4J-06

大規模コードクローン検出のための包含文字列を対象とした省メモリ化プログラム変換の予備的な性能評価

山崎 徹郎* 佐藤 芳樹 † 千葉 滋 ‡

東京大学情報理工学系研究科*‡§ 東京大学情報基盤センター †§

1 はじめに

膨大な量のソースコードから類似したコード片を見つける大規模コードクローン検出はソフトウェアリポジトリマイニングにおける重要な研究テーマの一つである．このようなコードクローン検出の中でも Type 3 と呼ばれる種類のコードクローンの検出は，しばしば大量の計算資源を必要とする．コードクローンは，主にソースコードのコピー＆ペーストにより作られるが，ペーストされたコード片に多少の文が挿入されたり，一部の文が削除または修正される場合，元のコード片と完全には一致しなくなる．このようなコードクローンは Type 3 と分類されるが，それを検出するためにはソースコードを抽象構文木に変換して，制御フローやデータフローのグラフも作るなどし，抽象構文木の類似する部分木を探すことになる．これは計算時間やメモリを大量に消費する．

実際に Type 3 コードクローン検出器である Scorpio [1] を用いた我々の予備実験では，入力サイズである解析対象のプログラムの大きさに対して約 200 倍のメモリを消費する場合があった．我々は大規模コードクローン検出のために，スーパーコンピュータを用いて大規模並列処理をおこなうことを計画しているが，実現のためにはクローン検出器が必要とするメモリ量を減らすことが必要である．大規模並列計算機では，ノード計算機あたりに多数の計算コアを搭載するので，計算コアあたりのメモリ量が相対的に少なくなるからである．例えば富士通 FX10 スーパーコンピュータの場合，ノード計算機あたり 16 コアかつ 32 GB メモリを搭載するので，計算コアあたり 2 GB しかメモリがない．

本論文は，我々が Scorpio の消費メモリ量の削減に向けて検討している Java の文字列オブジェクトの実装の工夫，およびそのためにおこなった実験の結果を報告する．Scorpio の動作を調査した結果，Scorpio は実行中，解析対象のプログラムの無数の断片を文字列として保持しており，消費メモリの実に約半分が文字列オブジェクトによって占められていたからである．2 つの文字列オブジェクトがあり，それらが表す文字列の一方が他方を包含している場合，2 つの文字列オブジェクトは内部情報を共有して消費メモリを減らすことができる．この手法をとる場合，実行中に包含関係にある文字列オブジェクトを多数発見できれば消費メモリは減少するが，多く発見しようとするとそれだけ実行速度が低下する．我々はいくつかの設定の下，消費メモリと実行速度の測定をおこない，実行速度の低下をおさえつつ消費メモリを減少させる方法を検討した．

2 文字列オブジェクトの実装法

Java 言語の文字列オブジェクトの実装方法は，文字列処理が多いプログラムの場合，そのメモリ消費量を決める重要な要素である．メモリ消費を少しでも減らすため，Java 6 までは例えば substring メソッドにより，ある文字列から部分文字列を切り出した場合，部分文字列のオブジェクトは元の文字列のオブジェクトと内部の char 配列を共有していた．この配列には文字列の各文字

	ファイル数	合計サイズ
Compress	253	1.9MB
Collections	497	3.9MB
Validator	128	1.1MB

表 1 入力データセット

A Preliminary Experiment of a Memory Reduction Method Using Substrings Implemented by Program Transformation for Large-Scale Code Clone Detection
* Tetsuro Yamazaki, †Yoshiki Sato, ‡Shigeru Chiba
§The University of Tokyo

	Compress		Collections		Validator	
共有範囲	実行時間	最大メモリ	実行時間	最大メモリ	実行時間	最大メモリ
(a) 共有なし	40sec	1.55GB	125sec	1.64GB	12sec	1.55GB
(b) 全フィールド	9,090sec	239MB	27,500sec	909MB	1,830sec	322MB
(c) 全 private フィールド	316sec	926MB	695sec	1.17GB	68sec	1.54GB
(d) 特定のフィールド	306sec	1.13GB	692sec	1.20GB	62sec	1.55GB

表 2　実験結果 (最大メモリはメモリ消費が最大の時に確保したヒープのサイズ)

の文字コードが格納される．この実装は Java 7 Update 6 からは変更され，部分文字列のオブジェクトは独立した char 配列を内部に持つようになり，元の文字列のオブジェクトと配列を共有することはなくなった．これは，長い文字列からごく短い文字列を部分文字列として切り出す場合，長い文字列全体を格納する char 配列が共有されるのを避けるためである．さもないと長い文字列がゴミになった後も短い文字列が生きている間はその char 配列はゴミとして回収されず無駄である．

我々は，substring メソッドで切り出した部分文字列のオブジェクトだけでなく，任意の文字列オブジェクトが新しく生成される度に他の文字列オブジェクトとの間に文字列の共通部分がないか調べ，ある場合には char 配列を共有させる手法を検討している．これによりメモリ消費量は抑えられるが，共通部分を調べるために実行時オーバーヘッドを伴う．そこでプログラムの限られた範囲で作られる文字列オブジェクトの間でだけ，文字列の共通分がないか調べる手法を考えた．この手法は，共通部分がある文字列オブジェクトはプログラムの狭い範囲で主に生成されるという仮説に基づいている．

3　実験

我々は Scorpio を用い，文字列の共通部分がないか調べるオブジェクトが生成される範囲を色々に変えて，消費メモリ量と実行速度を測定した．解析対象のプログラムとして Apache Commons プロジェクトから大きさの異なるサブプロジェクトを選んで用いた (表 1)．Scorpio は Intel Core i7-4770S (3.10GHz)，Ubuntu 14.04LTS，OpenJDK 7 で実行した．

実験では，生成時に他と共通部分がないか調べる特別な文字列クラスを実装し，プログラム中に現れる String クラスを置き換えた．置き換えのためにプログラム変換器を作成し，機械的に置き換えた．元の String クラスと新しい文字列クラスは互換性がないので，必要に応じて両者間の変換を実行するコードを挿入した．置き換えの範囲は次の通りである．(a) まったく置換せずに全て元の String クラスを利用，(b) 全てのクラスの全てのフィールドを新しい文字列クラスに置換，(c) 全てのクラスの private フィールドを新しい文字列クラスに置換，(d) 事前の分析で選んだ特定のフィールドだけを新しい文字列クラスに置換．なお新しい文字列クラスは，同じクラスの全てのオブジェクトとの間で文字列の共通部分がないかを探す．全てのオブジェクトとの間で探すのではなく，一部の限られたオブジェクトとの間でだけ探して探索時間を短くするような工夫はおこなっていない．

実験結果を表 2 に示す．実行時間と最大消費メモリの間にはトレードオフがあることが見て取れる．確かに全てのフィールドを新しい文字列クラスに置換する (b) と，消費メモリ量を大幅に削減できた．一方で実験した方針 (c)，(d) では，新しい文字列クラスに置換する範囲を狭めると，実行時間は改善するものの，消費メモリ量が増大した．実行速度を改善しつつ (b) に比べて消費メモリ量を増やさない方針は見つけられなかった．

4　まとめ

大規模コードクローン検出に向けて Type 3 コードクローン検出器 Scorpio のメモリ消費量を減らすため，文字列オブジェクトの実装方法の見直しを検討した．今後も実験を継続して，実行速度の悪化を許容できる範囲におさえつつ，消費メモリを削減して大規模並列計算機の上で Scorpio を動かす方法を探る．

参考文献

[1] 肥後芳樹，楠本真二. プログラム依存グラフを用いたコードクローン検出法の改善と評価. **情報処理学会論文誌ジャーナル**, 51(12):2149–2168, 2010.