Vol.2015-GN-93 No.23
Vol.2015-CDS-12 No.23
Vol.2015-DCC-9 No.23
2015/1/27

Android における LRU に基づくプロセスメモリ管理とユーザのアプリケーション起動傾向に関する考察

野村駿[†1]　中村優太[†1]　坂本寛和[†1]　濱中真太郎[†2]　山口実靖[†2]

Android には，low memory killer と呼ばれる独自のプロセスメモリ管理システムが搭載されており，メモリ不足時には既定の基準に従いプロセスを強制終了し空きメモリを確保する．このプロセスの強制終了により，再度同じアプリケーションを使用する場合にプロセスの再起動の時間が必要となり，ユーザの利便性を低下させることがある．本稿では，LRU に基づく強制終了プロセス選定手法に着目し，様々なユーザのアプリケーション起動履歴を用いて LRU を用いる手法の妥当性の評価を行う．そして，アプリケーションの起動傾向について考察する．

1. はじめに

Android はスマートフォン，タブレット PC，音楽プレイヤーなど様々なデバイスの OS として採用されており，2014 年第 3 四半期においてそのシェアは 84.4%[1]となっており，今も増加中である．この様に Android OS は広く普及しており，重要性が高まっていると考えられる．

Android には，low memory killer と呼ばれるプロセスメモリ管理システムが搭載されており，独自のルールに従ってメモリの管理を行なっている．low memory killer は空きメモリ量が少なくなると起動され，メモリの空き容量を確保するためにプロセスを強制終了する．このプロセスの強制終了により，ユーザが再度同じアプリケーションを使用する場合に，プロセスの再起動が必要となりユーザの待ち時間を増加させることがある．これに対して我々は，LRU を用いた強制終了プロセス選定手法を提案し，実機に実装し評価した[2].

本稿では，ユーザのアプリケーションの起動の傾向や偏りに着目し，ユーザのアプリケーション起動履歴の解析を行う. そして LRU を用いることの妥当性について考察を行う．

2. low memory killer

Android には，low memory killer という独自のプロセスメモリ管理システムが搭載されている．これは，システム全体のメモリ空き容量が閾値以下まで下がった場合に起動され，後述の *adj* と *minfree* の関係に基づいてプロセスを選定し強制終了するプログラムである. 多くの携帯端末同様に，Android はユーザがアプリケーションの終了処理を行わずに使用できるように設計されている．すなわち，ユーザはそのときに使用したいアプリケーションの起動(開始)のみを行えば良く，アプリケーションの終了操作を行う必要がない．よって，ユーザがアプリケーションを終了させることなく稼働アプリケーション数を増やし続けると OS が管理する空きメモリ量が単調に減り続け，必然的にシステムはメモリ不足の状態に陥る．low memory killer はこの問題を解決するために(換言すると，ユーザにアプリケーション終了操作を要求しないシステムを実現するために)用意されており，システム全体の空きメモリ量が少なくなると自動的にアプリケーションを終了する．

システム内のプロセスには *adj* と呼ばれる値が設定されており，*adj* の値が小さいプロセスほど強制終了されづらく，重要なプロセスほど小さな *adj* が設定されている．表 1 に Android 4.0.3 におけるプロセスの状態と *adj* の値の関係を示す．最も強制終了されづらいフォアグラウンドのアプリケーションに 0 が，サービスアプリケーションに 5 が，最も強制終了されやすいバックグラウンドのアプリケーションに 15 が割り当てられている．

low memory killer が起動されると，二段階で強制終了プロセスの選定が行われる．まず第一次判定として，*adj* を用いた比較が行われる．第一次判定で強制終了対象を一意に定めることができない場合は第二次判定としてメモリ使用量による比較が行われる．具体的には，起動している全てのプロセスの *adj* を比較し，*adj* の数値がより高いプロセスを強制終了する候補にあげる(第一次判定). 最高 *adj* のプロセスが 1 個しか無い場合は第一次判定にて選定は終了する．最高 *adj* のプロセスが複数存在する場合，メモリ使用量を比較し，メモリ使用量のより多いプロセスを強制終了する候補にあげる(第二次判定).

前述の通り，low memory killer はシステムの空きメモリ量が少なくなると起動され，空きメモリ量が指定の値を超えるまで *adj* が大きなプロセスを強制終了する．low memory killer が起動する空きメモリ量の閾値(*minfree* と呼ばれる)および強制終了対象とする *adj* の閾値の組み合わせは，`/init.rc` で定義されており，low memory killer において強制終了するプロセスを選定する際に参照される．*minfree* は，プロセス強制終了実行の閾値となる空きページ数であり，*adj* によってランク分けされている．例えば Android4.0.3 では *minfree*=9607[4KB]=38428[KB]と *adj*=9 の組み合わせが定義されており，メモリ空き容量が 38428[KB]以下になると *adj* の値が 9 以上の数値を持つプロセスが強制終了される候補に上がる．そして，空きメモリ量が 38428[KB]以上になるか，*adj* が 9 以上のプロセスがなくなるまでプロセスを強制終了する．

†1 工学院大学大学院
Kogakuin University Graduate School
†2 工学院大学
Kogakuin University

Vol.2015-GN-93 No.23
Vol.2015-CDS-12 No.23
Vol.2015-DCC-9 No.23
2015/1/27

表 1 プロセスの状態，種類による *adj* の値

Adj	プロセスの状態，種類
0	FOREGROUND_APP
1	VISIBLE_APP
2	PERCEPTIBLE_APP
3	HEAVY_WEIGHT_APP
4	BACKUP_APP
5	SERVICE
6	HOME_APP
7	PREVIOUS_APP
8	SERVICE_B
9	HIDDEN_APP_MIN
15	HIDDEN_APP_MAX

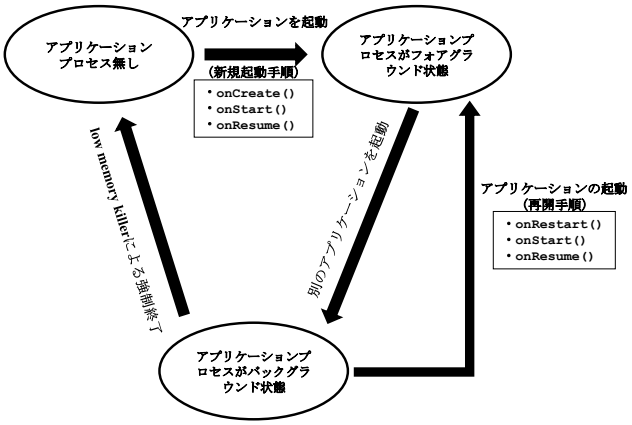

図 1 アプリケーション起動状態遷移図

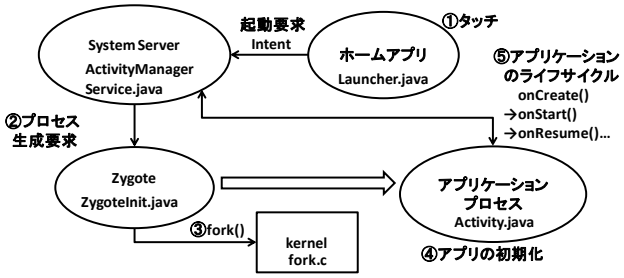

図 2 アプリケーションの起動手順

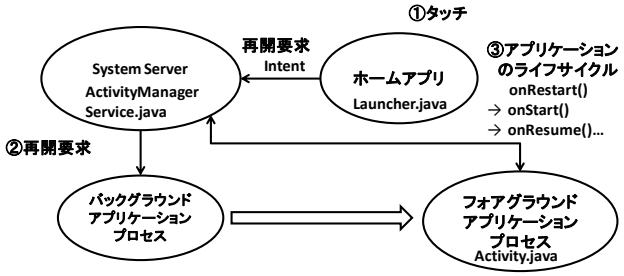

図 3 アプリケーションの再開手順

3. アプリケーションの起動手順

Android においてアプリケーションのプロセスは「プロセスが存在しない」「プロセスがフォアグラウンド状態で存在する」「プロセスがバックグラウンド状態で存在する」の 3 種類の状態があり得る．また，アプリケーション起動の方法は「新規起動」と「再開」の 2 種類がある．前者はアプリケーションプロセスが存在しない状態からアプリケーションプロセスを生成する起動方法であり，後者はバックグラウンド状態のアプリケーションプロセスが存在する状態から既存プロセスを再利用して起動する方法である．

これらの 3 種類のアプリケーションプロセスの状態と，2 種類の起動方法を図 1 に示す．OS が起動した直後はアプリケーションのプロセスは存在しておらず「アプリケーションプロセス無し」の状態である．ここで，ユーザがアプリケーションを起動すると「新規起動」の方法を用いてアプリケーションプロセスが生成され,「アプリケーションプロセスがフォアグラウンド状態」に移行する．次にユーザが別のアプリケーションを起動すると，当該アプリケーションのプロセスは「バックグラウンド状態」に移行する．バックグラウンド状態に移行している間にシステム全体のメモリ量が不足すると，当該プロセスは low memory killer により強制終了されてしまい「プロセス無し」の状態に移行する．メモリ不足が発生しなければ当該プロセスは「バックグラウンド状態」で存在し続ける．その後にユーザが当該アプリケーションを再度使用しようとしたときの動作は，アプリケーションプロセスの状態により異なる．「プロセス無し」の状態であれば「新規起動」の手順により起動され,「バックグラウンド状態」であれば「再開」の手順により起動される．

Android のアプリケーションを新規に起動する場合，図 2 の様な手順に従って起動される．①ユーザがアプリケーションのアイコンをタッチするとホームアプリケーション(Launcher)がアプリケーション起動要求のインテントを ActivityManager に送信する．②ActivityManager が Zygote にプロセス生成要求を送信する．③Zygote が自分自身を fork し，子プロセスを生成する．④新しいプロセスが初期化される．⑤アプリケーションのライフサイクルに従って `onCreate()`, `onStart()`, `onResume()` が呼び出される．

また，起動済みであるがバックグラウンド状態にあるプロセスの再開は，図 3 の様な手順に従って行われる．①ユーザがアプリケーションのアイコンにタッチするとホームアプリケーションが再開要求のインテントを ActivityManager に送信する．②ActivityManager は対象のバックグラウンドアプリケーションプロセスに再開要求を出す．③バックグラウンドアプリケーションプロセスは再開要求を受けてアプリケーションプロセスとして起動しフォアグラウンド状態となる[3]. 本稿では前者を「新規起動」,

Vol.2015-GN-93 No.23
Vol.2015-CDS-12 No.23
Vol.2015-DCC-9 No.23
2015/1/27

後者を「再開」と呼ぶ.

起動済みプロセスが low memory killer によって強制終了されることなく再利用された場合は，上記の「再開」の手続きが行われるが，プロセスが強制終了された後に再利用された場合は「新規起動」の手続きが行われる．通常，「再開」よりも「新規起動」の方が要する時間が長いため，後に再利用するプロセスの強制終了はユーザの待ち時間を増加させてしまう.

4. LRU に基づく強制終了プロセス選定

本章にて，我々が提案した LRU に基づく強制終了プロセス選定手法[2,4]を紹介する．本手法は標準 low memory killer をベースとしており，標準手法と同様に第一次判定，第二次判定により構成される．標準手法の第一次判定の判定方法を改変することにより本手法は実現されている.

本手法では，ユーザが実行したアプリケーションを実行履歴として一定数記録しておく．履歴は LRU により管理し，最後に参照されてからの時間が短いものの *adj* を下げ強制終了の対象となりにくくする.

評価用実装[4]では以下の様に実装されている．長さ 15 の配列を用意し，ユーザが起動したアプリケーション履歴を保持する．配列は LRU にて管理する．そして，low memory killer における強制終了プロセス選定の際，アプリケーションの履歴内の順位を調査し，履歴の *n* 番目にあるプロセスの *adj* を *n*/2 に変更する．順位は新しい方から数え，最小は 0 とし，小数点以下は切り捨てとする．ただし，変更前の *adj* が変更後の *adj* より低い場合は，変更前の *adj* を優先する．この手法により，最近使用されたアプリケーションのプロセスが強制終了されにくくなり，ユーザの待ち時間を低減させることができると期待される．履歴内に記録のないアプリケーションの *adj* は変更を加えない．履歴の長さや，*adj* の下げ幅はチューニングパラメータである．ただし，履歴内のアプリケーションを履歴外のアプリケーションよりも優先するために，履歴の最後尾のアプリケーションの調整後の *adj* が履歴外のアプリケーションの *adj* より小さくなる様に設定することを推奨している．文献[4]の実験例の場合，SERVICE_B や HIDDEN_APP_MIN の *adj* が 8 や 9 である．よって，履歴長を 15，*adj* 下げ幅を *n*/2 として，履歴最後尾のアプリケーションの調整後の *adj*(15/2)が 8 や 9 を下回る様に設定している.

5. アプリケーション起動の偏り

一般に，データアクセスに「参照の時間的局所性」[5]が存在するとキャッシュ置換アルゴリズム LRU は効果的に機能する．参照の局所性とは，「一度アクセスされたデータは近い将来再度アクセスされる可能性が高い」という性質であり，多くのアプリケーションのメモリアクセスにてこの性質が確認されている.

本章にて，ユーザのアプリケーション起動履歴の解析を行い，アプリケーション起動の局所性について考察する.

5.1 アプリケーション起動履歴

4 人の被験者の許可を得て，その被験者が使用したアプリケーションの起動履歴を入手した．アプリケーションの起動履歴は，文献[3]のアプリケーションの起動時間調査方法を改変し，起動したアプリケーションの名前のみを記録できるように Android OS に修正を加え，修正済み Android OS が搭載されている端末を使用してもらうことにより得た．得られた履歴は 7 件であり，本稿では log1 から log7 と呼ぶ．履歴は 7 件であり，このうち 4 件を付録に記す.

被験者のアプリケーションの起動履歴は log1 から log7 まであり，log1 から log7 の各履歴の長さは順に 436，201，95，55，43，42，30 である.

また，アプリケーション使用履歴は使用アプリケーション(フォアグラウンドアプリケーション)の変化時にのみ記録しており，1 つのアプリケーションを連続して使用しても履歴においては 1 つとする．よって，同一アプリケーションが再度履歴に登場するまでに少なくとも 1 つの他のアプリケーションが履歴内に登場する必要があり，特定のアプリケーションの出現頻度は最高で 50%となる．low memory killer は使用アプリケーションの変化時に起動されバックグラウンドアプリケーションを強制終了することが多いため，この方式を用いて記録を行った.

5.2 使用アプリケーションの種類

まず，各 log で使用されているアプリケーションの種類を図 4 に示す.

図より，少ないユーザで 8 種類，多いユーザで 20 種類以上のアプリケーションが使用されていることが分かる.

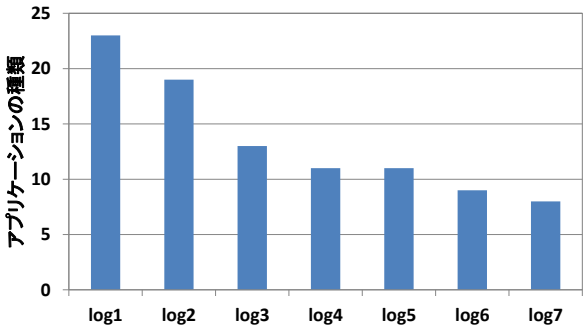

図 4 使用アプリケーションの種類

次に，各履歴におけるアプリケーションごとの使用頻度(履歴における占有率)を図 5 に示す.

Vol.2015-GN-93 No.23
Vol.2015-CDS-12 No.23
Vol.2015-DCC-9 No.23
2015/1/27

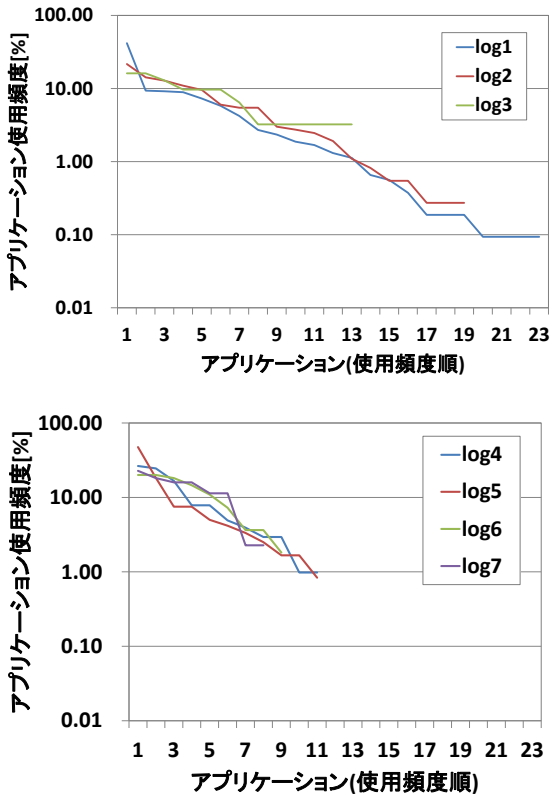

図 5 アプリケーションごとの使用頻度

図の横軸は各履歴におけるそのアプリケーションの使用頻度の順位である．図より，最も高い頻度で使用されているアプリケーションの使用頻度は20%前後であることが多く，50%に近い頻度のユーザもいる．

また，これを積分したものを図 6 に示す．

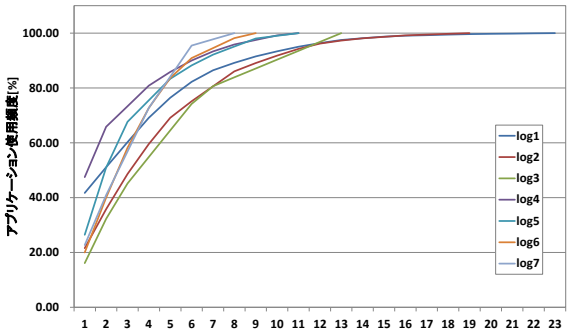

図 6 アプリケーション使用頻度(積分)

図 6 において，横軸 x，縦軸 y の点のプロットは，頻度順位が 1 位から x 位の x 個のアプリケーションの合計使用頻度が y%であることを意味している．

図より，多くのユーザにおいて，上位数個(4～7 個)の使用頻度のアプリケーションにより全使用の 80%を占めており，ユーザのアプリケーション起動には大きな偏りがあることがわかる．

5.3 アプリケーションの使用間隔

続いて，アプリケーション使用間隔に着目し，考察を行う．

本稿では“アプリケーションの使用間隔”という語を「あるアプリケーションを使用して，次に同じアプリケーションを再度使用するまでに他のアプリケーションを何個使用しているか」の意味で用いる．アプリケーション起動履歴の x 個目にアプリケーション A が登場し，次にアプリケーション A が登場するのが y 個目であれば，今回のアプリケーション使用間隔は $y\text{-}x\text{-}1$ となる．LRU は参照の時間的局所性を期待しており，最近使用されたアプリケーションが短い使用間隔で使用されると効果的に機能する．アプリケーション使用間隔の発生頻度とその積分値を図 7 から図 13 に示す．図より，多くのユーザにおいて，アプリケーション使用間隔は短いことが多く，LRU が効果的に機能すると予想できる．また，積分値に着目すると，ほとんどの履歴において使用間隔 5 個以下から 8 個以下で 80%を網羅しており，端末が過去 5 個から 8 個分のプロセスを保持できればアプリケーション再開率 80%を達成できることが分かる．

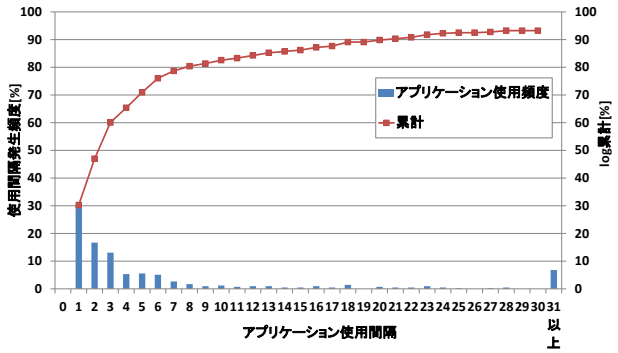

図 7 log1 におけるアプリケーション使用間隔の発生頻度

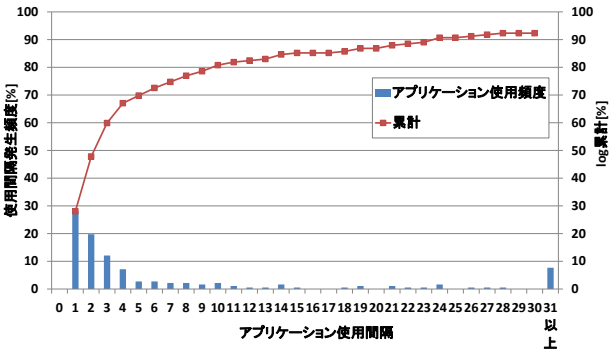

図 8 log2 におけるアプリケーション使用間隔の発生頻度

Vol.2015-GN-93 No.23
Vol.2015-CDS-12 No.23
Vol.2015-DCC-9 No.23
2015/1/27

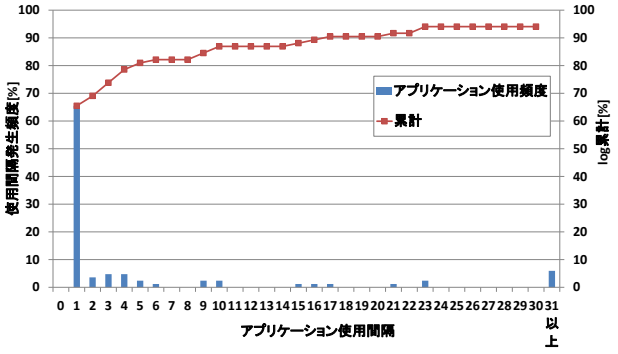

図 9 log3 におけるアプリケーション使用間隔の発生頻度

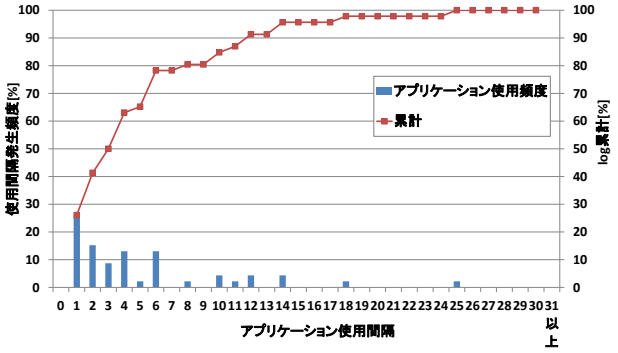

図 10 log4 におけるアプリケーション使用間隔の発生頻度

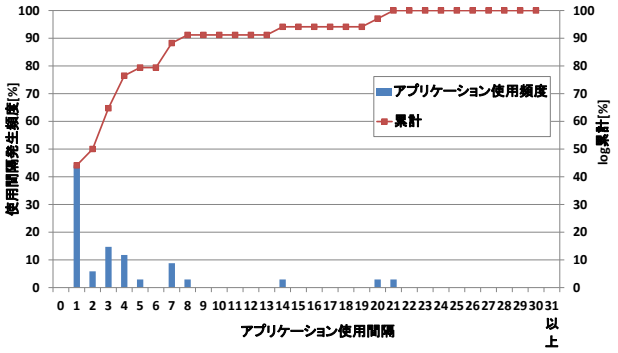

図 11 log5 におけるアプリケーション使用間隔の発生頻度

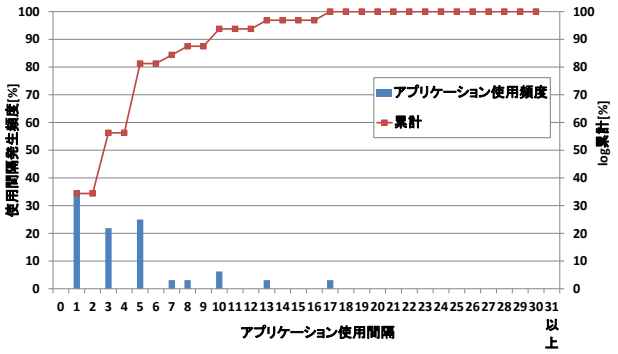

図 12 log6 におけるアプリケーション使用間隔の発生頻度

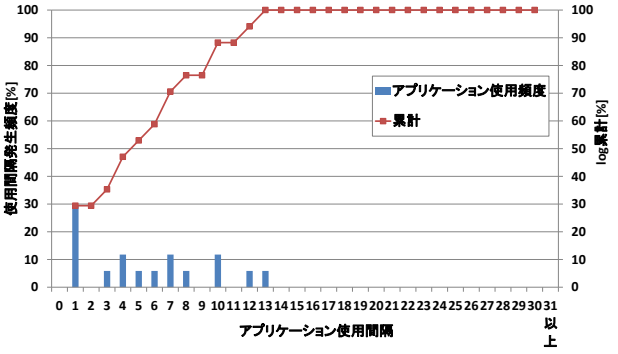

図 13 log7 におけるアプリケーション使用間隔の発生頻度

5.4 アプリケーションの短期間内占有率

次に，特定のアプリケーションの短期間内占有率の時間変化について述べる．“短期間内占有率”とは，履歴内の連続する領域(10 アプリケーション)にて，あるアプリケーションが何回登場するかを表している．同一アプリケーションが 2 回連続して履歴に登場することがないため，登場回数は最高で 5 回であり，占有率は最高で 50%である．図 14 と図 15 に log1 と log4 の，①同履歴内で最も使用頻度の高いアプリケーション，②使用頻度の順位が 2 番であるアプリケーション，③使用頻度の順位が全体の中央であるアプリケーション，④使用頻度の順位が最下位であるアプリケーションの占有率の推移を示す．

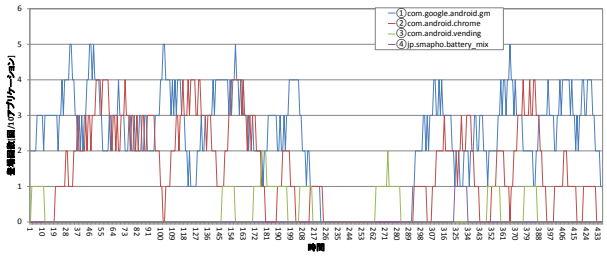

図 14 log1 におけるアプリケーションの占有率の時間変化

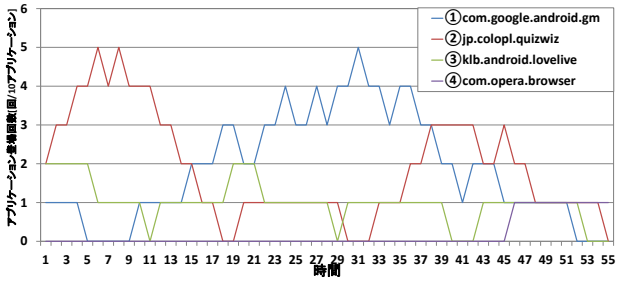

図 15 log4 におけるアプリケーションの占有率の時間変化

図より，履歴全体で見ると使用頻度が高いアプリケーションであっても，占有率は時間とともに大きく変化し，占有率が 50%近くと非常に高くなる時間帯と 0%になる時間帯があることが分かる．すなわち，ユーザがあるアプリケ

Vol.2015-GN-93 No.23
Vol.2015-CDS-12 No.23
Vol.2015-DCC-9 No.23
2015/1/27

ーションを使用している時間帯と使用していない時間帯の二極化が生じていることを意味しており，あるアプリケーションを使用している時間帯はそのアプリケーションを高確率で起動していることが分かる．よって，参照の時間的局所性(あるアプリケーションが使用されると，そのアプリケーションが近い将来に再度使用される確率が高い)に非常に近い現象であり，LRU が効果的に機能すると期待することができる．また時間帯ごとに使用される確率が高いアプリケーションが変化するため，LFU の様に重要視するアプリケーションが時間とともに変化しづらい手法は効果的に機能しないと予想される．

次に，log1 から log7 の全アプリケーションの占有率の発生頻度を図 16 から図 22 に示す．

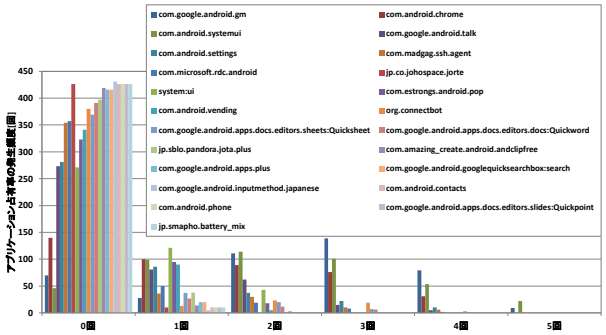

図 16 log1 におけるアプリケーションの占有率の発生頻度

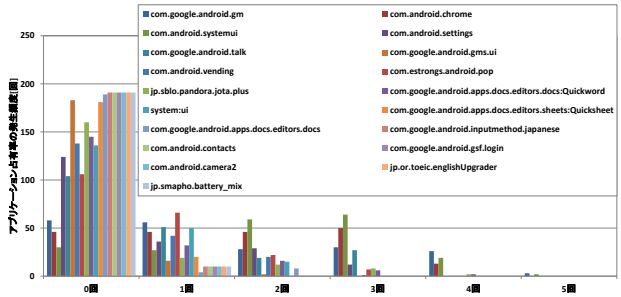

図 17 log2 におけるアプリケーションの占有率の発生頻度

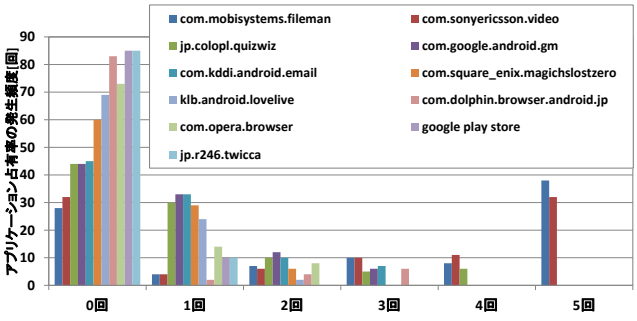

図 18 log3 におけるアプリケーションの占有率の発生頻度

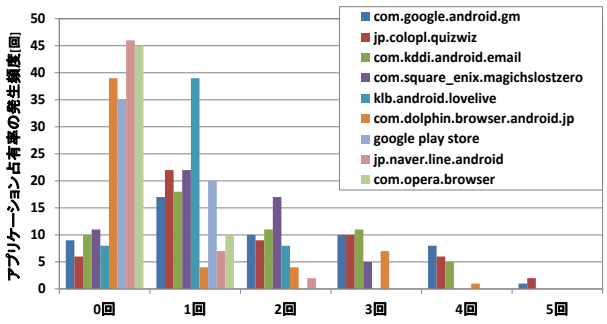

図 19 log4 におけるアプリケーションの占有率の発生頻度

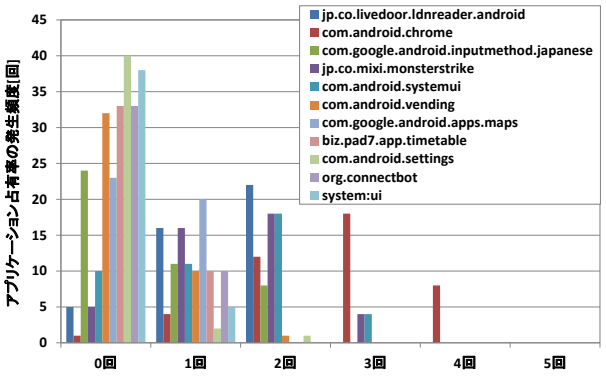

図 20 log5 におけるアプリケーションの占有率の発生頻度

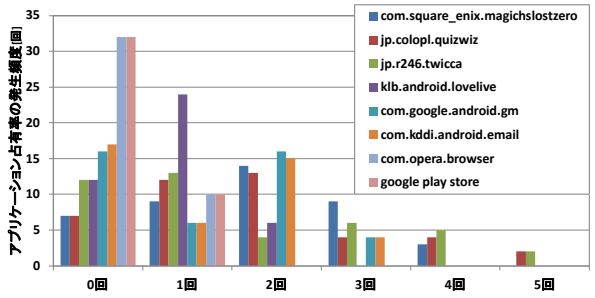

図 21 log6 におけるアプリケーションの占有率の発生頻度

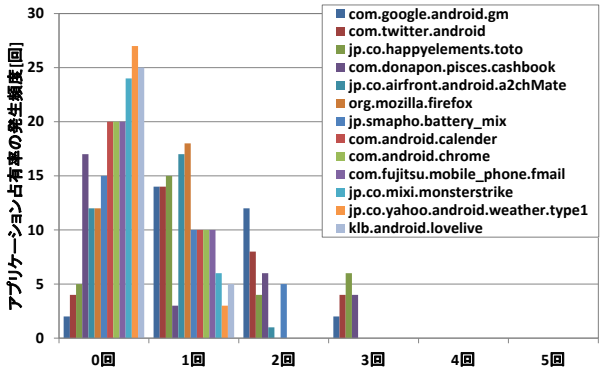

図 22 log7 におけるアプリケーションの占有率の発生頻度

図より，短期間内占有率 0%(0 回)が非常に多いにも関わらず，占有率 30%以上(3 以上)も多く，あるアプリケーショ

Vol.2015-GN-93 No.23
Vol.2015-CDS-12 No.23
Vol.2015-DCC-9 No.23
2015/1/27

ンが使用中になると集中的に使用される例が多いことがわかる．

参考のために，log2 の全アプリケーションにおける占有率の発生頻度と，マルコフ過程における占有率の発生頻度を図 23 に示す．マルコフ過程での発生頻度は，アプリケーションが 10 個あり，それらが等確率で起動される(ただし，同一のアプリケーションが連続して起動されることはない)ことを前提に，あるアプリケーションの短期間内占有率をシミュレーションにより求めた．

マルコフ過程を仮定すると(すなわち時間的局所性がないと仮定すると)，偶然特定のアプリケーションが短期間で集中して使われることが低いことが分かる．逆に，本履歴にはマルコフ過程と比較して非常に強い時間的局所性が存在することが分かる．このことからも，実履歴においてLRU が効果的に機能すると期待することができる．

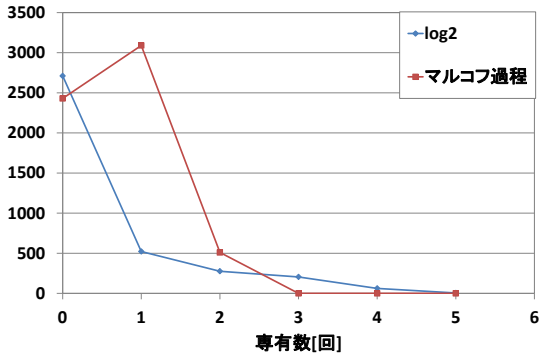

図 23 log2 におけるアプリケーション占有率の発生頻度とマルコフ過程による仮定

6. 関連研究

Android アプリケーションの起動時間の調査方法に関する研究としては，文献[3][6]の研究がある．これらにおいて，ユーザによるスクリーンのタッチからアプリケーション起動の完了までの時間の計測方法が示されている．また，実際のアプリケーション起動時間の測定結果と考察が示されている．本稿における起動時間の測定は当該研究で提案されている手法に基づき行われている．

Linux カーネルのモニタリングツールとしては，FTrace[7][8], SystemTap[9][10], LTTng[11][12], OProfile[13]がある．これらは Linux 用に構築されている．しかし，Linux 標準のアプリケーションプロセスの設計思想や実装と Android のアプリケーションプロセスの設計思想や実装は大きく異なるため，これらツールは Android の解析には適さない．具体的には本研究で必要となるアプリケーションフレームワークや Dalvik VM の動作の解析ができない．例えば，アプリケーションフレームワーク内におけるアプリケーションライフサイクル(`onCreate()`, `onRestart()`, `onStart()`, `onResume()`)の時刻を取得することができず，アプリケーション起動時間の調査をすることができない．

アプリケーション起動時間の短縮に関する研究としては，Zygote の preload クラスの増加による起動時間の短縮の研究[14]がある．当該研究は，Zygote による読込済みクラスの数を増加させることによりアプリケーション起動時におけるストレージからのクラス読み込み量を減少させ，起動時間の短縮を実現している．しかし，強制終了されたアプリケーションの起動時間の短縮を考える当該研究の手法と，強制終了するアプリケーションの最適化を考える本稿の手法はアプローチが大きく異なっている．また両手法は排他的な関係になく，両手法を同時に用いていくことが好ましいと考えられる．

7. おわりに

本稿で我々は low memory killer における強制終了プロセスの選定手法に着目し，LRU を用いる手法の妥当性を実際のユーザのアプリケーション起動履歴を用いて考察した．考察の結果，実際のアプリケーションの起動には強い偏り(特に参照の時間的局所性)があることがわかり，LRU に適していることが分かった．

今後は，端末の種類，使用用途，職業などによるアプリケーション起動の傾向の違いについて考察をしてく予定である．

謝辞

本研究は JSPS 科研費 24300034, 25280022, 26730040 の助成を受けたものである．

参考文献

1）Worldwide Smartphone OS Market Shere: http://www.idc.com/prodserv/smartphone-os-market-share.jsp
2）野村 駿, 中村 優太, 坂本 寛和, 山口 実靖：Android における LRU を用いた終了メモリ選定，第 10 回コンシューマ・デバイス＆システム(CDS)研究発表会，CDS10-7
3）永田恭輔，山口実靖：Android アプリケーションの起動性能解析システムとその評価，マルチメディア，分散，協調とモバイル(DICOMO2012)シンポジウム，pp83-90(2012)
4）野村 駿, 中村 優太, 坂本 寛和, 濱中 真太郎, 山口 実靖：Android OS におけるアプリケーション起動履歴を用いたプロセスメモリ管理，マルチメディア，分散，協調とモバイル(DICOMO2014)シンポジウム, pp. 1475 - 1481
5）Andrew S Tanenbaum (Author), Albert S Woodhull, "Operating Systems Design and Implementation," Prentice Hall, 3 edition, 2006
6）Kyosuke Nagata, Saneyasu Yamaguchi, "An Android Application Launch Analyzing System ," 8th ICCM: 2012 International Conference on Computing Technology and Information Management, (2012/04/24)
7）Steve Rostedt. ftrace tracing inftrastructure. http://lwn.net/Articles/270971/. SystemTap http://sourceware.org/systemtap/
8）Frank Ch. Eigler. “Problem solving with systemtap,” In Proceedings of the Ottawa Linux Symposium 2006, 2006.
9）LTTng Project http://lttng.org/
10）T. Bird, “Measuring Function Duration with Ftrace,” in Proc. of the Japan Linux Symposium, 2009.
11）M. Desnoyers, M. R. Dagenais, "The LTTng Tracer: A low impact

Vol.2015-GN-93 No.23
Vol.2015-CDS-12 No.23
Vol.2015-DCC-9 No.23
2015/1/27

performance and behavior monitor of GNU/Linux" Proceedings of Ottawa Linux Symposium 2006, 2006, pp. 209-223.
12）J. Levon and P. Elie. Oprofile: A system profiler for linux. http://oprofile.sf.net, September 2004.
13）Valgrind http://valgrind.org
14）Kyosuke Nagata, Yuta Nakamura, Shun Nomura and Saneyasu Yamaguchi, "Measuring and Improving Application Launching Performance on Android Devices", 4th International Workshop on Advances in Networking and Computing (WANC '13)

付録

1. log

log4～log7 は以下のような内容になっている．

表 2 log4 と log5 のアプリケーション起動履歴

log4	log5
com.square_enix.magichslostzero	com.android.settings
jp.naver.line.android	com.android.vending
com.kddi.android.email	com.android.settings
com.google.android.gm	jp.co.mixi.monsterstrike
klb.android.lovelive	system:ui
jp.colopl.quizwiz	com.android.chrome
com.dolphin.browser.android.jp	com.google.android.inputmethod.japanese
jp.colopl.quizwiz	com.android.chrome
jp.naver.line.android	com.android.systemui
klb.android.lovelive	jp.co.mixi.monsterstrike
jp.colopl.quizwiz	com.android.vending
com.dolphin.browser.android.jp	biz.pad7.app.timetable
jp.colopl.quizwiz	jp.co.livedoor.ldnreader.android
com.dolphin.browser.android.jp	com.android.chrome
jp.colopl.quizwiz	com.android.systemui
com.dolphin.browser.android.jp	jp.co.mixi.monsterstrike
jp.colopl.quizwiz	jp.co.livedoor.ldnreader.android
com.kddi.android.email	com.android.chrome
com.google.android.gm	com.android.systemui
com.square_enix.magichslostzero	com.android.chrome
klb.android.lovelive	com.android.systemui
google play store	jp.co.mixi.monsterstrike
com.kddi.android.email	jp.co.livedoor.ldnreader.android
com.google.android.gm	com.android.chrome
com.square_enix.magichslostzero	com.google.android.inputmethod.japanese
com.kddi.android.email	com.android.chrome
com.google.android.gm	com.google.android.inputmethod.japanese
klb.android.lovelive	com.android.chrome
jp.colopl.quizwiz	com.google.android.apps.maps
com.kddi.android.email	com.android.chrome
com.google.android.gm	jp.co.livedoor.ldnreader.android
com.kddi.android.email	com.android.systemui
com.google.android.gm	jp.co.mixi.monsterstrike
com.square_enix.magichslostzero	com.android.systemui
com.kddi.android.email	jp.co.mixi.monsterstrike
com.google.android.gm	com.android.chrome
com.kddi.android.email	jp.co.livedoor.ldnreader.android
com.google.android.gm	com.android.chrome
klb.android.lovelive	jp.co.mixi.monsterstrike
com.google.android.gm	org.connectbot
com.square_enix.magichslostzero	jp.co.livedoor.ldnreader.android
jp.colopl.quizwiz	com.android.chrome
com.kddi.android.email	com.google.android.apps.maps
com.google.android.gm	
jp.colopl.quizwiz	
com.square_enix.magichslostzero	
jp.colopl.quizwiz	
google play store	
com.square_enix.magichslostzero	
com.kddi.android.email	
com.google.android.gm	
klb.android.lovelive	
com.square_enix.magichslostzero	
jp.colopl.quizwiz	
com.opera.browser	

表 3 log6 と log7 のアプリケーション起動履歴

log6	log7
com.square_enix.magichslostzero	jp.co.happyelements.toto
klb.android.lovelive	com.twitter.android
com.square_enix.magichslostzero	jp.co.yahoo.android.weather.type1
klb.android.lovelive	com.google.android.gm
com.square_enix.magichslostzero	klb.android.lovelive
jp.colopl.quizwiz	jp.co.mixi.monsterstrike
com.square_enix.magichslostzero	com.donapon.pisces.cashbook
com.kddi.android.email	org.mozilla.firefox
jp.r246.twicca	jp.co.airfront.android.a2chMate
com.google.android.gm	jp.smapho.battery_mix
com.square_enix.magichslostzero	com.donapon.pisces.cashbook
klb.android.lovelive	com.google.android.gm
com.kddi.android.email	com.donapon.pisces.cashbook
com.google.android.gm	jp.co.happyelements.toto
jp.colopl.quizwiz	jp.smapho.battery_mix
com.square_enix.magichslostzero	com.twitter.android
com.kddi.android.email	com.android.calender
com.google.android.gm	jp.co.airfront.android.a2chMate
jp.colopl.quizwiz	org.mozilla.firefox
klb.android.lovelive	com.google.android.gm
com.square_enix.magichslostzero	com.twitter.android
com.kddi.android.email	com.android.chrome
com.opera.browser	com.twitter.android
com.google.android.gm	com.fujitsu.mobile_phone.fmail
com.kddi.android.email	jp.co.happyelements.toto
com.google.android.gm	com.google.android.gm
jp.colopl.quizwiz	jp.co.happyelements.toto
google play store	com.google.android.gm
jp.colopl.quizwiz	jp.co.happyelements.toto
jp.r246.twicca	com.twitter.android
jp.colopl.quizwiz	
jp.r246.twicca	
jp.colopl.quizwiz	
jp.r246.twicca	
jp.colopl.quizwiz	
com.square_enix.magichslostzero	
jp.r246.twicca	
com.square_enix.magichslostzero	
jp.r246.twicca	
com.square_enix.magichslostzero	
jp.r246.twicca	
klb.android.lovelive	