Vol.2016-SE-194 No.3
2016/11/17

CMS GC におけるヒープメモリ断片化の要因

髙雄慎二[†1]　高木浩則[†1]　和氣弘明[†1]　湯口徹[†1]

概要：Java の実行環境(JVM)において，コンカレント・マーク・スイープ(CMS)型のガベージコレクション(GC)を使用すると発生するヒープメモリの断片化は，過度に進行すると性能に大きく影響する場合がある．しかし，断片化の進行の度合いを直接に把握することは難しく，また，断片化の進行を緩やかにするためのチューニングの指針も知られていなかった．本稿では，ソフトウェアの試験段階などにおいて，ヒープ断片化の進行の度合いを推定するための統計的なモデルを構築し，これを運用段階などにおいて活用し，標準的なログからヒープ断片化を推定できるようにすることを提案する．

キーワード：ソフトウェア性能工学，ガベージコレクション，メモリ断片化，機械学習, Java

An Analysis on Factors of Java Heap Memory Fragmentation

SHINJI TAKAO[†1]　HIRONORI TAKAKI[†1]
HIROAKI WAKI[†1]　TORU YUGUCHI[†1]

Abstract:

Keywords: Software Performance Engineering, Garbage Collection, Memory Fragmentation, Machine Learning, Java

1. はじめに

Java*言語プログラムの実行基盤 (Java Virtual Machine : JVM) における コンカレント・マーク・スイープ・ガベージコレクション(CMS GC) は，GC に伴うアプリケーション停止時間を僅かに留めることができるため，広く使用されている．近年，より簡易なチューニングで良いとされるガベージ・ファースト GC (G1 GC) なども提供されているが, CMS GC は, 適切に使用すれば性能が良好であること，使用実績が多くソフトウェア品質的にも安定していることなどから，依然として広く使用されている．

CMS GC は，Java ヒープメモリ（ヒープ）を断片化させてしまう特徴がある．ヒープ断片化が過度に進行すると，アプリケーションが一時的に完全停止するなどの影響がある．しかし，基本的には影響は一時的であるため，従来は JVM パラメータ設計などに際して，ヒープ断片化は特に考慮されてこなかった．

しかし，負荷が大きいなどのある条件下では，断片化に起因する幾つかの事象は，性能を大きく低下させる場合がある．システム運用中に，ヒープ断片化の度合いを直接確認して対処できれば良いが，システムのリソース消費が大きい詳細なログ出力が必要なため, 現実的には困難である．また，ヒープ断片化がどのような要因と条件で過度になり緩和されるかが現状明らかでないため，JVM の動作パラメータ設計時に検討/対処することも困難である．

本稿では，JVM 内部の知見を基に，ヒープ断片化に影響を与える主要要因を特定し，各要因がヒープ断片化に寄与する度合を統計的にモデル化することを検討した．また，このモデルを活用し，運用時のリソース消費の少ない通常ログの情報から主要要因の値を算出して断片化の進行度合いを判断する手順を検討した．この手順によって，システム開発の試験工程におけるデータを基に統計的なモデルを構築し，それを活用することで，運用時にヒープ断片化の進行度合いを推定できるようになる．また，その推定を，ヒープ断片化の緩和を目指すチューニングを行う際の指標としても活用できる．

なお，近年のソフトウェア性能工学研究における GC の研究では，GC の総合的な性能に着目しており，ヒープ断片化などの詳細な挙動までには及んでいない．本稿では，ヒープ断片化の因果関係の知見に基づくモデル化を実施したものである．

2. 先行研究

2.1 ソフトウェア性能工学

ソフトウェアの性能低下は，しばしば開発・運用時に重大な影響を与える場合がある．その実態についての文献は少ないが，ある情報技術系企業に対する調査では，半数以上の企業が，関与したアプリケーションのうちの 2 割から 8 割に，許容できない予期せぬ性能問題があったとしている[1]．このように影響の大きいソフトウェア性能の管理は，従来は技術者の技能に依存する部分が大きかったが, 近年,

[†1] 日本電信電話株式会社

*Java は，Oracle Corporation およびその子会社，関連会社の米国およびその他の国における登録商標である．

これを工学的に扱うことを目指したソフトウェア・パフォーマンス・エンジニアリング（性能工学）の研究が行われつつある．

ソフトウェア性能工学とは, ソフトウェア工程を通して，ソフトウェア性能要件への適合を目指して行われる一連のソフトウェア工学活動，およびそれに関連する分析の総体と定義される[2]．また，ここで言う性能とは，処理件数等の容量と適時性に関するものと解される．特に適時性に関する応答時間(response time)は，ユーザビリティやビジネスへ与える影響が大きい[3][4].

このような最近の性能工学研究において，GC はわずかな設定の変更が性能を大きく左右する場合があるため，GC のチューニングと性能の関係を予測するためのモデル化を試みた研究が行われている[5][6][7]．しかし，JVM などのプログラム実行基盤は，様々な処理を実行しているため, 総合的な性能を予測することは困難が大きい. 例えば，GC 以外にも，実行頻度統計に基づく動的かつ反復的な最適化コンパイル，部分的な自動チューニング，アプリケーションの並列実行管理，メモリ割当時のライトバリアの管理，等を行っており，これらが総合的な性能に影響している．したがって，ソフトウェア性能を事前に予測することは困難となっており，多くの実務家の間で，小規模な性能試験によって性能を実測することが好まれている[7]．なお, この性能の実測とチューニングによるアプローチを，近年発達している機械学習技術は性能工学でも活用されつつあり[8]，機械学習により GC チューニングを自動化した試みでは，大幅な性能改善が行なえたという報告もある[9].

これらの既存研究は，プログラム実行基盤の総合的な性能について，平常状態の平均的性能の向上に注目したものである．しかし，本稿が問題とする非正常な状況で発生する性能低下事象や，それに関係するヒープ断片化の要因に着目したものではない.

3. 課題

3.1 CMS GC のヒープ断片化事象

CMS GC は，GC スレッドとアプリケーションスレッドを可能な限り並行(concurrent)に実行することで，GC におけるアプリケーション停止を最小限に抑えている. しかし，並行処理においては，使用中オブジェクトの再配置(relocation)を行うことは難しい. そのため, CMS GC では，基本的にオブジェクト再配置を伴うヒープ断片化の解消（デフラグメント）を行うことができない. その代り, CMS GC では，空き領域を管理するデータ(Free List)によって，断片化した空き領域をサイズ毎に管理し，新しいオブジェクトに対して Free List から適当なサイズの空きメモリのブロック同士を探し出し，これを割り当てる[10]．なお，要求されるオブジェクトサイズの統計に従い，Free List の中のメモリのブロックを，より要求の多いサイズに分割したり，隣接するブロックを結合する，等の管理作業が，CMS GC の concurrent sweep と呼ばれる並行フェーズの一つにおいて行われる.

さらに，CMS GC は他の JVM の GC と同様に，世代別 GC (generational GC)である．世代別 GC は，生存期間の短い（若い）オブジェクトと，長い（古い）オブジェクトを別領域で管理する．これは，大部分のオブジェクトの生存期間は短く，したがって比較的狭い領域で頻繁に GC を行えば，大分部を回収できるとする仮説に従ったデザインである．この領域(Young 世代領域)は，比較的狭くて済むため, アプリケーション停止の影響も小さい. そのため Young 世代領域に対しては，アプリケーション停止を伴う，フラグメント化の発生しない GC を実行する．このおかげで，長命なオブジェクトが移される領域(Old 世代領域)の GC の発生をわずかな回数に抑えることができる[10][11]. この世代別 GC と組み合わせることで，CMS GC によるヒープ断片化は許容できる範囲に抑えられ，かつ，アプリケーション停止が少ないというメリットを享受できている.

一方，CMS GC が高負荷となる場合，すなわち，Old 世代領域のオブジェクトがさほど長命でなく，大部分が回収され，かつその回数が多い場合，ヒープ断片化は許容できない程度まで進行する可能性がある．その場合，まず Free List 管理を行う concurrent sweep フェーズの時間が増加し，最終的には Free List によるメモリ割り当てに失敗し，アプリケーションの停止を伴う GC がヒープ全体に対して実行される(Full GC).

3.2 大幅な性能低下の可能性がある状況

Full GC が実行されれば，基本的にはヒープ断片化は解消される．その後，世代別 GC が機能すれば，次に Full GC が発生するまで相当程度の期間を要する．そのため，CMS GC によるヒープ断片化の影響は一時的なものに限定され，特に注意を要しないと考えられてきた.

しかし, 世代別 GC が想定通りに実行されている状況は，必ずしも常に実現しているわけではない．実際には，リソースの制約や設定の間違い，運用後の状況の変化等の様々な理由のため，世代別 GC が理想的に実行されていない状況で運用せざるを得ないシステムは多いと考えられる．また，システムへの処理要求が増し，負荷が過大となっている状況では，一回の Full GC 発生による処理性能の低下の影響が大きいばかりでなく，CMS GC の頻発，Full GC の頻発，concurrent sweep 時間増大による負荷増大などの事象が複合的に発生し, 性能が大きく低下する可能性がある.

したがって，最小限必要なヒープサイズを確保するだけでは不十分で，CMS GC が過負荷となり難いよう，余裕を持ったキャパシティプランニングとチューニングが必要と考えられる．しかし従来は，このようなことを考慮する必要性はあまり重視されてこなかった.

3.3 チューニングの困難さ

なお，ヒープ断片化の過度な進行を事前に察知する方法として，これを直接に測定し，増加傾向等を監視することが考えられる．しかし，JVM でヒープ断片化を監視することは，現状ではログ出力量の負荷が高く，通常運用時のログとしては取得することが難しい．

また，ヒープ断片化の要因の寄与を知ることができないので，これを避けるようにチューニングを行うための情報が，現状では不十分である．

4. アプローチ

本節では，本稿でとったアプローチを述べる．

4.1 課題解決のステップ

前節で述べたように，ヒープ断片化の度合いを直接確認することが困難であり，また，ヒープ断片化がどのような要因と条件で過度になり緩和されるかが明らかでないという課題に対して，本稿は以下のようなステップで解決することを提案する．

(1) ソフトウェア試験段階での推定モデルの構築
ソフトウェア開発の試験段階において，断片化進行度を直接に把握できるログを取得し，そのログに基づき，当該システムにおいてヒープ断片化の進行度を推定できる統計モデル（以後，**推定モデル**と呼ぶ）を構築する．試験段階であれば，断片化を直接に把握するログが取得できる場合が多いと考えられるためである．

(2) ソフトウェア運用段階での推定モデルの利用
ソフトウェアが運用段階へ移行した後は，通常のログから得られるデータを上記モデルに入力し，ヒープ断片化の進行度を推定する．ヒープ断片化が過度に進行したと推定された場合，ヒープ断片化推定の元となった指標を改善させることを目標に，JVM のチューニングを行う．

4.2 推定モデル構築の詳細

前項の解決策のステップ 1 に必要な，推定モデルを構築する手順は以下の通りである．なお，使用するデータは，ヒープ断片化の実測値を含むものを，試験段階などで取得し，学習用と評価用にランダムに分割しておく．

1 因果関係構造の構築
- 1.1 JVM の知見に基づく因果関係の明確化
  ヒープ断片化を進行させる因果関係を JVM の知見に基づき明確化する．
- 1.2 パス解析の反復
  前記の因果関係が単に机上のものでなく，実際のデータと適合していることを裏付けるため，パス解析を行う．その結果，適合度が低いモデルは見直し，再度パス解析を行う過程を繰り返す．
- 1.3 推定モデル向け因果関係の構築
  パス解析で確定した因果関係を基に，ヒープ断片化進行度を推定するための重回帰分析の式を作成する．この段階では，説明変数の重み付けにあたる偏回帰係数は計算しない．

2 推定モデルを構成する要因の重み付け
推定モデルに試験段階で得たデータを適用し，モデルを構成する要素の重み付け（偏回帰係数）を計算する．

3 推定モデルの精度の確認
推定モデルに評価用データを適用し，推定精度を確認する．

4.3 推定モデルの構成要素

推定モデルを構成する各要素について検討した候補を，表 1 に示す．これらのうち，反復的なパス解析による因果関係の検討の過程で，結果的に使用しなくなったものもある．

表 1 推定モデルの構成要素
Table 1 Elements of estimation model.

名称	概要
昇格閾値(*TT*)	オブジェクトを Young 世代領域から Old 世代領域へ昇格されるまでに，Young 世代領域で行われる GC 回数
Old 世代領域のメモリ使用率の変動(*V*)	Old 世代領域のメモリ使用率の標準偏差
CMS GC 発生間隔(*I*)	CMS GC の平均発生間隔(秒)
ヒープ断片化進行度(*Frag*)	Frag 値の最大値
concurrent sweep フェーズ時間(*S*)	concurrent sweep フェーズの最大持続時間(秒)
エラー発生回数(*ERROR*)	CMS GC 失敗，Full GC 発生などの事象発生回数

これらは，Frag 値を除き，標準的なログから取得できる．Frag 値とは，GC ログの-XX:PrintFLSStatistics=2 オプションで取得できる，ヒープ断片化の進行度を示す指標である．これにより，ヒープ断片化の進行度を直接に知ることが可能であるが，このオプションで出力されるログは膨大で，必要なディスク容量等のコストが高いため，通常の運用時に取得することは難しい．

4.4 JVM の知見に基づく因果関係の定義

本節では，推定モデル構築手順の 1.1 JVM の知見に基づく因果関係の明確化について述べる．

まず，JVM の動作に関する知見をもとに，ヒープ断片化を進行させる原因，また，ヒープ断片化が及ぼす影響について，幾つかの仮説を設定した．図 1 は，これらの仮説を図示したものである．

なお，CMS GC が世代別 GC として理想的に動作していれば，ヒープ断片化の影響は小さく，世代別 GC が理想的に動作していない場合は，影響が大きくなることを 3.1 節で述べた．このことをより詳細に検討し，以下の 6 つの仮説を設定した．

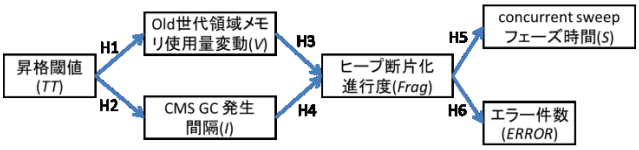

図 1　ヒープ断片化進行因果関係仮説

Figure 1　Hypotheses of heap fragmentation causality

Young 世代領域での GC が十分におこなれない場合，昇格閾値(TT)の値が減少する．本来であれば Young 世代領域で回収されることが望ましい短命なオブジェクトが，短い回数の Young 世代 GC を経ただけで，Old 世代領域に移動する. Old 世代領域にある短命オブジェクトは, 早期に CMS GC によって回収される．このことは，Old 世代領域のメモリ使用率の変動(V)を増大させる原因となる(図 1 仮説 H1). また同時に，CMS GC を頻発させ，CMS GC 発生間隔(I)を短くする原因ともなる(図 1 仮説 H2).

また，3.1 節で述べた機序により，CMS GC で回収される Old 世代領域が多いほど，また，CMS GC の発生頻度が多いほど，ヒープ断片化は進行すると考えられる(図 1 仮説 H3,H4).

ヒープ断片化が進行すると，Free List に含まれる空きメモリ領域の結合や分割などの管理作業の負荷がまし，concurrent sweep フェーズの時間が増加する(図 1 仮説 H5). また，ヒープ断片化の進行が極端になると，メモリ割り当てに失敗し，CMS GC 実行失敗のエラーや，長時間のアプケーション停止を伴う Full GC などの発生回数が増加する(図 1 仮説 H6).

5. 推定モデルの構築事例

5.1 データの取得方法

本稿では，データ生成のために，SPECjvm2008[12]を活用した．SPECjvm2008 とは，JVM の性能を測定することを目的として作られたベンチマークセット(benchmark suite)であり, JVM の核となる機能に焦点をあてた幾つかのベンチマークや実際のアプリケーションからなる．表 2 に, SPECjvm2008 を構成する各ベンチマーク／アプリケーションの概要と，本稿の目的で使用した際の採否を示す．これらのうち startup は, JVM プロセスの起動速度に焦点を当てており，本稿が着目する状況とは異なるため，採用しなかった.

SPECjvm2008 の正統的結果 (compliant result) は，これらすべてを実行して結果を幾何平均することで求める．しかし，本稿では正統的結果は不要であり，学習・評価用データのためにログを収集する目的で，各々のベンチマーク結果を独立したアプリケーションと見做して使用した.

SPECjvm2008 を構成する各ベンチマークを，ウォームアップ 5 分，本番実行 30 分の合計 35 分間実行した．また，SPECjvm2008 は，実行時のスレッド数によって負荷が変化することから，実行時に使用する CPU コア数を，最小値 1 から最大値 8 まで順次変化させて実行し, 各種の負荷状況でのデータを取得した．JVM 起動オプションとしては，著者らの所属組織の業務にて広く使用されているチューニングオプションを使用した．これは，CMS GC について広く使用されている一般的なオプションを組み合わせたものである.

表 2　SPECjvm2008 を構成するベンチマーク

Table 2　Benchmarks Composing SPECjvm2008.

名称	概要	採否
compiler.compiler	javac による javac 自身のコンパイル	○
compiler.sunflow	javac による sunflow のコンパイル	○
compress	LZW によるデータ圧縮	○
crypto.aes	AES,DES による暗号・復号化	○
crypto.rsa	RSA による暗号・復号化	○
crypto.signverify	署名確認 (MD5,SHA1,SHA256/DSA,RSA)	○
derby	Open Source のデータベース	○
mpegaudio	Mp3 オーディオのデコード	○
scimark.fft.large	浮動小数点演算，32Mbyte データ	○
scimark.lu.large	〃	○
scimark.sor.large	〃	○
scimark.sparse.large	〃	○
scimark.fft.small	浮動小数点演算，512Kbyte データ	○
scimark.lu.small	〃	○
scimark.sor.small	〃	○
scimark.sparse.small	〃	○
scimark.monte_carlo	浮動小数点演算，モンテカルロ法	○
serial	オブジェクトを出力ストリームへ書き出す直列化と復元	○
sunflow	画像レンダリング	○
xml.transform	XML 文書処理, style sheet の適用	○
xml.validation	XML 文書処理，妥当性確認	○
startup	各ベンチマークを新しい JVM プロセスを起動して一回実行	×

ベンチマーク実行にあたっては，JVM の情報として，以下のログを出力させた.

- GC ログ：GC の実行状況を逐次記録するログ．以下の関連オプションを設定した
  - -XX:+PrintGCDetails (GC の詳細情報)
  - -XX:PrintFLSStatistics=2 (フラグメント化についての統計情報)
- Jstat ログ：GC の実行状況を統計的に要約するログ．以下の関連オプションを設定した.
  - -gc (OLD 世代領域の容量と使用量など)

SPECjvm2008 を実行した環境を以下に示す.

- ハードウェア
  - CPU: Intel(R) Core(TM) i7-4770K CPU @ 3.50GHz

キャッシュサイズ 8192KB (x1)
- メモリ: 16GB (DDR3 DIMM 1600MHz 8MB) (x2)
- Java ヒープサイズ：1200MB

- OS/ソフトウェア
  - OS:CentOS Linux 7.2 (64bi)
  - カーネル: 3.10.0-327.18.2.el7
  - glibc: glibc-2.17-106.el7_2.6
  - JVM: java-1.8.0-openjdk-1.8.0.91-1.b14.el7_2

5.2 パス解析の反復と推定モデルの確定

4.2 節の推定モデルの定義で行う，パス解析の反復を，以下のように行った．

まず，断片化の因果関係の構造の初期モデルを構築(4.4 節)した．その後，取得したデータによりパス解析を行い，モデルとデータの適合を確認した．この段階で適合しないモデルは見直し，再度パス解析を行う過程を繰り返した．ただしモデルの見直しは，因果関係の知見と矛盾してはならず，それをよりよく説明することを目指すことに注意した．

見直しによって新たに加えられた因果関係は，Old 世代領域のメモリ使用率の変動(*V*)から，CMS GC 発生間隔(*I*)へ向かうパスである．Old 世代領域の変動が激しいことは，Old 領域に短命なオブジェクトが多いことを意味する．短命オブジェクトの量は多いため，これが Old 領域に移動している状況では，Old 領域は早期に使用し尽され，CMS GC の発生間隔も短くなると考えられる．

一方で，短命オブジェクトの Old への過早な昇格の存在を示すと思われた昇格閾値(*TT*)は，パスは統計的に有意であったが，モデル全体の適合度を低くしていた．そのため，予測モデルのための要素からは削除した．

また，エラー発生回数(*ERROR*)は，ヒープ断片化から影響を受けると考えられたが，パスは有意ではなかった．エラー事象の発生には，タイミングや経過時間なども関係しているため，今回のデータ取得方法では十分なデータ数を得られなかった可能性が考えられる．

最終的に得られた因果関係図が図 2 である．

図 2　ヒープ断片化進行因果関係図
Figure 2　Causality diagram of heap fragmentation.

図 2 で示された因果関係図を確認した，最終的なパス解析結果を表 3 に示す．なお，パス解析の実行にあたっては，オープンソースの統計処理言語である R [15]の sem パッケージ[16]を使用した．

表 3　パス解析結果
Table 3　Results of Spath analysis.

パス	係数	P 値
V→I	-0.85469985	1.140271e-75
V→Frag	0.33763076	2.166414e-16
I→Frag	-0.66659418	4.004903e-59
Frag→S	0.71035596	1.577754e-29

Model Chisquare = 0.5598298　Df = 2 Pr(>Chisq) = 0.7558481(脚注 a 参照)
Goodness-of-fit index =　0.9977707
Adjusted goodness-of-fit index =　0.9888533
RMSEA index =　0　90% CI: (NA, 0.1206623)
SRMR =　0.005033157
AIC =　16.55983

最終的に得られた因果関係図を元に，推定モデルを以下のように確定した．4.2 節の 1.3 で述べたように，この段階では，各要素の重み付け(偏回帰係数)はまだ計算しない．

$$Frag = \beta_0 + \beta_1 \cdot V + \beta_2 \cdot I \quad (2)$$

Frag: ヒープ断片化進行度
V: Old 世代領域のメモリ使用率の変動
I: CMS GC 発生間隔
$\beta 0$:切片
$\beta 1, \beta 2$：偏回帰係数

5.3 重回帰分析

ヒープ断片化を推定する重回帰モデルの，要因の重み付けにあたる偏回帰係数 $\beta 0, \beta 1, \beta 2$ を推計することと，統計的な有意性を確認するため，重回帰分析を行った．

5 で述べたように SPECjvm2008 構成ベンチマークを活用して生成したデータを，ランダムに学習用データと評価用データに分割した．次に，上記学習用データをモデルに適用し，重回帰分析を行った．結果を以下の式(3)に示す．

$$Frag = 0.5108 + 2.244 \cdot V - 0.0003099 \cdot I \quad (3)$$

P: 1.84e-14***, 3.57e-13***, 2e-16***
Adjusted R-squared: 0.9421

モデル全体の説明力を示す修正済み決定係数(Adjusted R-squared)は 0.9421 と，高い値となった．さらに，各説明変数の寄与率にあたる偏回帰係数も，P 値が 1%を下まわっており，統計的に有意な結果であることが確認できた．

さらに，上述の評価用データを，偏回帰係数計算済みの上記モデルに適用した．目的は，説明変数にあたる V, I のデータを入力して得られた Frag 推定値を，実測した frag 値と比較することである．推計値と実測値の差分を図 3 に示す．推計値と実測値の差分の平均値は 0.041，最大値は 0.266 であり，推計値と実測値の間に大きなかい離は無かった．

a ここでの Chisquare は”解析されたモデルは真のモデルに適合する”という帰無仮説について検定するので，p 値は大きいことが望ましい．

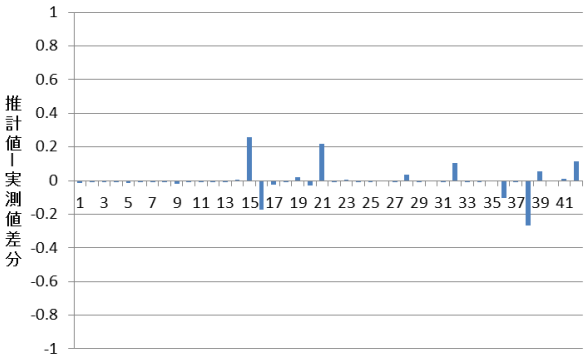

図 3　frag 推定値と実測値の差分

Figure 3　Difference between estimated and actual frag values.

5.4 推定モデルの利用

以上から，ヒープ断片化の進行度合いに対して，Old 世代領域のメモリ使用率の変動と CMS GC 発生間隔の 2 要因の影響が大きいことがわかった．この 2 要因に対するチューニングの方法はすでに明確であるので，これによって間接的にヒープ断片化の進行度合いをチューニングすることができる．本稿ではその詳細には触れないが，概要を以下に示す．

- Young 世代領域のチューニング
  - 最大昇格閾値の設定
  - Young 世代領域内の Survivor 領域と Eden 領域の割合の調整
  - Young 世代領域の拡張
- Old 世代領域の拡張
- Java ヒープ領域の拡張

ヒープ領域の拡張が困難などの理由で，上記での改善が困難な場合は，CMS GC 以外への変更も考えられる．

6. まとめと今後の課題

本稿では，ソフトウェアの試験段階などにおいて，ヒープ断片化の進行の度合いを推定するための統計的なモデルを構築し，これを運用段階などにおいて活用し，標準的なログからヒープ断片化を推定できるようにすることを提案した．また，そのために必要な推定モデルの構築手順を示した．実際に推定モデルを構築した結果，因果関係をよりよく記述でき，精度の高いモデルを構築できた．

今後，本稿のアプローチの適用事例を増やし，傾向を確認してゆく．また，ヒープ断片化の要注意レベルを判断するための基準について，さらなる検討を行ってゆく．

謝辞　本研究の機会を与えていただいた NTT オープンソースソフトウェアセンタの小西史和氏, 現 NTT ソフトウェア株式会社の岩田雅彦氏，本稿の内容にコメントを下さった同僚の久保田祐史氏，山崎秀夫氏に謹んで感謝の意を表する．

参考文献

[1] Compuware. Applied performance management survey (October 2006), http://www.cnetdirectintl.com/direct/compuware/Ovum APM/APM _Survey_Report.pdf (last retrieved 2016-10-06)
[2] Woodside, M., Franks, G., and Petriu, D. C., The Future of Software Performance Engineering (FOSE'07), 2007.
[3] Wall, M., How long will you wait for a shopping website to load? available at http://www.bbc.com/news/business-37100091 (last retrieved 2016-10-11)
[4] Nielsen, J., Website Response Times, June 21, 2010, available at https://www.nngroup.com/articles/website-response-times/ (last retrieved 2016-10-11)
[5] Libič, P., Bulej, L., Horky, V., and Tůma, P., On the limits of modeling generational garbage collector performance. Proc. 5th ACM/SPEC Int. Conf. Perform. Eng. (2014), 15–26.
[6] Blackburn, S. M., Cheng, P., and Mckinley, K. S., Myths and Realities : The Performance Impact of Garbage Collection. Sigmetrics (2004), 25–36.
[7] Horky, V., Libic, P., Steinhauser, A., and Tuma, P., DOs and DON ' Ts of Conducting Performance Measurements in Java. Proc. 6th ACM/SPEC Int. Conf. Perform. Eng. (2015), 337–340.
[8] Didona, D., and Romano, P., Hybrid machine learning/analytical models for performance prediction: A tutorial. ICPE 2015 - Proc. 6th ACM/SPEC Int. Conf. Perform. Eng. (2015), 341–344.
[9] Liu, J., Ramakrishna, R., and Wiltschko, A., Automated Tuning of the JVM with Bayesian Optimization, JavaOne (2016), CON8116.
[10]Pnntezis, T., and Detlefs, D., A Generational Mostly-concurrent Garbage Collector. 2000, 143–154.
[11] Lieberman, H., and Hewitt, C., A real-time garbage collector based on the lifetimes of objects. CACM 26 (6), 1983, 419-429.
[12] SPECjvm2008, https://www.spec.org/jvm2008/ (last retrieved 2016-10-07)
[13] TPC-W, http://www.tpc.org/tpcw/ (last retrieved 2016-10-07)
[14] Free space calculation of heap, http://mail.openjdk.java.net/pipermail/hotspot-gc-use/2012-June/001249.html (last retrieved 2016-10-11)
[15] The R Project for Statistical Computing, https://www.r-project.org/ (last retrieved 2016-10-07)
[16] sem: Structural Equation Models, https://cran.r-project.org/web/packages/sem/index.html (last retrieved 2016-10-07)