適応的分散アルゴリズム 第６章
無待機システム

川染翔吾

6.3.5 スナップショットオブジェクト

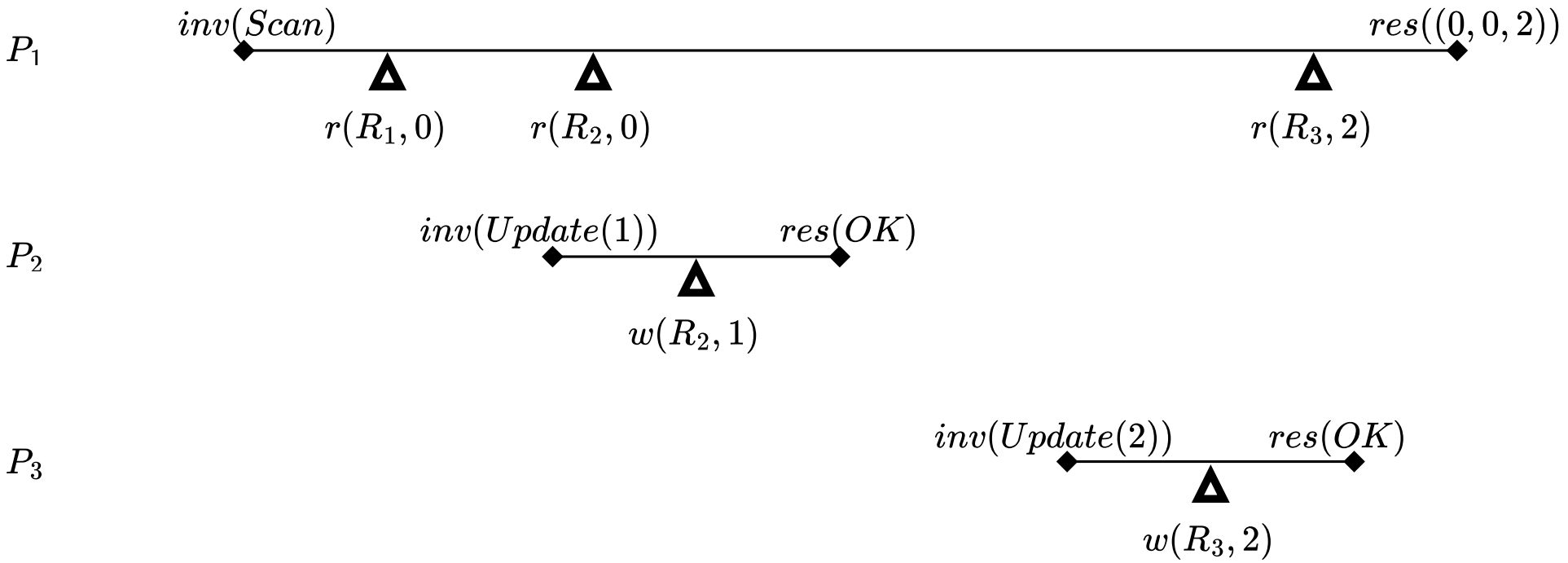
スナップショットオブジェクト

- スナップショットオブジェクト：$Update$ 命令と $Scan$ 命令の二つの操作が可能な共有オブジェクト
  - $Update_i(v)$ 命令：S_i に値 v を書き込む
  - $Scan_i$ 命令：$S_1, S_2, \ldots, S_n$ すべての値を読み出す
- 共有スナップショットオブジェクトを MRSW レジスタを用いて実現する

単純な方法

- 各セグメント S_i を nR1W レジスタ R_i で実現する
- $Update_i(v)$ 命令：P_i が v を R_i に書き込む
- $Scan_j$ 命令：P_j が $R_1, R_2, \ldots, R_n$ を順に読み出す

単純な方法


P_1
$inv(Scan)$
$res((0,0,2))$
$r(R_1,0)$
$r(R_2,0)$
$r(R_3,2)$
P_2
$inv(Update(1))$
$res(OK)$
$w(R_2,1)$
P_3
$inv(Update(2))$
$res(OK)$
$w(R_3,2)$

2重操作

- レジスタに時刻印をデータ値と共に書き込む
- 読み出し操作を繰り返し実行する
- 2回の連続した読み出しで同じ値を各レジスタから読み出せば、この間も$Update$命令が並行して実行されていないことが保証できる
- $Scan$ 命令の走査のたびに $Update$ で更新される場合、終了しない
- 全体では、いずれかのプロセスが実行している命令は必ず終了する
  - ッ全体がデッドロックで停止することはない（**無ロック性**）

SNAPSHOT

- 書き込むデータ値、時刻印、スナップショットの3項組をレジスタに格納
- $Update$ のときのスナップショットを格納しておき、$Scan$ 命令の実行期間に完全に含まれている $Update$ によって書き込まれたスナップショットを採用

SNAPSHOT

P_i 上の $Update(v)$ 命令

1. $view \leftarrow Scan$
2. $ts \leftarrow ts + 1$
3. $R \leftarrow v, ts, view$
4. `return` (OK)

P_i 上の $Scan$ 命令

1. `for` $k \leftarrow 1$ `to` n `do` $moved[k] \leftarrow false$
2. `while`
3. `for` $k \leftarrow 1$ `to` n `do` $First[k] \leftarrow R_k$
4. `for` $k \leftarrow 1$ `to` n `do` $Second[k] \leftarrow R_k$
5. `if` $First = Second$ `then` `return` $(First.val)$
6. `for` $k \leftarrow 1$ `to` n `do`
7. `if` $First[k].ts \neq Second[k].ts$

SNAPSHOT

定理

アルゴリズム SNAPSHOT は、n 個の nR1W レジスタを用いて、n プロセスが共有するスナップショットオブジェクトを実現する無待機アルゴリズムである

証明

無待機性

$Scan$ 命令の 2 以降の `while` で、$moved$ の値が $true$ の要素が増えない場合、命令の実行を完了する。したがって、2重走査は高々 $n+1$ 回しか実行されない。
$Update$ 命令は $Scan$ 命令と $write$ 命令が実行されるだけ

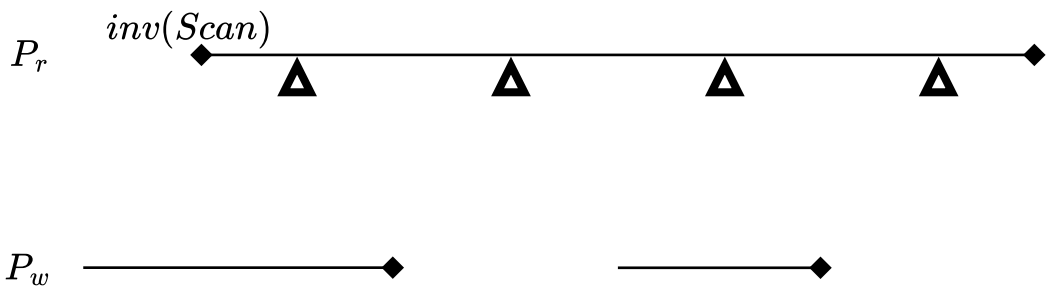
線形化可能性

次のように線形化ポイントを選ぶ
(1) $Update$ 命令に関しては、レジスタに値が書き込まれた時点。
(2) $Scan$ 命令に関して、5. で終了する場合は 3. と 4. の間の任意の時点。
(3) $Scan$ 命令に関して、7. で終了する場合は あるレジスタ R_k から読み出した $R_k.view$ をスナップショットとして採用している。プロセス P_k は $Update$ 命令の中で内部 $Scan$ 命令で得られたスナップショットを $R_k.view$ に書き込んでいるが、この内部 $Scan$ 命令の線形化ポイントの直後。

まず各命令の線形化ポイントが命令の実行期間内にあることを示す。
(1)と(2)は明らか。

(3)について
$Scan$ 命令を実行したプロセスを P_i とし、レジスタ R_k から読み出した $R_k.view$ をスナップショットとして返すものとする。
$Scan$ 命令の実行で P_i は $R_k.ts$ の値が異なる2重走査を 2 回実行したことになる。この2回の2重走査それぞれで2重走査の間に終了した $Update$ 命令が存在する。
op を2回目の2重走査の間に終了した P_k の最後の $Update$ 命令とすると、P_i は op が書き込んだ $R_k.view$ をスナップショットとして返している。

op の実行期間は P_i の $Scan$ 命令の実行期間に含まれる。



op が 5. で終了している場合、P_i の線形化ポイントは op の実行期間内にあり、したがって P_i の実行期間内にある。
op が 7. で終了している場合、この議論を繰り返すことで 5. で終了している内部 $Scan$ 命令が現れ、P_i の線形化ポイントがその実行期間内であることが示せる

次に、線形化ポイントに従って並べた命令の列がスナップショットオブジェクトの逐次仕様を満たすことを示す。

$Update$命令はレジスタに書き込んだ時点を線形化ポイントとしている。
また、5. で終了する $Scan$ 命令は線形化ポイントにおけるスナップショットを読み出している。
7. で終了する $Scan$ 命令については P_k の内部 $Scan$ 命令が 5. で終了する場合、線形化ポイントはこのスナップショットの直後としているので、これも線形化ポイントでのスナップショットとなっている。
内部 $Scan$ 命令が 7. で終了する場合も同様の議論を繰り返すといずれ 5. で終了した内部 $Scan$ 命令が現れ、すべての $Scan$ 命令で返すスナップショットはそれらの線形化ポイントでのスナップショットとなることが示せる。

6.4 無待機合意アルゴリズム

無待機合意問題

- 初期値の全集合 $B = \{0, 1\}$
- 各プロセス P_i は初期値と合意値を格納する局所変数 dP_i と wP_i をもつ
- 各プロセスは wP_i に最大1回だけある値を代入する

1. （**一様合意性**）すべてのプロセス P_i は合意値を持つならば、同じ値を合意値として持つ
2. （**無待機性**）すべての正常プロセスは有限個のステップ後に値を決定する
3. （**妥当性**）合意値は常にあるプロセスの初期値から選択される

- 型 T の共有オブジェクトと共有レジスタのみを用いて解くことが出来る無待機問題の最大プロセス数を、型 T のオブジェクトの**コンセンサス数**と呼ぶ

コンセンサス数

定理

型 T_1 のオブジェクトのコンセンサス数を x、型 T_2 のオブジェクトのコンセンサス数を y とし、$x < y$ とする。このとき、$n(> x)$ 個のプロセスで構成される分散システムにおいて、型 T_1 の共有オブジェクトと共有レジスタのみを用いて、型 T_2 の共有オブジェクトを構成する無待機アルゴリズムは存在しない

証明

T_2 を構成できると仮定すると、構成して T_2 無待機アルゴリズムを実行すれば、$\min(n, y)$ $(> x)$ 個のプロセスの無待機合意問題が解けることになる。

コンセンサス数

定理

型 T_1 のオブジェクトのコンセンサス数を x、型 T_2 のオブジェクトのコンセンサス数を y とし、$x \geq y$ とする。このとき、$n(\leq x)$ 個のプロセスで構成される分散システムにおいて、型 T_1 の共有オブジェクトと共有レジスタのみを用いて、型 T_2 の共有オブジェクトを構成する無待機アルゴリズムが存在する

共有オブジェクトがレジスタのみ

定理

共有オブジェクトとしてレジスタだけが使用可能な場合、プロセス数を 2 に限定しても、無待機合意問題は解けない

証明

- P_1 の初期値が 0、P_2 の初期値が 1 の状況 C_0 は、未決定初期大域状態
- 次にプロセス P_1 が動作すると 0-大域状態になり、プロセス P_2 が動作すると 1-大域状態になるような大域状態 C が存在する。ここで、P_1 の動作を e、P_2 の動作を f と表す

1. e, f が異なるレジスタに対する操作
    - e, f はどの順でもおこりえて、結果は変わらない
2. e, f が同じレジスタに対する操作でいずれかが $Read$
    - e が $Read$ とする。e の前後で P_2 とレジスタの状態は変わらないので f が実行できるが、このとき P_1 が停止すると e が先に実行されていたかわからない
3. e, f が同じレジスタに対する $Write$
    - e の後、f が実行されると、レジスタは上書きされる。P_2 とレジスタの状態は f が先に実行された場合と変わらないが、このとき P_1 が停止すると e が先に実行されていたかわからない

共有オブジェクトがキュー

定理

FIFO キューのコンセンサス数は 2 である

2-CONS-FIFO

Q は FIFO キューで初期値は $\{1, 0\}$

P_i 上のアルゴリズム

1. $R_i \leftarrow dP_i$
2. `if` $Dequeue(Q) = 1$ `then`
    - $wP_i \leftarrow dP_i$
3. `else`
    - $wP_i \leftarrow R_{\bar{i}}$

- ただし、$R_{\bar{1}} = R_2, R_{\bar{2}} = R_1$

共有オブジェクトがキュー

共有 FIFO キューでは 3 プロセス間の無待機合意問題が解けないことを示す。

- 未決定初期大域状態が存在する
- 次にプロセス P_1 が動作すると 0-大域状態になり、プロセス P_2 が動作すると 1-大域状態になるような大域状態 C が存在する。ここで、P_1 の動作を e、P_2 の動作を f と表す

1. e, f が共に $Dequeue$
    - P_3 の状態とキューの状態は変わらないので P_1, P_2 が停止すると P_3 はどちらが先に実行されていたかわからない

1. e, f の一方が $Dequeue$ で他方が $Enqueue$
    - キューが空でない場合、P_3 の状態とキューの状態は変わらないので P_1, P_2 が停止すると P_3 はどちらが先に実行されていたかわからない
    - キューが空の場合、e が $Read$ とする。e, f の順に実行された場合と f のみ実行された場合で、P_3 の状態とキューの状態は変わらないので P_1, P_2 が停止すると P_3 はどちらかわからない
2. e, f が共に $Enqueue$

ComPare & Swap

$CompSwap(CS, old_v, new_v)$：$CS$ が格納する値を読み出し、$CS = old\_v$ なら CS を new_v にする。

1. $x \leftarrow CompSwap(CS, -1, dP_i)$
2. `if` $x = -1$ `then`
    - $wP_i \leftarrow dP_i$
3. `else`
    - $wP_i \leftarrow x$

ComPare&Swap オブジェクトのコンセンサス数は ∞