A-012

分散グループ相互排除アルゴリズム
An Distributed Algorithm for Group Mutual Exclusion

尾崎 比呂人 † Hirohito Ozaki
角川 裕次 † Hirotsugu Kakugawa

1. はじめに

分散システムとは、互いに通信用リンクで接続された複数の計算機が通信を行いながら協調して動作し、ある共通の目的を達成するシステムである。分散処理における重要な問題である相互排除とは、多くの競合するプロセスの中から唯一つのプロセスを選出し、そのプロセスに対して資源を使用する為の特別な権利を与える競合解消である。これを拡張し、同一の資源を要求する複数のプロセスに資源の共有を許すようにしたものがグループ相互排除であり、これまでにも様々なアルゴリズムが考案されて来た。だがそのアルゴリズムを評価する方法も種々ある。中にはトレードオフの関係を持つものもあり全ての評価法について最善のパフォーマンスを持つアルゴリズムを作りだすことは困難である。そこで本研究ではグループ相互排除問題に於いて、あるパラメータを変化させることで各評価法についてのアルゴリズムの評価値を変化させ得るアルゴリズムを提案する。

2. グループ相互排除

$U = \{p_1, p_2, ..., p_n\}$ をプロセスの集合とし、$R = \{r_1, r_2, ..., r_m\}$ を資源の集合とする。プロセスはそれぞれ以下の状態を繰り返す。

- NCS：資源獲得要求が発生していない。
- $trying$：資源獲得要求が発生しているが、未だ獲得していない。
- CS：資源を獲得している。

グループ相互排除では同一の資源 r_i を要求するプロセス集合 G_i をグループ g_i と呼び、グループ g_i に属する複数のプロセスが同時に資源 r_i を獲得することが出来る。しかし、異なる資源 r_j が他のグループ g_j によって使用されていた場合、同時に r_i を使用することは出来ない。

このグループ相互排除問題を解く為には、以下の二つの条件を満たすアルゴリズムを構築する必要がある。

- **Mutual exclusion**：同一のグループのプロセスは同時に資源を獲得出来るが、異なるグループのプロセスは同時に資源を獲得出来ない。
- **Lockout freedom**：資源を要求するプロセスは必ず有限時間内に資源を獲得出来る。

このグループ相互排除を実現する単純な方法として、

- 他の全てのプロセスから許可を得る
- 調停を行う特別なプロセスを一つ用意し、全てのプロセスはそこから許可を得る

などが考えられるが、前者では一度許可を得るだけでも最低全プロセス数 N に比例する膨大な通信が必要となり、後者では調停を行うプロセスが故障しただけでシステムが停止する為、耐故障性に欠ける。

そこで、効率的に通信量を減らせ、かつ耐故障性を持つ有効な情報構造として知られているコータリを用いることを考える。

2.1 コータリによる相互排除

U を大きさ n のプロセス集合とし、空でない U の部分集合 Q_i の集合を C としたとき、

Intersection property：任意の $Q_i, Q_j \in C$ に対して $Q_i \cap Q_j \neq \emptyset$

Minimality：任意の $Q_i, Q_j \in C$ に対して $Q_i \not\subseteq Q_j$

という二つの条件を満たすならば Q_i をコーラムと呼び、この集合 C をコータリと呼ぶ。

Intersection priority によりコーラムは少なくとも一つのプロセスを共有するので、少なくとも一つのコーラムの全てのプロセスから許可を得るようにすれば、2つのプロセスが資源を獲得することは無い。コータリを用いることにより、資源獲得の為に必要な通信量は $O(|Q|)$($|Q|$ はコーラムの大きさ) となる。また、一つのコーラムの全てのメンバが生き残っていればシステムの運用が可能である為、耐故障性に優れているとも言える。

2.2 アルゴリズムの評価法

グループ相互排除アルゴリズムの評価法は主に3種存在する。

- **concurrency**：同時に資源を共有できるプロセスの最大数。
- **message complexity**：資源の使用許可を求めてから、資源を獲得し、解放するまでに交わされるメッセージの総数。
- **synchronization delay**：資源の使用許可を求めてから資源を獲得するまでに掛かる単位時間。

このような評価法の中には、アルゴリズムによってはトレードオフの関係となるものもある。この中でも特によく使われる **concurrency** と **message complexity** は、幾つかのアルゴリズムでトレードオフ関係を持っている [1][2]。

2.3 concurrency と message complexity のトレードオフ関係

相互排除に於ける競合解消は、多くの場合、あるプロセスに許可を出している状態でより高い優先度を持つプロセスから許可を求められた場合は、現在与えている許可を取り消し、より優先度の高いプロセスに許可を与え直すことで、システム全てのプロセスが最終的に最も優先度の高いプロセスに許可を与える、という方法によって成される。よって競合が発生した場合、許可の取り消しの為のメッセージが余分に増えることになる。

†広島大学大学院工学研究科情報工学専攻

concurrency が増加すると、一つのプロセスが一度に許可を出す要求の数も増える為、競合が発生した場合のメッセージの数もより増加する。

そこで、状況に応じて **concurrency** の値を変化させ、最悪時の **message complexity** の増大を抑えるという手法が考えられる。

3. 関連研究

グループ相互排除問題をコータリーを用いて解くものとして、文献 [1], [2] がある。文献 [1] では、surficial quorum system と呼ばれるコータリーの拡張を用いて、**concurrency** に制限を設けることで効率的にグループ相互排除問題を解くアルゴリズムが提案されている。このアルゴリズムはコーラム数を変更することで **concurrency** と **message complexity** のトレードオフが可能であるが、トレードオフを行う為にはコータリーを再構築し直すという大変な労力を伴う。また、文献 [2] では、[1] が資源の構造と **concurrency** に制限がある点に着目し、資源の構造と **concurrency** に制約の無いグループ相互排除アルゴリズムを提案している。このアルゴリズムでは **concurrency** に制約が無い代わりに、最悪時に於いて **message complexity** がプロセスの総数に比例した膨大な数になる。

本研究ではシステムから独立したパラメータを設け、その値を変更することで資源の構造やコータリーの構造等に依存すること無く **concurrency** と **message complexity** のトレードオフを行うアルゴリズムを提案する。

4. アルゴリズム

プロセスはそれぞれ識別子を持ち、各プロセスはメッセージパッシング方式により通信する。資源を獲得したプロセスは有限時間内に資源を解放すると仮定し、初期状態では資源を獲得しているプロセスはいないものとする。また、資源要求に優先度を付ける為に Lamport による時刻印 [3] を用い、時刻印の小さい方 (同じときは識別子の小さい方) をより優先度が高いとする。

以下では区別の為、コーラムに含まれるプロセスをノード、それ以外をプロセスと呼ぶ。

concurrency を定める為のパラメータ k を定義し、各ノードは属するコーラム毎に最大 k 個のプロセスに許可を与えるものとする。以下にアルゴリズムの概要を示す。

- **要求発生**：プロセス p_i はコータリーの中から任意に選択したコーラム Q_r 内の全てのノードに対し、資源獲得要求メッセージ *REQUEST* を送信する。
- **照会**：ノードは未だどのプロセスにも許可を出していないか、r として p_i と同じグループのプロセスに出した許可が未だ k に達していない場合は直ちにを与える。既に k に達していた場合は p_i よりも優先度の低いプロセスの許可を取り消し、p_i に許可を与える。p_i よりも低い優先度のプロセスがいなかった場合は p_i の要求は受けいれられない。もし異なるグループのプロセスに許可を与えている場合は、現在許可を与えているプロセス中で最も優先度の高いプロセス p_j と p_i の優先度を比較し、p_i に許可を出せるか照会する。もし p_i の優先度が高ければ、現在出している許可は取り消され、新たに p_i に許可が与えられる。許可要求が受け入れられなかった場合は、要求メッセージを待ち行列に挿入する。
- **資源獲得**：Q の全てのノードから *GRANT* を得ることが出来れば、p_i は資源を獲得出来る。
- **資源解放発生**：資源を獲得した p_i は、有限時間内に Q の全てのノードに資源解放メッセージ *RELEASE* を送信する。
- **資源解放**：p_i からの資源解放メッセージを受け取ったノードはカウンタをデクリメントし、その値が 0 になったかどうかを調べる。0 であった場合、カレントグループは初期化され、待ち行列より次に許可が与えられるグループが選ばれる。

  0 でなくとも、現在許可を与えている残りのプロセス全てよりも高い優先度を持つプロセスが待ち行列に存在した場合、許可は取り消され、もっとも高い優先度を持つプロセスのグループに許可が与えられる。

このアルゴリズムにおける **concurrency** は km、**message complexity** は最良の場合 $3Q$、最悪の場合 $3Q+3dkQ$ である。ここで、k は $1 \leq k \leq \frac{N}{m}$ の任意の値を取るパラメータ、N はプロセスの総数、m はコーラムの数、Q はコーラムの最大のノード数、d はあるノードを共有するコーラム数の最大である。k の値を変化させることで、**concurrency** と最悪時の **message complexity** のトレードオフが可能となっている。

5. おわりに

本研究では、コータリーに依存せず concurrency と最悪時 message complexity のトレードオフを行うコータリーを用いたグループ相互排除アルゴリズムを提案した。

しかし、提案したアルゴリズムには、要求が 1 つのコーラムに集中した場合 **concurrency** が最大値まで達しない状況が想定される、問題に適したパラメータの値を求める手法が確立されていない、といった問題点がある。

よって今後の課題としては、要求が集中した場合も **concurrency** が最大値まで達することを保障するようにアルゴリズムを改善する、最適なパラメータ値を算出する手法の確立などが挙げられる。

参考文献

[1] Y.-J.Joung, “ Quorum-based algorithms for group mutual exclusion, ”in Proceedings of the 15th Conference on Distributed Computiong (DISC), Vol. 2180 of LNCS, pp. 16-32, 2001.

[2] Mie Toyomura, Sayaka Kamei, Hirotsugu Kakugawa,“ A Quorum-Based Distributed Algorithm for Group Mutual Exclusion, ” in Proceedings of the 4th International Conference on Parallel and Distributed Computing, Applications and Technologies, 2003.

[3] L.Lamport, “ Time, clocks, and the ordering of events in a distributed system, ” Communications ACM, Vol. 21, pp. 558-565, 1978.