Vol.2018-GN-103 No.9
Vol.2018-CDS-21 No.9
Vol.2018-DCC-18 No.9
2018/1/26

アプリケーションのオブジェクト作成履歴に基づく Android 世代別 GC の Promote 制御の改善に関する一考察

森竜佑[†1]　神山剛[†2]　福田晃[†2]　小口正人[†3]　山口実靖[†1]

概要：ART (Android Runtime)の機能に，GC(Garbage Collection)があり，ART には様々な GC アルゴリズムが実装されている．ART の GC の 1 つに GSS と呼ばれる世代別 GC がある．世代別 GC では，ゴミになりづらいと判断したオブジェクトを Promote させ，不要な GC 処理を削減し，性能を向上させている．よって，その Promote の条件のチューニングは重要であると考えられる．過去の研究にて，アプリケーションが生成するオブジェクトのサイズと寿命の一般的な関係を考慮した Promote 制御手法が提案されている．しかし，当該手法に用いられているオブジェクトの統計情報は一般アプリケーションの情報であるため，対象アプリケーションにおけるオブジェクトの統計情報を取得する必要があると考えられる．本研究では，各アプリケーションにおけるオブジェクトのサイズや寿命などの統計情報を取得できるモニタ機能付き GSS を実装し，当該実装を用いて対象アプリケーションの統計情報の取得を行い，モニタ情報に基づき GSS の Promote 条件を制御する手法を提案する．そして，性能評価を行いモニタ情報に基づく手法が，モニタ情報を用いない手法よりも高い性能となることを示す．

キーワード：Android，ART，Garbage Collection，世代別 GC，オブジェクト寿命

1. はじめに

スマートフォンやタブレット PC が普及し，これら端末の重要性が高まっている．Android はこれらの端末のプラットフォームとして高いシェアを持ち，2017 年第 1 四半期の世界市場における全スマートフォンの出荷台数における Android のシェアは 85.0%である[1]．また，スマートフォン，タブレット PC 以外にも，腕時計，音楽プレイヤー，カーナビなど様々なデバイスで採用されており，重要性が高まっている．

Android アプリケーションは ART（Android Runtime）上で動作する．ART のメモリ管理機能の 1 つに GC があり，GC が不必要なメモリを解放する．ART の GC の動作を制御することにより，プロセスのメモリサイズやアプリケーションの性能を制御することができる．

ART に実装されている GC の一つに世代別 GC[2]がある．世代別 GC ではオブジェクトに年齢という概念を取り入れ，年齢ごとにオブジェクトを分類し GC を行う．ART における世代別 GC では，メモリ領域を新世代領域と旧世代領域に分類し，新世代領域内 GC で一定回数回収されなかったオブジェクトは旧世代領域へと Promote(昇進)する．また過去の研究にて，アプリケーションが生成するオブジェクトのサイズと寿命の関係[3][4]を考慮した Promote 制御手法が提案されている[5]．しかし，本手法で用いたオブジェクトの統計情報は一般的なアプリケーションにおける情報であり，GC 対象となっているアプリケーションにおける情報ではない．よって，対象アプリケーションのオブジェクトの統計情報を取得し，それにより Promote 条件を制御することによりさらなる性能向上を実現できると考えられる．

本研究では，オブジェクトのサイズや寿命などの統計情報を取得できる ART Monitor [2]を GSS に実装し，当該実装を用いて対象アプリケーションのオブジェクト寿命の推定を行い，推定寿命を用いて GSS の Promote 条件を制御する手法を提案する．そして，性能評価によりその有効性を示す．

2. 世代別 GC

2.1 Garbage Collection (GC)

GC は，ゴミオブジェクトを見つけて，そのメモリを解放する機能である．図 1 は、GC の動作の概要を示している．メモリ空間にはオブジェクトが割り当てられている．これらの中でどこからも参照されないオブジェクトをゴミオブジェクトと呼ぶ．アプリケーション実行環境は，オブジェクトがゴミオブジェクトであるかどうかを GC を実行して検出する．Mark and Sweep (MS) [7]などの GC は図 2 のようにルートオブジェクトからの参照を再帰的に参照し，ゴミオブジェクトを見つける．GC はルートオブジェクトから再帰的にアクセスできるオブジェクトをマークしていき，マークされていないオブジェクトはゴミオブジェクトであり，これらは解放（Sweep）される．また，GC の実行中はすべてのアプリケーションスレッドが停止する．この現象は Stop The World（STW）と呼ばれている．

†1 工学院大学
Kogakuin University
†2 九州大学
Kyusyu University
†3 お茶の水女子大学
Ochanomizu University

Vol.2018-GN-103 No.9
Vol.2018-CDS-21 No.9
Vol.2018-DCC-18 No.9
2018/1/26

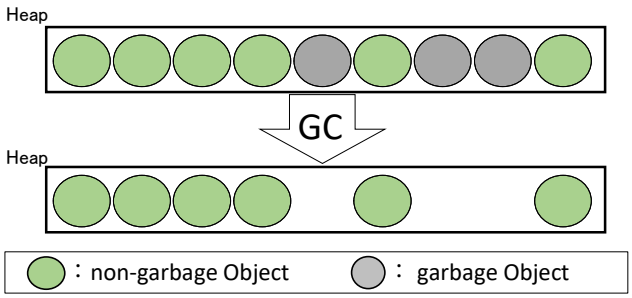

図 1 Garbage Collection

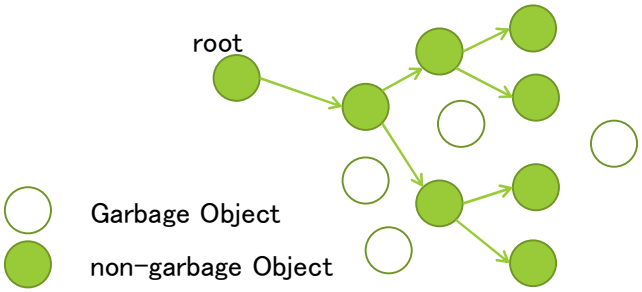

図 2 Mark and Sweep

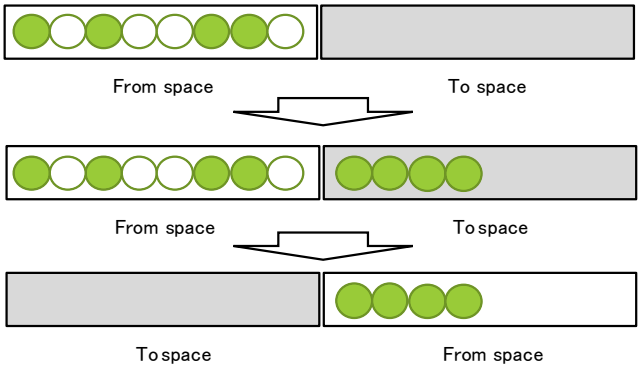

図 3 Copying GC

2.2 世代別 GC アルゴリズム

本節では，世代別 GC アルゴリズムについて述べる．世代別 GC とは,「生成されたばかりのオブジェクトはすぐに GC によって回収され，回収されずに長く生き残るオブジェクトは少ない」という経験に基づいた仮説をベースとしたアルゴリズムである．

生成されてすぐに死んでしまう短命オブジェクトと長時間生き残る長寿オブジェクトを分けて管理し，GC 対象となるオブジェクトを限定することで GC の処理時間の削減をはかっている．メモリ空間は図 3 のように新世代領域と旧世代領域に分けられ，新世代領域には生成されてから間もないオブジェクトが格納され，旧世代領域には生成されてから一定回数の GC を経験しても回収されなかったオブジェクトが格納される．

アプリケーションがオブジェクトを生成すると，オブジェクトはまず新世代領域に割り当てられ，新世代領域がいっぱいになると新世代領域を対象とした GC が行われる．一般的な世代別 GC では，新世代領域は図 4 に示すようなコピーGC[8]を用いて管理される．この GC によってゴミオブジェクトは回収され，一定回数の GC を生き延びたオブジェクトは新世代領域から旧世代領域へ移動される．これは Promote（昇進）と呼ばれる．また，旧世代領域がいっぱいになると両領域を対象とした GC を行い，旧世代領域も含めてゴミオブジェクトを回収する．

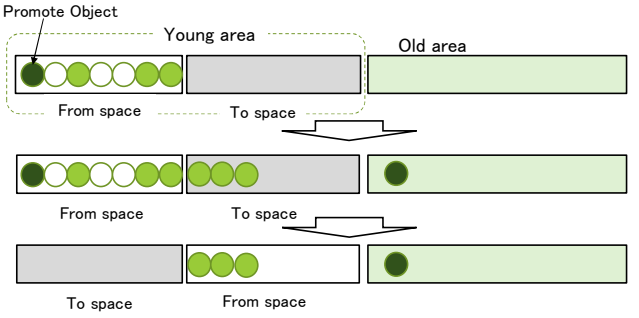

図 5 Generational semi space

2.3 Generational Semi Space

ART には Generational Semi Space(GSS)という世代別 GC が実装されている．GSS ではオブジェクトを新世代領域に入るものと旧世代領域に入るものの 2 グループに分けて管理し，範囲の異なる 2 つの GC を行う．

生成されたばかりのオブジェクトは新世代領域に置かれ，bpsGC(bump pointer space GC)によって高速にかつ積極的に回収される．2 回の bpsGC を経験し生き残ったオブジェクトは，旧世代領域へと Promote される．そして Promote されたオブジェクトの累積バイト数が PromotedThreshold（初期設定にて 4 MB）を超えるたびに whole GC が行われる．whole GC の対象範囲は両領域であり，GC 時間が長くなる．よって，頻繁に whole GC を行うことはアプリケーションの性能上は好ましくないと考えられる．ただし，旧世代領域に Promote されてから参照されなくなったオブジェクトは bpsGC の調査範囲外となってしまい，次回の whole GC 実行まで回収できなくなってしまう．よって，whole GC の回数の削減はヒープサイズ（メモリ消費量）拡大の原因になると考えられる．したがって，ヒープサイズの縮小と GC 処理時間の短縮はトレードオフの関係にあると考えられる．例えば，wholeGC の実行回数を増やすことにより，アプリケーションの性能を低下させつつ，アプリケーションのメモリ消費量の減少を実現できる．

Vol.2018-GN-103 No.9
Vol.2018-CDS-21 No.9
Vol.2018-DCC-18 No.9
2018/1/26

3. Android アプリケーションにおけるオブジェクトサイズや寿命の調査

本章にて，一般公開されている既存のアプリケーション(Google Map)における生成オブジェクトのサイズや寿命などの特性の傾向についての調査結果を示す．

3.1 ART Monitor in GSS

文献[3]にて，Android のアプリケーションにおいて生成オブジェクトの特性の調査方法が提案されている．同文献にて用いられている ART Monitor を改変し，GSS 実行時のオブジェクトのモニタリングを可能にした．

3.2 測定方法

GSS 用 ART Monitor を実装した Android OS を測定端末にインストールし，Google Map アプリケーションをインストールした．Google Map を起動し，同アプリケーションに文献[3]の処理を実行させ，GC の動作を観察した．測定環境は表 1 の通りである．

表 1 測定環境

端末名	Nexus7(2013)
OS	Android 6.0.1
CPU	Snapdragon S4 Pro 1.5GHz
メモリ	2GB

3.3 測定結果

図 5 にオブジェクトサイズ別の生成分布を示す．図より，17~32 バイトのオブジェクトが多く，サイズが大きくなるにつれて生成数が減少していることが確認できる．

図 6 にオブジェクトサイズと寿命の関係を示す．本稿では文献[3][4]と同様に経験した GC の回数をオブジェクトの年齢と定義し，回収されたときの年齢を寿命と定義する．例えば，生成されて最初の GC で回収されたオブジェクトの寿命は 0 であり，生成されて 3 回の GC を経験して生き残り 4 回目の GC で回収されたオブジェクトの寿命は 3 である．したがって，図 6 の Y 軸の「寿命 N 率」とは年齢 N で回収された（寿命 N）オブジェクトの割合である．図 6 より 8 バイト以下のオブジェクトは寿命 0 率が 40 %ほどで，寿命 3 率も同様に 40 %程度であり，サイズ 9~256 バイトやサイズ 1025~8192 バイトなどと比較して平均寿命が少し高くなっている．同じことがサイズ 257~1024 バイトのオブジェクトに対しても言える．また，9~16 バイト，65~256 バイトのオブジェクトは，寿命 0 率が非常に高く 80 %を超えていることがわかる．そして，16384 バイト以上のオブジェクトは寿命 0 率が非常に低く，長寿なオブジェクトの割合が高く平均寿命も高いことがわかる．

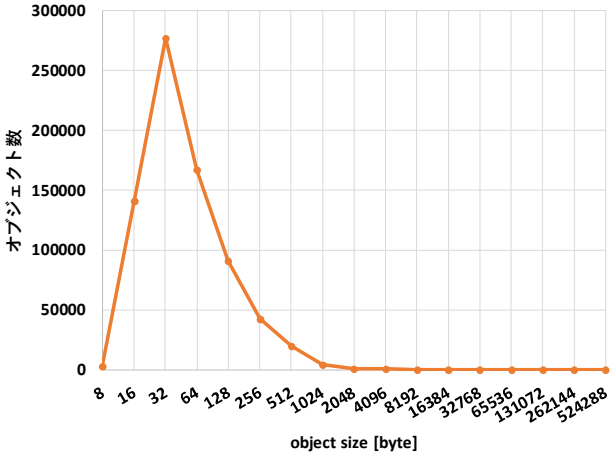

図 5 サイズ別のオブジェクト生成分布

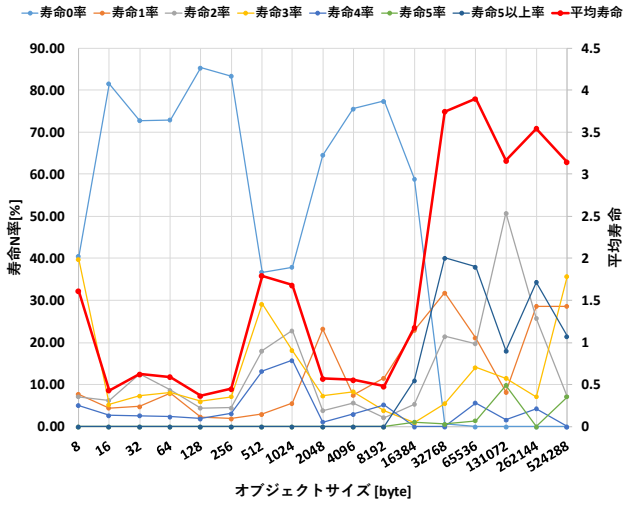

図 6 サイズと寿命の関係

3.4 考察

図 6 より，オブジェクトサイズを 9~16 バイトと 65~256 は平均寿命が特に短く，寿命 0 率が特に高くなっていることがわかる．これらのオブジェクトの Promote は，不適切（Promote させたが近い将来にゴミオブジェクトとなってしまい，長期間旧世代領域で回収されないメモリを消費し続ける）Promote となる可能性が高いことが予想でき，これらのオブジェクトに Promote を消極的にすることにより GSS の性能を改善できると予想できる．

Vol.2018-GN-103 No.9
Vol.2018-CDS-21 No.9
Vol.2018-DCC-18 No.9
2018/1/26

4. オブジェクトサイズを考慮した Promote 抑制手法

文献[5]では，オブジェクトの統計情報[3][4]を考慮した Promote 条件の制御によって，より少ない STW 時間増加でヒープサイズの縮小を実現する手法が提案されている．本章にて，オブジェクトサイズを考慮した Promote 抑制手法を紹介する．

本手法では，GC を 2 回以上経験したオブジェクトのうちサイズが m バイト以下のオブジェクトは確率 p で Promote させ，確率 1-p で Promote させない様に制御する．サイズがそれ以上のオブジェクトは，通常通り GC を 2 経験すると必ず Promote させる．

一般的なアプリケーションにおいて，サイズが小さいオブジェクトは短命である確率が高いという傾向が確認されている[3]．その傾向を用いて，サイズが小さなオブジェクトを Promote されることを抑制することにより，近い将来にゴミになってしまうオブジェクトが旧世代領域に Promote されてしまい GC による回収を長期間回避してしまう現象を減少できると期待できる．

ただし，本手法は一般的アプリケーションにおける傾向（統計情報）を個でのアプリケーションに対して適用しており，各アプリケーション固有の特徴は考慮されていない．同様に，当該文献[5]においてはパラメータ m の設定に関する議論はされていない．

5. 提案手法

本章では，対象アプリケーションのモニタリングに基づく世代別 GC の Promote 制御手法を提案する．

本手法では，GC を 2 回以上経験したオブジェクトのうちサイズが特定の範囲に入るオブジェクトは確率 p で Promote させ，確率 1-p で Promote させない様に制御する．サイズが同範囲外のオブジェクトは，通常通り GC を 2 経験すると必ず Promote させる．

Promote を抑制させる範囲は，ART Monitor 上で対象アプリケーションを事前に動作させ，オブジェクトサイズと平均寿命や寿命 0 率を調査し，その結果に基づき定める．

既存手法[6]と比較すると，図 7 のようになる．既存手法では，サイズが m バイト以下のオブジェクトは確率 p と 1−p で Promote を実行と保留をし，m はチューニングパラメータとして情報なく使用者が定める必要がある状態になっている．結果，既存の文献[6]では一般的なアプリケーションの情報[3][4]に基づき決定している．これに対して本稿の提案手法は，対象アプリケーションの情報を得て，それを基に定めることになる．

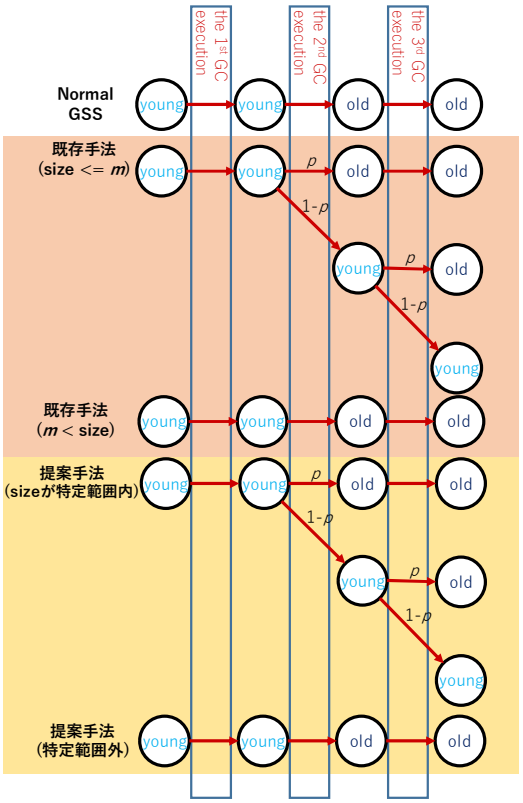

図 7 既存手法と提案手法

6. 性能評価

本章にて，提案手法の性能評価を行う．

6.1 評価方法

性能評価には Google Map（バージョン V9.50.2）を用い，アプリケーション起動中の GC の動作を観察した．Google Map 起動中の処理は全て自動化し，次のように処理をした．山手線の駅を東京駅から外回りに検索していき，最後の神田駅を検索し画面に表示された時点までの GC の動きを観察した．ある駅と次の駅を検索するまでの間隔は 5 秒である．

評価対象の手法と各パラメータの値は表 2 のとおりである．Promote を抑制する範囲は，既存手法[6]は同文献の通りの 16 バイト以下としてある．提案手法では，図 6 により，特に寿命 0 率が高く，平均寿命が短い「9~16 バイト」と「9~16 or 65~256 バイト」の 2 種類を用意した．前者は既存手法に合わせて範囲を一か所としてある．後者はモニタ結果に基づきより複雑な設定をした場合となっている．後者のように設定はモニタ結果が得られない状況では選択しづらく，モニタリングに基づく手法の特徴的な最適化と考えることができる．また 1 回の GC で Promote されるオブジェクトが少なくなることを考慮し，PromotedThreshold を 1 MB に変更した．測定に用いた端末は Nexus7 であり，仕様は表 1 のとおりである．ただし，OS は 3 章で行った

Vol.2018-GN-103 No.9
Vol.2018-CDS-21 No.9
Vol.2018-DCC-18 No.9
2018/1/26

測定とは別である．

表 2 評価対象の手法と各パラメータ

手法	パラメータ m	パラメータ p
Normal		
既存手法	0~16	0.5
提案手法	9~16	0.5
提案手法	9~16 or 65~256	0.5

6.2 評価結果

図 8 に平均ヒープサイズを示す．平均ヒープサイズとは，各 GC 処理後のヒープサイズの平均である．結果より，既存手法と提案手法いずれも通常手法と比較してヒープサイズを縮小できていることを確認できる．縮小の効果はほぼ同等であり，提案手法がわずかに上回っている状態である．

図 9 に各 GC 処理中の STW 時間の平均を示す．結果より，既存手法と提案手法いずれも通常手法と比較して増加していることが確認できる．図 10 に測定中に起きた GC の総 STW 時間を示す．既存手法といずれの提案手法も通常手法より STW 時間の増加が確認できる．前述のように，ヒープサイズと STW 時間はトレードオフの関係にあり，ヒープサイズの縮小を実現すると STW 時間の増加が生じるが，この増加をより小さく抑えることが重要であると言える．

既存手法と両提案手法を比較すると，両提案手法の方がより小さい STW 時間の増加で同等のヒープサイズの縮小を実現していることがわかる．特に「9~16 or 65~256」のより複雑な設定において小さな STW 時間増加を実現しており，モニタリングに基づく手法が有効であることがわかる．

7. 考察

本稿では GSS にオブジェクトのモニタリング機能を実装し，事前調査としてオブジェクト統計情報を取得しておくことで，取得したオブジェクト統計情報に基づき Promote 抑制手法のサイズパラメータ（範囲）m を決めた．既存研究では，一般的なアプリケーションのモニタリング結果に基づいていたため，アプリケーションの特徴が一般的アプリケーションと異なる場合などに効果的に機能しないことなどが予想されたが，本手法では個別のアプリケーションの特徴をモニタリングにより発見し対応することにより，より多くのアプリケー四にて効果を示せると期待することができる．

また，本稿で紹介した既存手法および提案手法いずれも Promote を抑制する手法である．したがって本稿のモニタリング結果から Promote 抑制に適した特性を利用した．本今回はアプローチの違いから利用はしなかったが，「16384 バイト以上のオブジェクトは寿命 0 率が非常に低く，長寿なオブジェクトの割合が高い」という特性も重要であり利用可能であると考えられる．この特性は積極的に Promote させる手法に適していると考えられる．具体的には，16384 バイト以上のオブジェクトは長寿と予想し，GC1 回経験で Promote させてしまうなどが考えられる．この積極的な Promote させる手法により，さらなる性能向上を実現でき

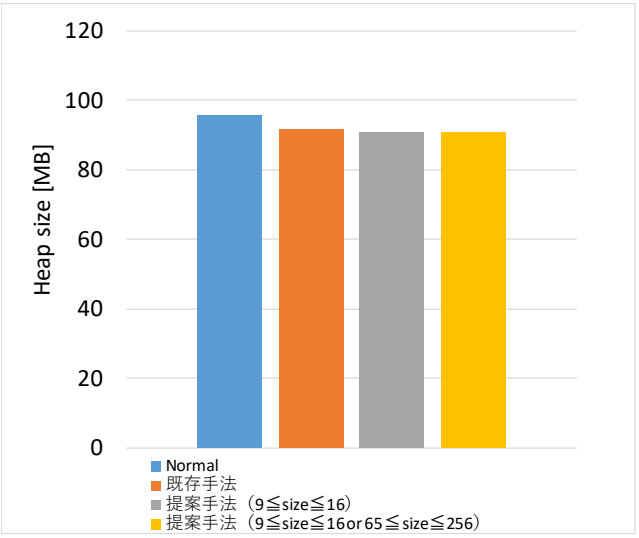

図 8 平均ヒープサイズ

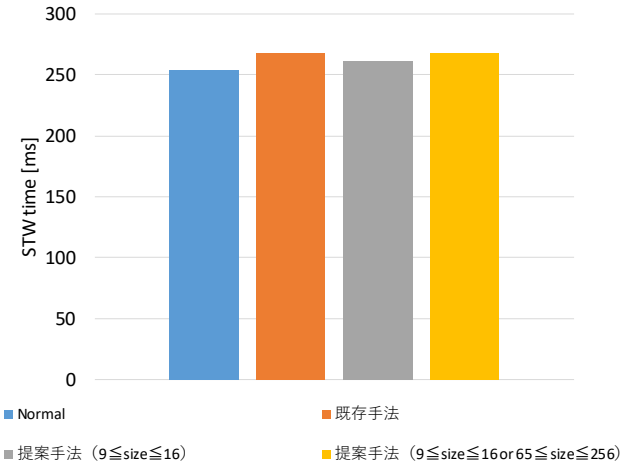

図 9 平均 STW 時間

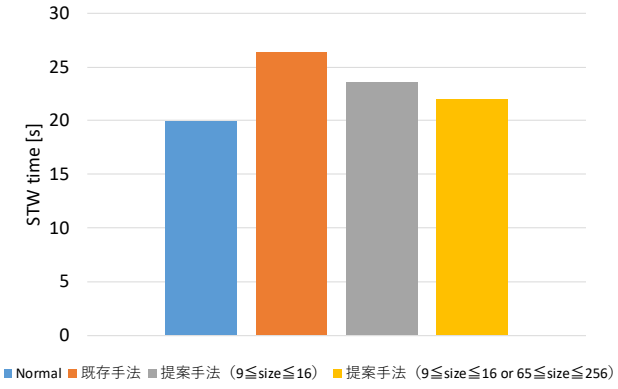

図 10 総 STW 時間

Vol.2018-GN-103 No.9
Vol.2018-CDS-21 No.9
Vol.2018-DCC-18 No.9
2018/1/26

る可能性がある.

8. おわりに

本稿では, モニタリング機能を実装した GSS にてオブジェクトの統計情報を事前に取得し，その情報に基づき既存手法を改善する手法を提案した．そして性能評価を行い，寿命の推定精度の向上により STW 時間の増加をより減少させ，アプリケーションのヒープサイズを縮小させることができることが確認され，提案手法の有効性を示すことができた.

今後は，別のアプリケーションや別端末での性能調査や, 積極的な Promote についての考察を行っていく予定である.

謝辞

本研究は JSPS 科研費 26730040, 15H02696, 17K00109 の助成を受けたものである.

本研究は，JST，CREST JPMJCR1503 の支援を受けたものである.

参考文献

[1] Smartphone OS Market Share, 2017 Q1 https://www.idc.com/promo/smartphone-market-share/os
[2] A. W. Appel.: Simple generational garbage collection and fast allocation, Softw. Pract. Exper. 19, 2, pp. 171-183, 1989. DOI=http://dx.doi.org/10.1002/spe.4380190206
[3] Shintaro Hamanaka，Shun Kurihara，Shoki Fukuda， Masato Oguchi，Saneyasu Yamaguchi，” A Study on Object Lifetime in GC of Android Applications” , CANDAR 2016 ,2016-11
[4] Shintaro Hamanaka，Shun Kurihara，Shoki Fukuda，Ryusuke Mori， Masato Oguchi，Saneyasu Yamaguchi，” Object Lifetime Trend of Modern Android Applications for GC Performance Improvement” , ACM IMCOM 2017 , 2017-01
[5] Mori, R., Oguchi, M., and Yamaguchi, S.: Memory Consumption Saving by Optimization of Promotion Condition of Generational GC in Android, Proc. of *2017 IEEE 6th Global Conference on Consumer Electronics (GCCE)* (2017).
[6] 森 竜佑 小口 正人 山口 実靖,” スマートフォンアプリケーションの特性を考慮した世代別 GC の Promote 条件に関する一考察”， 第 19 回 CDS 研究会，2017-5
[7] Paul R. Wilson, Uniprocessor Garbage Collection Techniques, IWMM '92 Proceedings of the International Workshop on Memory Management, Pages 1-42
[8] George E. Collins, A method for overlapping and erasure of lists, Communications of the ACM, Volume 3 Issue 12, Dec. 1960, Pages 655-657