プログラム実行に対するフェイズ検出を用いたログ取得量の動的変更手法の提案

溝内 剛[1,a)]　嶋利 一真[1,b)]　石尾 隆[2,c)]　井上 克郎[1,d)]

概要：デバッグにおいて，エラーや実行時の変数情報などの記録を目的としてロギングが行われる．ロギングによって取得した詳細なログは，プログラムの異常な挙動の原因を特定する際に役立つ．ログには詳細な情報が記録されていることが望ましいが，膨大な記録量により出力オーバーヘッドやストレージの問題が生じる恐れがあるため，サーバ運用などで常に詳細なログを取得することは現実的ではない．そこで本研究では，プログラム実行時の挙動に応じてログの取得量を動的に制御する手法を提案する．提案手法ではプログラムの実行を挙動ごとに分類する手法であるフェイズ検出を応用し，異常な挙動を検出した際に，詳細なログの取得を行う．本手法の適用により，プログラムの異常発生時に詳細なログの取得をしつつ，通常時はログの記録量を抑えたロギングの実現を目指す．

1. はじめに

プログラム実行時の情報を記録する手法として，ロギングが存在する．ソースコード中にログ出力文を追加することで，実行時のエラー情報や変数情報を取得できる．取得した情報はプログラム理解やデバッグなどに役立てられる．

しかし，サーバプログラムなどの長時間稼働し続けるプログラムでロギングを行った場合，ログの記録量が膨大になり大きなストレージコストがかかる場合がある．たとえば，あるソフトウェアシステムが 1 台のマシンで 1 日に出力するログの平均が 2 GB に及んだという例も存在する [1]．また，ログを出力する処理よるオーバヘッドが生じることで，プログラムの実行に悪影響を及ぼす恐れがある．たとえば我々の環境で Apache Tomcat8.5.34 をすべてのログを取得する設定で動かしたところ，ほとんどログを取得しない設定で動かした場合に比べ Apache Bench[2] によるテストの実行時間が約 3 倍に増加した．

このような課題に対して，Apache Log4j[3] 等のロギングライブラリを用いてログ取得量を制御する方法がある．これは，各ログ文にログレベルを設定しプログラム実行時に設定した出力レベル以上のログを取得するというものである．これにより，例えば開発者側のテストでは詳細なログを取得する設定でデバッグを行い，ユーザ側の実行環境では実行時エラーに関するメッセージのみを取得することでストレージコストやオーバヘッドを抑えるということが可能になる．

しかし，実際のデバッグにおいて実行時エラーに関するメッセージのみでは，デバッグに不十分である恐れがある．たとえば，ユーザ側の実行環境で異常な挙動が発生した場合，ログなどの情報をバグレポートとして開発者側に送る場合がある．この際に送られるエラーメッセージのみでは，異常な挙動の再現には不十分である場合が多いことが報告されている [4]．異常な挙動の再現に必要な情報は，ユーザのプライバシー等の問題から取得することが困難である場合もあるため，デバッグにおいては異常な挙動が起きた実行環境での詳細なログを取得できることが望ましい．

そこで本研究では，プログラム実行時の挙動に応じてログの取得量を動的に制御する手法を提案する．提案手法では，プログラムの実行を挙動ごとに分類するフェイズ検出手法を応用する．フェイズ検出手法を用いてプログラムの通常時のフェイズ情報を記録し，通常時にない未知のフェイズを異常な挙動が発生しているフェイズとして検出する．異常なフェイズが検出されるとログの出力レベルを切り替え，デバッグのために詳細なロギングを行う．本手法の適用により，プログラムの異常な挙動の発生時に詳細なログの取得をしつつ，通常時はログの記録量を抑えたロギングの実現を目指す．

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University.
[2] 奈良先端科学技術大学院大学先端科学技術研究科
Graduate School of Science and Technology, Nara Institute of Science and Technology.
[a)] m-tys@ist.osaka-u.ac.jp
[b)] k-simari@ist.osaka-u.ac.jp
[c)] ishio@is.naist.jp
[d)] inoue@ist.osaka-u.ac.jp

2. 提案手法

本研究では，プログラム実行時の挙動に応じてログの取得量を動的に制御する手法を提案する．提案手法では，プログラムの実行を挙動ごとに分類するフェイズ検出手法を応用し，過去に観測されたことのないフェイズの出現を異常な挙動として検出し，ログの取得量を動的に切り替える．これにより，通常の挙動を行っている間はエラーメッセージなどの重要な情報だけを含むログを取得し，異常な挙動が発生した際にログ取得量を切り替え詳細なログを取得する．

2.1 フェイズ検出

フェイズとは，実行履歴から切り出された連続するイベント列のうち，開発者にとって意味のある処理の実行に対応するものを指す [5]．本研究で使用する Reiss の手法 [5] は，図 1 に示すように，まずプログラムの実行を一定時間で区切り，区間ごとにベクトル V を作成する．なお，実装ではこの区間の長さを 0.5s とする．このベクトル V の次元はプログラムが持つメソッドの数，ベクトルの要素は区間内での各メソッドの実行時間である．次に，ある区間 t のベクトル V_t を一つ前の区間のベクトル V_{t-1} とコサイン類似度を用いて比較する．類似度が閾値以上であれば t と $t-1$ は同じフェイズであり，閾値以下であれば t からフェイズが変わったと判定する．

本研究では，さらに V_t を既知フェイズの各ベクトルと比較することで，区間 t のフェイズが既知か未知かの判定も可能である．具体的には，すべての既知ベクトルと V_t を比較し，各類似度の中で最大のものが閾値以上ならば t は既知のフェイズであり，閾値以下であれば未知のフェイズとする．提案手法では，このフェイズ検出手法を用いてプログラムの異常な挙動の発生を検出する．まず，対象プログラムの通常時（異常な挙動が発生していない時）の各フェイズのベクトル情報を記録しておく．プログラムの実行時に，通常時にない未知のフェイズが検出されると異常な挙動が発生しているフェイズと判定する．

2.2 ログ取得量の動的制御

フェイズ検出手法を用いてログ取得量を動的に切り替える手法を，図 2 を用いて説明する．フェイズ検出によって通常時の挙動をしていると判定されている間（p[3]）は通常のログ取得量でロギングを行う．未知のフェイズ（p[?]）が検出され，異常な挙動が始まったと判定されると，ログの取得量を詳細なロギングが可能なものに切り替える．未知のフェイズが終了し通常時のフェイズ（p[4]）に移行すると，ログ取得量を通常時のものに戻す．これにより，異常な挙動が発生した時のみ詳細なロギングを行う．

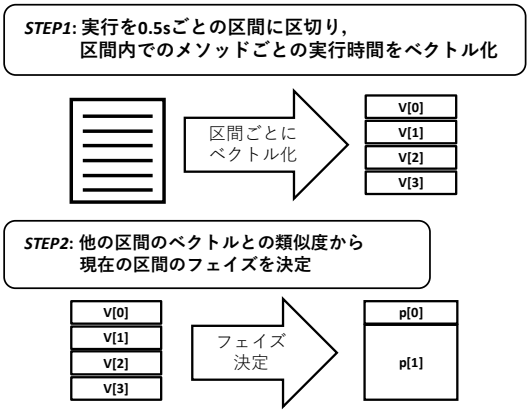

図 1　フェイズ検出

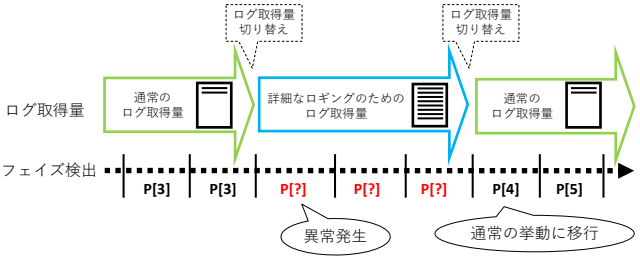

図 2　ログ取得量の動的制御

3. まとめと今後の課題

本研究では，プログラムの異常発生時に詳細なログの取得をしつつ，通常時はログの記録量を抑えたロギングの実現のために，プログラム実行時の挙動に応じてログの取得量を動的に制御する手法に取り組んでいる．今後の課題として，異常な挙動に関するより詳細な情報の取得のため，異常な挙動の発生前から発生に至るまでの詳細なロギングの実現がある．

謝辞　本研究は JSPS 科研費 JP18H03221，JP18KT0013，JP18H04094 の助成を受けたものです．

参考文献

[1] Fu, Q., Zhu, J., Hu, W., Lou, J.-G., Ding, R., Lin, Q., Zhang, D. and Xie, T.: Where Do Developers Log? An Empirical Study on Logging Practices in Industry, *Companion Proceedings of the 36th International Conference on Software Engineering*, pp. 24–33 (2014).

[2] Apache Software Foundation: ab - Apache HTTP server benchmarking tool, `http://httpd.apache.org/docs/2.4/programs/ab.html`.

[3] Apache Software Foundation: Apache Logging Services - Log4, `http://logging.apache.org/log4j`.

[4] Yuan, D., Mai, H., Xiong, W., Tan, L., Zhou, Y. and Pasupathy, S.: SherLog: Error Diagnosis by Connecting Clues from Run-time Logs, *SIGARCH Computer Architecture News*, Vol. 38, No. 1, pp. 143–154 (2010).

[5] Reiss, S. P.: Dynamic Detection and Visualization of Software Phases, *Proceedings of the Third International Workshop on Dynamic Analysis*, pp. 1–6 (2005).