NAIST-IS-MT1251077

修士論文

オブリビアス敵対スケジューラ下での**MRSW**レジスタを用いた乱択合意アルゴリズム

中島　悟

2014年3月7日

奈良先端科学技術大学院大学
情報科学研究科 情報科学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士 (工学) 授与の要件として提出した修士論文である。

中島　悟

審査委員：

井上　美智子 教授　（主指導教員）
伊藤　実 教授　（副指導教員）
米田　友和 助教　（副指導教員）
大和　勇太 助教　（副指導教員）

オブリビアス敵対スケジューラ下での MRSW レジスタを用いた乱択合意アルゴリズム*

中島　悟

内容梗概

本研究では, 共有メモリ分散システム上において, オブリビアス敵対スケジューラ下で, 全ての正常プロセスが一つの値に合意をとる乱択合意アルゴリズムを提案する. オブリビアス敵対スケジューラは, 各プロセスの状態を観測せずにプロセスの実行順番をスケジュールする. 乱択合意アルゴリズムとして, $1-\epsilon(0<\epsilon<1)$ の確率で合意をとる Conciliator アルゴリズムと合意がとれたことを検出する Adopt-Commit オブジェクトから構成する手法が知られている. アルゴリズムは, アルゴリズムが終了するのにかかるラウンド数とアルゴリズムの終了までに各プロセスが行った共有メモリへの演算数であるプロセスステップ計算量で評価される. 上記の乱択合意アルゴリズムにかかるプロセスステップ計算量は, Conciliator アルゴリズムと Adopt-Commit オブジェクトのそれぞれの実現にかかるプロセスステップ計算量の和と漸近的に等しい. 本研究では, Multi-Reader Single-Writer(MRSW) レジスタを用いて, Conciliator アルゴリズム, および Adopt-Commit オブジェクトの実現アルゴリズムを提案する.

計算量の解析では, 提案する MRSW レジスタを用いた Conciliator アルゴリズムは, 合意確率 $1-\epsilon(0<\epsilon<1)$ を保証するのに, ラウンド数, プロセスステップ計算量の上界が, それぞれ, $O(\log\log n+\log(1/\epsilon))$, $O(n\log\log n+n\log(1/\epsilon))$ であることを示す. この上界は, 既存アルゴリズムの計算量と漸近的に等しい. ここで, n は全プロセス数である. もう一方の, 提案する MRSW レジスタを用いた Adopt-Commit

*奈良先端科学技術大学院大学 情報科学研究科 情報科学専攻 修士論文, NAIST-IS-MT1251077, 2014 年 3 月 7 日.

アルゴリズムは, プロセスステップ計算量が $O(n)$ であり, 既存手法のプロセスステップ計算量の $\Theta(\min(n\frac{\log m}{\log\log m}, n^2))$ に比べ, 時間計算量を改善する. ここで, m はアルゴリズムの入力値の値域の大きさである. 二つの提案アルゴリズムから, 提案する乱択合意アルゴリズムのプロセスステップ計算量の期待値は, $O(n\log\log n)$ である. 既存の乱択合意アルゴリズムにかかるプロセスステップ計算量の期待値は $O\left(n\log\log n + \min\left(n\frac{\log m}{\log\log m}, n^2\right)\right)$ であり, アルゴリズムの入力値の値域の大きさ m が $(\log n)^{O(\log\log\log n)}$ を超えるとき, 提案手法は, 既存手法と比べ, プロセスステップ計算量が改善されている. それ以外の m では, 提案手法は既存手法と漸近的に等しい.

さらに, Conciliator アルゴリズムについては, 計算機実験による時間計算量の評価を行った. 実験結果から, 提案したアルゴリズムは, 実験した全てのプロセス数で, 任意の合意確率において, 既存手法と比べて時間計算量が少なく済むことがわかった. 特に, プロセス数 $n = 2048$ の場合, 提案したアルゴリズムは合意確率が 1.6%以上において, 既存手法よりも 1 ラウンド早く到達することがわかった.

さらに, 提案した Conciliator アルゴリズム中で, 各プロセスが write か read を選択するステップにおいて, write を選択する確率として, 計算機実験によって求めた最適値を使用する手法の提案を行う. この手法と, 解析で求めた write 確率を使用する場合の Conciliator アルゴリズムに対して, 計算機実験による比較評価を行った. 実験結果から, 最適な write 確率を使用した手法のほうが, 任意の合意確率において, 時間計算量が優れていることがわかった. 特に, プロセス数 $n = 2048$ の場合, 合意確率が 80%以上において, 最適な write 確率を使用するほうが 1 ラウンド早く到達することがわかった.

キーワード

分散アルゴリズム, MRSW レジスタモデル, 乱択合意問題, オブリビアス敵対スケジューラ,

Randomized consensus algorithm using MRSW registers under oblivious adversary*

Satoru Nakajima

Abstract

In this study, we propose an efficient randomized consensus algorithm that solves the randomized consensus problem where every correct process reaches agreement on a common value in distributed Multi-Reader Single-Writer (MRSW) shared register model under an oblivious adversary. An oblivious adversary decides a schedule of processes without checking processes' states. The randomized consensus algorithm is known to be constructed by a Conciliator algorithm and an Adopt-Commit object. The Conciliator algorithm solves the problem that every correct process reaches agreement with probability at least $1 - \epsilon (0 < \epsilon < 1)$ and the Adopt-Commit object detects agreement. An algorithm is evaluated by individual step complexity that is the number of steps each process takes in an execution of the algorithm. A Conciliator algorithm is also evaluated by the number of rounds that are building blocks of the algorithm. The expected individual step complexity of the randomized consensus algorithm above is asymptotically equal to the sum of the individual step complexities of the Conciliator and the Adopt-Commit object. We propose a Conciliator algorithm and an Adopt-Commit object implementation algorithm using MRSW registers.

In theoretical analysis, the proposed Conciliator algorithm reaches agreement with agreement probability $1 - \epsilon$ with $O(\log \log n + \log(1/\epsilon))$ rounds and $O(n \log \log n + n \log(1/\epsilon))$ individual step complexity. n is the number of processes. These upper

*Master's Thesis, Department of Information Science, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT1251077, March 7, 2014.

bounds of the complexities are asymptotically equal to exiting algorithms. On the other hand, the proposed Adopt-Commit algorithm takes individual step complexity of $O(n)$ and improves the best known individual step complexity of $\Theta(\min(n\frac{\log m}{\log\log m}, n^2))$, where m is input values. From two proposed algorithms, we can construct a proposed randomized consensus algorithm with $O(n\log\log n)$ expected individual step complexity. This algorithm improves the best known individual step complexity of $O\left(n\log\log n + \min\left(n\frac{\log m}{\log\log m}, n^2\right)\right)$ when the range m is larger than $(\log n)^{O(\log\log\log n)}$. In the range m except it, the proposed algorithm is asymptotically equal to exiting algorithms.

We also evaluate the time complexity of proposed Conciliator algorithm by experiments. Experimental results show that The time complexity of the proposed Conciliator algorithm is more effective than the conventional method in any agreement probability and all of the number of process that we experimented on. In particular, in the case of $n = 2048$, the proposed algorithm arrives at least one round early in comparison with the conventional method when agreement probability is greater than or equal to 16%.

Furthermore, we propose a method to choose an optimal write probability with simulation for our Conciliator algorithm, where there is a step in which each process decides a write or read operation according to the write probability. Experimental results show that the time complexity of this method is more effective in any agreement probability and all of the number of process that we experimented on. In particular, in the case of $n = 2048$, this method arrives at least one round early in comparison with the proposed algorithm using the write probability from the analysis when agreement probability is greater than or equal to 80%.

Keywords:

MRSW register model, randomized consensus problem, oblivious adversary scheduler

目 次

図 目 次

表 目 次

1. はじめに

分散システムにおける基本的な問題の一つに合意問題がある. 合意問題は次のような問題である:各プロセスは一つの初期値を持ち, 全てのプロセスは初期値の中から, ただ一つの値に合意をとる. 合意問題は, 分散システムをモデル化した非同期メッセージパッシングモデル, 非同期アトミック共有レジスタモデルの両方ともに, プロセス集合のうち, 一つのプロセスでも故障すると解くことが不可能な問題であることがそれぞれ, Fischer ら [1], Herlihy[2] により証明されている. しかしながら, プロセスが確率的に選択を行うことにより, 確率的に実行を終了する乱択合意問題であれば, 解くことが可能であることが知られている [3][4][5].

本研究では, 計算モデルとして, プロセスが故障する可能性のある非同期アトミック共有レジスタモデルを対象とする. 本論文で扱う非同期アトミック共有レジスタモデルは, 各レジスタに対し全プロセスが読み込み, 書き込みが可能である Multi-Reader Multi-Writer(MRMW) レジスタモデルと, 全プロセスが読み込み可能だが, 書き込みは特定の一つのプロセスのみである Multi-Reader Single-Writer(MRSW) レジスタモデルの二つである. また, プロセスの故障としては, 故障したプロセスはステップの実行を行わなくなる停止故障を想定する.

非同期アトミック共有レジスタモデル上での乱択合意アルゴリズムの評価として, アルゴリズムが終了するまでに全プロセスが行う共有レジスタへの演算数である総ステップ計算量, または総ステップ計算量の期待値, アルゴリズムが終了するまでに各プロセスが行う共有レジスタへの演算数であるプロセスステップ計算量, またはプロセスステップ計算量の期待値および, 共通の値で合意する合意確率を用いる.

乱択合意問題を解くのに必要な総ステップ計算量, プロセスステップ計算量や合意確率は, 敵対スケジューラの能力に強く影響される. 敵対スケジューラは, アルゴリズムの実行において, 次の演算を実行するプロセスを選択する. 敵対スケジューラには, アダプティブ敵対スケジューラ, ロケーションオブリビアス敵対スケジューラ, コンテンツオブリビアス敵対スケジューラ, オブリビアス敵対スケジューラなどいくつかのクラスが研究されている [3][8] [6] [7]. 敵対スケジューラには, スケジューラに与えられる情報量の差による強弱関係がある. ある敵対

スケジューラ S の環境下で合意問題を解くアルゴリズムは, S よりも弱い敵対スケジューラ S' の環境下でも合意問題を解くことが可能である. システム全ての状態を観測してプロセスの選択を行うアダプティブ敵対スケジューラは, コンテンツオブリビアス敵対スケジューラ, オブリビアス敵対スケジューラの両方ともに対して厳密に強いスケジューラであるということが証明されている [6]. また, オブリビアス敵対スケジューラは, 実行するアルゴリズムと各プロセスの初期値は知っているが, 各プロセスの確率的に選択した処理や各プロセスの内部変数などの実行状況を観測することなく, 前もってプロセスの実行順番をスケジュールする. そのため, 他の敵対スケジューラよりも弱い敵対スケジューラである.

現在までに, 各レジスタモデル, 敵対スケジューラに対して, 提案されているアルゴリズムを表 1 に示す. ここで, n はプロセス数, m は乱択合意問題における初期値の値域の大きさを表す. 既存のアルゴリズムには, 入力値として, 2 値を扱うものと, 任意の m に対する m 値を扱うものがある. また, 共有レジスタとして, MRMW レジスタを扱うものと MRSW レジスタを扱うものがある.

表 1 乱択合意アルゴリズム

					MRSW レジスタモデル上での m 値の乱択合意問題	
論文	問題の種類 / レジスタモデル	敵対スケジューラ	プロセスステップ 計算量	総ステップ 計算量	プロセスステップ 計算量	総ステップ 計算量
[4]	2 値 / MRMW	アダプティブ	$O(n)$		$O(n^2 \min(\log m, \log n))$	
[10]	2 値 / MRMW	アダプティブ	$\Omega(n/\log n)$			
[3]	2 値 / MRMW	アダプティブ		$\Theta(n^2)$		$O(n^3 \min(\log m, \log n))$
[8]	m 値 / MRMW	ロケーションオブリビアス	$O(\log m + \log n)$	$O(n \log m)$	$O(n \log n)$	$O(n^2 \log m)$
[5]	m 値 / MRMW	オブリビアス	$O(\log\log n + \min(\log m/\log\log m, n))$		$O(n \log\log n + \min(n \log m/\log\log m, n^2))$	
[9]	2 値 / MRSW	アダプティブ	$\Theta(n)$	$\Theta(n^2)$	$O(n \min(\log m, \log n))$	$O(n^2 \min(\log m, \log n))$
[6]	2 値 / MRSW	コンテンツオブリビアス	$\Theta(n)$	$O(n\log^4 n)$	$O(n \min(\log m, \log n))$	$O(n \log^4 n \min(\log m, \log n))$
本論文	m 値 / MRSW	オブリビアス	$O(n \log\log n)$		$O(n \log\log n)$	

MRMW レジスタを用いた乱択合意アルゴリズムに関しては, 2 値の場合, 最適な総ステップ計算量が知られている [3]. また, MRSW レジスタを用いる場合, アダプティブ, コンテンツオブリビアススケジューラに対し最適なプロセスステップ計算量が知られている [9][6].

表 1 の右 2 列に, 各アルゴリズムを用いて, m 値の乱択合意問題を解く MRSW レジスタモデルでのアルゴリズムを実現する場合の計算量を示す. これらの計算量

は, 以下のように求めることができる. 2 値のアルゴリズムから m 値のアルゴリズムの構成法が, [14, p.329] で提案されており, 2 値のアルゴリズムの実行を $O(\log m)$ 回繰り返すことで, m 値の乱択合意アルゴリズムを得ることができる. また, $m > n$ の場合は, プロセス ID で合意を取り, そのプロセスが提案する値を出力すればよいので, m 値の乱択合意問題は 2 値の乱択合意アルゴリズムの $O(\min(\log m, \log n))$ 倍のプロセスステップ計算量で解くことができる. また, MRMW レジスタモデルの各 $O(1)$ の演算は, MRSW レジスタモデル上で考えたとき, $O(n)$ のプロセスステップ計算量で実現できることが知られている [12].

本研究では, 最適なプロセスステップ計算量がまだわかっていないオブリビアス敵対スケジューラを対象として, MRSW レジスタモデルにおける乱択合意アルゴリズムの提案を行う.

オブリビアス敵対スケジューラを対象とした既存手法として, Aspnes[5] は, 確率的に全プロセスで一つの値に合意をとることができる Conciliator アルゴリズムと, 合意をとったことを検出できる Adopt-Commit オブジェクト [9] を組み合わせるた乱択合意アルゴリズムを提案している. [5] の構成法では, 乱択合意問題を, Conciliator アルゴリズムと Adopt-Commit オブジェクトのそれぞれにかかるプロセスステップ計算量の和と漸近的に等しいプロセスステップ計算量の期待値で解くことができる. [5] で提案されている乱択合意アルゴリズムにかかるプロセスステップ計算量の期待値は, $O\left(n \log\log n + min\left(n\frac{\log m}{\log\log m}, n^2\right)\right)$ である. 本研究では, [5] と同様の構成法を使用し, Conciliator アルゴリズムと Adopt-Commit オブジェクトの実現アルゴリズムを新規に提案し, 乱択合意アルゴリズムの提案を行う.

Conciliator アルゴリズムとは, 妥当性, 停止性, 確率的合意性を満足するアルゴリズムである. Aspnes は, [5] において, Multi-Reader Multi-Writer(MRMW) レジスタモデル上, オブリビアス敵対スケジューラ下で, 合意確率 $1 - \epsilon (0 < \epsilon < 1)$ の Conciliator アルゴリズムを $O(\log\log n + \log(1/\epsilon))$ のプロセスステップ計算量で実現している. このアルゴリズムを MRSW レジスタモデル上で実現した場合のプロセスステップ計算量は $O(n \log\log n + n \log(1/\epsilon))$ で実現できる. 本論文で提案する Conciliator アルゴリズムは, MRSW レジスタモデルにおいて, プロセスステップ計算量の上界が $O(n \log\log n + n \log(1/\epsilon))$ であることを示す. このプロセスス

テップ計算量の上界は, 既存手法と漸近的に等しい. Conciliator アルゴリズムに関しては, 計算機実験を用いての時間計算量の評価も行っている.

Adopt-Commit オブジェクトとは, 妥当性, 停止性, 一貫性, 収束性を満足する adopt-commit 演算のみを備えるオブジェクトである. Aspnes は, [9] において, Multi-Reader Multi-Writer(MRMW) レジスタモデル上, オブリビアス敵対スケジューラ下で, $O\left(min\left(\frac{\log m}{\log\log m}, n\right)\right)$ のプロセスステップ計算量で Adopt-Commit オブジェクトを実現している. ここで, m は Adopt-Commit オブジェクトへの入力値の値域の大きさを表す. この実現方法を MRSW レジスタモデル上で考えた場合, $O\left(min\left(n\frac{\log m}{\log\log m}, n^2\right)\right)$ で実現できる. 本論文で提案する Adopt-Commit アルゴリズムは, MRSW レジスタモデルにおいて, プロセスステップ計算量が $O(n)$ である Adopt-Commit オブジェクトを実現するアルゴリズムである. よって, 提案手法は, 既存手法に対して, プロセスステップ計算量が優れている.

二つの提案したアルゴリズムから構成される, 乱択合意アルゴリズムのプロセスステップ計算量の期待値は, $O(n\log\log n)$ となる. よって, アルゴリズムの入力値の値域 m が $(\log n)^{O(\log\log\log n)}$ を超える場合, 提案した乱択合意アルゴリズムは, 既存手法よりも高速なプロセスステップ計算量となる. それ以外の m においては, 既存手法と漸近的に等しい.

計算機実験を用いた Conciliator アルゴリズムの時間計算量の評価では, 提案アルゴリズムと [5] のアルゴリズムを対象として比較実験を行い, 提案アルゴリズムが総ステップ計算量, 合意確率の点で優れていることを示す.

さらに, 提案した Conciliator アルゴリズム中で, 各プロセスが write か read を選択するステップにおいて, write を選択する確率として, 計算機実験によって求めた最適値を使用する手法の提案を行う. この手法においても, 解析で求めた write 確率を使用した MRSW レジスタを用いた Conciliator アルゴリズムと計算機実験による比較評価を行った. 実験の結果として, この手法のほうが総ステップ計算量, 合意確率の点で優れていることを示す.

本稿の構成は以下のとおりである. 2 節では本研究で扱う計算モデルおよび, 合意問題などについて定義する. 3 節では Conciliator アルゴリズムの実現アルゴリズムを示す. 4 節では Adopt-Commit オブジェクトの実現アルゴリズムを示す. 5

節では Conciliator アルゴリズムと Adopt-Commit オブジェクトを用いた乱択合意アルゴリズムについて示す. 6 節では, Conciliator アルゴリズムに関して, 提案手法を既存手法との比較実験により, 総ステップ計算量, 合意確率について評価するとともに, 計算機実験を利用した手法について示す. 最後に, 7 節で本研究をまとめる.

2. 諸定義

本研究で対象とするモデルや合意問題などの定義を示す.

2.1 計算モデル

分散システムを表現する計算モデルには, メッセージパッシングモデルや共有メモリモデルなど, いくつかの種類が存在する. 本研究で対象とする計算モデルは, 故障する可能性がある n 個の非同期プロセスと複数の共有メモリオブジェクトからなる非同期共有メモリモデル [12] であり, プロセスは共有メモリオブジェクトへの演算の実行によりプロセス間通信を行う. プロセスの故障としては, 故障したプロセスはステップの実行を行わなくなる停止故障を想定する. このモデル上では, 共有メモリオブジェクトに対する演算がアトミックに行われる場合, そのオブジェクトをアトミック共有メモリオブジェクトと呼ぶ.

本研究で扱うアトミック共有メモリオブジェクトは, アトミック MRMW レジスタとアトミック MRSW レジスタの二つである. どちらも write 演算と read 演算の二つを備えている. MRMW レジスタは, システム内のどのプロセスも read, write 可能なレジスタである. それに対して, MRSW レジスタは, read はどのプロセスでも可能であるが, write は決められた特定の一つのプロセスのみが可能なレジスタである. レジスタのサイズに関しては, 任意のサイズを想定する. また, その他のオブジェクトに対する演算は, 上記の各アトミックレジスタを用いて実現している.

あるアルゴリズムが終了するまでに各プロセスが行う共有レジスタへの演算数を, そのアルゴリズムのプロセスステップ計算量と呼ぶ. また, あるアルゴリズムが終了するまでに全プロセスが行う共有レジスタへの演算数を総ステップ計算量と呼ぶ.

各プロセス p の MRMW レジスタへの read 演算, write 演算は, それぞれ, プロセス p の MRSW レジスタへの $O(n)$ 個の演算で実現できる [12]. よって, MRMW レジスタを用いた総ステップ計算量 N_t, プロセスステップ計算量 N_p のアルゴリズムは, MRSW レジスタを用いて総ステップ計算量 $O(nN_t)$, プロセスステップ計算量 $O(nN_p)$ で実現できる.

2.2 敵対スケジューラ

各プロセスの実行のタイミングは, 敵対スケジューラが生成するプロセスの識別子の系列からなるスケジュールに従う. 本研究では, オブリビアス敵対スケジューラを対象とする [5]. オブリビアス敵対スケジューラは, アルゴリズムのコードと各プロセスの初期値は知っているが, そのアルゴリズムの実行内容とは無関係に実行を行うプロセスの順番を表すプロセス集合を生成する. あるプロセスがアルゴリズムの実行を終了しても, スケジューラに指定されることがあるが, この場合, 指定されたプロセスは何も実行しないと考える.

2.3 合意問題

合意問題とは, 各プロセスが一つの初期値を持ち, 最終的に次の三つの性質を満たす, 共通の一つの合意値を全プロセスが出力する問題である.

- 合意性　全ての正常プロセスは同じ値を合意値とする.
- 停止性　全ての正常プロセスは, 有限の演算の実行数で, 合意値を決定する.
- 妥当性　合意値はあるプロセスの初期値から選択される.

合意問題は, 分散システムが非同期である場合, メッセージパッシングモデル, 共有レジスタモデルの両方ともに, プロセス集合のうち, 一つのプロセスでも故障すると解くことが不可能な問題であることが証明されている [1][2].

上記の不可能性から, 非同期分散システムにおいて合意をとる場合, 合意問題の性質の一部を確率的に保証する乱択合意問題が考えられる.

2.4 乱択合意問題

乱択合意問題とは, 合意性, 妥当性を保持し, 確率的停止性を満足する問題である.

- 確率的停止性　任意の正常プロセスに対し, k ステップ実行後に合意値を決定し停止する確率を $Pr[k]$ とするとき, $\lim_{k\to\infty} \Sigma_{i=1}^{k} Pr[k] = 1$ である.

Aspnes[5] は, 乱択合意問題を, 確率的に全プロセスで一つの値に合意をとることができる Conciliator アルゴリズムと, 合意をとったことを検出できる Adopt-Commit オブジェクトを組み合わせることで解けることを証明している. 具体的には, 定数確率で合意をとる Conciliator アルゴリズムを最初に実行し, その結果である各プロセスの合意値を Adopt-Commit オブジェクトに入力する. Conciliator アルゴリズムの結果として, 全てのプロセスが同じ値に合意をとっていた場合, 各プロセスは Adopt-Commit オブジェクトからの返り値として, 合意がとれていることを知ることができ, 乱択合意アルゴリズムの出力としてその合意値を出力する. 合意がとれていない場合, Adopt-Commit オブジェクトから返ってきた値を各プロセスは採用し, 再度 Conciliator アルゴリズムを実行する. この繰り返しによって乱択合意アルゴリズムを実現する. その構成法による乱択合意アルゴリズムにかかるプロセスステップ計算量の期待値は, Conciliator アルゴリズムと Adopt-Commit オブジェクトのそれぞれにかかるプロセスステップ計算量の和と漸近的に等しい.

2.5 Conciliator アルゴリズム

Conciliator アルゴリズムは, 確率的合意性, 停止性, 妥当性を満足する.

- 確率的合意性　ある $\delta > 0$ が存在し, 全ての正常プロセスが同じ値を合意する確率が少なくとも δ である.

Conciliator アルゴリズムは合意を形成するが, 合意がとれることを必ず保証こと も検出することもできない.

2.6 Adopt-Commit オブジェクト

Adopt-Commit オブジェクト [9] は, 合意を検出するオブジェクトである. Adopt-Commit オブジェクトは, 一つの演算 adopt-commit(v) のみを備えている. 各プロセ

スは, 高々一回 adopt-commit(v) を実行する. その演算は, ある値 v を入力値とし, 次の四つの性質を満たす, ある出力値 v' に対する ($commit, v'$) もしくは ($adopt, v'$) を返り値として返す. ここで, 一つ目の成分は判定ビットであり, プロセスが v' に決定を行う ($commit$) か, v' を採用する ($adopt$) かを表す.

- 妥当性　全ての演算の出力値 v' は, ある演算の入力値 v から選択される.
- 停止性　全ての演算は, 有限の時間で終了する.
- 一貫性　ある演算が ($commit, v$) を返したならば, 全ての演算は同じ出力値 v を持つ ($adopt, v$) もしくは, ($commit, v$) を返す.
- 収束性　全ての入力値が v ならば, 全ての演算の返り値は ($commit, v$) である.

文献 [9] において, Adopt-Commit オブジェクトを, 入力された値の中で衝突が起こっているかを検出する m 値の Weak Confilict-Detector オブジェクトを用いて実現するアルゴリズムが提案されている. ここで, m はオブジェクトへ入力される値の値域の大きさである. この Adopt-Commit アルゴリズムは, Weak Conflict-Detector オブジェクトの実現にかかるプロセスステップ計算量と定数回の共有メモリへの演算で実現できる.

2.6.1 Weak Conflict-Detecotr オブジェクト

Weak Conflict-Detector オブジェクトは, check(v) 演算のみを備えている. その演算は, 次の二つの性質を満たす, 入力値 v に対する $true$ もしくは $false$ を返す.

- $v \neq v'$ である check(v) と check(v') 演算を含んだ任意の実行において, それらの演算の少なくとも一つは $true$ を返す.
- 全ての check 演算が同じ入力値を持つような任意の実行において, それら全ては $false$ を返す.

返り値 $true$ は, 各プロセスが入力した値の間で衝突が発生したことを表す.

3. Conciliator アルゴリズム

Conciliator アルゴリズムは, 妥当性, 停止性, 確率的合意性を満たすアルゴリズムである. 本研究では, MRSW レジスタモデルにおいて, オブリビアス敵対スケジューラ下での Conciliator アルゴリズムを提案する.

文献 [5] で, MRMW レジスタモデルにおいて, オブリビアス敵対スケジューラ下での Conciliator アルゴリズムが提案されている. 本研究では, 提案する Conciliator アルゴリズムが, [5] の Conciliator アルゴリズムと同程度の時間計算量で実現できることを証明する. 3.1 節で, [5] の Conciliator アルゴリズムのアイディアとそれにかかるプロセスステップ計算量を示す. 3.2 節では, 本研究で提案する Conciliator アルゴリズムのアイディアとそれにかかるプロセスステップ計算量について示し, 既存手法との差異を示す. また, 各定理, 補題に関する証明についても示す.

3.1 MRMW レジスタを用いた Conciliator アルゴリズム [5]

図 1 に, MRMW レジスタモデルにおいて, 確率 $1 - \epsilon (0 < \epsilon < 1)$ で合意性を満たす Conciliator を実現するアルゴリズムを示す.

アルゴリズムの基本的なアイディアは, ラウンド R という系列を導入することである. ラウンドとは, プロセスが現在持っている値から, read もしくは write を一回だけ確率的に選択し, 自分の持つ値を決定する一連の処理を表す. ラウンドごとに各プロセスは提案値を持ち, 各ラウンド i に用意された共有レジスタ $r[i]$ を使用して合意をとることを試みる. 合意をとる方法として, 各ラウンドで, 各プロセスは自身の持つ値を共有レジスタに write することで提案するか, 提案されている値を共有レジスタから read するかを確率的に選択して一回だけ実行する.

プロセスが write を選択した場合, そのプロセスが持つ値は次のラウンドにも使用される. それに対して, read を選択した場合, 共有レジスタの中身が $\perp$ である場合のみ, そのプロセスの持つ値は次のラウンドにも使用される. レジスタから他のプロセスが提案した値を読み込んだ場合, 読み込んだ値を自身の提案値として採用し, それ以降のラウンドで使用する. 上記の処理を各ラウンドごとに繰り返し行う.

```
    shared data:
        registers r[1 ... R], initially ⊥;
    local data:
        l, initially ⊥; v, initially ⊥;
        Let R = ⌈log log n⌉ + ⌈log_{4/3}(8/ε)⌉
        Let chooseWrite be a vector of R independent random Boolean
        variables with Pr[chooseWrite[i] = 1] = p_i

1.  procedure ConciliatorMW(input)
2.  begin
3.      v ← input
4.      for i ← 1 ... R do
5.          if chooseWrite[i] = 1 then
6.              r[i] ← ⟨v, chooseWrite⟩
7.          else
8.              l ← r[i]
9.              if v ≠⊥ then
10.                 ⟨v, chooseWrite⟩ ← l
11.     end
12.     return v
13. end
```

図 1 MRMW レジスタアルゴリズム

write と read の処理から, ラウンドの初めに存在する各プロセスが持つ異なる提案値の個数は, 次のラウンドの初めまでに確率的に減少する. その結果, 有限のラウンド後で, ある確率で一つにまで異なる提案値の個数を減らし, 合意をとることが可能となる.

このような処理を行うために, 各プロセスはラウンドごとに write か read のどちらを実行するか決定する系列 chooseWrite を生成する. 本研究では, オブリビアス敵対スケジューラを想定しているため, 各プロセスの生成する chooseWrite とスケジュールは独立している. そのため, 各プロセスは, アルゴリズムの最初に chooseWrite を生成し, 初期値と結び付けておく. このことにより, read の実行により同じ値を保有するプロセスが発生した場合, 同じ値を持つプロセスは, それ以降のラウンドでの write, read の選択は全て同じ選択を行うことになる.

また, このアルゴリズムでは, 各プロセスは非同期にラウンドを進めていく. そ

のため, 各プロセスは自身のラウンドのみを知ることができ, 他のプロセスのラウンド数を気にすることなく, 自身のラウンド数が R に達した場合, ラウンド R での決定値を Conciliator アルゴリズムの決定値として出力する.

ラウンドごとに, 各プロセスが write か read を選択する確率を変えることで, 次のラウンドの異なる提案値の個数の期待値を効率的に減らしていくことができる. 具体的な各ラウンド i での write を選択する確率は, $p_i = 2^{1-2^{-i+1}}(n-1)^{-2^{-i}}$ である. n はプロセスの全数を表す. この確率で選択を行うことで, 各プロセスが $R = \lceil \log\log n \rceil + \lceil \log_{4/3}(8/\epsilon) \rceil$ のラウンド数を実行すれば, 合意確率 $1-\epsilon(0<\epsilon<1)$ の Conciliator アルゴリズムが実現できる.

文献 [5] の Conciliator アルゴリズムを, MRSW レジスタモデル上で実現することを考える. アルゴリズムの処理では, 各ラウンドで MRMW レジスタに一度 read もしくは write を行うので, MRSW レジスタモデル上では $O(n)$ のプロセスステップ計算量がかかる. そのため, Conciliator アルゴリズムを実現するためには $O(n\log\log n + n\log(1/\epsilon))$ のプロセスステップ計算量がかかる.

3.2 MRSW レジスタを用いた Conciliator アルゴリズム

3.1 節の Conciliator アルゴリズムでは, 各ラウンドにおいて, MRMW レジスタを用いているため, 各プロセスが read 演算によって読みだす値は, その read 演算の直前に write 演算を行ったプロセスの値となる. 本研究では, MRSW レジスタを用いて, read 演算によって, それ以前に行われた write 演算で書き込まれた値全てを読み出し, その中で最大の値を選択することで, 効率的に合意に近づけていくことを考える.

図 2 に, MRSW レジスタモデルにおいて, 確率 $1-\epsilon(0<\epsilon<1)$ で合意性を満たす Conciliator を実現するアルゴリズムを示す.

MRSW レジスタモデルでは, 各ラウンド i で, プロセス間の通信のために n 個のアトミック MRSW レジスタ $r[i]$ が使われる. また, 3.1 節のアルゴリズムと同様に, ラウンドの系列を導入する. 各ラウンド i で各プロセス j には, MRSW レジスタ $r[i][j]$ が割り当てられる.

```
shared data:
        MRSW registers r[1 ... R][1 ... n], initially ⊥;
local data:
        l[1 ... n], initially ⊥; v, initially ⊥;
        Let R = ⌈log log n⌉ + ⌈log_{4/3}(8/ϵ)⌉
        Let chooseWrite be a vector of R independent random Boolean
        variables with Pr[chooseWrite[i]= 1]= p_i

1.  procedure ConciliatorSW(input) for process x
2.  begin
3.      v ← input
4.      for i ← 1 ... R do
5.          if chooseWrite[i]= 1 then
6.              r[i][x] ← ⟨v, chooseWrite⟩
7.          else
8.              for j ← 1 ... n do
9.                  l[j] ← r[i][j]
10.             if ∃l[j] ≠⊥ then
11.                 ⟨v, chooseWrite⟩ ← max(l[j].v)
12.     return v
13. end
```

図 2 MRSW レジスタアルゴリズム (プロセス p_x に対するコード)

各ラウンド i で, 各プロセスは write もしくは read を確率的に選択し, 一回だけ実行する. 各プロセスは, 各ラウンドで write か read のどちらを選択するかを表す系列 chooseWrite をアルゴリズムの最初に生成する. write を選択したプロセスは, 自身に割り当てられたレジスタに提案値を書き込む. read を選択したプロセスは, 用意された全てのレジスタ $r[i]$ を読み込み, ローカル変数 l に保存する. そして, l のどれかに $\perp$ 以外の値が保存された場合, $\perp$ 以外の値の中で最大の値を自身の値として採用し, 結び付けられた chooseWrite とともに, それ以降のラウンドで使用する. 全てのレジスタの値が $\perp$ であった場合, 自身の持つ値をそのまま次のラウンドでも使用する.

図 2 のアルゴリズムが Conciliator の三つの性質を満足することと, 合意確率を $1-\epsilon$ としたとき, それを満足するのに必要なラウンド数が $O(\log\log n + \log(1/\epsilon))$ であることをそれぞれ証明する. MRMW レジスタを用いた Conciliator アルゴリズム [5] は, i ラウンド後に残る異なる提案値の個数 Y_i と余分な異なる提案値の個

数 $X_i = Y_i - 1$ に対し, $E[X_{(i+1)}|X_i] \leq \min\left(p_{i+1}X_i + \frac{1}{p_{i+1}}, \left(1 - p_{i+1} + p_{i+1}^2\right)X_i\right)$ が成り立つことを利用して, 各ラウンドでの write 確率, 必要なラウンド数を求めている. まず, 同じ不等式が提案アルゴリズムに対して成り立つことを示す.

補題 1. Y_i を図 2 のアルゴリズムの i ラウンド後に残っている異なる提案値の個数, $X_i = Y_i - 1$ を余分な異なる提案値の個数とする. そのとき,

$$E\left[X_{i+1}|X_i\right] \leq \min\left(p_{i+1}X_i + \frac{1}{p_{i+1}}, \left(1 - p_{i+1} + p_{i+1}^2\right)X_i\right)$$

が成り立つ.

証明 . X_i の値が大きいときには, *min* の最初のケースが有効となる. 最初の式を得るために, 初めに $E\left[Y_{i+1}|Y_i\right]$ を求め, それを利用して $E\left[X_{i+1}|X_i\right]$ を計算する.

i ラウンドで決定値として少なくとも一つのプロセスに選ばれた提案値が, $i+1$ ラウンドにおいて, 最初に次のタイミングのどちらかが生じた順番に並べた系列 $a_1, \ldots, a_{Y_i}$ を考える.

(I) 各提案値が, $r[i+1]$ のどれかに *write* されたタイミング

(II) *read* の結果として全ての l が $\perp$ のとき, その *read* 処理での最初の *MRSW* への読み込みが行われたタイミング

$i+1$ ラウンドでの各提案値とプロセスとの結び付きは, 1 から i ラウンドにおける *chooseWrite* と敵対スケジューラが生成するスケジュールから決定される. $i+1$ ラウンドにおける *chooseWrite* は, 1 から i ラウンドにおける *chooseWrite* と敵対スケジューラが生成するスケジュールの二つとは独立している.

各提案値 a_j において, 上記のタイミングが起こり, ラウンド $i+1$ で採用されるのは次の場合のみである.

(a) a_j を持つあるプロセスが, それに結びづけられた *chooseWrite*$[i+1]$ で 1 を見て, a_j を *write* した場合

(b) a_j を持つあるプロセスが, *chooseWrite*$[i+1]$ で 0 を見て *read* を行ったが, その結果 $\perp$ であった場合

a_j と $a_{j'}(j' < j)$ の全てにおいて, 結びづけられた *chooseWrite*$[i+1]$ が *0* である場合のみ, *(b)* は起こる.

上記の場合が起こる確率は, *(a)* は p_{i+1}, *(b)* は $(1-p_{i+1})^j$ 以下である. ここで, *(b)* が成り立つ j の集合を J_b とするとき, これらの確率を全ての j で合計すると, $E[Y_{i+1}|Y_i] \leq p_{i+1}Y_i + \sum_{j' \in J_c}(1-p_{i+1})^{j'}$ が得られる. ここで, $\sum_{j' \in J_c}(1-p_{i+1})^{j'} \leq \sum_{j=1}^{\infty}(1-p_{i+1})^j = 1/p_{i+1} - 1$ が成り立つ. この期待値に $X_{i+1} = Y_{i+1} - 1$ と $Y_i = X_i + 1$ を代入すると,

$$\begin{aligned} E[X_{i+1}|X_i] &\leq p_{i+1}(X_i+1) + 1/p_{i+1} - 2 \\ &= p_{i+1}X_i + 1/p_{i+1} + p_{i+1} - 2 \\ &< p_{i+1}X_i + 1/p_{i+1} \end{aligned}$$

となる

X_i が小さいとき, *min* の二つ目のケースが有効となる. この境界を求める際, 最初のプロセス q が *read* か *write* のどちらを選択したかによって場合分けを行って考える. q が *write* の場合, ラウンド $i+1$ で採用される値は, すべて *write* された値である. また, q が *read* を選択した場合は, 全ての提案値が消えることなく, 全て採用されるという単純な上界を考える. 上記で示した考え方から, 期待値は,

$$\begin{aligned} E[X_{i+1}|X_i] &= (1-p_{i+1})E[X_{i+1}|X_i, qreads] + p_{i+1}E[X_{i+1}|X_i, qwrites] \\ &\leq (1-p_{i+1})X_i + p_{i+1}^2 X_i \\ &= (1-p_{i+1}+p_{i+1}^2)X_i. \end{aligned}$$

となる. よって, 補題が成り立つ. □

補題 1 から, アルゴリズムの最初に存在する余分な異なる提案値の個数 X_0 としたとき, X_1 を最小とする write 確率 p_1 は, $p_1 = 1/\sqrt{X_0}$ である. これから, $E[X_1] \leq 2\sqrt{X_0} \leq \sqrt{n-1}$ が得られる. この手順を続くラウンドで再帰的に繰り返し行うことで, $x_0 = n-1$, $p_{i+1} = 1/\sqrt{x_i}$, $x_{i+1} = p_i x_i + 1/p_i = 2\sqrt{x_i}$ としたとき,

$$x_i = 2^{2-2^{-i+1}}(n-1)^{2^{-i}}$$

が得られる. また, 同様に,

$$p_i = 2^{-(1-2^{-i+1})}(n-1)^{-1/2^i}$$

が得られる.

補題 1, x_i, p_i から, MRMW レジスタに対する Conciliator アルゴリズム [5] と同様に以下の補題, 定理が成り立つ.

補題 2. X_i を図 *2* のアルゴリズムにおいて, $i = \lceil \log \log n \rceil$ ラウンド目まで p_i を *write* 確率として用いたときの, i ラウンド後に残っている異なる提案値の個数とする. その時, $1 \ldots \lceil \log \log n \rceil$ の全てのラウンドにおいて, $E[X_i] \leq x_i$ である.

補題 3. X_i を図 *2* のアルゴリズムにおいて, $i = \lceil \log \log n \rceil$ ラウンド目まで $p_i = 2^{-(1-2^{-i+1})}(n-1)^{-1/2^i}$, $i > \lceil \log \log n \rceil$ のラウンドでは $p_i = 1/2$ を *write* 確率として用いたときの, i ラウンド後に残っている異なる提案値の個数とする. また, $j > 0$ とする. そのとき, $E[X_{\lceil \log \log n \rceil + j}] \leq 8 \cdot (3/4)^j$ である.

定理 1. 図 *2* のアルゴリズムは, ラウンド数 $R = \lceil \log \log n \rceil + \lceil \log_{4/3}(8/\epsilon) \rceil$ とすると, 合意確率 $1 - \epsilon$ の *Conciliator* を実現できる.

証明 . 停止性に関しては, 図 *2* のアルゴリズムから, 必ず *return* が実行されるため, 保証される.

妥当性に関しては, 図 *2* のアルゴリズムから, 合意値として決定される値はアルゴリズムの返り値 v である. v は, アルゴリズムの最初に, 各プロセスの入力値が代入され, 各ラウンド i の処理では, *MRSW* レジスタ $r[i]$ に格納されている値のうち, 最大の値が v に代入される. $r[i]$ に格納されるのは各プロセスが持つ v の値のみなので, v に代入される可能性があるのは, 各プロセスが最初に持つ初期値のみである. よって, 妥当性も保証される.

最後に, 確率的合意性に関しては, 補題 *3* から, $R = \lceil \log \log n \rceil + \lceil \log_{4/3}(8/\epsilon) \rceil$ ラウンド後に残っている異なる提案値の個数 X の期待値は, 高々

$$8 \cdot (3/4)^{\log_{4/3} 8/\epsilon} = 8 \cdot (\epsilon/8) = \epsilon$$

となる. X は非負で整数の確率変数であるため, マルコフの不等式より,

$$Pr[X_{\lceil \log \log n \rceil + \lceil \log_{4/3}(8/\epsilon) \rceil} > 0] = Pr[X_{\lceil \log \log n \rceil + \lceil \log_{4/3}(8/\epsilon) \rceil} \geq 1] \leq \epsilon$$

が成り立つ. □

3.1 節と 3.2 節の結果から, MRSW レジスタを用いて Conciliator を実現する提案アルゴリズムにおいて, 合意確率 $1 - \epsilon$ を満足するのに必要なラウンド数の上界は, $O(\log\log n + \log(1/\epsilon))$ となり, MRMW レジスタを用いた Conciliator アルゴリズム [5] のラウンド数と漸近的に等しい. また, 提案アルゴリズムの 1 ラウンドにかかるプロセスステップ計算量は, 図 3.2 のアルゴリズムから, $O(n)$ となる. そのため, アルゴリズムの終了にかかるプロセスステップ計算量の上界は $O(n\log\log n + n\log(1/\epsilon))$ となり, MRMW レジスタを用いた Conciliator アルゴリズム [5] のプロセスステップ計算量と漸近的に等しい.

6 節では, 計算機実験により, 提案手法と既存手法の合意確率, 総ステップ計算量を比較する.

4. MRSWレジスタを用いたAdopt-Commitオブジェクト

Adopt-Commit オブジェクトは, 文献 [9] から, Weak Conflict-Detector オブジェクトと定数回の共有メモリへの演算で実現できることが証明されている. [9] の Adopt-Commit オブジェクトを実現するアルゴリズムを図 3 に示す.

```
    shared data:
            MRMWregisters proposal, initially ⊥;
            weak Conflict-Detector wCD;
            1-bit MRMW register conflict, initially ⊥;
    local data:
            v, initially ⊥; u, initially ⊥;

1.   procedure adopt-commit(input)
2.   begin
3.       v ← input
4.       if wCD.check(v) then
5.           conflict ← true
6.       u ← proposal
7.       if u ≠ ⊥ then
8.           v ← u
9.       else
10.          proposal← v
11.      if conflict = true then
12.          return (adopt, v)
13.      else
14.          return (commit, v)
15.  end
```

図 3 Weak Conflict-Detector オブジェクトを用いた Adopt-Commit アルゴリズム

あるプロセスに (*commit*, *v*) が返ってきた場合, その return が行われるまでに他の値 $v'(v \neq v')$ を持ったプロセスが 4 行目の check 演算を行うことはない. そのため, v' を持ったプロセスが 7 行目で, $u = \perp$ であることはない. その結果, 共有レジ

スタ proposal に v' が書き込まれることが無いため, v' を return として返すプロセスは存在しない. このため, $(commit, v)$ が返ってくる場合では, $(commit, v)$ もしくは $(adopt, v)$ のみが返ってくる. そのため, Adopt-Commit オブジェクトの一貫性を満足している. また, 全ての adopt-commit 演算で同じ値 v が入力された場合, weak Conflict-Detector オブジェクトで衝突が検出されないので, conflict が true になることが無いため, Adopt-Commit オブジェクトの収束性を満足している.

本研究では, 上記のアルゴリズムを基に, Weak Conflict-Detector オブジェクトの実現アルゴリズムを新規に提案することで, Adopt-Commit オブジェクトの実現を行う.

4.1 MRSW レジスタを用いた Weak Conflict-Detector オブジェクト

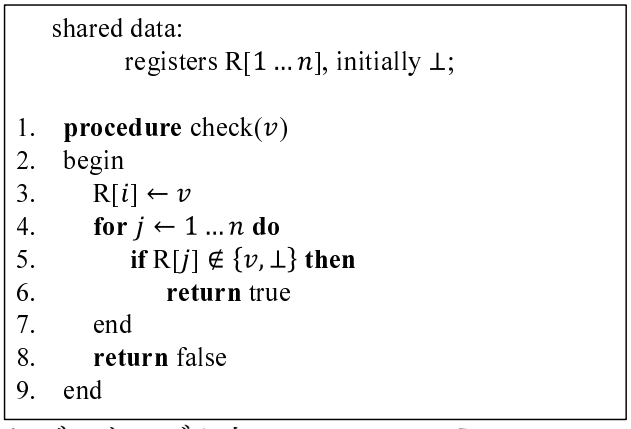
文献 [9] では, MRMW レジスタを用いて, m 値の Weak Conflict-Detector オブジェクトの備える check 演算を $\Theta(\min(\log m / \log\log m, n))$ で実現している. MRSW レジスタモデル上で, 文献 [9] のアルゴリズムを実現すると, check 演算の実行に $\Theta(\min(n \log m / \log\log m, n^2))$ のプロセスステップ計算量がかかる.

本研究では, MRSW レジスタを用いて, 任意の入力値の値域で実現可能な Weak Conflict-Detector をプロセスステップ計算量 $O(n)$ で実現するアルゴリズムを提案する. 図 4 に, MRSW レジスタモデル上での Weak Conflict-Detector オブジェクトの実現アルゴリズムを示す.

図 4 のアルゴリズムでは, 各プロセス i に MRSW レジスタ R[i] を割り当てる. check 演算では, プロセスは自身の割り当てられたレジスタに write を行い, その後に, 全てのレジスタに read を実行する. 図 4 から, 最悪の場合での各プロセスの check 演算のプロセスステップ計算量は $O(n)$ となる.

定理 2. 図 *4* は, *check* 演算のプロセスステップ計算量 $O(n)$ で *Weak Conflict-Detector* を実現するアルゴリズムである.

証明 . 全ての *check* 演算の入力値が同じ値 v の場合, レジスタ $R[i]$ には v のみが書き込まれる. *5* 行目で v もしくは $\perp$ 以外の値が R から読み込まれることはないため, 全ての演算の出力は $false$ が返ってくる.



```
    shared data:
          registers R[1 ... n], initially ⊥;

1.  procedure check(v)
2.  begin
3.      R[i] ← v
4.      for j ← 1 ... n do
5.          if R[j] ∉ {v, ⊥} then
6.              return true
7.      end
8.      return false
9.  end
```

図 4 MRSW レジスタモデル上での Weak Conflict-Detector 実現アルゴリズム

図 4 では, 各プロセスは自身のレジスタに *write* を行ってから全レジスタに *read* を行うので, 二つのプロセスが異なる値 $v \neq v'$ の *check(v)* 演算と *check(v′)* 演算を実行した場合, 先に *write* を行ったプロセスが書き込んだ値は, 後で *write* を行うプロセスが必ず *read* することになり, 値の衝突が起こったことを検出する. すなわち, *check(v)* 演算, *check(v′)* 演算 *(v ≠ v′)* のどちらかは *true* を返す. □

定理 2 から, MRSW レジスタモデル上では, 提案アルゴリズムによる Weak Conflict-Detector オブジェクトの実現にかかるプロセスステップ計算量は, 文献 [9] アルゴリズムよりも優れている.

5. Consiliator と Adopt-Commit を用いた乱択合意アルゴリズム

本研究では, MRSW レジスタを用いた乱択合意アルゴリズムの提案を行う. 文献 [5] において, Conciliator アルゴリズムと Adopt-Commit オブジェクトを使用することで, 乱択合意アルゴリズムを実現できることが示されている. 具体的には, 定数確率で合意をとる Conciliator アルゴリズムを実行し, その結果である各プロセスの合意値を Adopt-Commit オブジェクトの adopt-commit 演算の入力値として用いて, 演算の実行を行う. adopt-commit 演算の出力として $(commit, v)$ を得たプロセスは, v を合意値と決定しアルゴリズムを終了する. $(adopt, v)$ を得たプロセスは, 出力値 v を自分の提案値として採用し, 再度 Conciliator アルゴリズム, adopt-commit 演算を実行する. あるプロセスが $(commit, v)$ が返ってきた場合, その Adopt-Commit オブジェクトで返ってくる出力は $(commit, v)$ もしくは $(adopt, v)$ となるため, 他の値を合意値として決定し終了するプロセスは存在しない. 上記の処理を繰り返すことで, 乱択合意問題を解決することができる.

上記の乱択合意アルゴリズムにかかるプロセスステップ計算量の期待値は, Conciliator アルゴリズムのプロセスステップ計算量と Adopt-Commit オブジェクトを実現するプロセスステップ計算量の総和と漸近的に等しい. 文献 [5] で提案されている乱択合意アルゴリズムは, MRMW レジスタを用いて, $O(\log\log n + \log m / \log\log m)$ のプロセスステップ計算量の期待値で実現される. このアルゴリズムを MRSW レジスタモデル上で考えた場合, アルゴリズムの実現にかかるプロセスステップ計算量の期待値は, $O(n \log\log n + n \log m / \log\log m)$ である. 本研究で提案する乱択合意アルゴリズムは, 3 節と 4 節の結果から, MRSW レジスタを用いて, $O(n \log\log n)$ で実現することができる.

表 4 に, 各アルゴリズムに対する提案手法と既存手法それぞれのプロセスステップ計算量を示す.

提案手法と既存手法を比べると, アルゴリズムの入力値の値域 m が $(\log n)^{O(\log\log\log n)}$ を超える場合, 提案した乱択合意アルゴリズムは, 既存手法 [5] よりも, プロセスステップ計算量において優れている. m が $(\log n)^{O(\log\log\log n)}$ 以下の間では, 提案手法

表 2 各アルゴリズムの計算量 (乱択合意アルゴリズムでは計算量の期待値)

	既存手法	提案手法
Conciliator	$O(n \log\log n + n \log \frac{1}{\epsilon})$[5]	$O(n \log\log n + n \log \frac{1}{\epsilon})$
Adopt-Commit	$\theta\left(\min\left(n\frac{\log m}{\log\log m}, n^2\right)\right)$[9]	$O(n)$
乱択合意	$O\left(n \log\log n + \min\left(n\frac{\log m}{\log\log m}, n^2\right)\right)$[5]	$O(n \log\log n)$

と既存手法のプロセスステップ計算量は漸近的に等しい.

6. Conciliatorアルゴリズムに関する計算機実験

3節では, 提案したアルゴリズムの時間計算量の上界が, 既存手法と漸近的に等しいことを証明した. しかし, Conciliatorアルゴリズム中の各ラウンド処理において, 既存手法はread演算で読み込む値が, その演算の直前のwrite演算で書き込まれた値のみであるのに対して, 提案手法では, read演算の実行以前に行われたwrite演算全ての値を読み込むことが可能である. このことから, 各ラウンドにおいて, 合意がとれる確率が提案手法の方が高くなると考えられる. そのため, 提案手法は, ある合意確率を得るのに必要なラウンド数が既存手法よりも少なく済む可能性が考えられる. 6.1節では, 上記の考えを, 計算機を用いてアルゴリズムをシミュレーションすることで検証, 評価を行う.

また, 提案したConciliatorアルゴリズム中のwrite確率として, 3節で示したwrite確率 p_i ではなく, 計算機シミュレーションを用いて最適なwrite確率 p_{opti} を用いた手法の提案を行う. その手法も同様に, 計算機実験を用いて, p_i を使用する提案手法を比較対象として, 合意確率, 総ステップ計算量に関して比較評価を行う. 6.2節として, 提案手法の説明と評価実験の結果について示す.

6.1 提案手法と既存手法を比較した評価実験

評価実験では, 既存アルゴリズムとしては, 3.1節のMRMWレジスタを用いたConciliatorアルゴリズムをMRSWレジスタモデル上で実現する. 提案アルゴリズムは, 3.2節のMRSWレジスタを用いたConciliatorアルゴリズムを実現する. 提案アルゴリズムと既存アルゴリズムの中で使用する各ラウンド i でのwrite確率は3.2節で示した p_i を使用する. オブリビアス敵対スケジューラについては, 一様確率によるプロセスの選択を行うスケジューラとして実現した. 各プロセスが持つ初期値は, 0からプロセス数 n の10倍までの値域の乱数値とする. 評価実験の条件としては, プロセス数, ラウンド数を考慮する. また, 以降の結果では, アルゴリズムを3,000回試行した平均を用いる.

実験結果として, プロセス数を32, 128, 1024, 2048とし, 各ラウンド数を実行した場合の合意確率を表3, 図5,6,7,8に示す. また, 各ラウンド数を実行した場合の

アルゴリズムの総ステップ計算量を表 4, 図 9, 10, 11, 12 に示す.

表 3 提案と既存のアルゴリズムでの合意確率 (%)

プロセス数	32		128		1024		2048	
アルゴリズム	既存	提案	既存	提案	既存	提案	既存	提案
ラウンド数 1	0.0	1.4	0.0	0.1	0.0	0.0	0.0	0.0
2	2.3	30.3	1.1	21.3	0.0	7.2	0.0	3.2
3	10.3	54.9	6.1	52.7	1.3	42.0	1.6	37.6
4	20.9	73.1	10.8	71.6	6.8	65.6	6.1	62.4
5	31.1	84.2	21.6	83.2	16.0	81.1	14.9	79.6
6	45.5	90.4	36.0	90.6	28.7	88.9	27.7	90.4
7	55.6	94.2	46.5	94.6	40.0	94.4	39.0	94.6
8	65.7	96.4	58.6	97.2	54.2	96.9	49.9	96.4
9	75.1	97.9	70.1	98.2	62.7	98.2	60.5	97.8
10	80.9	98.2	75.0	99.0	70.3	99.3	70.3	99.0
11	84.8	99.1	81.9	98.7	80.1	99.5	77.0	99.8
12	89.4	99.5	84.1	99.6	83.7	99.6	80.1	100.0
13	93.1	99.6	90.2	99.8	88.7	99.7	87.0	100.0
14	93.8	99.6	91.5	99.8	89.7	99.9	89.4	99.8
15	95.3	99.8	94.2	99.8	91.3	100.0	93.7	100.0
16	95.6	99.9	94.4	100.0	94.1	100.0	94.9	100.0
17	97.5	100.0	97.3	100.0	95.6	99.9	96.4	100.0
18	98.2	99.9	97.4	100.0	96.9	99.9	96.6	100.0
19	97.9	100.0	97.5	100.0	97.9	100.0	98.8	100.0
20	99.2	100.0	98.6	100.0	98.4	100.0	98.9	100.0

表 3, 表 4 の結果から, どのプロセス数においても, ほとんどのラウンドで, 提案アルゴリズムが既存アルゴリズムの合意確率以上の確率で合意がとれており, より少ない計算量でアルゴリズムを終了していることがわかる. 合意確率において, 特定の確率を満足するのに必要なラウンド数を比較すると, 提案アルゴリズムと既存アルゴリズムに大きな差が生まれている. このことから, 合意確率を指定し

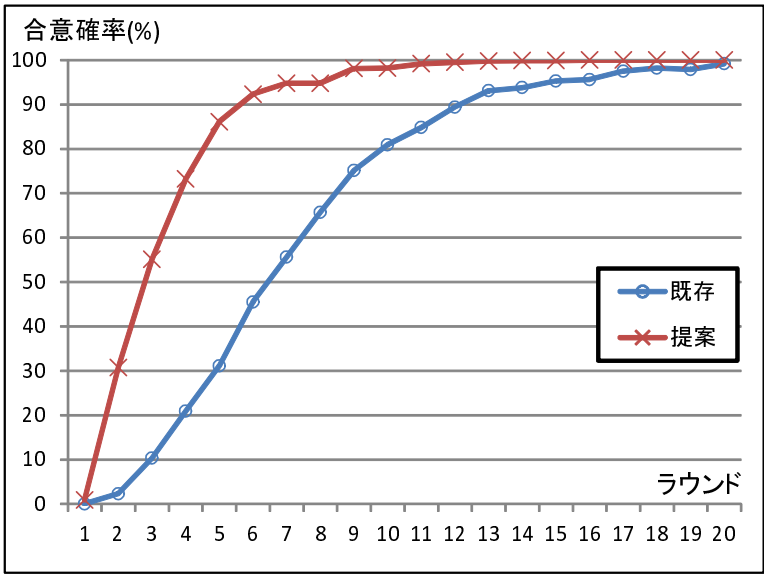


図 5 32 プロセスでの合意確率

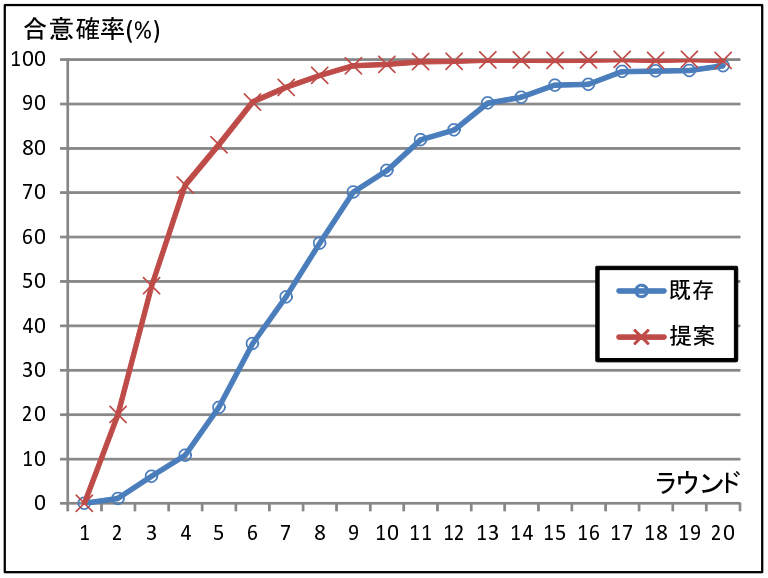


図 6 128 プロセスでの合意確率

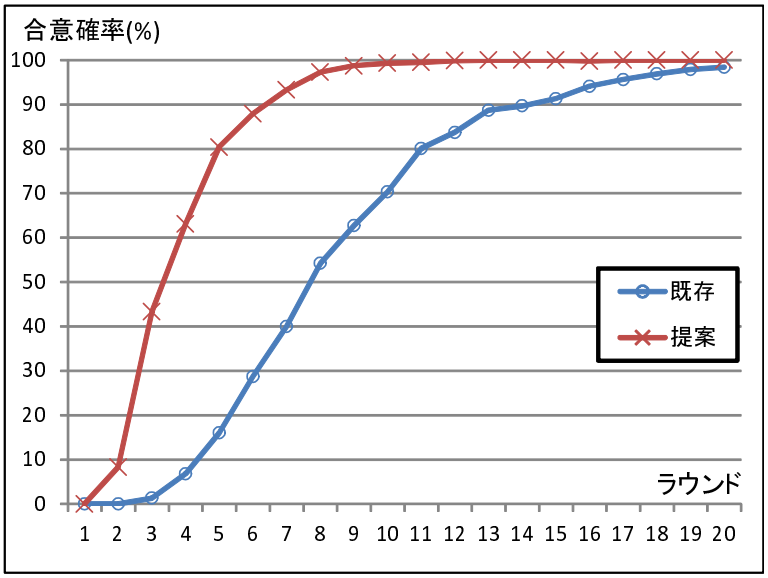


図 7 1024 プロセスでの合意確率

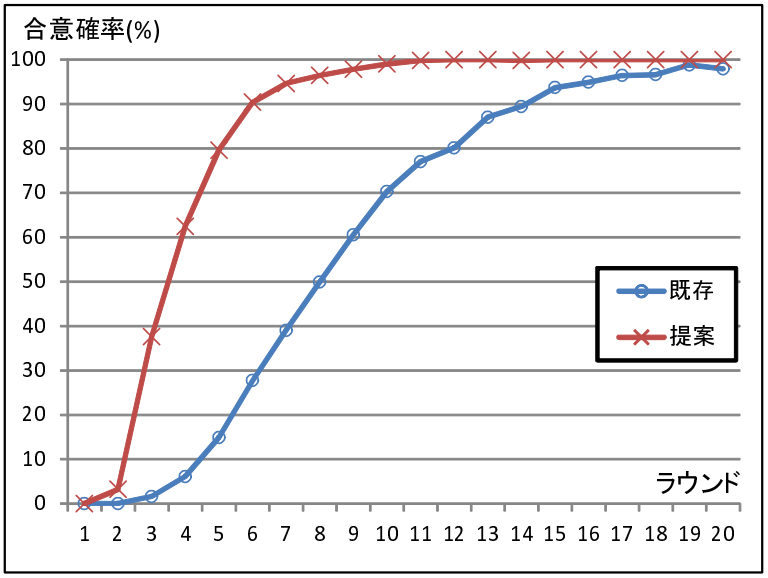


図 8 2048 プロセスでの合意確率

表 4 提案と既存のアルゴリズムでの総ステップ計算量

プロセス数	32		128		1024		2048	
アルゴリズム	既存	提案	既存	提案	既存	提案	既存	提案
ラウンド数 1	1,024	848	16,384	14,946	1,048,576	1,015,908	4,194,304	4,101,264
2	2,048	1,551	32,768	27,824	2,097,152	1,931,390	8,388,608	7,826,191
3	3,072	2,163	49,152	36,731	3,145,728	2,390,609	12,582,912	9,675,801
4	4,096	2,667	65,536	44,448	4,194,304	3,103,257	16,777,216	12,316,169
5	5,120	3,149	81,920	51,846	5,242,880	3,572,691	20,971,520	14,580,198
6	6,144	3,692	98,304	60,448	6,291,456	4,088,067	25,165,824	16,438,244
7	7,168	4,189	114,688	68,677	7,340,032	4,588,308	29,360,128	18,117,258
8	8,192	4,776	131,072	77,754	8,388,608	5,109,255	33,554,432	20,664,734
9	9,216	5,230	147,456	85,624	9,437,184	5,640,243	37,748,736	22,463,390
10	10,240	5,796	163,840	93,672	10,485,760	6,201,333	41,943,040	24,503,596
11	11,264	6,321	180,224	102,375	11,534,336	6,758,814	46,137,344	27,027,768
12	12,288	6,863	196,608	110,251	12,582,912	7,150,910	50,331,648	28,416,066
13	13,312	7,448	212,992	118,333	13,631,488	7,756,761	54,525,952	30,840,724
14	14,336	7,905	229,376	126,697	14,680,064	8,245,275	58,720,256	32,621,048
15	15,360	8,429	245,760	133,782	15,728,640	8,838,323	62,914,560	34,832,448
16	16,384	8,973	262,144	143,098	16,777,216	9,339,998	67,108,864	36,970,256
17	17,408	9,469	278,528	150,915	17,825,792	9,875,289	71,303,168	39,829,360
18	18,432	10,016	294,912	158,875	18,874,368	10,360,401	75,497,472	40,959,516
19	19,456	10,528	311,296	167,978	19,922,944	10,932,598	79,691,776	42,978,260
20	20,480	10,990	327,680	175,819	20,971,520	11,427,274	83,886,080	45,640,252

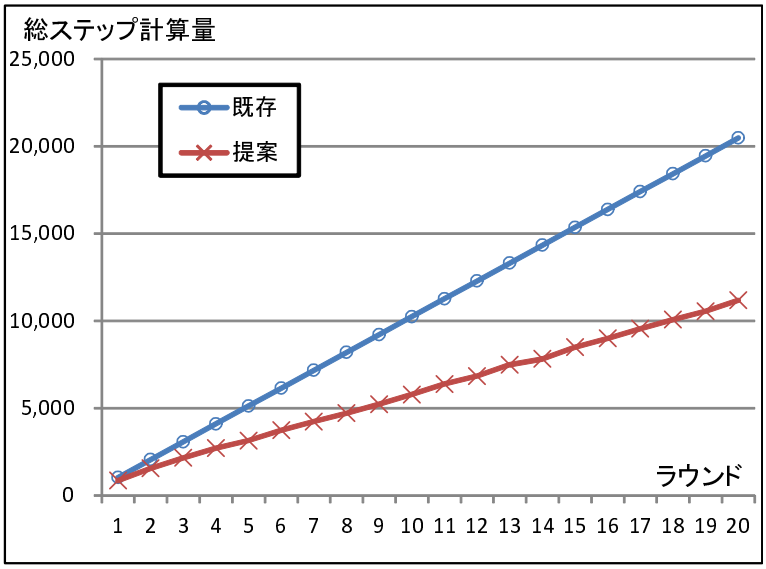


図 9 32 プロセスでの総ステップ計算量

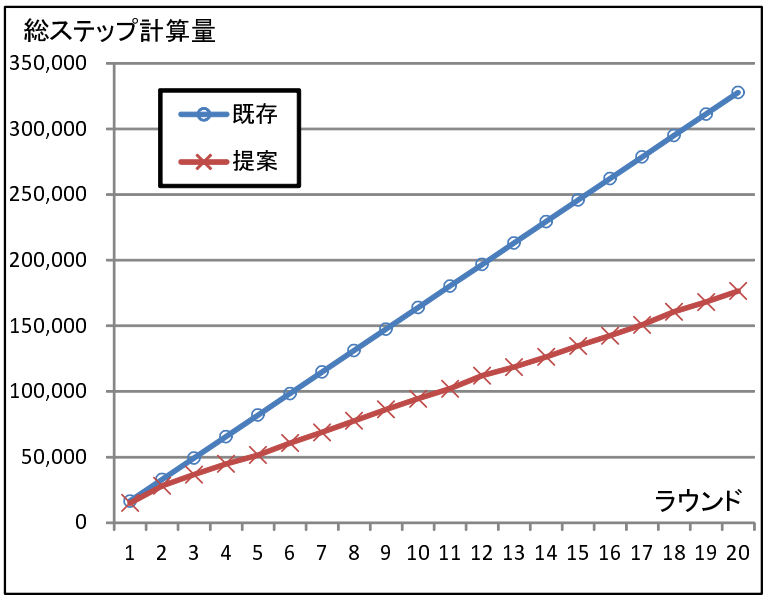


図 10 128 プロセスでの総ステップ計算量

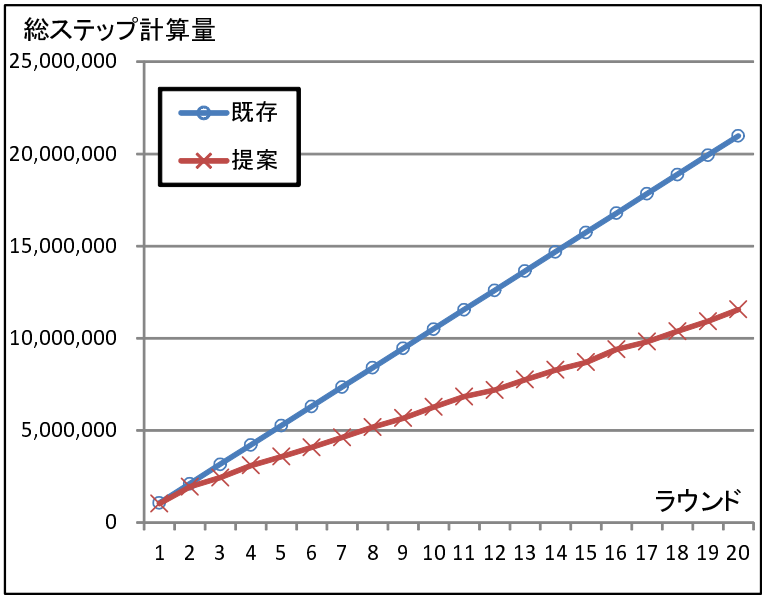


図 11 1024 プロセスでの総ステップ計算量

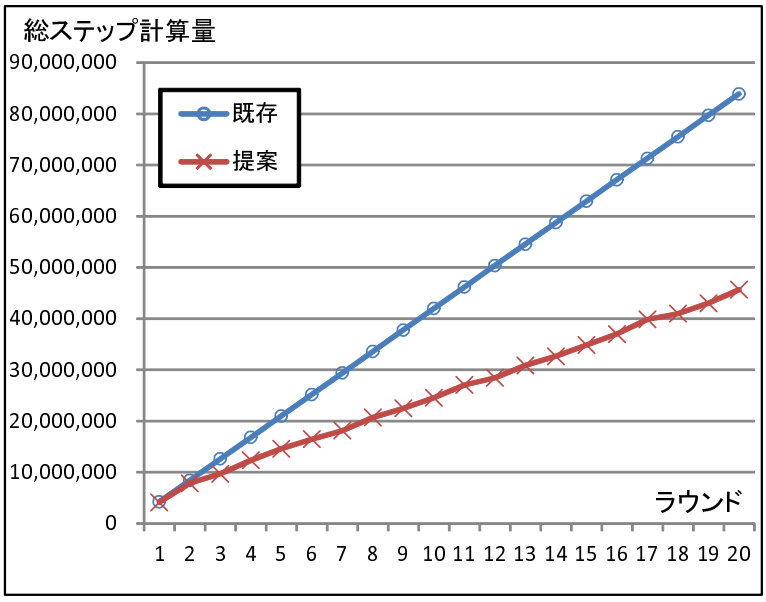


図 12 2048 プロセスでの総ステップ計算量

た場合の必要なプロセスステップ計算量は, 提案アルゴリズムが既存アルゴリズムに比べ, 高速に処理できることがわかる. 例えば, 1024 プロセスで合意確率 98%を保証するアルゴリズムのプロセスステップ計算量を考えると, 既存アルゴリズムでは 20,971,520(20 ラウンド) なのに対し, 提案アルゴリズムは 5,640,243(9 ラウンド) であり, 約 3.72 倍高速である. また, 2048 プロセスの場合, 提案アルゴリズムは合意確率が 1.6%以上において, 既存手法よりも 1 ラウンド早く到達することがわかった.

6.2 計算機シミュレーションを用いた提案手法と評価実験

提案アルゴリズムにおいて, 3 節で示した各ラウンド i で使用する write 確率 p_i は, 各ラウンドで残っている異なる提案値の個数 X_i の単純な上界を基に計算した確率である. そのため, 計算機シミュレーションにより各ラウンドでの X_i の数を最小とする最適な write 確率 p_{opti} を求めて使用することで合意確率は, 更に高くなると考えられる. そこで, 提案した MRSW を用いた Conciliator アルゴリズム中の write 確率として p_{opti} を使用する手法を提案する. この手法において, ある合意確率を得るのに必要なラウンド数は, 計算機シミュレーションで得た各ラウンドでの合意確率が指定した合意確率を上回ったラウンド数とする.

この手法に関しても, 計算機実験により, p_i を用いた Conciliator アルゴリズムと比較評価を行い効果を確認した.

実験結果として, プロセス数を 32, 128, 1024, 2048 とした時に, 各ラウンドでの X_i の数を最小とする write 確率 p_{opti} を求めた結果を表 5 に示す. また, p_{opti} を使用した場合の各ラウンド数での合意確率と総ステップ計算量を表 6 に示す. また, 図 13, 14, 15, 16 に p_i と p_{opti} を使用した場合それぞれの合意確率をグラフで示す. また, 図 17, 18, 19, 20 に p_i と p_{opti} を使用した場合それぞれの総ステップ計算量をグラフで示す.

表 5 と表 6 の結果から, p_{opti} を使用することで, より高い合意確率を早いラウンドで得ることができることが分かった. また, プロセス数が大きくなるにつれ, 各ラウンドでの合意確率はより高くなっている. 特に, プロセス数 $n = 2048$ の場合, 合意確率が 80%以上において, p_{opti} を使用するほうが 1 ラウンド早く到達することがわかった. 上記の実験結果から, write 確率 p_{opti} を用いた Conciliator アルゴリズムは p_i を使用した場合と比べ, 時間計算量, 合意確率において優れていることがわかった.

表 5 提案アルゴリズムの各ラウンドでの write 確率 (%)

プロセス数	32		128		1024		2048	
write 確率	p_i	p_{opti}	p_i	p_{opti}	p_i	p_{opti}	p_i	p_{opti}
ラウンド数 1	0.1796	0.1068	0.0887	0.0411	0.0313	0.005	0.0221	0.0034
2	0.2997	0.321	0.2106	0.243	0.125	0.19	0.1051	0.15
3	0.3871	0.376	0.3245	0.35	0.25	0.33	0.2293	0.35
4	0.5	0.43	0.5	0.4	0.3536	0.41	0.3386	0.49
5	0.5	0.44	0.5	0.42	0.5	0.45	0.5	0.46
6	0.5	0.47	0.5	0.47	0.5	0.54	0.5	0.49
7	0.5	0.47	0.5	0.56	0.5	0.53	0.5	0.51
8	0.5	0.47	0.5	0.43	0.5	0.53	0.5	0.53
9	0.5	0.49	0.5	0.47	0.5	0.48	0.5	0.5
10	0.5	0.46	0.5	0.47	0.5	0.51	0.5	0.49
11	0.5	0.56	0.5	0.54	0.5	0.51	0.5	0.5
12	0.5	0.44	0.5	0.54	0.5	0.52	0.5	0.5
13	0.5	0.5	0.5	0.5	0.5	0.49	0.5	0.5
14	0.5	0.53	0.5	0.45	0.5	0.49	0.5	0.5
15	0.5	0.54	0.5	0.51	0.5	0.5	0.5	0.5
16	0.5	0.44	0.5	0.49	0.5	0.5	0.5	0.5
17	0.5	0.5	0.5	0.52	0.5	0.5	0.5	0.5
18	0.5	0.49	0.5	0.52	0.5	0.5	0.5	0.5
19	0.5	0.44	0.5	0.47	0.5	0.5	0.5	0.5
20	0.5	0.49	0.5	0.49	0.5	0.5	0.5	0.5

表 6 実験で求めた write 確率 p_{opti} を用いた場合の合意確率と総ステップ計算量

プロセス数	32		128		1024		2048	
	合意確率 (%)	総ステップ	合意確率 (%)	総ステップ	合意確率 (%)	総ステップ	合意確率 (%)	総ステップ
ラウンド数 1	9.4	918	1.6	15,698	3.0	1,043,141	1.0	4,180,327
2	42.7	1,591	38.7	28,300	39.3	1,893,701	37.3	7,704,466
3	63.8	2,250	64.5	37,829	56.0	2,562,637	59.1	10,224,390
4	77.9	2,832	76.8	46,932	77.8	3,134,918	76.0	12,063,052
5	86.7	3,393	87.0	56,654	90.4	3,713,714	91.1	14,488,786
6	91.7	3,984	93.4	65,790	95.8	4,231,999	94.2	16,689,292
7	95.2	4,515	94.9	73,490	96.4	4,709,261	97.9	18,640,054
8	96.8	5,085	97.5	81,975	98.7	5,099,652	100.0	20,635,682
9	97.9	5,617	98.6	91,454	100.0	5,796,179	99.0	22,648,234
10	98.7	6,171	98.9	99,243	100.0	6,238,645	100.0	24,891,460
11	99.2	6,670	99.5	106,822	100.0	6,657,156	100.0	27,233,250
12	99.5	7,269	99.7	114,101	100.0	7,382,954	100.0	28,979,466
13	99.7	7,775	99.7	123,283	100.0	7,845,255	100.0	30,963,380
14	99.8	8,294	99.9	131,827	100.0	8,399,880	100.0	33,405,528
15	99.7	8,739	100.0	141,078	100.0	8,881,243	100.0	35,793,024
16	99.9	9,321	100.0	150,186	100.0	9,334,403	100.0	37,559,732
17	99.9	9,886	100.0	154,847	100.0	9,928,948	100.0	39,564,676
18	100.0	10,372	100.0	163,247	100.0	10,379,337	100.0	41,849,276
19	100.0	10,995	99.8	172,335	100.0	11,015,188	100.0	44,386,528
20	99.9	11,499	100.0	183,795	100.0	11,686,194	100.0	45,755,244

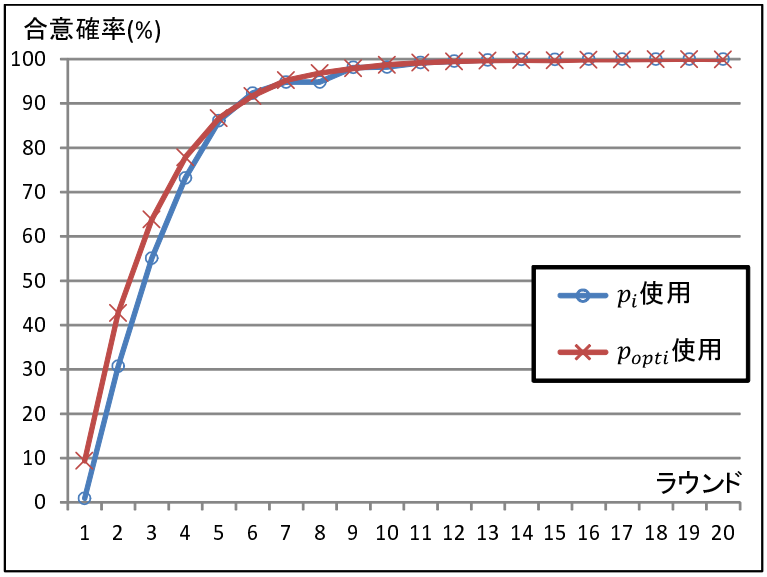


図 13 write 確率による 32 プロセスでの合意確率

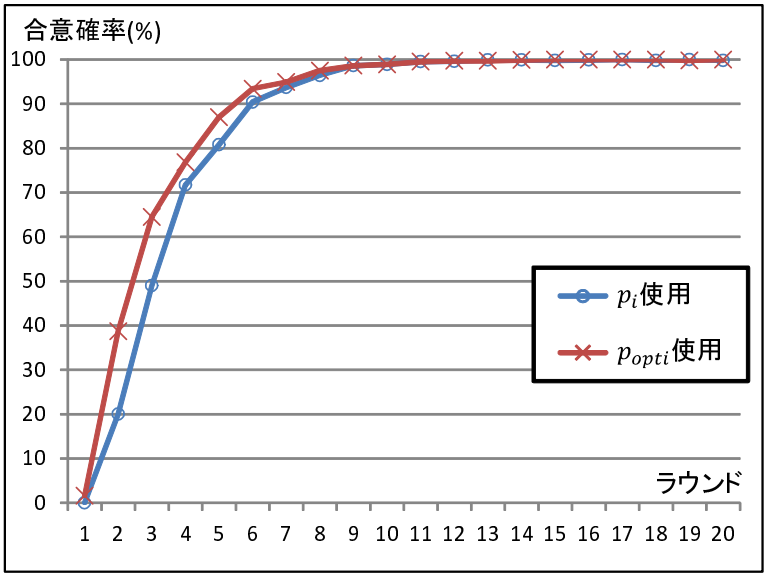


図 14 write 確率による 128 プロセスでの合意確率

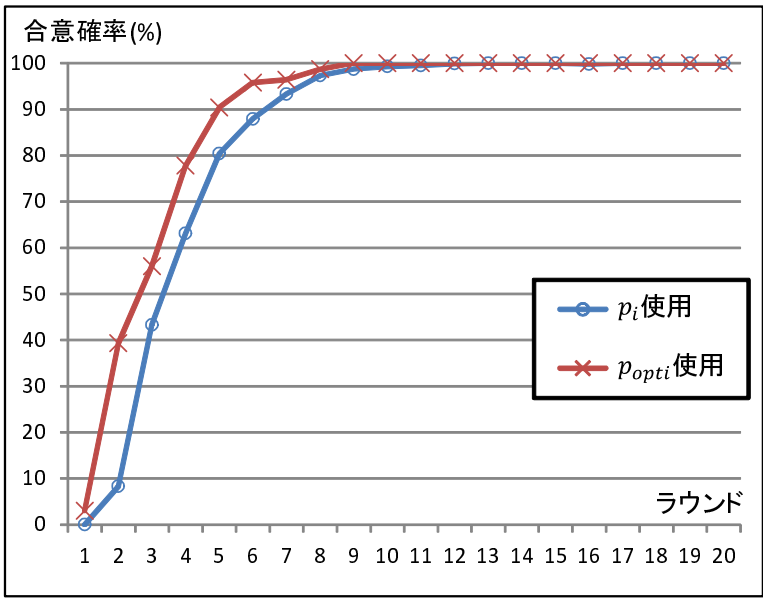


図 15 write 確率による 1024 プロセスでの合意確率

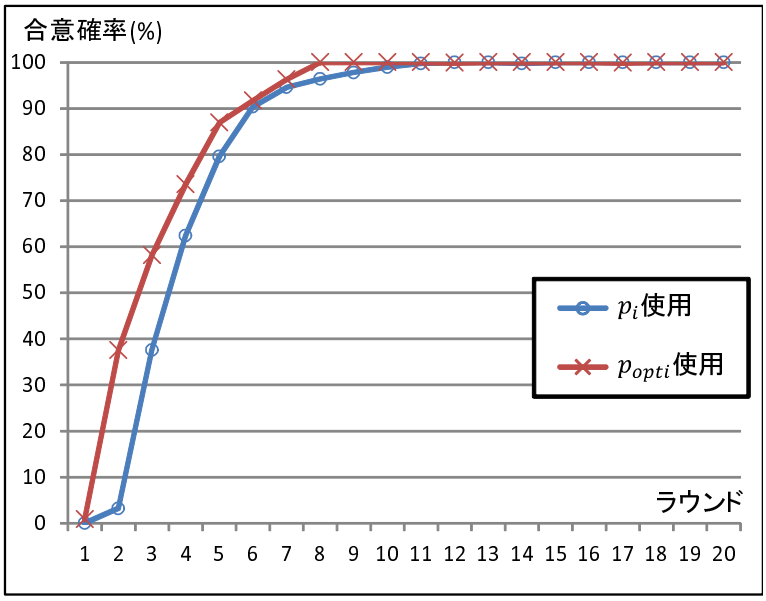


図 16 write 確率による 2048 プロセスでの合意確率

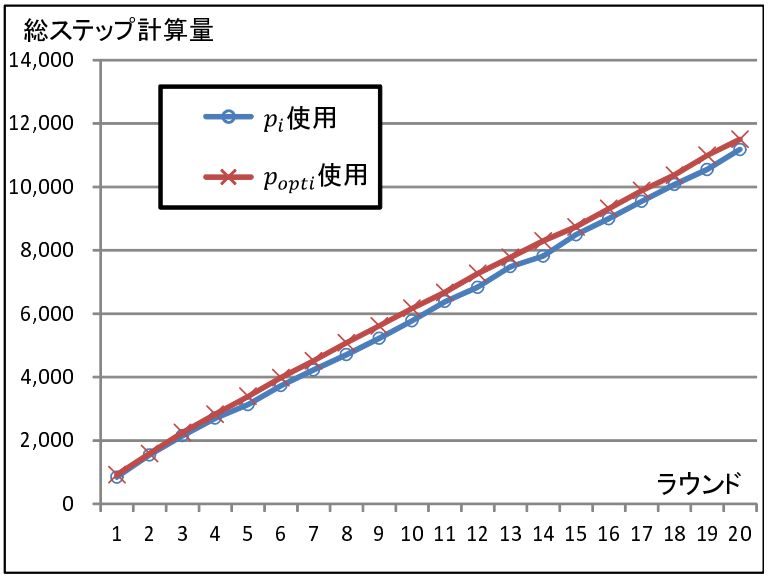


図 17 write 確率による 32 プロセスでの総ステップ計算量

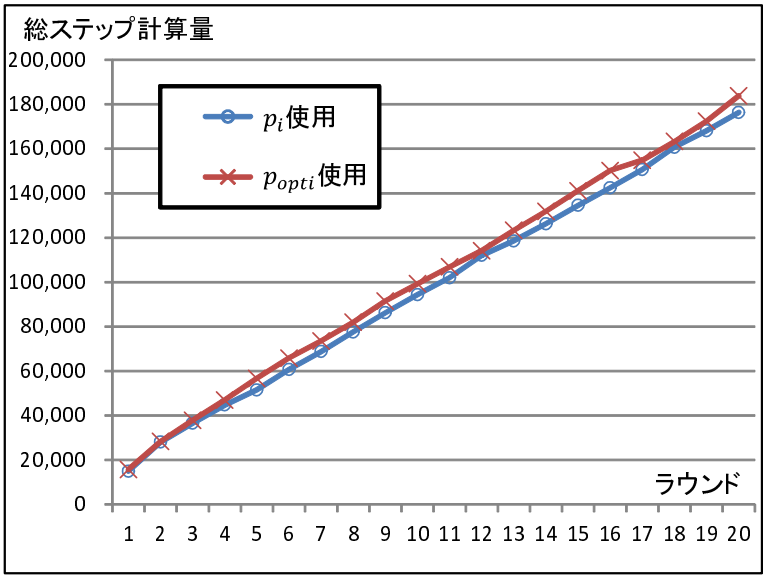


図 18 write 確率による 128 プロセスでの総ステップ計算量

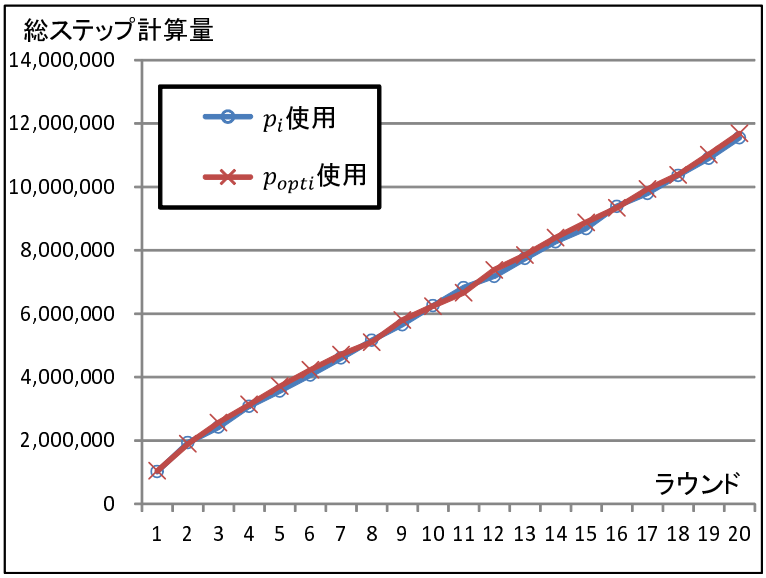


図 19 write 確率による 1024 プロセスでの総ステップ計算量

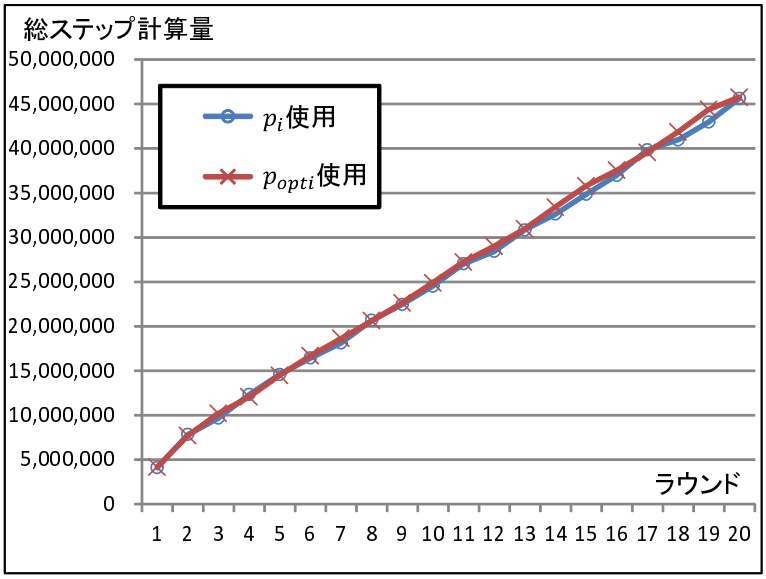


図 20 write 確率による 2048 プロセスでの総ステップ計算量

7. まとめ

本研究では, MRSW レジスタを用いた, 乱択合意アルゴリズムの提案を行った. 具体的には, Conciliator アルゴリズムと Weak Conflict-Detector オブジェクトを実現するアルゴリズムを提案し, 乱択合意アルゴリズムを実現した.

提案した Conciliator アルゴリズムでは, 合意確率が $1-\epsilon(0 < \epsilon < 1)$ である Conciliator を, MRSW レジスタを用いて $O(n \log\log n + \log(1/\epsilon))$ のプロセスステップ計算量で実現できる. これは, 文献 [5] の Conciliator アルゴリズムのプロセスステップ計算量と漸近的に等しい.

Weak Conflict-Detector オブジェクトを実現するアルゴリズムでは, 文献 [9] の実現方法では, プロセスステップ計算量が $\Theta(\min(n\frac{\log m}{\log\log m}, n^2))$ であるのに対して, 提案アルゴリズムでは, プロセスステップ計算量が $O(n)$ で実現できる.

乱択合意アルゴリズムは, Conciliator アルゴリズムと Weak Conflict-Detector オブジェクトの組み合わせで実現でき, そのプロセスステップ計算量はそれぞれにかかるプロセスステップ計算量の和と漸近的に等しいため, 提案したアルゴリズムのプロセスステップ計算量は $O(n \log\log n)$ である. この結果は, アルゴリズムの入力値の値域 m が $m = (\log n)^{O(\log\log\log n)}$ を超える場合において, 同様の構成方法である文献 [5] の乱択合意アルゴリズムよりも優れている. それ以外の場合においては, 提案手法のプロセスステップ計算量は既存手法と漸近的に等しい.

また, Conciliator アルゴリズムについては, 計算機実験により, 既存手法 [5] と比較評価を行った. 実験結果として, 提案した Conciliator アルゴリズムが, 指定した合意確率を満足するのに必要な総ステップ計算量が, 既存手法よりも短いことを示し, 提案アルゴリズムの有用性を確認した. さらに, 提案手法として, 乱択合意アルゴリズム中で使用する write 確率として, 計算機シミュレーションによる最適値 p_{opti} を用いる手法を提案した. この手法に対しても, 計算機実験により, 3 節で示した write 確率 p_i を用いる提案手法と比較評価を行った. 実験結果として, p_i を用いた場合よりも p_{opti} を用いた手法のほうが, 時間計算量, 合意確率において優れていることがわかった.

今後の課題としては, 提案アルゴリズムの優位性に関して, 評価実験ではなく計算量による解析を行い, 証明する必要が考えられる.

謝辞

研究を進めるにあたり, 日頃より懇切丁寧なご教示を賜りました主指導教員の井上美智子教授に心から感謝申し上げます. 本研究に際して, 貴重な御指導を賜りました副指導教員の伊藤実教授に深く感謝申し上げます. 本研究を行うにあたり, 的確な御助言, 御指導をいただきました副指導教員の米田友和助教に心から感謝申し上げます. 本研究を行うにあたり, 貴重な御助言, 有益な御指導をいただきました副指導教員の大和勇太助教に深く感謝申し上げます. また, 本研究を行うにあたり, 的確な御助言をいただきました畠山一実特任教授に心から感謝申し上げます. 最後に, 日頃からさまざまな面で支えてくださったディペンダブルシステム学研究室のスタッフ, 日頃の研究活動を通じて活発な議論にお付き合いいただいたディペンダブルシステム学研究室の同期の伊藤君, 西井君, 宮本君, Yussuf 君に感謝致します.

参考文献

[1] Michael J. Fischer, Nancy A, Lynch, and Michael S. Paterson, ”Impossibility of distributed consensus with one faulty process,” Journal of the ACM, 32(2):374–382, April 1985.

[2] Murice Herlihy, ”Wait-Free Synchronization,” ACM Transactions on Programming Languages and Systems, Vol.11, No 1, pages 124–149, January 1991.

[3] Hagit Attiya, Keren Censor, ”Tight bounds for asynchronous randomized consensus,” Journal of the ACM, 55(5):20, October 2008.

[4] James Aspnes, Keren Censor, ”Approximate shared-memory counting despite a strong adversary,” Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms, pages 441–450, Philadelphia, PA, USA, 2009.

[5] James Aspnes, ”Faster randomized consensus with an oblivious adversary,” Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing, pages 1–8, July 2012.

[6] Yonatan Aumann, ”Efficient asynchronous consensus with the weak adversary scheduler,” Proceedings of Sixteenth Annual ACM Symposium on Principles of Distributed Computing, pages 209–218, NY, USA, 1997.

[7] Tushar Deepak Chandra, ”Polylog randomized wait-free consensus,” In Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, pages 166–175, Philadelphia, Pennsylvania, USA, May 1996.

[8] James Aspnes, ”A modular approach to shared-memory consensus, with applications to the probabilistic-write model,” Distributed Computing, 25(2):179–188, May 2012.

[9] James Aspnes, Faith Ellen, ”Tight bounds for anonymous adopt-commit objects,” Proceedings of 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, pages 317–324, June 2011.

[10] James Aspnes, ”Lower bounds for distributed coin-flipping and randomized consensus,” Journal of the ACM, 45(3):415–450, May 1998.

[11] James Aspnes, ”Randomized consensus in expected $O(n^2)$ total work using single-writer registers,” Proceedings of 25th International Symposium on Distributed Computing, pages 363–373, Springer-Verlag, September 2011.

[12] 増澤利光, 山下雅史, ”適応的分散アルゴリズム,” 共立出版, 2010.

[13] Eli Gafni, ”Round-by-round fault detectors: Unifying synchrony and asynchrony (extended abstract),” PODC 17th, pages 143-152, 1998.

[14] Gadi Taubenfeld, ”Synchronization algorithms and concurrent programming,” Prentice Hall, July 2006.

業績リスト

1. 中島 悟, 井上 美智子, ”オブリビアス敵対スケジューラ下での MRSW レジスタを用いた乱択合意アルゴリズム,” 電気通信学会技術研究報告 (COMP2013-68), vol. 113, No. 488, pp. 53–60, March. 2014.