様　式　Ｃ－１９、Ｆ－１９、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

平成 ２８ 年　 ６ 月 ２０ 日現在

機関番号：１２６０１
研究種目：基盤研究(A)（一般）
研究期間：2011～2015
課題番号：２３２４０００３

研究課題名（和文）クラウドコンピューティングミドルウェアのソフトウェアモデル検査手法

研究課題名（英文）Software model checking methods for cloud computing middleware

研究代表者
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

交付決定額（研究期間全体）：（直接経費）　　37,200,000円

研究成果の概要（和文）：クラウドコンピューティングに代表される，ネットワークアプリケーションを対象としたソフトウェア検証の手法を開発した．先行研究で開発したモデル検査手法に基づくツールを，さまざまなプロトコルや通信形態に対応できるように拡張するとともに，モデルベーステスト手法に基づくツールも新たに開発することで，検証可能なアプリケーションの範囲を広げた．クラウドコンピューティング環境におけるプロセス協調ソフトウェアであるZookeeperをはじめとするさまざまなネットワークソフトウェアへの適用を通して，ツールの有効性を確立した．

研究成果の概要（英文）：We have developed methods of software verification for network applications that are typical in cloud computing environments. Our software model checking tool has been enhanced so that it can handle various protocols and application configuration. We also have developed a novel tool based on model base testing so that we can verify a wider range of applications. The usefulness of the tools has been confirmed by experiments with network applications such as Zookeeper, software for coordination of distributed processes in cloud computing environments.

研究分野：ソフトウェア

キーワード：仕様記述・仕様検証　ソフトウェアモデル検査　クラウドコンピューティング

１．研究開始当初の背景

クラウドコンピューティングは，巨大なネットワーク環境における分散処理である．分散処理ではその本質的な非決定性のため，競合条件，デッドロックなど，テストによる発見が難しく，システムに深刻なダメージを与えるバグを作り込みやすい．モデル検査は，このようなバグを検出する有効な手法である．

クラウド環境上で動作するアプリケーションは，大量のデータを効率よく扱うために，定められたパターン，たとえば map-reduce に従って設計・実装されることが多い．このようなパターンに則している限り，アプリケーションプログラムの誤りのために，上述のような問題が発生する可能性はあまり大きくない．クラウドコンピューティング環境を提供するミドルウェアによって，分散処理特有の誤りを混入させやすい部分が隠蔽されるからである．したがって，ミドルウェア自体が上述のようなバグを含まずに正しく動作することが非常に重要である．なお，ここではミドルウェアという言葉をやや広くとらえ，アプリケーションプログラムを動作させるために必要なライブラリ，サーバプログラムなどを指している．

しかし現実には，ミドルウェアにバグが無いとは言い難い．Apache Hadoop は，オープンソースによる代表的なミドルウェアであるが，この種のバグがしばしば報告されており，完全に修復されるまでに長い時間がかかる傾向にある．一例を挙げれば，2010 年 5 月 12 日に報告された競合条件に関するバグは，7 回にわたる誤った修正を経て 8 回目でようやく完全に修正されたが，それまでに 30 日以上を要している．ソフトウェアモデル検査が可能であれば，誤った修正はサブミット前に発見され，修正完了に要する時間はずっと短かったと考えられる．

また，クラウドコンピューティング環境では，ハードウェアは故障する可能性があるという前提の元に，多重化によって可用性を確保している．単一故障点を作らないようにするため，基本的にはどのマシンも対等であり，このため，リーダ選出などの複雑な協調処理が必要となる．このような処理が正しく動作することは，システムの安定のために決定的に重要であるが，Hadoop では，協調処理は Java のプログラム中で原子操作を組み合わせて実現することになっており，誤りが混入する可能性が大きい．このようなプログラムをソフトウェアモデル検査によって検証することの価値は高い．

クラウドコンピューティング環境においてソフトウェアモデル検査を実現する上での本質的な困難は，通信を行う複数のプロセスを対象にしなければならない点にある．ソフトウェアモデル検査器は，本来的に 1 つのプロセスのみを対象とするものであるが，通信する複数のプロセスのうちの 1 つだけをモデル検査の対象とすると，バックトラックの際に他のプロセスとの通信の同期が取れなくなるからである．

我々はこの問題の解決法として「I/O キャッシュ層」を提案していた．通信の実行を監視する層を設け，バックトラックを行うモデル検査対象とその他のプロセスとの整合をとるものである．この方式に基づき，広範囲のネットワークアプリケーションのソフトウェアモデル検査を可能とする基本的な枠組みを構築していた．しかし，クラウドコンピューティングミドルウェアにおけるソフトウェアモデル検査を実用的なレベルで実現するためには，まだ多数の課題を解決しなければならない状況にあった．

２．研究の目的

ネットワークソフトウェアモデル検査の手法をさらに発展させ，より多様なネットワークソフトウェアのソフトウェア検証を可能とする．特に，クラウドコンピューティングミドルウェアのような，複雑な協調処理を行うことが必要となるソフトウェアに適用できる手法を確立し，検証ツールを実装すること．

３．研究の方法

前項の目的を達成するため，以下の項目について研究を進める．

(1) モデルベーステスティング

ソフトウェアモデル検査は，スレッド等の計算主体の実行順序の全ての可能な場合を調べ尽くす極めて強力な手法であるが，全システムに適用するには，計算量が多すぎるという問題がある．したがって，効果の高い箇所に絞って適用することになる．

システム全体の検証にはテスト手法が有効であり，その際には，テストケースを自動的かつ効果的なケースを生成することがきわめて重要である．しかし，従来のアプローチは，イベント駆動や入出力駆動という特徴をもつネットワークソフトウェアには，うまく適合していない．特に，非ブロッキング入出力や，通信切断の際の例外処理といった箇所へのサポートが求められる．

モデルベーステスティング手法は，このような状況にうまく適合すると考えられるので，この手法に基づくテストケース生成ツールを開発することとする．当然，ネットワークアプリケーションへの適用が主要な目的ではあるが，可能な限り，一般性を保つように設計を行うこととする．

(2) net-iocache v2

net-iocache は，我々の先行研究で開発した，ネットワークアプリケーションを対象としたソフトウェアモデル検査ツールであり，Java PathFinder (JPF) のプラグインとして動作

する．強力なツールではあるが，個別システムに対応して開発を繰り返してきたことによる問題点も内包している．

本研究では，net-iocache の再設計・構築・実装を行い，net-iocache v2 を実現するとともに，曖昧な点のある意味論の明確化を図る．

(3) UDP プロトコル検証

net-iocache 拡張の一部ではあるが，独立性が高いため，個別項目として研究を実施する．

(4) 事例研究

上述のネットワークソフトウェア検証ツールを実際のシステムに適用することで，ツールの性能や実用性を評価する試みである．以下の 3 つのシステムを対象とする．

① Zookeeper

Zookeeper は，クラウドコンピューティング環境として広く用いられている Hadoop において，プロセス間協調動作を担当する，サーバソフトウェアである．クライアントに対して API を提供しており，これらの API は同期的もしくは非同期的に呼び出される．

Modbat によるモデル化の能力を確認するための事例研究を行う．

② OpenFlow

OpenFlow は，ネットワークの制御をソフトウェアで行う方式（SDN）の一つである．SDN によって，ネットワーク制御の柔軟性が向上した反面，検出しにくいソフトウェアのバグによってネットワーク全体が停止するなどの問題も生じるようになってきている．この項目では，ネットワークをモデル化して JVM 上で動作させ，コントローラをピアプログラムとみなすという形で，net-iocache によるソフトウェアモデル検査を行うことで，不具合の検出を試みる．

③ MQTT

MQTT は，IoT のアプリケーションレイヤのプロトコルとして代表的なものであり，信頼性が低いネットワークやデバイスを想定して設計されている．このプロトコルで通信するクライアントを対象に，net-iocache によるソフトウェアモデル検査を行う．

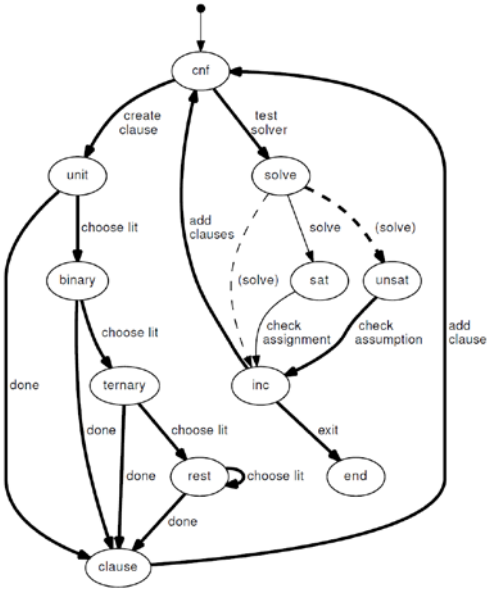
(5) その他

UDP プロトコルにおける検証がその例であるように，モデル検査手法で探索すべき空間が巨大となるため，単純な全探索が不可能であることがある．その場合であっても，探索範囲を適切に制限することができれば，モデル検査手法による網羅的で再現可能な探索は，不具合検出のために依然として有用な手法である．この項目では，探索範囲の適切な制限を行う手法を開拓する．

４．研究成果

(1) モデルベーステスティング

モデルベーステスティング手法を実現するツール Modbat を開発した．モデルベーステスティングでは，テストケースをテストデータではなく，実行すべきアクションの列として定義する．アクション列を生成するタイプの従来のツールでは，ユーザは提供された API を用いてモデルを構築する．このインタフェースでは，モデルの意味論との間に抽象度のギャップがあり，システムの振舞いの記述の際に困難を引き起こすことが指摘されていた．Modbat では，拡張有限状態機械（EFSM）と呼ぶ構造を用いて，システムの振る舞いを記述する．EFSM は，グラフィカルな表示をもち，直観的な理解がしやすい（後述の SAT ソルバーのテストケース生成のための EFSM を下図に示す）．利用者は，Modbat が提供する Scala 上に構築されているドメイン固有言語（DSL）を利用することで，EFSM をさらに詳細化することができる．たとえば，事前条件を設定することが可能である．



テストケースは，テスト対象システムの API の呼び出し列として作成され，テスト結果はアサーションでチェックすることもできるし，また，モデル中の変数に格納して，後のテストで参照することも可能である．また，例外処理や，非決定的な動作も自然にモデル記述することができる．

Modbat は，後述のように，Zookeeper の検証でも有用性が確認されているが，その他にも，SAT ソルバーのような複雑な構造を持つソフトウェアもモデル化し，テストケースを生成することができる．SAT ソルバー専用に C で書かれたテストケースジェネレータとの比較分析によって，テストケースの品質として同等であることが確認されている．

Java nio（ネットワークライブラリ）のような，非決定的なアクション（ノンブロッキング IO）を持つシステムもモデル化すること

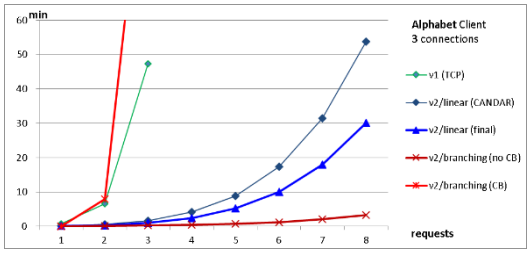
が可能である．Modbat が生成したテストケースによって，Java で記述されているライブラリの，これまで知られていなかった不具合を発見することができた．

Modbat の強みは，提供している DSL が軽量かつ柔軟である点にあり，モデル記述があいまいさなく，簡潔に表現できる．このため利用者は，システムの意味を表現することに集中することができ，モデル開発工数の削減や，モデル定義の際の誤りの混入のリスクの低減などの効果がある．

(2) net-iocache v2

net-iocache の再設計・構築・実装を行い，net-iocache v2 としてリリースした．v2 では，v1 と比較して，以下の改善を行った．

- プロトコルサポート：TCP に加え，UDP もサポートする．詳細は次項で説明する．他にも ftp など，良く用いられるプロトコルのサポートを行った．
- Java.net API のサポート範囲を拡張した．従来，対応できていなかった非ブロッキング入出力に対応した．また，例外処理にも対応した．
- 試験的な実装として，実行時検証技術との関連で，ログ機能との連携を行うようにした．
- 多様な接続形態をサポートする：peer-to-peer，サーバークライアント，複数のチャネルでの通信，中心化等に対応した．接続形態には，v1 では暗黙の仮定が存在した．この仮定を明示化し，それ以外の環境でも動作するオプションを与えた．
- スケーラビリティの改善：速度面では，v1 に比較して，オーダ単位での改善を実現した（下図参照）．NASA で開発されている同種のソフトウェアである jpf-nas （ただし中心化方式であり，リモートピアはサポートしていない）と比較して同程度の速度である．メモリ消費量では，v1 および jpf-nas よりも優位である．leader election ベンチマークでは，v1：6 スレッド，jpf-nas：10 スレッドに対し，v2 は 15 スレッドを動作させた．



(3) UDP プロトコル検証

net-iocache v2 における対応プロトコル強化の一環として，UDP プロトコルへの対応を行った．UDP においては，パケットは消失，重複，順序変更を起こしうる．ソフトウェアがこのような現象の下でも正しく動作することを検証する手段として，従来から，確率的な分布に基づいたテストが実施されている．我々の，ソフトウェアモデル検査によるアプローチは，空間を網羅的に探索することが，従来の方法にはない利点であると考えられる．特に，検出した問題の再現性が確実である点がすぐれている．

net-iocache v2 上で，上述の 3 要因に基づくパケットのパターンを実装した．どの要因を用いるのか，単独の要因を考えるのか，複合させるのか，また，各要因毎にどの範囲で現象を発生させるのかを，パラメタによって制御できるようにした．現実的な時間内に探索が終了するようにパラメタを調整して，検証を実施することが可能となった．

UDP を用いたファイル転送プログラムにこのツールを適用して，従来手法では見つかっていなかった不具合を検出した．この不具合の発生する条件はまれなものであり，組織的な探索を行わない限り，検出は難しい．

UDP プロトコル検証ツールの開発に加え，UDP 通信において生じうるパケット総数に関する理論的な評価を行った．

この評価は，UDP を用いるネットワークアプリケーションのソフトウェアモデル検査において，探索計算量見積もりの基礎となるものである．UDP において発生しうるすべてのパケットの組み合わせは無限にあるので，すべてを探索することは，不可能である．したがって，現実のモデル検査においては，上記の 3 要因をどのように組み合わせた範囲を探索するか決定することが重要となる．

要因ごとに範囲が指定された場合に，組み合わせ総数を明示的に求めることができた．この結果を用いることにより，モデル検査を計画する際の指針がたてやすくなる．

ツールを用いた実際のモデル検査の性能測定結果と比較したところ，見積もりが示す通りの傾向を示すことが確認された．

(4) 事例研究

① Zookeeper

Zookeeper は規模の大きなサーバプログラムであり，クライアントも様々なパターンで API を呼び出すことができる．適用作業を通して，Modbat は高いモデリング能力を持ち，この複雑なシステムを十分にモデル化できることが示された．

しかし，テストオラクルの記述において問題点が検出された．並行したリクエストが行われた際，ありうる結果を全てカバーするオラクルを人手で記述するのは，大きな工数がかかり現実的な選択とは言えないことがわかった．

我々は，この問題をモデル検査器を導入することで解決した．モデル検査器の網羅的探索機能を利用して，並行リクエストのアクション順序を列挙するものである．SPIN や Java PathFinder のプラグインなどの，既製の部品の流用も検討したが，Modbat への組込みにそれぞれの理由で困難があり，直接実装を行うこととなった．この結果，テストオラクルの自動的な生成が可能となった．

本事例研究を通して，Modbat に有用な機能を組み込むことができ，Zookeeper に対する効果的なテストケースが生成できた．この機能は，他の類似したアクセスパターンを持つアプリケーションにも適用可能である．

② OpenFlow
モデル化されたネットワークの状態空間を探索の対象とし，検証対象のプログラムをピアプログラムとして扱う，という通常と異なる構成でのモデル検査を実施した．net-iocache v2 での通信構成多様化の恩恵により，これが可能となった(v1 では対応していない構成であった)．あらかじめ作り込んでおいた不具合が，検出できることを確認した．(Proxy を作成する必要はあったが)検証対象のコントローラが Java で記述されていない(今回は Ruby+C) 状況でも検証が実施できたことは，net-iocache の高い柔軟性を示している．

③ MQTT
MQTT のクライアントライブラリである Paho とともに配付されているクライアントプログラムを対象に net-iocache によるモデル検査を行い，不具合を検出することができた．この不具合は開発者に報告され，未検出のものであったことが確認された．不具合の構造自体はそれほど珍しいものではないが，通常のネットワークでのテストでは，みつかる可能性は低いものである．IoT の実行環境において発現するかもしれないこのような不具合の検出に，net-iocache が効果的に働きうることが確認されたことになる．なお，サーバは C++で実装されているものを使用しており，ここでも，net-iocache の柔軟性が役立った形である．

(5) その他
モデル検査の対象となる状態空間の探索範囲を制限する手法として，深さ優先ヒューリスティック探索を考案し，これを JPF 上に実装した．従来のヒューリスティック探索手法が，幅優先探索をベースにして，探索候補に優先度を付与するのに対し，本手法では深さ優先探索をベースにして，探索を打ち切るための評価を与えるという点に特色がある．従来手法からある程度組織的に提案手法への変換ができることも有利な点である．変換前の探索よりも返還後の探索の方が良い成績（早く不具合をみつける）を取るケースが多いことを，実験によって示すことができた．

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 30 件）

1. Franz Weitl, Nazim Sebih, Cyrille Artho, Masami Hagiya, Yoshinori Tanabe, Yoriyuki Yamagata, Mitsuharu Yamamoto: Cardinality of UDP Transmission Outcomes. Proc 1st International Symposium on Dependable Software Engineering: Theories, Tools, and Applications. pp. 120-134, 2015. DOI: 10.1007/978-3-319-25942-0_8. 査読有
2. 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 Networking and Computing. Vol. 5 (2015), pp. 347-372. 査読有
3. Nazim Sebih, Masami Hagiya, Franz Weitl, Mitsuharu Yamamoto, Cyrille Artho, Yoshinori Tanabe. Software Model Checking of UDP-based Distributed Applications. International Journal of Networking and Computing, Vol. 5 (2015), pp. 373-402. 査読有
4. 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. Proc 30th IEEE/ACM International Conference on Automated Software Engineering. pp. 858-863, 2015, DOI: 10.1109/ASE.2015.95. 査読有
5. Cyrille Artho, Klaus Havelund, Rahul Kumar, Yoriyuki Yamagata: Domain-Specific Languages with Scala. Proc 17th International Conference on Formal Engineering Methods, pp. 1-16, 2015, DOI: 10.1007/978-3-319-25423-4_1. 査読有
6. Watcharin Leungwattanakit, Cyrille Artho, Masami Hagiya, Yoshinori Tanabe, Mitsuharu Yamamoto, Koichi Takahashi: Modular Software Model Checking for Distributed Systems. IEEE Transactions on Software Engineering, 40(5), pp.483-501, 2014. DOI: 10.1109/TSE.2013.49. 査読有
7. Richard Potter, Cyrille Artho, Kuniyasu Suzaki, Masami Hagiya: A Knoppix-based demonstration environment for JPF. ACM SIGSOFT Software Engineering Notes. Vol.39, 2014, pp.1-5. DOI: 10.1145/2557833.2560574. 査読無
8. Cyrille Artho, Masami Hagiya, Richard Potter, Yoshinori Tanabe,

Franz Weitl, Mitsuharu Yamamoto: Software Model Checking for Distributed Systems with Selector-Based, Non-Blocking Communication. Proc 28th Int. Conf. on Automated Software Engineering, pp. 169-179, 2013. DOI: 10.1109/ASE.2013.6693077. 査読有
9. Cyrille Artho, Armin Biere, Masami Hagiya, Martina Seidl, Eric Platon, Yoshinori Tanabe, Mitsuharu Yamamoto: Modbat: A Model-based API Tester for Event-driven Systems. Proc 9th Haifa Verification Conference, pp. 112-128, 2013. DOI: 10.1007/978-3-319-03077-7_8. 査読有
10. Watcharin Leungwattanakit, Cyrille Artho, Masami Hagiya, Yoshinori Tanabe, Mitsuharu Yamamoto: Model checking distributed systems by combining caching and process checkpointing. Proc 26th International Conference on Automated Software Engineering, pp. 103-112, 2011. DOI: 10.1109/ASE.2011.610043. 査読有

〔学会発表〕(計 39 件)

1. Cyrille Artho: Model-based testing. Shonan Meeting 2014-16, 2014.12.01-04, 湘南国際村センター (神奈川県)
2. C. Artho, K. Hayamizu, R. Ramler, Y. Yamagata: With an Open Mind: How to write good models. 2nd Int. Workshop on Formal Techniques for Safety-Critical Ssytems (FTSCS 2013). 2013.10.29. Crowne Plaza Hotel, Queenstown (ニュージーランド)
3. A. Biere, M. Seidl, C. Artho: Model-based Testing for Verification Backends. 7th International Conference on Tests and Proofs (TAP 2013). 2013.06.18. Budapest University of Technology and Economics, Budapest (ハンガリー)
4. Cyrille Artho: Analysis of networked software. 第 4 回新世代ネットワークおよび将来インターネットに関する日欧シンポジウム, 2012.01.19, TKP 東京駅八重洲カンファレンスセンター (東京都)
5. Cyrille Artho: Analysis of networked and cloud computing software. International workshop on Embedded Security of System and Software (IWESSS'12), 2012.02.13, National University of Singapore (シンガポール)

〔その他〕
ツール公開サイト
(jpf net-iocache)
http://babelfish.arc.nasa.gov/trac/jpf/wiki/projects/net-iocache
(jpf net-iocache v2)
https://bitbucket.org/weitl/jpf-net-iocache
(Modbat)
https://staff.aist.go.jp/c.artho/modbat/

６．研究組織
(1)研究代表者
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

(2)研究分担者
アルト　シリル (ARTHO, Cyrille)
産業技術総合研究所・
情報セキュリティセンター・研究員
研究者番号：30462831

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

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

横山　重俊 (YOKOYAMA, Shigetoshi)
国立情報学研究所・
アーキテクチャ科学研究系・特任教授
研究者番号： 10600968
(2011 年～2013 年)

(3)連携研究者
該当無し

(4)研究協力者
ヴァイテル　フランツ (WEITL, Franz)
ポッター　リチャード (POTTER, Richard)
レオンワタナキット　ワチャリン
(LEUNGWATTANAKIT, Watcharin)