分散処理を用いたコーディングパターン検出ツールの実装

悦田 翔悟† 伊達 浩典‡ 石尾 隆‡ 井上 克郎‡
†大阪大学基礎工学部 ‡大阪大学大学院情報科学研究科

1 はじめに

オブジェクト指向プログラミングには，継承や多態性など，モジュール化された部品を活用するための機構がある．しかし，すべての機能がモジュール化されるわけではなく，ロギングや同期処理などは，複数のモジュールに分散した定型的なコード片，すなわち，コーディングパターンとして実装される．

コーディングパターンを検出することにより，定型的なライブラリの使い方や，エラー処理などソフトウェアを理解する上で有益な情報が得られる．我々の研究グループは，シーケンシャルパターンマイニング手法の 1 つである PrefixSpan アルゴリズム [1] を用いて，Java のソースコードに対するコーディングパターン検出手法を実現している [2]．しかし，コーディングパターン検出は対象ソースコードの規模が増大するにつれて，計算コストが大幅に増加する．そのため数百万行におよぶ大規模ソフトウェアに対しては，既存手法によるパターン検出が困難である．

そこで，本研究ではコーディングパターン検出における PrefixSpan アルゴリズムを分散実行させることにより，パターン検出の高速化を実現し，大規模ソフトウェアへの適用を可能にする．

2 コーディングパターンマイニング

コーディングパターンとは，ソースコードの複数個所に出現する類似した構造をもつコードである．我々の研究グループでは，コーディングパターンをメソッド呼び出し要素と制御構造要素で構成される定型的な列と捉えたパターンマイニング手法を提案している [2]．図 1 は，画像編集ソフト

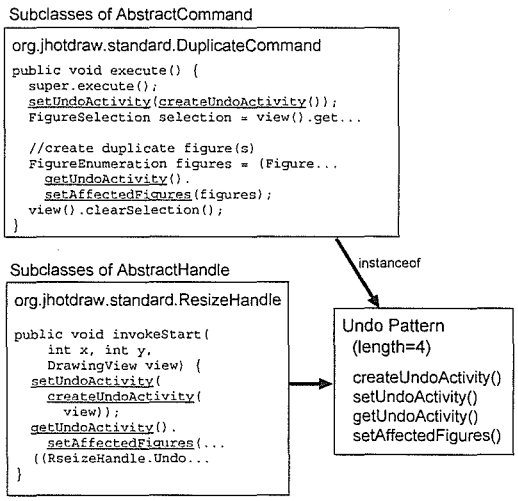

図 1: JHotDraw 5.4b1 における画像の編集作業を「元に戻す」実装パターン

JHotDraw 5.4b1 から検出されたコーディングパターンの例で，編集作業を「元に戻す」ことを可能とするための実装の一部である．

既存手法では，Java ソースコードをメソッド単位に分割し，メソッド呼び出し要素と制御構造要素の特徴列として正規化する．それにより得られた配列データベースに対して，PrefixSpan を適用し，頻出する部分列，すなわち，コーディングパターンを検出する．

この手法は，解析対象のソースコードが大きくなると，長時間の解析を必要とする．そこで，既存手法における PrefixSpan アルゴリズムの計算を分散実行することで解析時間を短縮する．具体的には，マスタ・ワーカ法による PrefixSpan の分散処理法 [3] を適用し，パターンマイニング部分を複数のジョブに分割し，複数のワーカでパターン検出を行う．本研究の対象データは，従来の負荷分散手法の実験が対象としていたアミノ酸などのデータに比べて，要素の種類が多く，列の長さが短いという特徴があることから，どの程度分散

Implementation of distributed coding pattern detection tool

†Shogo ETSUDA, ‡Hironori DATE, ‡Takashi ISHIO, ‡Katsuro INOUE
†School of Engineering Science, Osaka University
‡Graduate School of Information Science and Technology, Osaka University

表 1: 実験対象のデータ

実験対象	行数 (KLOC)	列数	ジョブ数
SableCC-3.2	35	1904	101
ANTLR-3.0.1	56	2045	208
JHotDraw-7.0.9	90	4981	537
jEdit-4.3pre11	160	6125	628

表 2: 実験対象ごとの解析時間と性能向上比

実験対象	ワーカ 1 台	ワーカ 2 台	ワーカ 3 台
SableCC	107 秒	72 秒 (1.5 倍)	57 秒 (1.8 倍)
ANTLR	1002 秒	542 秒 (1.8 倍)	416 秒 (2.4 倍)
JHotDraw	895 秒	533 秒 (1.7 倍)	414 秒 (2.2 倍)
jEdit	4767 秒	2421 秒 (1.9 倍)	2324 秒 (2 倍)

処理により性能が改善されるか，実験を行った．

3 適用実験

適用実験における分散処理の構成を示す．本研究ではマスタ PC には，Pentium4 3.2GHz を搭載する計算機を，ワーカ PC には，PentiumM 1.2GHz を搭載する計算機を用いた．そしてこれらの計算機を 100base-TX で接続した．OS は WindowsXP，通信には Java RMI を使用した．

実験対象には，表 1 に示す 4 つのソフトウェアを用いた．各ソフトウェアで，少なくとも 6 つのメソッドに出現するコーディングパターンを検出する処理を，ワーカの台数を 1〜3 台として実行した．解析時間を計測した結果を表 2 に示す．

ワーカ台数を 1 台から 2 台に増やすと性能は 1.5〜1.8 倍に向上し，3 台に増やすと 1.8〜2.4 倍に向上した．ワーカ台数を増やすごとに性能は上がっていくが，jEdit はワーカ台数を増やしても性能はさほど向上しなかった．

本実験における jEdit 解析時の各ワーカの解析時間は表 3 のとおりである．表 3 に示すように，各ワーカの解析時間には偏りが見られる．特にワーカ台数が 3 台のとき，ワーカ 3 の解析時間は他の 2 台のワーカと比べてかなり大きい．これは特定のジョブの処理時間が他のジョブに比べて大きく，それを担当したワーカの負荷が他のワーカよりも大きくなったためと考えられる．今回の実験では，1 つのジョブの解析が終わらないために，全体の解析時間が延びるケースが多く見られた．また，各ジョブの処理時間の偏りは，実験対象によって異なっていたため，実験対象によって性能向上比にばらつきが見られた．

表 3: jEdit 解析時の各ワーカの解析時間

ワーカ台数	ワーカ 1	ワーカ 2	ワーカ 3
1 台	4767 秒	-	-
2 台	2336 秒	2340 秒	-
3 台	971 秒	1474 秒	2233 秒

4 まとめ

コーディングパターン検出における課題の 1 つとして，対象を大規模ソフトウェアとしたときの解析時間の増加があげられる．本研究ではマスタ・ワーカ法を用いて PrefixSpan アルゴリズムを分散計算させることにより，パターン検出において，平均して台数 $\times 0.77$ 倍の高速化を実現した．今回の実験では，ジョブの処理時間に大きなばらつきが見られた．処理時間が長いジョブがパターン検出の後半で出現すると，特定のワーカに負荷が偏り，性能向上比が低くなる．これを改善するためにはワーカの負荷を均等にする必要があり，動的に負荷分散を行うタスク・スティール法などが有効であると考えられる．今後は，負荷分散の手法を取り入れ，パターン検出のさらなる高速化を目指す．

謝辞

本研究は，科学研究費補助金基盤研究（A）（課題番号:17200001）および萌芽研究（課題番号:18650006）の助成を得た．

参考文献

[1] Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U. and Hsu, M.: PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth, *Proc. 17th International Conference on Data Engineering*, pp.215-224 (2001).

[2] Ishio, T., Date, H., Miyake, T. and Inoue, K.: Mining Coding Patterns to Detect Crosscutting Concerns in Java Programs, *Proc. 15th Working Conference on Reverse Engineering*, pp.123-132 (2008).

[3] Sutou, T., Tamura, K., Mori, Y. and Kitakami, H.: Design and Implementation of Parallel Modified PrefixSpan Method, *Proc. 5th International Symposium on High Performance Computing*, pp.412-422 (2003).