6J-2

Dalvik VM コンカレントGCのSTW時間の短縮に関する考察

永田　恭輔†　　中村　優太†　　野村　駿†　　山口　実靖†

†工学院大学大学院工学研究科電気・電子工学専攻

1. はじめに

スマートフォンやタブレットPCが普及し，これらの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，特に重要なプラットフォームとなっている．Androidには Dalvik VMという仮想機械が搭載されており，Androidのアプリケーションは Dalvik VMの上で動作する．Dalvik VMには GC機能が搭載されているため，アプリケーション開発者はメモリ開放を自ら行う必要がない．しかし，GCが動作するとアプリケーション（ミューテータ）の停止あるいは性能低下が生じるため，リアルタイム性の高いアプリケーションにおいては GCの起動が大きな問題となり，GCの起動を回避する様な開発が要求されることがある．本稿では，GCのストップザワールド（STW）時間に着目し，この時間を短縮する手法について考察する．具体的には，Dalvik VM GCによるミューテータの停止時間を調査する．そして，GCの影響の低減手法について考察する．

2. Dalvik VMのGC

Dalvik VMは GCアルゴリズムとしてマーク&スイープを採用している．マーク&スイープの利点として，実装が容易であること，参照カウントと異なり GCが動いてないときのオーバーヘッドが小さいこと，循環参照も解放できることが挙げられる．一方，欠点としては，全てのオブジェクトを辿らなければならないため時間がかかることが挙げられる．マーク&スイープはマークフェーズとスイープフェーズに分かれており，マークフェーズでは非ゴミ（使用中）オブジェクトにマークを付け，スイープフェーズではマークの付いていない（不使用）オブジェクトの回収を行う．

マークフェーズでは，最初に VMから直接参照されているオブジェクトをルートオブジェクトとしてマークする．次に，ルートオブジェクトから直接的あるいは間接的に参照されているオブジェクトを再帰的にリンクを辿ることによりマークする．これらのルートオブジェクトから参照可能なオブジェクトが非ゴミ（使用中）オブジェクトとなる．次に，スイープフェーズでは前述のマークフェーズにてマークされなかったオブジェクトをゴミ（不使用）オブジェクトとして回収する．また Dalvik VMの GCでは，CoWにてプロセス間で共有されているメモリにマーキングによる書き込みが生じないように，ビットマップマーキング方式が採用されている[1].

A Study on Decreasing STW Time of Dalvik VM Concurrent GC

†Kyosuke Nagata, Yuta Nakamura, Shun Nomura, Saneyasu Yamaguchi

†Electrical Engineering and Electronics, Kogakuin University Graduate School

Android 2.2以前の Dalvik VMの実装では，GC処理全体が STW処理であるノンコンカレント GCが実装されていた(図 1(a)). このため，GCによりアプリケーションが 100ms以上停止することがしばしば発生すると指摘されており，この時間は許容範囲内ではないとの主張もなされている[2]. 一方，Android 2.3以降の Dalvik VMの実装では，コンカレント GCが採用されており，図 1(b)の様に GC処理の一部とミューテータを並列して動作させることが可能である．コンカレント GCの処理は以下の通りである．まず，イニシャルマーク処理としてルートオブジェクトに印をつける．この間は，ミューテータは停止（STW）する．次に，コンカレントマーク処理としてルートオブジェクトから辿れるオブジェクトにマークしていく．この間は，ミューテータと GCが並列に動作し，ミューテータは停止しない．次に，リマーク処理としてコンカレントマーク処理中に生じた書き込みの整合性をとるために，ミューテータを停止（STW）させて変更が生じたオブジェクトを基にマークを行う．最後に，コンカレントスイープ処理としてマークの付いていないオブジェクトをゴミオブジェクトとみなし，フリーリストに繋ぎ，再利用可能な領域とする．コンカレントスイープ処理はミューテータと並列に処理が行われる．非コンカレント GCでは全てのマーク処理中ミューテータが停止（STW）するのに対し，コンカレント GCでは 2度の停止（STW）の間ミューテータが動作するため，停止（STW）時間が大幅に削減される．

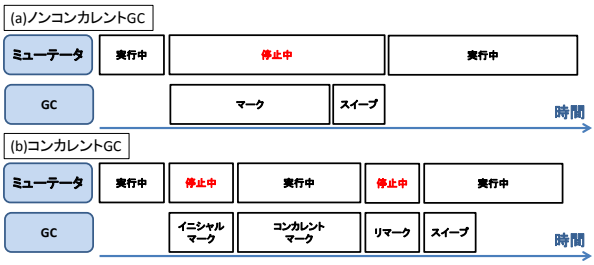

図 1 GCの処理

3. 性能評価

本章で Dalvik VM GCの性能の評価を行う．

3.1. 測定環境

Nexus7上で自作のベンチマークを実行し，GCの各処理の時間を測定した．測定に用いた端末の仕様は OS：Android 4.1.2，CPU：NVIDIA Tegra3 T30L 1.3GHz クアッドコア，メモリ：1GB，ストレージ：16GBである．

3.2. ベンチマーク

本節では，ベンチマークについて説明する．ベンチマ

ークは初めに，指定数のオブジェクト（これは非ゴミオブジェクトとなる）を生成する．次に，ゴミオブジェクトを生成し続けることで，GC を起動させる．本測定では，オブジェクト同士のリンクの書き換え頻度を変化させ，GC の処理にどのような影響を与えるかの評価を行う．測定では GC のそれぞれのフェーズに要する時間を取得し，以下では STW 時間に着目して考察する．

3.3. リンク書き換え頻度

本節でリンク書き換え頻度について説明する．各非ゴミオブジェクトは，一様分布乱数でランダムに選択した非ゴミオブジェクトに対するリンクを 1 個保持しており，ベンチマーク実行中はゴミオブジェクト 1 個生成するたびに，ランダム選択された n 個のオブジェクトのリンクを別のものに変更する．この n が書き換え頻度であり，書き換え頻度 n の時は，ゴミオブジェクト 1 個生成するたびに n 個のオブジェクトのリンクが変更されることになる．コンカレントマーク中にリンク書き換えが行われたオブジェクトは，リマークにおける再調査の対象となる．

3.4. リンク書き換え頻度と GC 時間の関係

本節では，リンク書き換え頻度を 1, 2, 4, 8, 16 と変化させたときのそれぞれのフェーズの時間について述べる．全ての測定において，ベンチマークの非ゴミオブジェクト数は 10^6 個である．STW 処理の時間を図 2 に示す．図より，書き換え頻度を増加させるにしたがって，リマーク時間が増加することが確認できる．これは，コンカレントマーク中に変更されたオブジェクトが増えるに従い，リマーク処理で調査しなおすオブジェクト数が増えるためだと考えられる．最長の例においては 50ms を越えており，一般的フレームレート（25fps や 30fps）におけるフレーム間の時間より大きく，人間もミューテータの停止を認知できる時間であると予想される．

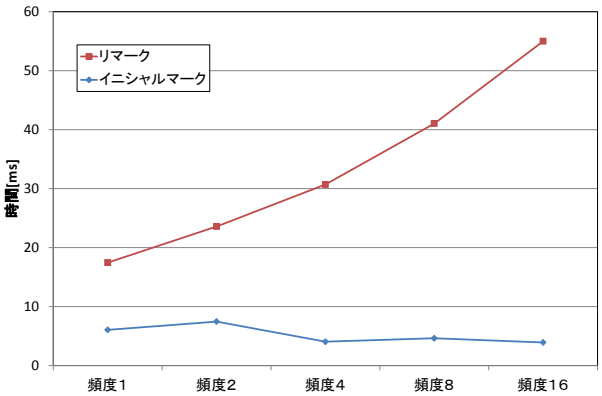

図 2 書き換え頻度の変化時の STW 時間

3.5. Dirty カード数

本節では，Dirty カード数と書き換え頻度の関係を調査した結果を示す．Dalvik VM のコンカレント GC はカードテーブルを用いており，コンカレントマーク中の変更をカードテーブルのカードを Dirty にすることで記録する．リマーク処理では，このカードテーブルの Dirty カード部分のみ調査することで，マーク時間を削減している．

書き換え頻度の変化による Dirty カード数の変化を図 3 に示す．この図から，書き換え頻度増加に伴い Dirty カード数が増加することが確認でき，Dirty カード数が 3.4 節のリマーク時間の増加の原因であることが確認できた．

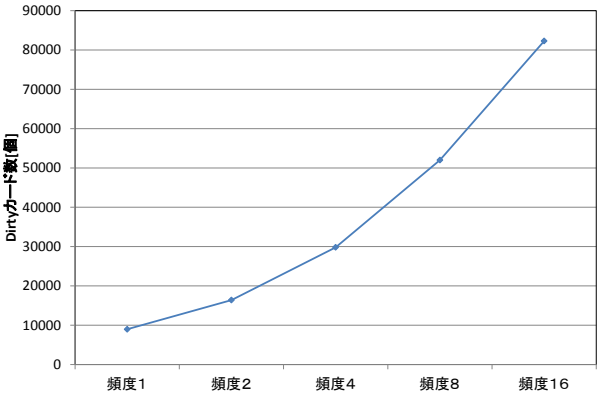

図 3 書き換え頻度と Dirty カード数

4. STW 時間短縮の考察

前章では GC の基本測定を行った．測定より，オブジェクト数が多くかつ，オブジェクトに対する変更が多い場合を除き，STW 時間は 10ms 未満であり，オブジェクト数が多くかつ変更頻度が高い場合にのみ 50ms を越えるという結果が得られた．以上より，書き換え頻度が高い環境におけるリマーク処理による STW 時間の短縮により STW によるアプリケーション停止を大きく抑制できると考えられる．

リマーク処理による STW 時間の短縮についてはリマーク処理もミューテータとコンカレントに行い，リマーク中に生じた変更の整合性を取る STW 処理であるリリマーク処理を追加することにより実現できると考えることができる．同様にリリマークによる STW 時間が問題となる可能性が考えられるが，この問題はリリマーク処理のコンカレント化により改善ができると期待できる．一方，上記のようにリマークのコンカレント化などを追加で実装していくと，不要な処理の追加を招く可能性が考えられるが，図 2，図 3 の結果から，Dirty カード数と整合処理の時間に強い相関があることが分かり，Dirty カード数の大小によりさらなるコンカレント処理の追加と STW 処理を動的に選択することによりこの問題を回避できると考えられる．

5. おわりに

本稿では，Dalvik VM GC によるミューテータの停止時間の評価を行い，GC がミューテータ性能に大きな影響を与える条件の調査を行った．調査結果をもとに GC の影響の低減手法について考察を行った．

今後は，Dalvik VM に低減手法を実装し GC 時間の調査を行う予定である．

謝辞

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

参考文献

[1] TOMOHARU UGAWA, HIDEYA IWASAKI, TAIICHI YUASA, “Improvements of Recovery from Marking Stack Overflow in Mark Sweep Garbage Collection”, IPSJ, Vol.5, No.1 (2012).
[2] Patric Dubroy, “Memory Management for Android Apps”, Google I/O 2011