「マルチメディア，分散，協調とモバイル
(DICOMO2014)シンポジウム」 平成26年7月

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

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]

近年，Android OS は様々なデバイスで採用されており，普及率も高く重要性が高まっている．Android OS 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．本研究では，LRU に基づく強制終了プロセス選定手法，アプリケーションの起動時間に基づく手法，メモリ使用量に基づく手法，およびこれらを組み合わせた手法を提案する．そして，これら提案手法を Android OS に実装し，実機においてこれら提案手法を評価した結果を示す．さらに，これら提案手法と Android OS 標準手法におけるプロセスの新規起動，再開状況の詳細を示し，提案手法の有効性に関する考察を示す．

Process memory management using application history in Android Low Memory Killer

SHUN NOMURA[†1] YUTA NAKAMURA[†1] HIROKAZU SAKAMOTO[†1]
SHINTARO HAMANAKA[†2] SANEYASU YAMAGUCHI[†2]

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，2013 年第 2 四半期においてそのシェアは 81.3%[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．

本研究では，強制終了するプロセスの選定の改善によりプロセス再起動にともなうユーザの待ち時間を低減することを目的として，新しい選定手法を提案する．そして提案手法を Android に実装し，評価を行う．

†1 工学院大学大学院 工学研究科 電気・電子工学専攻
Electrical Engineering and Electronics, Kogakuin University Graduate School
†2 工学院大学 工学部 情報通信工学科
Department of information and Communications Engineering, Kogakuin University

2. Android OS

2.1 アーキテクチャ

Android OS はアプリケーション，アプリケーションフレームワーク，ライブラリ，Android ランタイム，Linux カーネルで構成されている[2].

アプリケーション，アプリケーションフレームワーク，ライブラリ, Android ランタイムには Android のために新規に開発された実装が多く含まれている．カーネルには Linux カーネルが使用されており，多くの部分は変更されることなくそのまま使用されている．ただし，次節で述べる low memory killer など Android 固有の機能も追加されている．

2.2 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，メモリの空き容量が閾値以下まで下がった場合に起動され，*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． low memory killer が起動されると，強制終了するプロセスの選定のために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．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．

2.3 *adj* と *minfree*

adj と *minfree* の関係は，`/init.rc` で定義されており，

前述の様に，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．*adj* はプロセスの状態を表しており，プロセスの状態が変化するたびに数値が増減する．プロセスの状態による *adj* の値を表 1 に示す．*adj* の値が小さいプロセスほど強制終了されにくく，フォアグラウンドのプロセスが最も強制終了されにくい．*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．Android4.0.3 における *adj* と *minfree* の関係を表 2 に示す．表 2 中の *minfree* の値の単位はページ[4KB]である．例えば，メモリ空き容量が 38000[KB]（9500[4KB]）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](9607[4KB])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．

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

表 2 Android4.0.3 における *adj* と *minfree* の関係

adj	*minfree*[4KB]
0	3674
1	4969
2	6264
4	8312
9	9607
15	11444

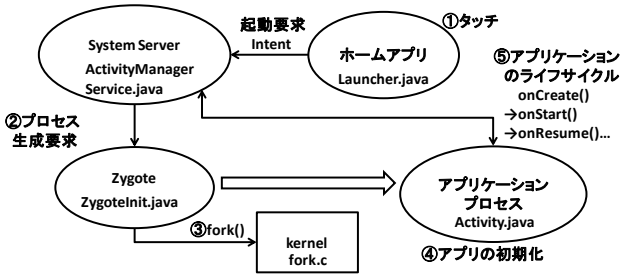

図 1 アプリケーションの新規起動手順

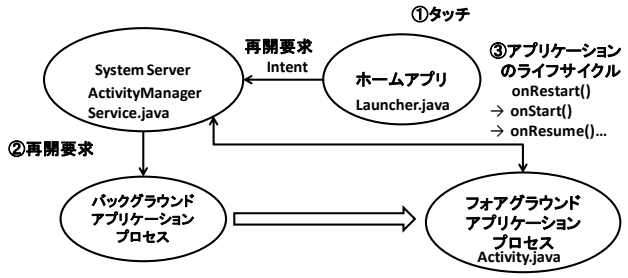

図 2 アプリケーションの再開手順

3. アプリケーションの起動手順

Android のアプリケーションを新規に起動する場合，図 1 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 2 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]. 本論文では前者を「新規起動」，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. 提案手法

本章で，LRU に基づいて low memory killer の強制終了プロセスを選定する LRU 手法と，アプリケーションの新規起動時間をもとに選定する起動時間手法と，これらを組み合わせた手法を提案する.

前述のように，low memory killer は第一次判定として *adj* による判定を行い，第二次判定としてメモリ使用量による判定を行う．以下における F-LRU および F-起動時間手法は第一次(First)判定に LRU や起動時間を用いる手法であり，

S-標準手法および S-メモリ×LRU 手法は，第二次(Second)判定にメモリ量やメモリ量×LRU を用いる手法である．

4.1 F-LRU 手法

本節で，強制終了するアプリケーションの一次判定を LRU 方式を用いて行う F-LRU 手法を提案する．

本手法では，ユーザが起動したアプリケーションを LRU により管理し，LRU 順にて新しい(最後の起動からの時間が短い)ものの *adj* を下げ強制終了の対象となりにくくする．

後述の評価用実装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す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．

4.2 F-起動時間手法

本節で，アプリケーションの新規起動時間を考慮して強制終了するプロセスの一次判定を行う F-起動時間手法を提案する．本手法では，アプリケーションの新規起動時間を計測する．そして，新規起動時間が長く，強制終了した場合に増加させるユーザの待ち時間が大きいアプリケーションを強制終了されにくくする．

アプリケーションの新規起動時間は，アプリケーションのライフサイクルにおける `onCreate()` の呼び出しから `onResume()` の呼び出しまでの時間とし，提案システムでは 5 章での手法によりアプリケーションごとの新規起動時間の履歴を取る．そして，履歴内のアプリケーションに対して新規起動時間の長い順に順位(rank)をつける．例えば，新規起動時間の一番長いアプリケーションのプロセスは，rank=0 となる．low memory killer による強制終了プロセス選定の際，アプリケーション履歴内に存在するアプリケーションのプロセスの rank を参照し，*adj* を rank/2 にすることで，新規起動時間の長いアプリケーションを強制終了されにくくする．これにより，アプリケーションの起動に長い時間を要しユーザの待ち時間を大きく増加させることを回避できると期待される．

後述の評価用実装では，アプリケーションの新規起動時間の履歴は最新の 15 個を保持し，それより古いものを破棄する．よって，長時間前に起動し最近使用していないアプリケーションが長期にわたり強制終了の対象とならないことは生じない．履歴の長さと *adj* の下げ幅はパラメータである．

4.3 F-標準手法

比較のため，第一次判定を *adj* の比較で行う標準 low memory killer の手法を本稿では F-標準手法と呼ぶ．

4.4 F-LRU×起動時間手法

本節で，F-LRU 手法と F-起動時間手法を組み合わせて第一次判定を行う F-LRU×起動時間手法を提案する．

本手法では，F-LRU 手法と F-起動時間手法を並列して動作させ，両値の小さい方をそのプロセスの *adj* とする．

4.5 S-メモリ×LRU 手法

本節では，まず S-標準手法を定義し，続いて S-メモリ×LRU 手法の提案を行う．

4.5.1 S-標準手法

本手法では，強制終了プロセス選定の第二次判定をプロセスのメモリ使用量で行い，よりメモリ使用量の多いプロセスを強制終了する対象とする．

本手法は，標準 low memory killer の動作である．比較のため，これを S-標準手法と呼ぶ．

4.5.2 S-メモリ×LRU 手法

本項で，強制終了プロセス選定の第二次判定をプロセスのメモリ使用量と LRU 順の積で行う手法を提案する．アプリケーションが LRU の履歴内に存在しない場合は，LRU 順として(履歴長+1)を用いる．

5. 実装

提案手法を実装するために，Android のカーネルおよびフレームワークに変更を加えた．以下にその詳細について述べる．

5.1 カーネル内部

カーネル内部ではまず，`lowmemorykiller.c` において low memory killer が強制終了するプロセスを選定する箇所に変更を加えた．

F-LRU 手法では，次節のユーザ空間プログラムよりアプリケーション起動時刻の情報を得て，アプリケーションの起動順をカーネル内の配列にて LRU を用いて管理する．

F-起動時間手法では，カーネルがユーザ空間より `/proc` インターフェイスを介してアプリケーションの新規起動時間の情報を受け取り，それをカーネル内の配列に格納して履歴として管理し，新規起動時間の長い順に rank をつける．そして，メモリが不足し強制終了プログラムを選定する際には，アプリケーション履歴内に存在するアプリケーションのプロセスの rank を参照し，*adj* を rank/2 とする．

5.2 ユーザ空間部

ユーザ空間部では，アプリケーションフレームワークの実装にアプリケーションライフサイクルメソッド(`onCreate()` など)の呼び指し時刻を記録する機能と，記録されたメソッド呼び出し時刻とアプリケーション起動時間を `/proc` インターフェイスを通じてカーネル内の low memory killer に伝える機能を追加した．F-LRU 手法ではア

プリケーションの起動時刻を，F-起動時間手法ではアプリケーションの新規起動時間を伝えている．

6. 評価

本章にて，標準 low memory killer(以下標準手法と呼ぶ)および提案手法のアプリケーションの合計起動時間の評価を行う．

6.1 評価方法

標準手法および提案手法が実装された Android スマートフォンにおいて，複数のアプリケーションを順に起動し，その起動に要した時間を測定した．実験は，表 3 に示す仕様のスマートフォンを用いて行った．

表 3 実験環境

Device name	Nexus S
OS	Android4.0.3
CPU	Cortex A8（Hummingbird）Processer 1GHz
Memory	512MB

6.2 評価シナリオ

本稿では，起動するアプリケーションの順番を定めたものを “シナリオ”と呼ぶ．本実験では，第三者から許可を得て取得したユーザの実際の 1 日のアプリケーション使用履歴であるシナリオを用意し，標準手法と全ての提案手法の評価を行った．(本シナリオにおけるアプリケーション起動順は表 5 参照．)

アプリケーション起動とアプリケーション起動の間にホームアプリケーション(Launcher)を起動し，実際の使用順に近いものとしている．

6.3 合計起動時間

本シナリオにおける評価結果を図 3 に示す．図の縦軸は，シナリオのホームアプリケーション（Launcher）以外の全アプリケーションの起動時間の合計であり，少ないほど優れていると言える．

また，本シナリオにおけるそれぞれの手法のアプリケーションの新規起動，再開の数を表 4 に示す．

通常，新規起動による起動時間の方が再開による起動時間より長いため，基本的に新規起動の回数が少ないほど合計起動時間が短くなり，優れた手法であると考えることができる．ただし，新規起動時間はアプリケーションにより異なるため，新規起動時間の回数が同一でも合計起動時間に差が生じる．

図と表より，両シナリオにおいて標準手法より全ての提案手法の方が合計起動時間が短く，また新規起動の回数も少ないまたは同等であることが確認でき，標準手法よりも優れた手法であることがわかる．

提案手法同士を比較すると，F-LRU-S-メモリ手法と F-LRU-S メモリ×LRU 手法が両シナリオにおいて優れた性能を示しており，第一次判定においては LRU を用いることが好ましいことが分かる．

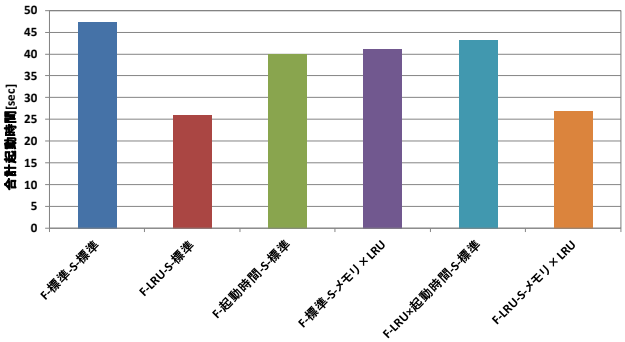

図 3 シナリオにおける評価結果

表 4 シナリオにおける新規起動数と再開数

手法	新規起動回数	再開回数
F-標準-S-標準	76	320
F-LRU-S-標準	35	361
F-起動時間-S-標準	74	322
F-標準-S-メモリ×LRU	76	320
F-LRU×起動時間-S-標準	33	363
F-LRU-S-メモリ×LRU	32	364

6.4 アプリケーションごとの起動時間

本節にて，各手法のアプリケーションごとの新規起動回数，再開回数の評価を行う．

各アプリケーションの新規起動時間と再開時間の平均および標準偏差を図 4~7 に示す．図 5 や図 7 の一部のアプリケーションはグラフが表示されていないが，これは再開回数が 0 や 1 であるものである．

また，本シナリオ内における起動回数(新規起動回数＋再開回数と同じ)を図 8 に示す．

各手法の再開率(再開回数/(新規起動回数＋再開回数))を図 9 に示す．新規起動時間や再開時間に着目すると，アプリケーションごとに大きな差があることが分かる．また，それらの標準偏差に着目すると，一部のアプリケーションにて大きな値がみられ，アプリケーションの新規起動や再開の時間に大きなばらつきがあることが分かる．

手法ごとの再開率に着目すると，標準手法(F-標準-S-標準)は，いずれのアプリケーションも再開率が低く，提案手法とくらべ良い動作をできていないことが分かる．提案手法の中でも，F-LRU-S-標準手法や F-LRU-S-メモリ×LRU 手法に着目すると，シナリオの中で起動回数が多いアプリケーションにおいて，高い再開率を実現していることが分かる．

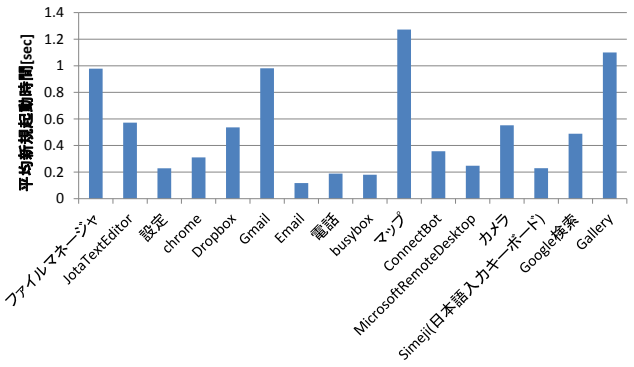

図 4 各アプリケーションの平均新規起動時間

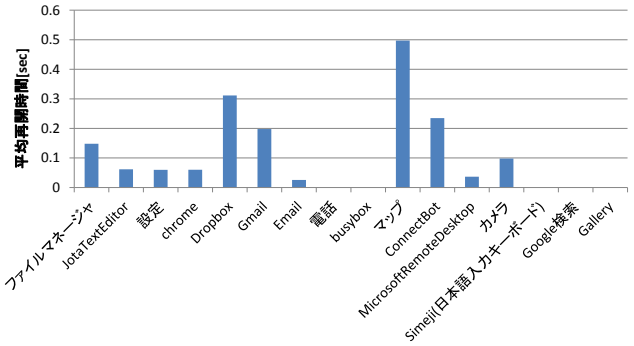

図 5 各アプリケーションの平均再開時間

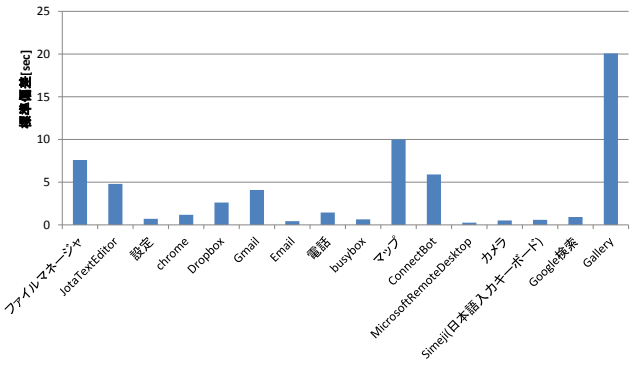

図 6 各アプリケーションの新規起動時間の標準偏差

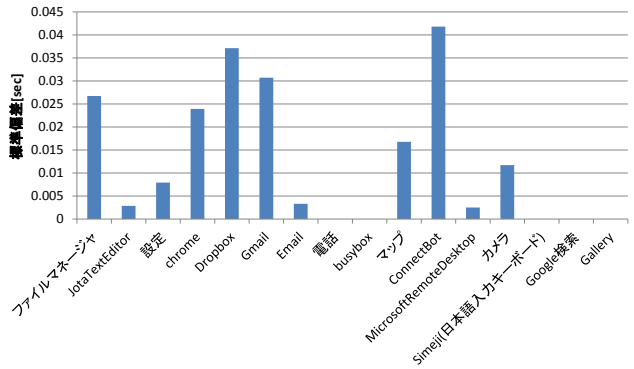

図 7 各アプリケーションの再開時間の標準偏差

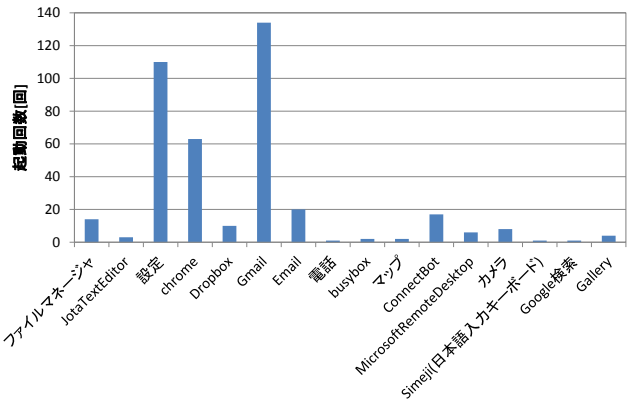

図 8 各アプリケーションの起動回数

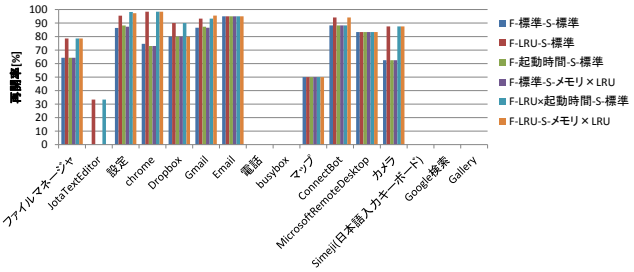

図 9 各手法のアプリケーションごとの再開率

7. 考察

今回提案した手法では，アプリケーションの新規起動時間は `onCreate()` の開始時刻から `onResume()` の開始時刻までとしており，再開時間は `onRestart()` の開始時刻から `onResume()` の開始時刻までとしている．しかし，アプリケーションによっては `onResume()` が開始した後にデータの読み込みやページの読み込みを行うことがあり，これによりユーザには待ち時間が生じてしまうことが考えられる．この様な場合，実際にユーザの待ち時間の増加の程度は，本稿で定義した「アプリケーション起動時間の増大量」を大きく上まわることになり，提案手法により得られるユーザの体感的待ち時間の低減は前章の評価をさらに上まわるものになると期待できる．例えば，Web ブラウザアプリケーションの多くがこれに該当し，プロセスが強制終了されてしまった後にアプリケーションを使用すると `onResume()` の開始後に以前閲覧していたページの再読み込みが発生し，ユーザはこの間待たされることとなる．また，タブ型のブラウザの場合は閲覧タブの変更を行うたびに以前は読み込み済であったページの再読み込みが生じユーザが待たされることとなる．そのため，本稿で定めたアプリケーションの起動ではなく，`onResume()` の開始後に発生する処理に要する時間も考慮した手法を適用することでユーザの待ち時間のさらなる軽減を図ることができると予想される．

8. 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][4]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[5][6]，SystemTap[7][8]， LTTng[9][10]，OProfile[11]がある．しかし，これらは Linux 用に構築されているため Android の解析には適さず，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，Zygote の preload クラスの増加による起動時間の短縮の研究[12]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込量を減少させ，起動時間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目的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．

9. おわりに

本論文で我々は 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 による選定手法とアプリケーションの起動時間を考慮した選定手法等を提案した．そしてこれらの手法の評価結果を示し，提案手法が標準手法よりもアプリケーション起動時間を短縮できることを示した．また，全提案手法の中でも特に LRU による手法が優れていることを示した．

今後は，`onResume()`の開始後に発生する処理に要する時間も考慮した選定手法について考察していく予定である．

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．

表 5 シナリオ B におけるアプリケーション起動順

File Manager	Email	Google Chrome	Setting
JotaTextEditor	Email	Dropbox	Gmail
File Manager	Email	Dropbox	Gmail
Setting	Setting	Dropbox	Gmail
Setting	Setting	Google Chrome	Gmail
Setting	Setting	Dropbox	Google Chrome
Setting	Setting	Dropbox	Gmail
Setting	Setting	Dropbox	Gmail
Setting	Setting	Dropbox	Gmail
Google Chrome	Setting	Dropbox	Gmail
Dropbox	Setting	Dropbox	Gmail
Google Chrome	Email	Map	Gmail
Gmail	Email	Gmail	Google Chrome
Gmail	Setting	Gmail	Gmail
Gmail	Setting	Gmail	Google Chrome
Google Chrome	Setting	Gmail	Gmail
Setting	Email	Gmail	Gmail
Setting	Setting	Setting	Gmail
Google Chrome	Setting	Setting	Gmail
Google Chrome	Setting	Gmail	Gmail
Gmail	Gmail	JotaTextEditor	Gmail
Gmail	Google Chrome	Gmail	Gmail
Gmail	Google Chrome	Gmail	Gmail
Setting	Setting	Gmail	Camera
Setting	Setting	Gmail	Gmail
Gmail	Google Chrome	Gmail	Gmail
Google Chrome	Google Chrome	Map	Camera
Google Chrome	Google Chrome	ConnectBot	File Manager
Google Chrome	Setting	ConnectBot	Gallery
Google Chrome	Setting	ConnectBot	File Manager
Gmail	Setting	ConnectBot	File Manager
Google Chrome	Setting	ConnectBot	Gmail
Google Chrome	Setting	ConnectBot	File Manager
Gmail	Setting	ConnectBot	Google Chrome
Google Chrome	Setting	ConnectBot	Setting
Google Chrome	Setting	Google Chrome	Setting
Google Chrome	Setting	Setting	Setting
Gmail	Setting	Setting	Setting
Gmail	Setting	Setting	Gmail
Gmail	Setting	Setting	Gmail
Gmail	Setting	Google Chrome	Gmail
Gmail	Google Chrome	ConnectBot	Gmail
Gmail	Setting	ConnectBot	Gmail
Gmail	Setting	ConnectBot	Gmail
Gmail	Setting	ConnectBot	Gmail
Gmail	Setting	ConnectBot	Google Chrome
Google Chrome	Setting	ConnectBot	Gmail
Google Chrome	Phone	ConnectBot	Gmail
Google Chrome	Setting	MicrosoftRemoteDesktop	Gmail
Google Chrome	Setting	MicrosoftRemoteDesktop	Camera
Google Chrome	Setting	MicrosoftRemoteDesktop	File Manager
Gmail	Gmail	MicrosoftRemoteDesktop	Camera
Google Chrome	Gmail	ConnectBot	File Manager
Google Chrome	Gmail	ConnectBot	Gallery
Google Chrome	Google Chrome	MicrosoftRemoteDesktop	File Manager
Google Chrome	Setting	MicrosoftRemoteDesktop	Gallery
Google Chrome	Setting	Google Chrome	File Manager
Setting	Setting	Setting	Gallery
Setting	Setting	Setting	File Manager
Setting	Gmail	Setting	File Manager
Setting	Gmail	Setting	Gmail
Setting	Gmail	Setting	Gmail
Setting	Gmail	Gmail	Gmail
Setting	Gmail	Gmail	Gmail
Setting	Gmail	Gmail	Gmail
Setting	Gmail	Gmail	Simeji(日本語入力キーボード)
Setting	Gmail	Gmail	Gmail
Gmail	Gmail	Gmail	Gmail
Gmail	Gmail	Google Chrome	Gmail
Gmail	Gmail	Google Chrome	Gmail
Gmail	Gmail	Google Chrome	Gmail
Gmail	Gmail	Google Chrome	Gmail
Gmail	Gmail	Gmail	Google Chrome
Gmail	Gmail	Gmail	Google Chrome
Gmail	JotaTextEditor	Gmail	Google Chrome
Gmail	Setting	Gmail	Gmail
Gmail	Setting	Gmail	busybox
Setting	Google Chrome	Gmail	Google Chrome
Setting	busybox	Gmail	Google Chrome
Setting	Setting	Setting	Gmail
Email	Setting	Setting	Google Chrome
Setting	Setting	Setting	Google Chrome
Email	Setting	Setting	Google Chrome
Setting	Setting	Setting	Google Chrome
Email	Setting	Camera	Google Chrome
Setting	Setting	Camera	Google Chrome
Email	Gmail	Setting	Google Chrome
Email	Google Chrome	Setting	Google検索
Email	Gmail	Setting	Gmail
Email	Gmail	Gmail	Gmail
Email	Google Chrome	Gmail	File Manager
Gmail	Setting	Setting	Gmail
Email	Setting	Setting	Gmail
Email	Setting	Setting	Gmail
Email	Setting	Setting	Gmail
Email	Google Chrome	Camera	Gmail
Email	Google Chrome	Camera	File Manager
Email	Google Chrome	Setting	Gmail

参考文献

1）Android Captures Record 81 Percent Share of Global Smartphone Shipments in Q3 2013: https://blogs.strategyanalytics.com/WSS/post/2013/10/31/Android-Captures-Record-81-Percent-Share-of-Global-Smartphone-Shipments-in-Q3-2013.aspx

2）What is Android?|Android Developers: http://developer.android.com/guide/basics/what-is-android.html

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）Kyosuke Nagata, Saneyasu Yamaguchi, "An Android Application Launch Analyzing System ," 8th ICCM: 2012 International Conference on Computing Technology and Information Management, (2012/04/24)

5）Steve Rostedt. ftrace tracing inftrastructure. http://lwn.net/Articles/270971/. SystemTap http://sourceware.org/systemtap/

6）Frank Ch. Eigler. “Problem solving with systemtap,” In Proceedings of the Ottawa Linux Symposium 2006, 2006.

7）LTTng Project http://lttng.org/

8）T. Bird, “Measuring Function Duration with Ftrace,” in Proc. of the Japan Linux Symposium, 2009.

9）M. Desnoyers, M. R. Dagenais, "The LTTng Tracer: A low impact performance and behavior monitor of GNU/Linux" Proceedings of Ottawa Linux Symposium 2006, 2006, pp. 209-223.

10）J. Levon and P. Elie. Oprofile: A system profiler for linux. http://oprofile.sf.net, September 2004.

11）Valgrind http://valgrind.org

12）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)