TM-1284

分散環境におけるデッドロックの WTC 方式を用いた検出方式

六沢　一昭、近山　隆、
市吉　伸行（三菱総研）

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191 ~ 5

Institute for New Generation Computer Technology

分散環境におけるデッドロックの WTC方式を用いた検出方式

六沢 一昭　　近山 隆　　市吉 伸行
(財) 新世代コンピュータ技術開発機構　　(株) 三菱総合研究所

1 はじめに

非同期メッセージ通信を行なう分散環境では大域的な状態の把握が困難である。この大域的な状態のひとつにデッドロックがある。これは、計算は終了していないが、すべてのプロセスが実行を停止しており、メッセージも存在しない状態である。

本稿では、終了検出の一方式である WTC方式を利用したデッドロック検出方式を述べる。

2 計算モデル

- ひとつの制御プロセスと有限個の子プロセス(以下"プロセス"と略す)が存在し、互いに非同期メッセージによって通信する。
- メッセージは送信されてから受信されるまでに任意の時間がかかる。このため任意の時刻において、「送信はされたがまだ受信されていないメッセージ("通信中のメッセージ"と呼ぶ)」が存在しうる。
- プロセスには実行中/中断/終了の3つの状態がある。中断状態とは、他のプロセスからのメッセージを待って計算を中断している状態であり、メッセージを受信すると実行中になる。実行中のプロセスはメッセージを送信することがあり、計算を終了すると終了状態になる。またメッセージを待って中断状態になることもある。終了状態のプロセスはメッセージを受信すると実行中になる。

2.1 デッドロックとその検出のむずかしさ

計算は終了していないが、実行中のプロセスも通信中のメッセージも存在しない状態がデッドロックである。

この状態を検出することは容易ではない。例えば、制御プロセスが「状態を尋ねるメッセージ(*check*)」をブロードキャストし、すべてのプロセスから「中断状態を示すメッセージ(*suspended*)」を受信したとしても、デッドロックが起こっているとは限らない。各プロセスが*suspended*を送信した時刻は同じではなく、また通信中のメッセージも存在しうるからである。

Distributed Deadlock Detection Using Weighted Throw Counting
Kazuaki ROKUSAWA (ICOT), Takashi Chikayama (ICOT),
Nobuyuki ICHIYOSHI (MRI)

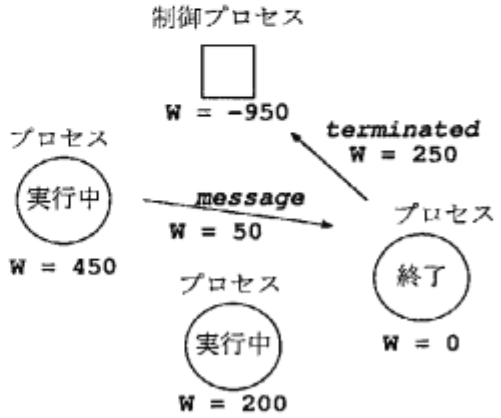

図 1: WTC方式

3 WTC方式

WTC方式[1]は、制御プロセスとすべてのプロセス及び通信中のメッセージに"重み"を持たせる。制御プロセスの重みは負の整数、プロセス及び通信中のメッセージは正の整数である。そして『重みの合計がゼロ』を保ちながら重みの割付けを行なう。この結果すべてのプロセスが終了し通信中のメッセージも存在しない時にのみ制御プロセスの重みがゼロになる。

送信するメッセージには適当な重みを割付け自分の重みはその分だけ減らす。重みが1である場合は、メッセージを送信することができないので、制御プロセスへ重みの供給を要求する。メッセージを受信すると付いていた重みを自分の重みに加える。計算が終了すると重みをすべてメッセージ(*terminated*)で制御プロセスへ返す。*terminated*を受信し重みの加算を行なった結果ゼロになったならばすべての計算の終了が検出される。

4 デッドロックの検出

4.1 検出のアイデア

WTC方式によりプロセスと通信中のメッセージの重みの合計を制御プロセスが把握しているので以下の操作によりデッドロックを検出することができそうである。

制御プロセスが*check*をブロードキャストする。プロセスは*check*を受信すると、中断状態ならばすべての重

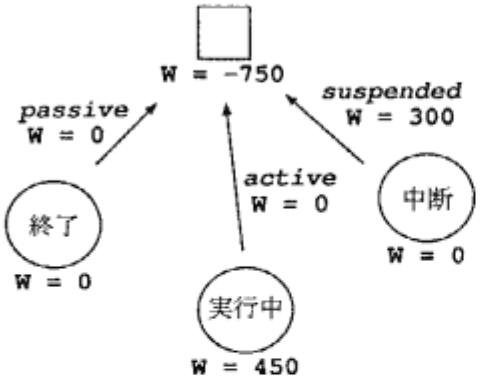

図 2: デッドロックの検出

みを *suspended*で返却し、実行中あるいは終了状態ならば *active*あるいは *passive* (いずれも重みゼロ)を返信する。制御プロセスは *suspended*を受信すると重みを加算し、ゼロになったならばデッドロックが検出される。「*active*を受信した」あるいは「*suspended*あるいは *passive*のみを受信したが重みがゼロにならない」場合はデッドロックしていないことがわかる。前者は実行中のプロセスが存在した場合、後者は通信中のメッセージが存在した場合である。

4.2 考慮すべき状況

正常終了or デッドロック?

実行中のプロセスはいつでも終了するので、制御プロセスはデッドロック検出中(*check*をブロードキャストしてからすべての応答を受信するまで)に *terminated*を受信するかもしれない。

ところが検出中に *terminated*を含むメッセージを受信し重みがゼロになった場合、起こっている状況として正常終了とデッドロックのどちらも考えられる。例えば *suspended*と *terminated*をひとつずつ受信して重みがゼロになった場合、2つが同じプロセスから *suspended*,*terminated*の順に送信されたのならば正常終了であるが、2つの送信元が異なる、あるいは逆の順序で送信された場合はデッドロックである。

このため *terminated*も受信して重みがゼロになった場合はあらためて正常終了/デッドロックを判断する。

重みゼロプロセス

中断状態のプロセスはデッドロック検出処理によって重みがゼロになるが、この「重みゼロプロセス」は危険な存在である。デッドロックが起こっていなかった場合、実行中のプロセスがすべて終了し重みが返却されると、この重みゼロプロセスを残したまま制御プロセスの重みがゼロになってしまうからである。

これを避けるため、デッドロックの起こっていないことがわかった段階で制御プロセスは *supply*をブロードキャストして重みを供給する。この重みの供給は上述した誤った終了検出を防止するだけでなくプロセスの重みが1になる確率を低くする働きもする。

4.3 検出処理

4.3.1 制御プロセスの処理

*check*の応答メッセージ数をカウントするカウンタ(初期値 = プロセス数)と、*supply*の送信を記憶するフラグ(初期値 =OFF)を用意する。

処理の開始 *check*(重みゼロ)をブロードキャストする。

***active*を受信** カウンタを −1 し、フラグが OFF ならば ON にセットして *supply*をブロードキャストする。

***suspended*を受信** 重みを加算し カウンタを −1 する。

***passive*を受信** カウンタを −1 する。

***terminated*を受信** 重みを加算する。

カウンタゼロを検出 重みがゼロで *terminated*未受信ならばデッドロックである。重みがゼロで *terminated*受信済みならば、再度 *check*をブロードキャストし、その応答がすべて *passive*ならば正常終了、ひとつでも *suspended*を受信したらデッドロックである。重みがゼロでなければデッドロックは起こっていないので、フラグを調べて OFF ならば *supply*をブロードキャストする。

4.3.2 (子)プロセスの操作

各状態における *check*受信時の処理を以下に示す。

実行中 *active*(重みゼロ)を返信する。

中断状態 すべての重みを *suspended*で返却する。

終了状態 *passive*(重みゼロ)を返信する。

5 おわりに

重み付けを利用したデッドロック検出を述べた。本方式によるデッドロック検出はUNIX/Cベースの KL1 処理系 KLIC[2] に実装する予定である。

参考文献

[1] Rokusawa, K. *et al.* : An Efficient Termination Detection and Abortion Algorithm for Distributed Processing Systems. *ICPP'88*. Vol.I. pp.18–22. 1988.

[2] 仲瀬明彦他：ポータブル KL1 処理系 KLIC の概要, 情報処理学会 第47回全国大会, 3D-7, 1993.