# PLP-Raft：部分的漏洩耐性を持つ合意アルゴリズム

永見 拓人[1,a)]　駒野 雄一[1,b)]

**概要**：現在、Kubernetes をはじめ様々な分散システムにおいて、合意アルゴリズム Raft が基盤技術として使われている。Raft は全サーバーにログを完全複製するアルゴリズムであるため、1 つでもサーバーが侵害されればデータが漏洩してしまう恐れがあり、構造的に漏洩耐性が低いという問題を抱えている。漏洩耐性を上げる手法として一般的には暗号化が挙げられるが、大半の暗号化は複雑な鍵管理を要し、運用次第で結局漏洩耐性が上がらないケースがしばしば存在する。本稿では、データを分散保存しているサーバーのうちしきい値未満の台数のサーバーが侵害されても元のデータが復元できない性質を「部分的漏洩耐性」と定義し、Raft の派生アルゴリズムとして $(k, n)$ しきい値秘密分散法を利用して部分的漏洩耐性を持たせた Partially Leak-Proof Raft Consensus Algorithm (PLP-Raft) を提案する。PLP-Raft 合意アルゴリズムは、秘密分散によって生成されたシェアをそれぞれのサーバーに配置することによって部分的漏洩耐性を獲得しており、用いる秘密分散アルゴリズムやそのしきい値を変えることにより漏洩耐性と障害耐性のバランスを調整することができる。実験を通して、PLP-Raft は部分的漏洩耐性を現実的なオーバーヘッドで実現できることが明らかになった。

**キーワード**：合意アルゴリズム, Raft, 秘密分散法

# PLP-Raft: A Partially Leak-Proof Consensus Algorithm

Takuto Nagami[1,a)]　Yuichi Komano[1,b)]

***Abstract:*** The consensus algorithm Raft is currently used as the underlying technology for various distributed systems, including Kubernetes. Because Raft replicates logs across all servers, even a single server compromise can result in data leaks, making it structurally less resilient. While encryption is a common approach to improving resilience, most encryption methods require complex key management and often fail to achieve high resilience depending on their implementation. In this paper, we define "partial leak resistance" as the property that prevents data from being restored even if a number of servers (less than a threshold) that store data in a distributed manner are compromised. We propose a partially leak-proof Raft consensus algorithm (PLP-Raft), which utilizes a $(k, n)$ threshold secret sharing scheme to achieve partial leak resistance. The PLP-Raft consensus algorithm achieves partial leak resistance by allocating shares generated by secret sharing to each server. The balance between leak resistance and fault tolerance can be adjusted by changing the secret sharing algorithm and its threshold. Through the experimentation, it became clear that PLP-Raft can achieve partial leak resistance with relatively low overhead.

***Keywords:*** Consensus Algorithm, Raft, Secret Sharing Scheme

## 1. はじめに

2014 年に Ongaro と Ousterhout[7] によって提案された合意アルゴリズム Raft は、現在 Kubernetes をはじめ様々な分散システムにおいて基盤技術として使われている。

しかし、Raft は全ノードにログを完全複製するアルゴリズムであるため、任意のノードが侵害されれば機密情報が漏洩してしまう恐れがあり、構造的に機密性の問題を抱えている。etcd に代表される Raft を組み込んだプロダク

[1] 千葉工業大学
Chiba Institute of Technology
[a)] s2232703wf@s.chibakoudai.jp
[b)] yuichi.komano@p.chibakoudai.jp

© 2025 Information Processing Society of Japan

This paper is work in progress and not peer-reviewed.

トは、TLS 通信や証明書認証などで機密性を強化しているが、ノードへの侵害や物理アクセスを伴うメモリダンプが起こった場合、データの奪取や改ざんには対抗できない。

ノードへの侵害や物理アクセスを伴うメモリダンプによる情報漏洩を防ぐアプローチとして、AES[6] に代表される共通鍵暗号や RSA 暗号 [8] に代表される公開鍵暗号による暗号化がある。実際に etcd 等では暗号化機能を提供しているが、大半の暗号化は複雑な鍵管理を要し、運用次第で結局漏洩耐性が上がらないケースも多く存在する。

鍵の管理なしで情報の機密性を維持するアプローチに秘密分散法 [2], [10] がある。その一つである $(k, n)$ しきい値秘密分散法では、機密情報を $n$ 個のシェアに分割し、$n$ 個のノードにシェアを一つずつ保存する。このとき、$k$ 個 (しきい値) 以上のノードがシェアを持ち寄れば機密情報を復元できるが、$k$ 個未満のシェアからは元の機密情報に関する情報を得ることはできない。

本研究では、Raft が持つ構造的な機密性の問題を鍵管理無しで解決するために、分散システムにおける新たなセキュリティ評価指標として「部分的漏洩耐性」を定義する。そして、Raft の派生アルゴリズムとして $(k, n)$ しきい値秘密分散法を利用して部分的漏洩耐性をもたせた Partially Leak-Proof Raft Consensus Algorithm (PLP-Raft) を提案し、実装評価によりその性能を評価する。

### 1.1 関連研究

Raft に秘密分散法を組み合わせて新しいアルゴリズムを提案する研究はいくつか存在するが、それらの貢献は本稿と大きく異なる。

#### 1.1.1 VSSB-Raft

Tian ら [11] によって提案された Verifiable Secret Sharing Byzantine Fault Tolerance Raft Consensus Algorithm (VSSB-Raft) は、Raft にゼロトラストを導入することでビザンチン障害耐性をつける試みである。

VSSB-Raft では、Raft のリーダー選挙 (2.1.1 節) に SM2 署名アルゴリズムと秘密分散 (2.2 節) を組み合わせることで、あるノードが不正にリーダーに選ばれることがない新たな選挙アルゴリズムが用いられている。

しかし、この研究では Raft に秘密分散が組み込まれているもののリーダー選挙にしか使われておらず、本稿が目的としている部分的漏洩耐性 (3.1 節) を実現するものではない。

#### 1.1.2 RaBFT

Bai ら [1] によって提案された Byzantine fault tolerance consensus algorithm based on raft (RaBFT) も、Raft に秘密分散を組み込んだ新しいアルゴリズムの例として挙げられる。RaBFT も VSSB-Raft[11] と同様に、Raft にビザンチン障害耐性をつける試みである。

RaBFT では、Log Replication(2.1.2 節) において Leader と Follower の間に Committee という新たな役割のサーバーを追加し、Leader が元のデータを秘密分散によって変換したシェアをそれぞれの Committee に渡し、全ての Committee が全ての Follower にシェアをブロードキャストして各 Follower が検証することで、悪意を持った Committee を検出することができる。

しかし、RaBFT でも最終的に全ての Follower が同一のログを持つことになるため、本稿が目的としている部分的漏洩耐性 (3.1 節) を実現するものではない。

### 1.2 本稿の貢献

本稿の貢献は以下のとおりである。

- 分散システムにおける新たなセキュリティ評価指標として「部分的漏洩耐性」を定義し、それが有効に働くサイバー攻撃のシナリオを確認する。
- Raft のアルゴリズムをベースに、保存する State を秘密分散でシェアに変換し、各サーバーのステートマシンに異なるシェアを保存することで部分的漏洩耐性を実装した合意アルゴリズム「PLP-Raft」を提案する。
- PLP-Raft と Raft における読み書きのレイテンシを計測し、比較評価する。

本論文の構成は以下のとおりである。2 節では準備として、合意アルゴリズム Raft とその問題点、さらに $(k, n)$ しきい値秘密分散法について説明する。次に 3 節では、本稿で実現する部分的漏洩耐性を定義したのち、それとトレードオフの関係にある障害耐性の定義を述べる。その後 4 節では、Raft を秘密分散法と組み合わせて部分的漏洩耐性を持たせた派生アルゴリズム「Partially Leak-Proof Raft Consensus Algorithm (PLP-Raft)」を提案し、5 節でその性能を実装評価する。そして 6 節で機密性と障害耐性および処理効率について考察し、7 節でまとめを述べる。

## 2. 準備

本節では、Raft とその問題点について述べたのち、PLP-Raft で利用する秘密分散法について説明する。

### 2.1 Raft

Raft[7] は、分散システム上で複製されたログを管理するための合意アルゴリズムである。同じ目的を持ったアルゴリズムとして Paxos[3], [4] が長らく知られていたが、Ongaro と Ousterhout[7] は、Paxos が非常に理解しづらく、実用的なシステムの構築に不向きであると主張した。これを踏まえて、(multi-)Paxos と同じ結果・効率を持ちながらも Paxos よりもアルゴリズムが理解しやすく、実用的なシステムを構築するためのより良い基盤を提供することを目標として Raft が生み出された。

Raft は選挙と通常操作の 2 つのフェーズからなり、選挙フェーズでは Leader Election(2.1.1 節)、通常操作フェー

© 2025 Information Processing Society of Japan

ズでは Log Replication(2.1.2 節) が行われる。

### 2.1.1 Leader Election

Raft は合意を形成するために、特定の Leader を選出し、その Leader に複製されたログの管理に関する全ての責任を与える。このリーダーの選出に用いられるアルゴリズムは Leader Election(リーダー選挙) とよばれる。Leader Election は、同時に 2 つ以上の Leader が存在しないことを保障する重要なアルゴリズムである。

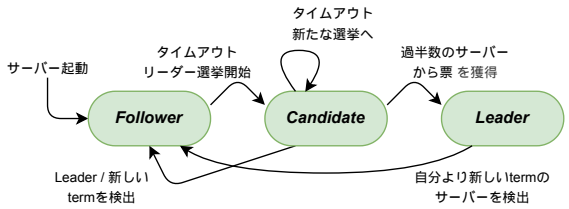


**図 1** Ongaro と Ousterhout[7] が提案した Raft における、サーバーの状態遷移

図 1 に示されているように、Raft において各サーバーは Leader、Follower、Candidate のいずれかの状態をとる。Raft は時間を term とよばれる整数値の長さの期間に区切り、各期間はリーダー選挙から始まる。

また、Follower は、Leader や Candidate から一定時間通信を受信しなかった場合、有効な Leader がいないと判断して新しいリーダー選挙を開始する。Follower は自身の term を 1 増やして Candidate に遷移し、投票したのちに他の全てのサーバーに投票をリクエストする。

Candidate がクラスタ全体の過半数となるサーバーから票を獲得した場合、当選したとみなされて Leader になる。当選した Leader は、自身の権限を確立し新たな選挙を防ぐために、他の全てのサーバに一定間隔でハートビートメッセージを送信し続ける。Leader が故障したり、疎通性が失われたりした場合、いずれかの Follower が Candidate になり新たな選挙が行われる。

### 2.1.2 Log Replication

Log Replication は、Leader が持つログをすべてのサーバーに複製し、一貫した状態を維持する仕組みである。

Raft は Replicated State Machine[9] 機構に則って構成されている。典型的な Replicated State Machine のアーキテクチャにおいては、ステートがそのまま複製されるのではなく、操作履歴 (ログ) が Raft などの合意アルゴリズムによって管理・複製される。State Machine はログを常に同じ順序で計算してステートを算出するため、全てのサーバーで同じステートを共有することができる。

Raft は強力なリーダーシップモデルを採用しており、ログエントリは Leader から Follower へ一方向のみに流れる。Leader がクライアントからのコマンドを受け取ると、それを自身のログに追記し、他のサーバーへ「AppendEntries RPC」を用いて並行して複製する。Leader は Follower のログを自身のログと一致させるように強制し、不整合なエントリがあった場合には Follower は自身のログを Leader のログで上書きする。それぞれのログエントリは、Leader からクラスタ内で過半数のサーバーに複製された時点でコミット (確定) したと見なされ、その時点をもってステートマシンに適用できるようになる。

### 2.1.3 Raft の問題点と既存の対策

2.1.2 節で説明したように、Raft は全てのサーバーに同じログが複製・保存されるため、任意のノードが侵害されれば機密情報が漏洩してしまう恐れがあり、構造的に機密性の問題を抱えている。etcd 等の Raft を組み込んだ既存のプロダクトは、TLS 通信や証明書認証などを備えて情報漏洩を防いでいるが、ノードへの侵害や物理アクセスを伴うメモリダンプが行われた場合、データの奪取や改ざんには対抗できない。

以上のような、ノードへの侵害や物理アクセスを伴うメモリダンプによる情報漏洩を防ぐアプローチとして、AES[6] に代表される共通鍵暗号や RSA 暗号 [8] に代表される公開鍵暗号による暗号化がある。実際に etcd 等では暗号化機能を提供しているが、大半の暗号化は複雑な鍵管理を要し、運用次第で結局漏洩耐性が上がらないケースも多く存在する。

## 2.2 $(k, n)$ しきい値秘密分散法

秘密分散法は、Shamir[10] と Blakley[2] によって独立に提唱された、データを複数の断片 (シェア) に分割し、その一部を知っていても元の情報を復元できないようにする暗号技術である。両者ともこの技術を暗号化鍵の管理手法として提案したが、理論的にはどのような機密情報でもシェアに分散することができる。Shamir と Blakley はそれぞれ、データ $D$ を $n$ 個のシェアに分割し、そのうち任意の $k$ 個のシェアが集まれば $D$ を容易に復元できるが、$k$ 個未満のシェアからは $D$ に関する情報が一切得られないようにする、$(k, n)$ しきい値分散法を提案した。

本稿は、特に Shamir[10] によって提唱された秘密分散法を使用する。この方式は、多項式補間に基づき、数値あるいは数値に変換可能な秘密データ $D$ を分散する。

シェアの生成においては、ディーラーとよばれる秘密データ $D$ を分散する人物が、次数が $k-1$ であり、定数項 $a_0$ が秘密データ $D$ そのものとなるランダムな多項式 $q(x)$ を選ぶ。そして、多項式 $q(x)$ に $x = i$ を代入することによって $n$ 個のシェア $D_i = q(x_i)$ を生成し、$n$ 人のユーザーに $i$ と $D_i$ を一組ずつ配布する。

シェアから秘密情報を復元する際は、任意の $k$ 個のシェア $D_i$ と、それに対応する $i$ が必要となる。これらの点から多項式補間により元の多項式 $q(x)$ の係数を特定することがで、定数項から秘密データ $a_0 = D$ を復元できる。

© 2025 Information Processing Society of Japan

Shamir[10] は「多項式補完のアルゴリズムは何でも良く、計算量 $O(n \log_2 n)$ のものがいくつか議論されているが、単純な計算量 $O(n^2)$ のものでも実用的な鍵管理用途であれば十分高速である」と主張しており、実際の実装ではラグランジュ補間が使われることが一般的である。

## 3. 本稿の提案システムが目指す安全性指標

本稿では、2.1.3 節で説明した Raft の構造的な機密性の問題を、鍵管理を必要としない方法で解決することを目的とする。これを定量的に評価するため、以下のような安全性の指標を定める。

### 3.1 部分的漏洩耐性

2.2 節で説明した $(k, n)$ しきい値秘密分散法では、データ $D$ を分割した $n$ 個のシェアのうち、任意の $k$ 個のシェアがあれば $D$ を復元できるが、$k$ 個未満のシェアからは $D$ に関する情報が一切得られないことが情報理論的に保障されている。

この秘密分散の安全性を分散システムに拡張すれば、データを分散保存しているサーバーのうち、しきい値未満の台数のサーバーが侵害されても元のデータが復元できない性質を生み出すことができる。本稿ではこの性質を「部分的漏洩耐性」と定義する。部分的漏洩耐性は「侵害されても機密情報が漏洩しない最大のサーバー台数」として定量的に評価する。

システムが部分的漏洩耐性をもつと、以下のようなケースにおいて攻撃からの情報漏洩を防ぐことができる。

- Web アプリケーションなど外部からアクセス可能なアプリケーションの脆弱性を利用して、単一のサーバーが攻撃・窃取されるケース
- 各サーバーが地理的/論理的に離れた地点に存在し、分散システムに属するサーバーに対して連鎖的な攻撃が難しいケース

一方、攻撃者によって複数のサーバーが一度に窃取されてしまう以下のようなケースでは、システムに部分的漏洩耐性をもたせる対策は有効に働かない。

- 全サーバーに権限を持つ管理者を狙ったソーシャルハッキングが攻撃の起点となるケース

### 3.2 障害耐性

Ongaro と Ousterhout[7] によれば、典型的な合意アルゴリズムは、過半数のノードが動いている限り正く動き続ける。すなわち、過半数に満たない台数であれば、サーバーの故障は許容できる。一定数までの故障を許容できる性質は、一般に「Resilience」とよばれるが、本稿では部分的漏洩耐性と対比させるために「障害耐性」とよぶこととする。障害耐性は、典型的に評価指標として用いられている「許容できる最大の故障台数」によって定量化する。

合意アルゴリズムに部分的漏洩耐性の要件を加えた場合、秘密データの復元に必要なサーバー台数のしきい値に依存して障害耐性が大きく変化する。6 節で述べるとおり、部分的漏洩耐性の強さと障害耐性の強さの間にはトレードオフの関係がある。

## 4. PLP-Raft

本節では、Raft の派生アルゴリズムとして $(k, n)$ しきい値秘密分散法を利用して部分的漏洩耐性をもたせた「Partially Leak-Proof Raft Consensus Algorithm (PLP-Raft)」を提案する。PLP-Raft は、分散システムの故障耐性という面においては、Raft が保障している Crash-Recovery 障害までの耐性を保障し、ビザンチン障害 [5] 耐性は保証しないアルゴリズムとして設計されている。

### 4.1 基本概念

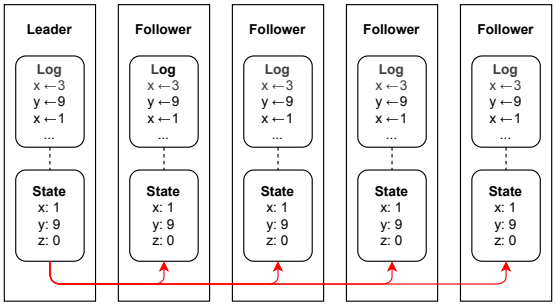


**図 2** Raft のデータ配置イメージ

2.1 節で説明したとおり、Raft においては図 2 のようにログおよびステートが複製される。

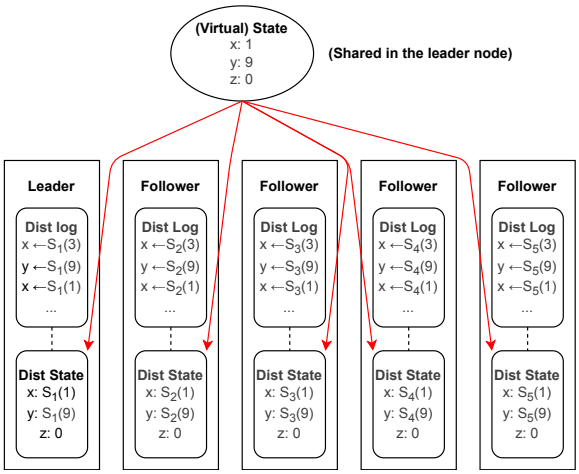


**図 3** PLP-Raft のデータ配置イメージ

一方 PLP-Raft においては、図 3 のような形でデータが配置される。クラスタ全体で保存されているデータは仮想ステート (Virtual State) とよばれ、これを秘密分散で変換したシェアがそれぞれのサーバーがもつ分散ステート

© 2025 Information Processing Society of Japan

(Distributed State) である。それぞれのサーバーは異なる分散ステートをもち、分散ステートの変更履歴である分散ログ (Distributed Log) の整合性をクラスタ内で保つ仕組みになっている。

### 4.2 サーバーの状態遷移と Leader Election

PLP-Raft は、図 1 に示した Raft のサーバー状態遷移と同一の状態遷移を用いる。また、リーダー選挙についても 2.1.1 節で述べた Raft のリーダー選挙をそのまま用いる。

### 4.3 Distributed Log Transmission

分散ログ転送 (Distributed Log Transmission) は Raft における Log Replication(2.1.2 節) に対応する概念であり、機密情報から生成された秘密分散のシェアをそれぞれのサーバーに転送するための仕組みである。

まず、Raft と同様に、ステートに対する変更は全て Leader へと送られる。Leader はこれを仮想ステートのログとして扱い、このログがコミットされるまで自身の揮発性ステートに一時保存する責任を負う。

追加ステート定義

**Leader における揮発性ステート** (太字が追加)

- *nextIndex[]*: 各 Follower に対して次に送られるべきログの index
- *matchIndex[]*: 各 Follower が保存していることが確定しているログの index
- ***uncommitedLog[]*: まだコミットされていない、仮想ステートに対するログ**
- ***recoveredDistLog[][]*: クラスタではコミット済みだが Follower に転送されていない、Follower ごとの分散ログ**

Leader はここで追加したログエントリから、$(k, n)$ 秘密分散 ($k$ はあらかじめ設定された値、$n$ はノードの台数) のディーラーとしてノード数分のシェアを生成する。シェアの生成時に多項式に代入される $i$ は、重複を防ぐためユニークにサーバーに割り振られた ID から決める。これによって、同じログエントリから同じノードに対して発行されるシェアは常に同じであることが保障される。

最終的に Leader は、各 Follower に対して、生成したシェアを分散ログのエントリとして Raft と同様の AppendEntries RPC で送信する。このエントリがコミット (4.3.2 節) された瞬間、Leader が一時保存していた仮想ステートの該当ログエントリは、**可及的速やかに消去**される必要がある。

#### 4.3.1 未配送分散ログの再送

Raft において、Leader が保存していて Follower が保存していないログは都度再送される。Raft では Leader が自分で保存しているログをそのまま再送すればよいが、PLP-Raft ではタイミングによって以下の 2 パターンに処理が分かれる。

- **コミット前**

コミット前は、Leader が仮想ステートのログを自身の揮発性ステートに一時保存しているため、そこからシェアを再生成して再送する。

- **コミット後**

コミット後は、仮想ステートのログを直接保存しているサーバーが存在しないため、Leader が以下の手順を踏んでシェアを再生成する必要がある。

まず、Leader は仮想ステートのログを復元するために、該当の Follower を除くすべての Follower の FetchLog RPC を呼び出す。FetchLog RPC はこのために生み出された新しい RPC 定義である。Leader はこれが早く帰ってきた順に値を読取り、自分が持っている分散ログエントリと合わせて $k$ 個以上のシェアが集まった段階で仮想ステートのログエントリを復元し、再度シェアを生成して配布する。

FetchLog RPC

**引数**

- *term*: Leader の term
- *index*: 欲しい分散ログの index

**結果**

- *term*: 呼び出し先 Follower の term。Leader が新しい term を検知する用途
- *success*: 該当する分散ログエントリが存在すれば true
- *entry*: 該当する分散ログエントリ

**レシーバーの実装**

(1) $term < currenTerm$ なら $success = false$ と空の *entry* で応答
(2) 該当するエントリが自分の元に存在しなければ、$success = false$ と空の *entry* で応答
(3) 該当するエントリを取得して *entry* にセットし、$success = true$ で応答

Leader はこの手順によって再生成されたシェアを、AppenEntries RPC に対して true が返ってくるまで自身の揮発性ステートに一時保存する責任を負う。AppendEntries RPC で true が返ってきて分散ログが正確に転送されたことが確認できたら、Leader が一時保存していた再計算済みシェアは、**可及的速やかに消去**される必要がある。

#### 4.3.2 コミット

Raft においては、過半数のサーバーにログが適切に複製されたとき、そのログがコミット (確定) される。Raft はコミットされたログに関しては耐久性があり、いずれ全ての動いている State Machine[9] で実行されることを保障する。リーダー選挙の制約として、他のサーバーがもつ全て

© 2025 Information Processing Society of Japan

の Log をもっていることが Leader になる条件として定義されており、これによって Leader の疎通性が取れなくなった場合も新しい Leader が常にコミットされたログをもっていることが保障できる。

一方、PLP-Raft においては、Leader だけで正確な (仮想) ログをもつことが不可能であるため、クラスタ全体で安全にコミットされたログを保持するための責任が Follower にもある。したがって、PLP-Raft のコミットは以下のような形をとることで、コミット直後に Leader のみ疎通性がとれなくなった場合でも分散ログを復旧できる状態を作り、コミットされたログの転送状況によって障害耐性 (6.2 節) を動的に変える、という構造をとる。

PLP-Raft におけるコミット

PLP-Raft において、ある index の分散ログエントリが
$max($ サーバーの過半数 $, k+1)$ 台
に転送されたことが確認できたとき、そのエントリはコミットされる。

また前述のとおり、エントリがコミットされたら、Leader が一時保存していた仮想ステートの該当ログエントリは、**可及的速やかに消去**される必要がある。

#### 4.3.3 ログ転送の安全性と検証の必要性

本節では、分散ログ転送の最中に分散ログから仮想ステートのログを復元する、検証操作の必要性について説明する。

結論として、PLP-Raft における分散ログは検証操作を挟む必要がない。Raft[7] では、Leader がログ管理に関する全ての責任を負うという特徴と、シンプルな一貫性チェックによって Log Matching Property が保障されている。

Log Matching Property

- もし異なる 2 つのログが同じ index と term のエントリをもっていたら、その 2 つのエントリは同じ操作である。
- もし異なる 2 つのログが同じ index と term のエントリをもっていたら、それより昔のログエントリも全く同じである。

Raft の強固なリーダーシップモデルと転送の仕組みに従ってログを転送しているため、PLP-Raft でも同じ Log Matching Property が保障される。Log Matching Property が成り立ち、かつビザンチンノードが存在しない前提の上では、それぞれのログエントリには Leader が生成したシェアがそのまま配置されていることが保障できる。

これらの理由により、PLP-Raft における分散ログは不正なシェアが混入することがなく、検証操作が不要である。

### 4.4 ステートの読み出し

PLP-Raft に保存されているステートを読み出す際、読み出しをリクエストされたノードは自分以外の全てのノードに対して FetchState RPC を呼び出す。FetchState RPC は、呼び出し元が指定した commitIndex までの分散ログを State Machine[9] で計算した分散ステートを取得するための、新しい RPC 定義である。ステートを読み出すノードはこれが早く帰ってきた順に値を読み取り、自分がもっている分散ステートと合わせて $k$ 個以上のシェアが集まった段階で仮想ステートを復元し、読み出し結果として値を返す。

FetchState RPC

**引数**

- *term*: 呼び出し元の term
- *commitIndex*: 呼び出し元の commitIndex
- *arg(optional)*: ステートの中で特定の値を取り出すためのオプショナルな引数

**結果**

- *term*: 呼び出し先の term
- *success*: 該当する分散ステートが計算できれば true
- *state*: 該当する分散ステート

**レシーバーの実装**

(1) $term < currenTerm$ なら $success = false$ と空の *state* で応答。
(2) 引数の $commitIndex >$ 自らの $commitIndex$ なら $success = false$ と空の *entry* で応答。
(3) 自身の持っているログの中で引数の $commitIndex$ までの index のログを State Machine で処理し、計算された分散ステートを *state* にセットし、$success = true$ で応答。

## 5. 実装評価

本節では、4 節で提案した PLP-Raft の実装評価に関して、評価環境と評価結果を説明する。

### 5.1 評価環境

Raft と PLP-Raft アルゴリズムをそれぞれ Go 言語によって実装した。通信のばらつきによる影響を抑えるため、どちらも 1 つの実行ファイルから複数のスレッドを立ち上げてサーバーとみなし、プロセス間通信 (Go の channel) を経由して RPC 呼び出しを行う形とした。

今回は簡易実験として、5 サーバー (スレッド) に固定し、全てのスレッドを CPU 1 コア、メモリ 2GB に制限したコンテナの中で起動することで条件をそろえることとした。

© 2025 Information Processing Society of Japan

**表 1** 処理時間の平均と分散（処理回数：100、単位：ナノ秒）

| | 書き込み | | 読み出し | |
|---|---|---|---|---|
| | 平均 | 分散 | 平均 | 分散 |
| Raft | 10179.12 | 248839319 | 24.16 | 42.1944 |
| PLP-Raft | 181073.61 | 265776566.4 | 41574.75 | 86595202.15 |

### 5.2 評価結果

今回の実験では、Raft と PLP-Raft の両方で 100 バイトの文字列を 100 個エントリとして書き込み、それを再度 1 つずつ読み込んで、各オペレーションにかかった時間を計測し、それらの平均と分散を求めて処理のレイテンシを評価することとした。PLP-Raft のしきい値 k は 3 とした。

実際に計測した結果は表 1 のとおりである。全て単位はナノ秒。

## 6. 考察

本節では、本システムが目指す安全性指標である部分的漏洩耐性と障害耐性、および 5 節の実装評価結果について考察する。

### 6.1 部分的漏洩耐性

PLP-Raft の部分的漏洩耐性 $L$ は、3.1 節で定義したとおり、「侵害されても機密情報が漏洩しない最大のサーバー台数」である。この値は、$(k, n)$ しきい値秘密分散の $k$ を使って以下の数式であらわすことができる。

PLP-Raft の部分的漏洩耐性

$$L = k - 1$$

### 6.2 障害耐性

PLP-Raft の障害耐性 $R$ は、3.2 節で説明したとおり、評価指標として典型的に用いられている「許容できる最大の故障台数」で評価する。この値は、$(k, n)$ しきい値秘密分散の $k$、$n$(全体サーバー台数)、Leader がある時点でコミットした分散ログエントリをもつサーバー台数 $l$(4.3.2 節で定めたコミットルールにより、$k + 1 < l <= n$) を使って以下の数式であらわすことができる（$\lfloor x \rfloor$ は $x$ を超えない最大の整数を返す床関数とする）。

PLP-Raft の部分的漏洩耐性

$$R = min(\lfloor (n-1)/2 \rfloor, l - k)$$

すなわち、故障したサーバーの台数がサーバー全体の過半数に満たないか、その時点でコミットされたログをもつサーバーが $k$ 台確保できる状態であれば、障害からの復旧が可能である。$l$ は時間によって変動する値なので、$R$ は時間によって 1 から $\lfloor (n-1)/2 \rfloor$ までの間で変化する。

### 6.3 処理効率

通信によるレイテンシがほぼ最小化されている分、秘密分散によるオーバーヘッドが顕著にあらわれた。書き込みに関しては一回あたり 0.17 ミリ秒の増加、読み出しに関しては一回あたり 0.04 ミリ秒の増加がみられた。

しかし、100 バイトのエントリに対して 1 ミリ秒に満たないほどのオーバーヘッドなのであれば、実際の通信遅延を考慮した際にはかなり実用的な値が出ていると言える。実際に離れた地点にサーバーを配置した実験は今後の課題としたい。

## 7. まとめ

現在広く使われている合意アルゴリズム Raft は、全てのサーバーに同じログが複製・保存されるため、任意のノードが侵害されれば機密情報が漏洩してしまう恐れがあり、構造的に機密性の問題を抱えている。etcd 等の Raft を組み込んだ既存のプロダクトは、TLS 通信や証明書認証などを備えて情報漏洩を防いでいるが、ノードへの侵害や物理アクセスを伴うメモリダンプが行われた場合、データの奪取や改ざんには対抗できない。このようなノードへの侵害や物理アクセスを伴うメモリダンプによる情報漏洩を防ぐアプローチとして暗号化があるが、大半の暗号化は複雑な鍵管理を要し、運用次第で結局漏洩耐性が上がらないケースも多く存在する。

本稿では、上記の問題を鍵管理無しで解決するために、データを分散保存しているサーバーのうち、しきい値未満の台数のサーバーが侵害されても元のデータが復元できない性質「部分的漏洩耐性」を定義した。そして、Raft の派生アルゴリズムとして $(k, n)$ しきい値秘密分散法を利用して部分的漏洩耐性をもたせた Partially Leak-Proof Raft Consensus Algorithm (PLP-Raft) を提案し、実装評価によりその性能を評価した。その結果、PLP-Raft は、この部分的漏洩耐性を現実的なオーバーヘッドで実現できることが明らかになった。

サーバーの動的な構成変更や Log の圧縮等、Raft が対応しているが PLP-Raft が対応できていない部分や、異なる秘密分散法を用いた場合のオーバーヘッドの評価、よりプロダクションに近い環境での実験及びデータ収集に関しては今後の課題である。

**謝辞:** 本研究は JSPS 科研費 JP24K14951 の助成を受けたものです。

### 参考文献

[1] Fenhua Bai, Fushuang Li, Tao Shen, Kai Zeng, Xiaohui Zhang, and Chi Zhang. Rabft: an improved byzantine fault tolerance consensus algorithm based on raft. *J. Supercomput.*, 80(14):21533–21560, June 2024.

[2] G. R. BLAKLEY. Safeguarding cryptographic keys. In

© 2025 Information Processing Society of Japan

*1979 International Workshop on Managing Requirements Knowledge (MARK)*, pages 313–318, 1979.

[3] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[4] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[5] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[6] National Institute of Standards, Technology (NIST), Morris J. Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence E. Bassham, E. Roback, and James Dray Jr. Advanced encryption standard (AES), 2001-11-26 00:11:00 2001.

[7] Diego Ongaro and John K. Ousterhout. In search of an understandable consensus algorithm. In Garth Gibson and Nickolai Zeldovich, editors, *Proceedings of the 2014 USENIX Annual Technical Conference, USENIX ATC 2014, Philadelphia, PA, USA, June 19-20, 2014*, pages 305–319. USENIX Association, 2014.

[8] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[9] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Comput. Surv.*, 22(4):299–319, December 1990.

[10] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[11] Siben Tian, Fenhua Bai, Tao Shen, Chi Zhang, and Bei Gong. Vssb-raft: A secure and efficient zero trust consensus algorithm for blockchain. *ACM Trans. Sen. Netw.*, 20(2), January 2024.

© 2025 Information Processing Society of Japan