3L-9

ベクトルクロックを利用した 分散プログラム用逆実行デバッガ

野田雄一朗† 太田剛‡
静岡大学大学院情報学研究科† 静岡大学情報学部‡

1. 背景

分散プログラムは複数のマシン上にある複数のプロセスが、互いにネットワークを介して必要な情報を交換し、協調しながら並列に動作する。

プログラムの動作をエラーが発生した状態から遡って原因を追跡するのは、プログラムの誤りを発見する有効な手段である。動作を遡るための逆実行を行う方法は、最寄のチェックポイントから再実行をするか、逆のオペレーションを繰り返すことで達せられる[1]。

複数のプロセスから構成され動作に非決定性を伴う分散プログラムを逆実行させることは、単一のプロセスによる逐次実行プログラムにおける逆実行と同じように行うことはできない。ある一つのプロセスを逆実行させることでそれに直接的間接的に関係する他のプロセスも逆実行させる必要性が生じるのである。あるプロセスがすでに送受信が完了したメッセージを送る前の状態にまで逆実行したとき、そのメッセージを受信したプロセスもまた受け取る前の状態にまで逆実行しなければ、送信者のないメッセージを受け取ったという矛盾した状態となってしまう。

分散プログラムは複数のプロセスの動作が複雑に絡み合うため、このような矛盾のない一貫した状態を追跡するには多大な計算量が必要になる。しかしベクトルクロックというものを用いることにより、比較的少ない計算量で、プログラム全体の動作の一貫性を損なうことなく各プロセスに逆実行させる範囲を特定することができる。

2. ベクトルクロック

2.1 因果先行

事象間の因果関係を示す半順序関係 → が Lamport によって定義された[2]。その定義は以下のとおりである。

A Reversible Debugger for Distributed Programs Using Vector Clock

† Yuhichiro Noda
Graduate School of Informatics, Shizuoka University
‡ Tsuyoshi Ohta
Faculty of Informatics, Shizuoka University

e_1 と e_2 を任意のプロセス上にある任意の事象とする。以下のいずれかが成り立つとき、e_1 は e_2 に因果先行するといい、$e_1 \rightarrow e_2$ と表記する。

1. e_1 と e_2 が同じプロセス上で発生する。かつ、e_1 が e_2 に先行する。
2. e_1 がメッセージ m の送信を示す事象である。かつ、e_2 が m の受信を示す事象である。
3. $e_1 \rightarrow e_3$ かつ $e_3 \rightarrow e_2$ となる事象 e_3 が存在する。

e_1 と e_2 の間に $e_1 \rightarrow e_2$ または $e_2 \rightarrow e_1$ のどちらも成り立たないとき、e_1 と e_2 は同時に発生しうる事象であり、$e_1 \| e_2$ と表記する。

2.2 ベクトルクロック

ベクトルクロックは各プロセスのタイムスタンプから事象を半順序付けする論理時計として Mattern によって定義された[3]。その定義は以下のとおりである。

プロセス p_i のローカル時刻は n 次元のベクトル $v_i = (v_i[1], \cdots, v_i[n])$ によって示される。ベクトル中の $v_i[j]$ は、p_i がその時点までに知り得た p_j のローカル時刻である。特に $v_i[i]$ は p_i 自身のローカル時刻である。v_i は p_i の各事象 e に与えるタイムスタンプ ts(e) となる。

v_i は以下の規則により更新される。

1. 各 p_i で事象 e が起きたとき
   $v_i[i] \leftarrow v_i[i] + d \quad (d > 0)$
2. p_i が p_j からメッセージ m を受信したとする。m は p_j が m を送信したときの p_j のローカル時刻を示す v_j を含む。このとき、各 $h(=1, \cdots, n)$ に対して $v_i[h] \leftarrow \max(v_i[h], v_j[h])$

v_i と v_j をふたつのプロセスのローカル時刻とする。このとき v_i と v_j の間に以下の順序を定義する。

1. 各 $h(=1, \cdots, n)$ について $v_i[h] \leq v_j[h]$ となるとき $v_i \leq v_j$ である。
2. $v_i \leq v_j$ かつ $v_i[h] \neq v_j[h]$ となる h が存在する

とき、$v_i < v_j$である。

3. $v_i \leq v_j$と$v_j \leq v_i$のどちらも満たさないとき $v_i \| v_j$である。

ここで、e_1, e_2は各々のプロセス p_i と p_j で発生し、タイムスタンプ $ts(e_1)=\langle i, v_i \rangle$, $ts(e_2)=\langle j, v_j \rangle$を持つとする。このとき $v_i < v_j$ ならば $e_1 \rightarrow e_2$ であり、$v_i \| v_j$ ならば$e_1 \| e_2$である。

2.3 ベクトルクロックの特長

以上の定義にしたがって構築されるベクトルクロックの最大の特長は、他のプロセスに他の事象の因果先行関係を参照することなく、任意のふたつの事象間の因果先行関係を求めることができるという点である。すなわち高々ベクトル長n程度の計算量で因果先行関係を求めることができるということである。

この特長を利用することで、ある基準となる事象 e に対し、e が他のプロセスの事象に影響を及ぼすことのない範囲、すなわち因果先行関係で後に発生する事象の直前までの範囲(以下 Possible Point)と、e が発生するのに最低限必要な範囲、すなわち因果先行関係で前に発生する事象の範囲(以下 Minimum Point)を求めることができる。

3. 分散プログラムの逆実行

ベクトルクロックを利用することで、ユーザが分散プログラムのあるプロセスを任意の事象 e まで逆実行させたいとき、e を基準とした Possible Point または Minimum Point をとることで、プログラム全体の一貫性を損なわないために他のプロセスを逆実行させる範囲をシステムが自動的に求めることができる。

求めた範囲の逆実行は、最寄のチェックポイントから再実行する方式をとる。しかし、分散プログラムには逐次プログラムにはない問題が存在する。分散プログラムの動作には非決定性を伴うため、単純に再実行するだけでは異なるふるまいをするおそれがある。そのため、正しい逆実行の結果を得るためにはふるまいを再現させる必要がある。また絶対時間が保証されないため、ふるまいを再現させるには工夫が必要となる。

そこで、メッセージの送受信事象の実行履歴を取得し、前述したベクトルクロックという論理時計を用いることで、事象の発生順序を実行履歴に従って再現させ、逆実行を実現する。

4. システムの概要

デバッガ制御下で動作させた分散プログラムは、まず通常実行し、送受信事象発生時にベクトルクロックを計算、実行履歴に記録する。

ユーザの任意のタイミングで特定のプロセスを任意の事象まで逆実行させることができる。逆実行を指示する入力が得られたら、すべてのプロセスに対してベクトルクロックを用いて逆実行する事象を求め、そこへ至る再実行のために必要な情報を構築する。

分散プログラムの再実行は、送受信事象に対してメッセージの到着順序の制御を行う。具体的には、送信事象においてタイムスタンプを ID としてメッセージに付加し、受信事象において到着したメッセージが順序を正しく再現するものであれば受理、そうでなければメッセージプールに格納し、正しい順番のメッセージが到着するのを待つ。メッセージプールに格納されたメッセージは受信事象が発生するたびに確認を行い、受信する順番がきたメッセージはメッセージプールから取り出して受信を受理するというふるまいをする。

逆実行を完了すると、分散プログラムは実行履歴を取得する通常実行を再開させる。

5, 評価

本システムは逐次プログラム用デバッガである gdb を基盤として作成された分散プログラム用再現実行デバッガ ddb をさらに拡張して作成した。

そして、分散プログラムの同期問題を説明する古典的な例題として挙げられる哲学者の食事問題を分散プログラムとして作成し、これを実験に用いた。その結果、分散プログラムを逆実行させるという目標を達せられた。

今回作成したデバッガでは目的とする機能の実装に重点をおいているため、取り扱うプロセス数、事象数、実行履歴の量などの定量的な評価には現段階では重点をおいていない。今回作成したシステムを雛型として、今後はこのような定量的な評価、改善を図っていく必要がある。

【参考文献】

[1] Shyh-Kwei Chen, W. Kent Fuchs, Jen-Yao Chung : Reversible Debugging Using Program Instrumentation, *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, VOL.27, NO.8, pp.715-727(2001)

[2] Lamport, L.: Time, Clock, and the Ordering of Events in a Distributed System, *Communications of the ACM*, Vol. 21, No. 7, pp. 558-564(1978).

[3] Mattern, F.: Virtual Time and Global States of Distributed Systems, *in Parrallel and Distributed Algorithms (Consnard, M. and Quinton, P. eds.), North-Holland, Amsterdam*, pp.215-216(1989).