2017 年 2 月 6 日

平成 28 年度　情報・通信工学科　　コンピュータサイエンスコース　卒業研究発表会

マルチスレッドプログラミングにおける デバッグ研究のためのバグ入りプログラムの作成

情報・通信工学科 学籍番号:1311171 寺田研究室 藤本 明優

1　背景

マルチスレッドプログラミング特有のバグを並行性バグといい, これは複数のスレッドにより並行処理を行う場合に, スレッド間の相互作用やメモリモデルの仕様によって発生するバグである.

並行性バグはスケジューリングに由来するため, バグの顕在化する確率の高さである再現性に問題がある.

再現性の低い並行性バグは発見と原因追及が困難であり, 今日まで様々なデバッグ手法が提案されてきた [4][6].

2　目的

本研究の目的はデバッグツールの開発におけるテストやデバッグ手法を学ぶためのサンプルとしてマルチスレッドのバグ入りプログラムを作成することである.

またバグ入りプログラムを自動生成することでサンプルを増やし, デバッグ手法やデバッグツールのベンチマークテストに利用できないか検討する.

3　マルチスレッドのバグ

3.1　バグの種類

並行性のバグには以下のような種類がある.

- メモリの可視性に由来するバグ
- デッドロック
- 競合状態
- データ競合
- 飢餓状態
- ライブロック

3.2　手作業でのデバッグ

手作業でデバッグを行う場合, デッドロックは再現したとき, Java Debugger や IDE のデバッグ機能によって検出が可能である.

対して競合状態やデータ競合では print 文や assert 文を使ってバグの原因箇所を突き止める必要がある.

3.3　Java Path Finder でのデバッグ

手作業でデバッグを行う場合, 並行性バグではバグが検出されるまでテストを繰り返す必要がある.

しかし Java Path Finder(JPF) を用いるとマルチスレッドのプログラムにおいて全てのスケジューリングを検査できる. JPF は Java のモデル検査器である. モデル検査を用いてデバッグを行う研究もされている [5].

> モデル検査とは, 対象システムを有限状態遷移系にモデル化し, そのモデルが時相論理式で表現した性質を満たすか否かをモデルの取り得る全状態空間の探索により網羅的に検査する手法である (モデル検査のデバッグへの適用 [5] より引用)

4　関連研究

4.1　A Study of Concurrency Bugs in an Open Source Software[2]

この研究は並行性バグと非並行性バグを比較, 検証するものである. オープンソースのフレームワークである Apache Hadoop[1] のバグレポートから並行性バグに関するものを抽出し集計している.

4.2　本研究との関連

この研究からデータ競合が並行性バグ全体の内 40% を占めていることがわかる. また深刻さが高くなるほどデッドロックの割合が増えていることや競合状態が一定量含まれていること, ライブロックや飢餓状態が占める割合が非常に少ないことがわかる.

また第 3.2 節の通りデッドロックは一度再現できればデバッグが可能である. したがって本研究では高い割合を占め, かつデバッグの容易でないデータ競合や競合状態に焦点を当てる.

5　作成したバグ入りプログラム

メモリの可視性は環境によって変化するため本研究ではそれに由来するバグを可能な限り避けることとする.

使用言語はマルチスレッドプログラミングがサポートされている Java である. Java SE は Java 8 を使用し, IDE は Eclipse Neon.1 Release (4.6.1) を使用した.

5.1　手作業で作成したバグ

5.1.1　BuggyBank

ロック順デッドロックの例としてよく用いられる仮想の銀行口座をサーバ/クライアント通信を使って再現するプログラムを基にロックの省略やロックの順序の変更によってバグ入りプログラムを作成した.

5.1.2　MapReduce モデル [3] に基づくプログラム

MapReduce とは分散プログラミングモデルの一つで, 入力を分割してキーと値の組を出力とする Map 処理とその処理の結果をキー毎にまとめたものを入力として出力を作る Reduce 処理を行う.

本実験では数字列を収めたファイル内の各数字の個数を出力する MapReduce モデルに基いたプログラムをサーバ/クライアント通信を用いて作った.

これを基にロックを省略する, 順序を制限するコードを取り払うなどしてバグ入りプログラムを作成した.

5.2 バグ入りプログラムの自動生成

正しいプログラムを基にバグ入りプログラムを自動生成するプログラムを作成した. 正しいプログラムとして BuggyBank で用いたプログラムと同様の処理に口座の削除とパスワードの変更を加えた仮想の銀行口座を作った.

正しいプログラムにおける各ロックについてそれぞれ消去した物とそのまま残したものを組み合わせた.

5.2.1 自動生成で作成できなかったバグ入りプログラム

ロックの消去以外にロックする区間をずらす方法を試みたが, 一つのファイルから 40 億通り以上の組み合わせが発生する場合があるため実行性能の関係上断念した.

また, データ競合について, 口座の削除とその口座への送金処理の間に二つの順序の制限を行っている. これらの制限を取り払った場合, データ競合のバグが発生するが, 自動処理では順序を制限するコードを特定できないケースがあった.

これらのバグを手作業で作成した.

6 評価

作成したバグ入りプログラムについて手作業によるデバッグと JPF によるデバッグを行い, それによってわかることをまとめた.

6.1 BuggyBank

手作業によるデバッグと JPF によるデバッグを行い, JPF によるデバッグがこのサイズのプログラムで動作することを確認した.

6.1.1 MapReduce モデル [3] に基づくプログラム

JPF でのデバッグは僅か四つの数字を数えるプログラムでも検査する空間が大きすぎて動作が停止せず, またその間バグを発見することもできなかった.

手作業によるデバッグでは例外など特殊な場合を再現するテストが必要になった.

6.2 自動生成したバグ入りプログラム

JPF でのデバッグはプログラムを口座の作成と送金のみに絞って 6 時間以上稼働しても停止せず, またその間バグを発見することもできなかった.

手作業によるデバッグでは assert 文を用いることでデバッグができた.

6.3 自動生成できなかったバグ入りプログラム

再現性の高いものは手作業によりデバッグできた.

再現性の極めて低いバグはそのバグを誘発しやすい命令のみを残し, スレッドプールの保持するスレッド数を減らすなどバグが顕在化しやすいようにしてテストを行うことでバグを発見, 修正することができた.

7 結論

7.1 まとめ

手作業では競合状態およびデータ競合, 自動生成ではロックの不足による競合状態のバグ入りプログラムを作成できた.

また, 手作業によるデバッグは柔軟性があるが再現性の低いバグのデバッグが困難であること, JPF によるデバッグは確実性があるが大きいプログラムのデバッグができないことを確かめることができた.

さらにバグ入りプログラムを JPF のベンチマークのサンプルにすることができたので, 作成したバグ入りプログラムをその他のデバッグツールのためのサンプルやベンチマークのために使用することが考えられる.

7.2 今後の課題

バグ入りプログラムの自動生成について, 非並行性バグを取り除く必要がある.

また, ロックの対象となるオブジェクトのクラスが同じものだけを組にしてその中だけでロックの有り無しを組み合わせるようにするなどで組み合わせの数をもっと減らすことで, より大きいプログラムからバグ入りプログラムを作成できるようになる.

組み合わせの数が減ることでロックの区間を変更する方法によるバグ入りプログラムの作成も可能になる可能性がある.

また関連研究 [2] より並行性バグのおよそ 40% はデータ競合であり, 例えば wait メソッド及びそれに付随する try-catch 文を取り除くなどの方法により実行順序を制限しているコードを取り除き, データ競合のバグ入りプログラムを自動生成できるようになることが望ましい.

参考文献

[1] Apache ソフトウェア財団. “Apache Hadoop”. https://issues.apache.org/jira/browse/HADOOP.

[2] Sara Abbaspour Asadollah, Daniel Sundmark, Sigrid Eldh, Hans Hansson, and Eduard Paul Enoiu. “A Study on Concurrency Bugs in an Open Source Software”. 12st International Conference on Open Source Systems, 2016.

[3] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. Sixth Symposium on Operating System Design and Implementation, 2004.

[4] 荒堀喜貴, 小宮常康, 多田好克. “マルチスレッド C プログラムのための高互換かつ高効率かつ高精度な競合検出法”. 第 53 回プログラミング・シンポジウム. 情報処理学会, 2012.

[5] 篠崎孝一, 太田弘, 早水公二, 星野光勇. “モデル検査のデバッグへの適用”. ソフトウェアテストシンポジウム 2006, 2006.

[6] 北野翔一郎, 片山徹郎. “Java マルチスレッドプログラム向けの拡張ペトリネットを用いた実行の再現を利用したデバッグ支援ツールの試作”. 研究報告ソフトウェア工学（SE) 2014-SE-185 23 号, pp. 1–8. 情報処理学会, 2014.