様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

平成　２９　年　　５　月　２２　日現在

機関番号：　１２５０１
研究種目：　基盤研究(B)（一般）
研究期間：　2014～2016
課題番号：　２６２８００１９

研究課題名（和文）実行時検証とモデル検査の融合によるネットワークソフトウェアの統合実行監視

研究課題名（英文）Integrated Runtime Monitoring of Network Software by Fusion of Runtime Verification and Model Checking

研究代表者

山本　光晴（Yamamoto, Mitsuharu）

千葉大学・理学（系）研究科（研究院）・准教授

研究者番号：００２９１２９５

交付決定額（研究期間全体）：（直接経費）　　13,200,000円

研究成果の概要（和文）：実行時検証とモデル検査とを融合させることにより、ネットワークシステムが全体としての仕様を満たすことを検証しつつ、それが検査対象プログラムのスケジューリングによらないことを検査するような枠組を提案し、そのプロトタイプを実装した。
実行時検証単独では検査対象のスレッドスケジューリングに依存する挙動の違いを捕えられず、モデル検査単独ではネットワークシステム全体として満たして欲しい性質と直接に結びつかない。両者を融合させることにより、検査対象プログラムに対する原始的な性質のモデル検査では顕在化しないようなバグを検出可能にする。

研究成果の概要（英文）：By fusing runtime verification and model checking, we proposed a framework to verify global specification of network systems with taking account of thread scheduling nondeterminism of the system under test, and implemented its prototype.
Runtime verification alone cannot deal with thread scheduling nondeterminism, and model checking alone cannot handle properties of the whole network system consisting of the system under test and remote peers. By fusing both techniques, we made it possible to detect bugs that cannot be manifested only by existing model checking against the system under test.

研究分野：　情報数理学

キーワード：　実環境モデル検査　実行時検証　ネットワークソフトウェア

１．研究開始当初の背景

マルチスレッドプログラムに代表される並行プログラムの作成においては、デッドロックなどのバグを作り込みやすく、さらにスケジューリングの非決定性により、テストなどの手法では潜在的バグが顕在化しにくいという問題がある。この種の問題を解決するための一つの手法として、モデル検査が知られている。

モデル検査とは、システムの状態空間を系統的・網羅的に探索することによって検証を行う手法である。並行プログラムに対するモデル検査においては、スケジューリングが持つ非決定性を網羅的に探索し、どのようなスケジューリングを行ってもシステムが不適切な状態に陥らないことを検証することがその目的となる。

旧来のモデル検査の対象は専用の言語で記述された擬似的なプログラムであったが、近年は実際のプログラムを検査対象とするソフトウェアモデル検査が研究対象となってきている。さらに**実環境ソフトウェアモデル検査**においては、検査対象のプログラムを、それが実際に使用される状況に近い環境で実行しながら検査を行うことにより、実際に起こりうる不具合を早期に発見することを目的としている。

我々は、平成 20 年度～22 年度の科研費研究課題「仮想計算機によるコミュニケーションバックトラッキングとモデル検査への応用」および平成 23～25 年度科研費研究課題 「分散チェックポインティングを用いたネットワークアプリケーションのモデル検査」(以下、**先行研究課題**とする)において、並行プログラムとして書かれたネットワークアプリケーションのモデル検査に取り組んだ。**検査対象(SUT, system under test)**のプログラムは Java プログラムのモデル検査器である Java Pathfinder (JPF)の上で動作し、スケジューリングの非決定性が系統的に制御される。一方で、検査対象に対する**通信相手(ピア)**となるプログラムは、JPF とは別に通常のプログラムとして動作する。このハイブリッドな構成は、通信相手も含めてモデル検査器の上で動作させる「中央集権化」による手法と比較して、通信相手側の実装言語を問わずに実環境のプログラムを用いることができ、また状態空間の爆発が抑えられるなどの利点がある。

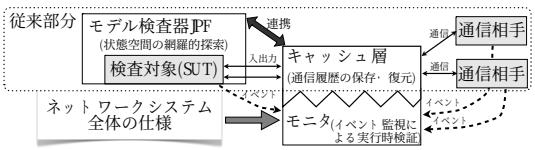
しかし、先行研究で検証可能な性質は、**検査対象のプログラムの振舞い**に関するものに限られていた。典型的な例としては、どのようなスレッドスケジューリングを行っても、デッドロック状態に陥らない、捕捉されない例外を発生しない、プログラム中に埋め込まれた assertion に違反しない、といった性質である。一方で、一般的なネットワークシステムにおいては、複数のプロセスが連携しながら協調して目的を達成することが意図されており、それ故単体のプログラムではなく、**ネットワークシステム全体の振舞い**に関する性質が満たされることを保証したいという要求がある。

他方、モデル検査とは別の検証手法として、**実行時検証**がある。実行時検証においては、システムの振舞いを監視し、ユーザが提供した形式的仕様に実際の実行トレース(イベント列)が違反していないかどうかを検証する。このような実行監視(モニタリング)は実行と同時にイベントを収集して行う場合も、ログを記録しておいて実行後に検証する場合もある。イベントの例としてはロックの取得・開放などが挙げられ、このようなイベント列を解析することで不適切なロックの使用が行われていないかどうかを検出する。

２．研究の目的

本研究課題の目的は、上記で述べた実行時検証とモデル検査とを融合させることにより、ネットワークシステムが全体としての仕様を満たすことを検証しつつ、それが検査対象プログラムのスケジューリングによらないことを検査するような枠組とその実装を提供することである。特に以下を達成することを目指している。

- 先行研究課題でネットワークアプリケーションのモデル検査に使用した**キャッシュ層**を、イベント取得や他コンポーネントとの連携、より柔軟なバックトラッキングに対応できるように再設計する。具体的内容は研究計画の部分で述べる。
- ネットワークシステム全体の振舞いに関する仕様を記述するのに、どのような論理・形式体系が適しているか検討し、設計・実装して評価する。
- イベント収集・解析部分を含め、全体の検証システムを開発し、応用例を用いて評価する。



本研究で提案する検証システムは、次のような例への応用が予想される。

- ロックグラフの監視による、複数のピア間での資源競合に起因するデッドロックの検出
- セッションの乗っ取り・複数セッション間競合によるデータ漏洩の検出
- セキュリティプロトコルに準拠しているかどうかの検査

実行時検証単独では検査対象のスレッドスケジューリングに依存する挙動の違いを捕えられない。一方、モデル検査単独では検査対象プログラムに対して記述される性質が原始的すぎて、ネットワークシステム全体と

して満たして欲しい性質と直接に結びつかない。両者を融合させることにより、検査対象プログラムに対する原始的な性質のモデル検査では顕在化しないようなバグを検出可能にすることが本研究の特色である。

３．研究の方法

以下の３つの部品それぞれに対応するサブプロジェクトに分割して研究を進める。

(1) net-iocache へのアクション・スコープの導入

net-iocacheはJavaプログラムのモデル検査器である Java Pathfinder(以下、JPF)上で**キャッシュ層**と呼ばれる機構を提供し、ネットワークアプリケーションのモデル検査を可能にするための拡張である。これは我々のグループが先行研究課題「仮想計算機によるコミュニケーションバックトラッキングとモデル検査への応用」(平成 20～22 年度)および「分散チェックポインティングを用いたネットワークアプリケーションのモデル検査」(平成 23～25 年度)において継続的に設計・実装してきたものである。

net-iocache はチェックポインティング機構との連携、非ブロッキング I/O への対応と順調に拡張を続けてきたが、一方で、それらの拡張の過程を通して初期の設計に問題が見え始めているのもまた確かである。このまま拡張を続けていくとコードの複雑化は避けられず、保守が極めて困難になることが予想される。

そこで、net-iocache に**アクション・スコープ**という概念を導入して、これを軸に既存のコードを整理・構造化・拡張することで、さらなる機能拡張や他のコンポーネントとの連携を容易に、かつアドホックでない形で行う。これにより、本研究で必要とされるイベントの取得や柔軟なバックトラッキングが可能となる。

(2) net-monitor の設計と実装

net-monitor は JPF 内外のネットワークプロセスからイベントを収集し、実行監視による実行時検査を行う部分である。具体的な実行監視手法は次項(3)の、論理/形式体系の評価と、実行監視フレームワークの定義の結果によって定まる。

net-monitor は最終的には従来研究課題で我々が開発した net-iocache と一体化し、JPF によるモデル検査と連携する。これは、モデル検査において SUT の状態のバックトラッキングを行うときに、実行監視側でも SUT の状態と同期させた形でイベント列に対するバックトラッキングが必要となるためである。そこで、初年度にどういった形で SUT の状態とイベント列を同期させるべきかを検討し、その設計を行う。

(3) 論理/形式体系の評価と、実行監視フレームワークの定義

実行監視のための論理/形式体系はこれまで多数提案されている。これらの間の違いは、主に実行時の効率と表現力のトレードオフをどのように解決するかである。本研究ではまずいずれの方式が本研究の目的に適切であるか、評価を行う。新たな実行監視のための体系の開発は行わない予定である。一方、これら既存のフレームワークが用いるシステム記述は、オートマトンや正規表現、ルールの集合といった比較的低水準なものが多い。本研究では、UML のステートチャート図や CSP (Communicating Sequential Processes)などの高レベル表現からこれらを導出する方法を検討する。

４．研究成果

(1) net-iocache へのアクション・スコープの導入

我々が先行研究課題でネットワークアプリケーションのモデル検査のために開発した net-iocache を、イベント取得や他コンポーネントとの連携、より柔軟なバックトラッキングに対応できるように再設計した。具体的には、アクション・スコープという概念を導入して、これを軸に既存のコードを整理・構造化・拡張した。

既存のキャッシュ層の問題点として、それが送受信されたデータのキャッシュしか持っておらず、ソケットの作成・接続・終了などに関する情報が保存されていないことが挙げられる。そのため、例えば一つの通信相手(ピア)に対し複数の接続を持つような検査対象(SUT, system under test)を扱うのが困難である。そのため、ソケットの接続のような作用をアクションに抽象化する。アクションは様々なレベルで作用し、ここではそれをスコープと呼ぶことにする。

例として、次のような Java プログラムの断片を考える。

コード	アクション	作用のスコープ
s=new Socket();	object の作成	オブジェクトスコープ
s.bind(portP);	portP への束縛	オブジェクト+システム(SUT)スコープ
s.connect(peerA);	peerA への接続	オブジェクト+ピアスコープ
s.close();	peerA への接続の終了	オブジェクト+システム+ピアスコープ

アクションの作用は、リソースの状態の変化を引き起こす。モデル検査の際に SUT の状態をバックトラックする際には、その変化を取り消さなければならない。しかし、上記 s.connect のような接続の場合は、変化が複数のスコープで起こるため、送受信データの復元だけでは不十分である。

そこで、この問題をアドホックでない方法で解決するために、作用のスコープ毎に個別のキャッシュを導入する。上記の例では、3 つ

のキャッシュを用意する。すなわち、1) s というソケットオブジェクトの専用キャッシュ 2) システムリソースのキャッシュ 3) ピアへの送受信データのキャッシュである。上記の例における作用は全てオブジェクトスコープで状態の変化を起こすので、1 番目のキャッシュに上記の全てのアクションが入る。しかし、2 番目のキャッシュには s.bind(portP) と s.close()しか入らない。これらは portP というシステムリソースを変更する作用だからである。同様に、3 番目のキャッシュに入るのは、s.connect(peerA)と s.close()という作用である。
このようにスコープ毎にキャッシュを保持することで、例外処理も含めて JPF 上の挙動と native JVM(Java Virtual Machine)上での挙動に一貫性を持たせられる、状態変化の影響範囲を精密に解析することで効率的なバックトラッキングが実現できる、UDP など別プロトコルの追加や他のコンポーネントとの連携が容易になる、などの利点が得られた。アクション・スコープの導入により、以前のバージョンに比肩する性能を保ちながら、長時間稼動させても安定して動作する堅牢性を兼ね備えていることを実験により確認している。
また、JPF を用いた検証をより広い範囲で行えるようにするため、JPF を Android の実機上で動作させる jpf-mobile を実装し、論文として発表した。
(2) net-monitor の設計と実装
net-monitor のアーキテクチャは以下の通りである。検証対象は分散システムであり、1 つの System Under Test(SUT)と複数の通信相手(ピア)とからなる。SUT は JPF 上で動作させ、ピアは通常の JVM 上で動作させる。
鍵となるアイデアは、拡張した JPF 上で SUT に対し網羅的検査を行い、ピアについてはコードインストルメンテーションにより関心のある内部状態を**モニターサーバ**に集約するというものである。JPF がスレッドスケジューリングの非決定性に対応するために通常行うスレッドインターリービングによる検証に加えて、net-monitor では(異なるホストで動作しているかもしれない)ピアの情報に依存する性質も検証する。さらに、ネットワークレベルの非決定性に影響される性質の検証に対応するため、SUT とその他のコンポーネントとの間の全ての通信は**プロキシキャッシュ**によって一旦横取りされ、例えば UDP におけるパケットの順序逆転や消失などのネットワークの非決定性がシミュレートされる。
上記の実行監視部分とモデル検査器を統合する net-monitor フレームワークの基本設計および、チャットサーバを用いた、1) SUT から送信されたメッセージは全てのピアで受信される 2) SUT が受信したメッセージは、ピアにとっては自身が以前に送信したものであるか、そのピアも受信したものであるといった、JPF の net-iocache のみでは検証不可能な大域的性質検証の実証実験の結果をショートペーパーとしてまとめ、発表した。
(3) 論理/形式体系の評価と、実行監視フレームワークの定義
実行監視のための論理/形式体系として、フレームワーク CSP_E を設計し、それ Scala 上の検査器を実装した。また、実例を用いた既存体系との比較、形式的意味論の構築を行い、論文として発表した。CSP_E はプロセス代数 CSP に基づく実行監視のためのフレームワークであり、並行システムをボトムアップに記述できる。これは既存のオートマトンベースの監視フレームワーク QEA がトップダウンに記述するのと対照的であり、システムコンポーネント間の正しいやりとりを記述しやすくすることを目的としている。

５．主な発表論文等

（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計 13 件）

① Cyrille Artho, Quentin Gros, Guillaume Rousset, Kazuaki Banzai, Lei Ma, Takashi Kitamura, Masami Hagiya, Yoshinori Tanabe, Mitsuharu Yamamoto,
Model-based API Testing of Apache ZooKeeper,
10th IEEE International Conference on Software Testing, Verification and Validation (ICST 2017),
査読有, 2017, 印刷中

② Alexander Kohan, Mitsuharu Yamamoto, Cyrille Artho, Yoriyuki Yamagata, Lei Ma, Masami Hagiya, Yoshinori Tanabe,
Java Pathfinder on Android Devices,
ACM SIGSOFT Software Engineering Notes,
査読有, 41(6), 2016, pp. 1 - 5,
DOI: 10.1145/3011286.3011292

③ Yoriyuki Yamagata, Cyrille Artho, Masami Hagiya, Jun Inoue, Lei Ma, Yoshinori Tanabe, Mitsuharu Yamamoto,
Runtime Monitoring for Concurrent Systems,
Runtime Verification - 16th International Conference, (RV 2016),
査読有, LNCS 10012, 2016, pp. 386 - 403,
DOI: 10.1007/978-3-319-46982-9_24

④ Cyrille Artho, Guillaume Rousset, Quentin Gros,
Precondition Coverage in Software Testing,
1st International Workshop on Validating Software Tests,
査読有, 2016, pp. 21 - 24,

DOI: 10.1109/SANER.2016.31

⑤ Cyrille Artho, Lei Ma,
Classification of Randomly Generated Test Cases,
1st International Workshop on Validating Software Tests,
査読有, 2016, pp. 29 - 32,
DOI: 10.1109/SANER.2016.32

⑥ Franz Weitl, Nazim Sebih, Cyrille Artho, Masami Hagiya, Yoshinori Tanabe, Yoriyuki Yamagata, Mitsuharu Yamamoto,
Cardinality of UDP Transmission Outcomes,
Dependable Software Engineering: Theories, Tools, and Applications - First International Symposium, SETTA 2015,
査読有, LNCS 9409, 2015, pp. 120 - 134,
DOI: 10.1007/978-3-319-25942-0_8

⑦ Cyrille Artho, Martina Seidl, Quentin Gros, Eun-Hye Choi, Takashi Kitamura, Akira Mori, Rudolf Ramler, Yoriyuki Yamagata,
Model-Based Testing of Stateful APIs with Modbat,
30th IEEE/ACM International Conference on Automated Software Engineering,
査読有, 2015, pp. 858 - 863,
DOI: 10.1109/ASE.2015.95

⑧ Lei Ma, Cyrille Artho, Cheng Zhang, Hiroyuki Sato, Johannes Gmeiner, Rudolf Ramler,
GRT: An Automated Test Generator Using Orchestrated Program Analysis,
30th IEEE/ACM International Conference on Automated Software Engineering,
査読有, 2015, pp. 842 - 847,
DOI: 10.1109/ASE.2015.102

⑨ Lei Ma, Cyrille Artho, Cheng Zhang, Hiroyuki Sato, Johannes Gmeiner, Rudolf Ramler,
GRT: Program-Analysis-Guided Random Testing (T),
30th IEEE/ACM International Conference on Automated Software Engineering,
査読有, 2015, pp. 212 - 223,
DOI: 10.1109/ASE.2015.49

⑩ Lei Ma, Cyrille Artho, Cheng Zhang, Hiroyuki Sato, Masami Hagiya, Yoshinori Tanabe, Mitsuharu Yamamoto,
GRT at the SBST 2015 Tool Competition,
IEEE/ACM 8th International Workshop on Search-Based Software Testing,
査読有, 2015, pp. 48 - 51,
DOI: 10.1109/SBST.2015.19

⑪ Cyrille Artho, Klaus Havelund, Rahul Kumar, Yoriyuki Yamagata,
Domain-Specific Languages with Scala,
Formal Methods and Software Engineering - 17th International Conference on Formal Engineering Methods, ICFEM 2015,
査読有, LNCS 9407, 2015, pp. 1 - 16,
DOI: 10.1007/978-3-319-25423-4_1

⑫ Nazim Sebih, Masami Hagiya, Franz Weitl, Mitsuharu Yamamoto, Cyrille Artho, Yoshinori Tanabe,
Software Model Checking of UDP-based Distributed Applications,
International Journal of Network and Computing,
査読有, 5(2), 2015, pp. 373 - 402,
DOI: 10.15803/ijnc.5.2_373

⑬ Cyrille Artho, Kuniyasu Suzaki, Masami Hagiya, Watcharin Leungwattanakit, Richard Potter, Eric Platon, Yoshinori Tanabe, Franz Weitl, Mitsuharu Yamamoto,
Using Checkpointing and Virtualization for Fault Injection,
International Journal of Network and Computing,
査読有, 5(2), 2015, pp. 347 - 372,
DOI: 10.15803/ijnc.5.2_347

〔学会発表〕(計 14 件)

① Cyrille Artho,
Model-based API Testing of Apache ZooKeeper,
10th IEEE International Conference on Software Testing, Verification and Validation (ICST 2017),
2017 年 3 月 14 日,
早稲田大学(東京都新宿区)

② Alexander Kohan,
Java Pathfinder on Android Devices,
Java Pathfinder Workshop 2016,
2016 年 11 月 18 日,
Seattle(USA)

③ Alexander Kohan,
Monitoring Distributed Applications with Java Pathfinder,
Java Pathfinder Workshop 2016,
2016 年 11 月 18 日,
Seattle(USA)

④ Yoriyuki Yamagata,
Runtime Monitoring for Concurrent Systems,
Runtime Verification - 16th International Conference, (RV 2016) ,
2016 年 9 月 29 日,
Madrid(Spain)

⑤ Cyrille Artho,
Precondition Coverage in Software Testing,

1st International Workshop on Validating Software Tests,
2016年3月15日,
大阪大学(大阪府吹田市)

⑥ Cyrille Artho, Lei Ma,
Classification of Randomly Generated Test Cases,
1st International Workshop on Validating Software Tests,
2016年3月15日,
大阪大学(大阪府吹田市)

⑦ Cyrille Artho,
Domain-Specific Languages with Scala,
Scala Matsuri 2016,
2016年1月31日,
東京国際交流館(東京都江東区)

⑧ Cyrille Artho,
Model-Based Testing of Stateful APIs with Modbat,
30th IEEE/ACM International Conference on Automated Software Engineering,
2015年11月13日,
Lincoln(USA)

⑨ Lei Ma,
GRT: An Automated Test Generator Using Orchestrated Program Analysis,
30th IEEE/ACM International Conference on Automated Software Engineering,
2015年11月12日,
Lincoln(USA)

⑩ Lei Ma,
GRT: Program-Analysis-Guided Random Testing (T),
30th IEEE/ACM International Conference on Automated Software Engineering,
2015年11月11日,
Lincoln(USA)

⑪ Cyrille Artho,
Domain-Specific Languages with Scala,
17th International Conference on Formal Engineering Methods, ICFEM 2015,
2015年11月5日,
Paris(France)

⑫ Franz Weitl,
Cardinality of UDP Transmission Outcomes,
Symposium on Dependable Software Engineering: Theories, Tools, and Applications,
2015年11月4日,
Nanjing(China)

⑬ Cyrille Artho,
Using Checkpointing and Virtualization for Fault Injection,
CANDAR 2014: The Second International Symposium on Computing and Networking - Across Practical Development and Theoretical Research,
2014年12月11日,
グランシップ(静岡県静岡市)

⑭ Nazim Sebih,
Software Model Checking of UDP-based Distributed Applications,
CANDAR 2014: The Second International Symposium on Computing and Networking - Across Practical Development and Theoretical Research,
2014年12月10日,
グランシップ(静岡県静岡市)

〔その他〕
ホームページ等
https://bitbucket.org/cyrille.artho/net-iocache

６．研究組織

(1)研究代表者
山本 光晴（YAMAMOTO, Mitsuharu）
千葉大学・大学院理学研究科・准教授
研究者番号：00291295

(2)研究分担者
萩谷 昌己（HAGIYA, Masami）
東京大学・大学院情報理工学系研究科・教授
研究者番号： 30156252

アルト シリル（ARTHO, Cyrille）
国立研究開発法人産業技術総合研究所・情報技術研究部門・主任研究員
研究者番号： 30462831

山形 頼之（YAMAGATA, Yoriyuki）
国立研究開発法人産業技術総合研究所・情報技術研究部門・主任研究員
研究者番号： 40415758

田辺 良則（TANABE, Yoshinori）
鶴見大学・文学部・教授
研究者番号： 60443199

(3)研究協力者
コハン アレクサンダー（KOHAN, Alexander）
千葉大学

ヴァイテル フランツ（WEITL, Franz）
千葉大学

馬 雷（MA, Lei）
千葉大学

セビ ナジム（SEBIH, Nazim）
東京大学