォンの消費電力を抑制するため，端末の状態の観測結果に応じて CPU の動作周波数を制御する．この手法では，端末を対話的に利用していない場合に着目し，典型的なのシナリオにおいて共通な端末の状態に着目し，その組み合わせで端末のシナリオを判定する．このシナリオに対して，CPU 設定を結びつけたデータベースを用意し，端末の状態の監視結果から対応する CPU 周波数の制御設定を決定し，OS のカーネルに設定する．

また，提案手法のプロトタイプを開発し，Google Nexus One を用い，端末に電池から流れ込む電流と，CPU の動作周波数の測定して性能を評価した．この評価の結果，ディスプレイが OFF でセンサ類を読み出している状態では，30%〜40%程度端末に流れ込む電流が削減できる上，ディスプレイ OFF かつ端末がスリープできない状態（待受）状態では最大 60%の電力が削減できた．

参考文献

[1] 鈴木裕紀，平石絢子，竹田千沙，中矢恭介，高津利樹：2008 年秋冬モデル搭載アプリケーション機能 (1) i コンシェル機能およびユーザメモリ一括バックアップ機能の開発，NTT DOCOMO テクニカルジャーナル， Vol. 16, No. 4, pp. 40–45 (2009).

[2] Google: Google Latitude, Google (online), available from ⟨http://www.google.com/intl/ja_jp/mobile/latitude/⟩ (accessed 2013-07-13).

[3] Google: Android Developer, Google (online), available from ⟨http://developer.android.com/⟩ (accessed 2013-07-12).

[4] Mozilla: Firefox OS, Mozilla Foundation (online), available from ⟨http://www.mozilla.org/en-US/firefox/os/⟩ (accessed 2013-08-09).

[5] Canonical: Ubuntu phone, Canonical (online), available from ⟨http://www.ubuntu.com/phone⟩ (accessed 2013-08-09).

[6] Project, T.: Tizen, Linux Foundation (online), available from ⟨https://www.tizen.org/⟩ (accessed 2013-08-09).

[7] Hopper, J.: Reduce linux power consumption, part 1: The cpufreq subsystem, Technical report, IBM (2009).

[8] Brodowski, D. and Golde, N.: CPUFreq governors, Linux カーネルドキュメント (online)，available from ⟨https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt⟩ （参照 2013-07-12）.

[9] HipKat: CPU Governors explained, xda-developers (online), available from ⟨http://forum.xda-developers.com/showthread.php?t=1663809⟩ (accessed 2013-08-07).

[10] Pallipadi, V. and Starikovskiy, A.: The ondemand governor, *Proceedings of the Linux Symposium*, Vol. 2, Linux Symposium, pp. 215–230 (2006).

[11] Google: Interactive Governor, GitHub (online), available from ⟨https://github.com/CyanogenMod/cm-kernel/commit/255f13bf41f368aa51638a854ed69cfc60f39120⟩ (accessed 2013-07-12).

[12] Dongili, M. and Staikos, G.: Cpufreqd (online), available from ⟨http://www.linux.it/~malattia/wiki/index.php/Cpufreqd⟩ (accessed 2013-07-13).

[13] Huang, M.: SetCPU for Root Users, setcpu.com (online), available from ⟨http://www.setcpu.com/⟩ (accessed 2013-07-12).

[14] cyanogen: CyanogenMod (online), available from ⟨http://www.cyanogenmod.org/⟩ (accessed 2013-07-12).