分散システムの状態変化に対する非集中化モニタリング手法の提案

長岡　拓美†　佐藤　晴彦‡　栗原　正仁‡
北海道大学 工学部†　北海道大学 大学院情報科学研究科‡

1 はじめに

システムが満たすべき性質を動的に検証する技術として実行時検証[1]が注目されてきている。実行時検証は静的に検証を行うモデル検査と違い、システムを実際に実行させ、その実行トレースから満たすべき性質を満たしているかを動的に検証するものである。満たすべき性質は時相論理を用いて記述される。実行時検証はモデル検査のような静的検証技術と違い、計算コストが少なく、拡張性に優れている。そのため、分散システムのような大規模なシステムに対しても低コストで検証可能であるという利点をもつ。ここでは、分散システムのうち、分散オブジェクト指向システムを対象とし、システムの実行中に発生するイベントの順序関係に基づいてシステムの時間を定義し、その時間を用いてシステムの状態遷移を定義する。また、検証の際に非集中的に検証を行う方法について述べる。本手法では木構造を用いて論理式を部分論理式に分解し、流れ作業で検証結果を求める。そのため、検証を行うプロセスを分散配置することが可能である。

2 分散オブジェクト指向システム

2.1 モデル

本研究では、分散オブジェクト指向システムのモデルとして、アクティブオブジェクトモデル[2]を使用する。このモデルでは、各オブジェクトは自身のスレッドをもつアクティブオブジェクトと、それ以外のパッシブオブジェクトに分類される。アクティブオブジェクト同士では、並列メッセージを用いて情報のやり取りを行い、パッシブオブジェクトとアクティブオブジェク間では直列メッセージを用いてパッシブオブジェクトのメソッドが呼び出される。並列メッセージは非同期通信であり、直列メッセージは同期通信として扱われる。

2.2 処理の流れ

システムの処理の流れは、イベントの流れとして定義される。イベントは、(1)内部状態の変化、(2)メッセージの送信、(3)メッセージの受信、の3つである。これらのイベントに対し、Lamport は[3]で事前発生関係を定義し、その関係をもつイベント間の順序関係が常に成り立つ時間の表し方を示している。この時間を論理クロックと呼ぶ。

また、[4]では、Lamport の論理クロックに対する実装について述べている。そこでは、分散システムの各プロセス P_i はローカルカウンタ C_i を持っており、このカウンタが各プロセスの時間を表す。C_i は以下のように更新される（図1）。

1. イベント（メッセージ送信、または内部イベント)を実行する前に P_i は $C_i \leftarrow C_i + 1$ を実行する。
2. プロセス P_i が P_j にメッセージ m を送信するとき、1 を実行したあと、m のタイムスタンプ $ts(m)$ を自身のカウンタ C_i の値でセットする。
3. P_j がメッセージ m を受信すると、P_j は自身のカウンタを $C_j \leftarrow \max\{C_j, ts(m)\}$ に調整する。その後、$C_j \leftarrow C_j + 1$ を実行する。

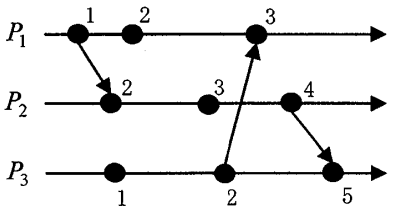

図1：システムの時間経過の例。黒点がイベント、矢印がメッセージ通信、イベントの上の数字が時間を表す。

本研究では、この方法により分散オブジェクト指向システムにおける各オブジェクトの時刻を定義し、時刻の変化をシステムの状態変化として検証を行う。つまり、時刻 n におけるシステム全体の状態を s_n としたとき、時刻が 1 増えて $n+1$ となったとき、状態は s_{n+1} に遷移すると考える。同じ時刻に複数のオブジェクトで起きたイベントは同時に起きたものとする。時刻 n でのシステムの状態について検証を行うとき、カウンタが n である点までのイベントの集合に対して検証を行う。ここで、この方法で時間を表したとき、ある時刻 l でイベントが発生しないオブジェクトが存在する。そのとき、そのオブジェクトは「何もしていな

Decentralized Monitoring for State Change of Distributed System
†Takumi NAGAOKA; Faculty of Engineering Hokkaido University
‡Haruhiko SATO, Masahito KURIHARA; Graduate School of Information Science and Technology, Hokkaido University

い」とし、オブジェクトに変化がないとする。

3 検証方法

3.1 記述言語

システムが満たすべき性質は、時相論理を用いて記述される。本研究では過去の事柄を扱う言語である PTLTL[5]を使用する。PTLTL は有限の実行トレースに対する検証と相性が良い。

以下に、PTLTL の構文と簡単な意味について述べる。AP を原子命題としたとき、構文は次のように帰納的に定義される。

- $p \in AP$ ならば、p は PTLTL 式である。
- F, G が PTLTL 式ならば、$\neg F$, $F \vee G$, $F \wedge G$, $F \rightarrow G$ も PTLTL 式である。
- F, G が PTLTL 式ならば、$(*)F$, $[*]F$, $<*>F$, $F\ \mathbf{S}\ G$ も PTLTL 式である。

$(*)F$ は1つ前の状態で F が成り立つことを表し、$[*]F$ は過去のすべての状態で F が成り立つ、$<*>F$ は過去のある状態で F が成り立つ、$F\ \mathbf{S}\ G$ は過去のある状態で G が成り立ち、かつその状態から現在の状態までのすべての状態で F が成り立つことを表す。これらのオペレータを用いてシステムの満たすべき性質を記述する。

3.2 モニタリングと検証

システムのイベントの発生を、モニタを用いて監視し、その情報から検証を行うことをモニタリングという。しかし、オブジェクトのすべての内部状態の変化をイベントとみなしてカウンタの更新を行うのは現実的ではない。そこで、内部状態の変化に対するカウンタの更新は、PTLTL で記述された性質に関係のあるイベントが起きたときに行われる。各オブジェクトには専用のモニタが取り付けられ、イベントが発生したかどうか監視される。

イベントが起こると、その内容がシステムの外部にある別のモニタに通知される。このモニタは得られた情報から検証を行うモニタである。モニタはその通知を受けると、それまでに得た情報から、システムが性質を満たしているかどうかについて検証を行う。

本研究では、記述された PTLTL 式より構文木を生成し、その木から検証用モニタを生成する。モニタの作成例を図 2 に示す。図 2 の例においては、モニタ 3 が結果をモニタ 2 に送信し、モニタ 2 が結果をモニタ 1 に送信する。モニタ 1 は自分の結果が全体の検証結果となり、検証後、何らかの行動を行う。例えば、すべてのモニタに通知、ログファイルに出力などがある。

このように情報の伝播を行うことで、検証を複数のモニタを用いて流れ作業で行うことができる。そのため、モニタを分散配置することで検証の負荷を分散させることが可能となる。ここで、送られてきた情報から検証を行う際に、システムの状態に矛盾が起きないように検証する必要がある。これは、時間が最も進んでいないオブジェクトに時間を合わせて検証を行うことで、矛盾の発生を防ぐことができる。

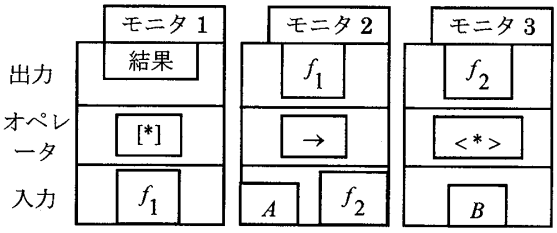

図 2： $[*](A \rightarrow <*> B)$ におけるモニタ作成の例。
例。
上段が出力、中段がオペレータ、下段が入力を表す。

4 評価・実装

現在、これまでに述べた手法を使用して検証を行うべく、簡単な検証システムの実装を進めている。PTLTL は過去のことに対する性質を記述するため、ある結果から、その前に起こる事象に対して検証するときに役立つ。例えば、「サーバーがクライアントに返答を返すとき、その前に必ずそのクライアントから要求がある。」というような性質を記述できる。このような分散システムにおけるイベントの順序関係は同期処理などの検証に利用可能である。

5 まとめ

分散システムにおけるイベントの順序関係を定式化し、その関係を満たすようにシステムの状態遷移を定め、その状態遷移の中でシステムが満たすべき性質について動的に検証する方法を示した。また、大規模な分散システムに対する検証における処理を分散的に行う方法を示した。今後は、この手法を用いたシステムの実装を行い、分散システムによく見られる性質に対して検証を行いたい。

参考文献

[1] M.Leucker, C.Schallhart:"A Brief Account of Runtime Verification", The Journal of Logic and Algebraic Programming Volume 78 pp.293-303 (2009)
[2] 丸山勝巳:"並列オブジェクト指向言語 COOL",情報処理学会論文誌 Vol.34, No.5 (1993)
[3] L.Lamport:"Time, Clocks, and the Ordering of Events in a Distributed System", Comm.ACM, Vol.21, No.7 pp.558-564 (1978)
[4] M.Raynal, M.Singhal:"Capturing Causality in Distributed Systems", IEEE Computer, (29)2:49-56 (1996)
[5] K.Havelund, G.Rosu:"Efficient Monitoring of Safety Properties", International Journal on Software Tools for Technology Transfer Volume 6, pp.158-173 (2004)