ソラナ:高性能ブロックチェーン v0.8.13 のための新しいアーキテクチャ

アナトリー・ヤ
コヴェンコ
anatoly@solana.io

法的免責事項このホワイトペーパーの何も、トークンを販売するオファー、または購入の申し出の勧誘です。Solanaは、一般の人々からのフィードバックやコメントを受け取るためだけに、このホワイトペーパーを公開しています。Solanaがトークン(または将来のトークンのための単純なAグリーメント)を販売することを申し出る場合、開示文書やリスク要因を含む決定的な提供文書を通じてそうします。これらの決定的な文書には、現在のバージョンとは大きく異なる可能性があるこのホワイトペーパーの更新版も含まれると予想されます。ソラナが米国でこのようなオファリングを行った場合、このオファーは認定投資家だけが利用できる可能性が高くなります。

このホワイトペーパーの中には、Solanasビジネスやトークンがどのように発展するか、またはトークンの有用性または価値を保証または約束するものとして扱われるべきではありません。このホワイトペーパーは、その裁量で変更される可能性のある現在の計画を概説し、その成功は、データおよび暗号通貨業界内の市場ベースの要因や要因など、ソラナスの管理外の多くの要因に依存します。将来の出来事に関する記述は、このホワイトペーパーに記載されている問題のソラナス分析のみに基づいています。その分析は間違っていることが判明するかもしれません。

要約

本論文では、歴史実証(PoH)に基づく新しいブロックチェーンアーキテクチャを提案する。 PoHは、元帳に信頼のない時間の経過をエンコードするために使用されます - 追加のみのデータ構造。仕事の証明(PoW)やステークの証明(PoS)などのコンセンサスアルゴリズムをアルオンサイドで使用すると、PoHはビザンチンフォルトトル-エラント複製ステートマシンのメッセージングオーバーヘッドを削減することができ、その結果、インは2秒以下の最終回時間を生み出します。この論文はまた、PoH台帳の時間キ

ーピングプロパティを活用する2つのアルゴリズムを提案しています - 任意のサイズのパーティションから回復できるPoSアルゴリズムと効率的なストリーミングレプリカプルーフ-tion(PoRep)。PoRep と PoH の組み合わせは、時間 (発注) と stor-age に関して元帳の偽造に対する防御を提供します。このプロトコルは 1 gbps のネットワークで分析され、このホワイト ペーパーは、現在のハードウェアでは毎秒最大 710k トランザクションのスループットが可能であることを示しています。

1　紹介

ブロックチェーンは、フォールト トレラントなレプリケートステートマシンの実装です。現在の一般に公開されているブロックチェーンは、時間に依存したり、参加者が時間を保つ能力について弱い仮定をしたりしません [4, 5].ネットワーク内の各ノードは、通常、ネットワーク内の他の参加者のクロックを知らなくても、独自のローカルクロックに依存します。信頼できる時刻ソースがない場合、メッセージのタイムスタンプを使用してメッセージを受け入れるか拒否しても、ネットワーク内の他のすべての参加者がまったく同じ選択をするという保証はありません。ここで紹介する PoH は、検証可能な時間の経過、すなわちイベントとメッセージの順序付けの間の期間を持つ元帳を作成するように設計されています。ネットワークのすべてのノードは、信頼せずに元帳に記録された時間の経過に依存することが期待されます。

2　概要

この記事の残りの部分は、次のように構成されています。システム全体の設計については、セクション3を参照してください。歴史の証明の詳細については、セクション 4で説明します。提案されたステークコンセンサスアルゴリズムの証明の詳細な説明は、セクション5で説明されています。 t の詳細については、彼が提案した高速レプリケーションの証明のセクション 6 で説明します。システムアーキテクチャとパフォーマンスの制限は、セクション7で分析されます。高性能GPUフレンドリーなスマート契約エンジンは、セクション7.5で説明されています

3　ネットワーク 設計

図 1に示すように、システムノードがリーダーとして指定され、履歴の証明シーケンスが生成され、ネットワーク global 読み取り一貫性と検証可能な時間の経過を提供します。リーダーは、ユーザー　・メッセージをシーケンスし、システム内の他のノードで効率的に処理できるように順序付けし、スループットを最大化します。RAM に格納されている現在の状態でトランザクションを実行し、トランザクションと最終状態の署名を Verifiers と呼ばれるレプリケーション ノードに公開します。検証者

は、状態のコピーに対して同じトランザクションを実行し、確認としてstateの計算された署名を pub-lishします。公開された確認は、コンセンサスアルゴリズムの投票として機能します。

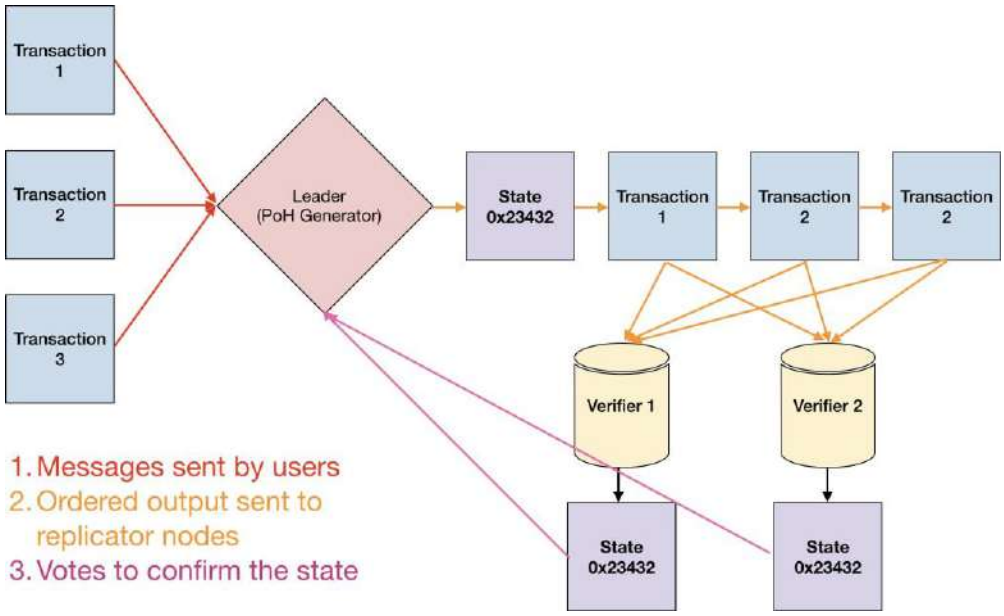


図1:ネットワーク全体のトランザクションフロー

パーティション化されていない状態では、任意の時点で、ネットワーク内にリーダーが1 つ存在します。各検証ツールノードはリーダーと同じハードウェア機能を持ち、リーダーとして選出できます。提案されたPoSアルゴリズムの選挙は、セクション5.6で詳細に説明されています。 CAP定理に関しては、パーティションのイベントにおいて、ほとんどの場合、整合性はAvail-能力よりも優先されます。大きなパーティションの場合、本論文では、ネットワークの制御をパーティションから回復するメカニズムを提案する。
任意のサイズ。これについては、セクション5.12で詳しく説明します。

4 歴史の証明

歴史の証明は、2 つのイベント間の時間の経過を暗号で検証する方法を提供できる計算のシーケンスです。これは、出力が入力から予測することができないように書かれたcryp-toグラフィカルに安全な関数を使用し、出力を生成するために完全に実行する必要があります。 この関数は、1 つのコアでシーケンスで実行され、その前の出力は

現在の入力は、定期的に現在の出力を記録し、その呼び出し回数。 出力は、別のコア上の各シーケンスセグメントをチェックすることにより、外部コンピュータによって並列に再計算および検証することができます。データ (またはデータのハッシュ)を関数の状態に追加することで、データをこのシーケンスにタイムスタンプすることができます。シーケンスに追加された状態、インデックス、およびデータの記録は、データが以前に作成されたことを保証するタイムスタンプを提供します。次のハッシュはシーケンス内で生成されました。この設計では、複数のジェネレータが互いのシーケンスに状態をミックスすることによって互いに同期できるため、水平方向のスケーリングもサポートされます。水平方向のスケーリングは、以下で深く議論されています。セクション 4.4

4.1 形容

システムは次のように動作するように設計されています。関数を実行しないと出力を予測できない暗号ハッシュ func-tion (例: sha256、ripemd など) を使用して、ランダムな開始値から関数を実行し、その出力を受け取り、同じ関数に入力として渡します。function が呼び出された回数と、各呼び出しでの出力を記録します。選択された開始ランダム値は、その日のニューヨーク時間の見出しのような任意の文字列である可能性があります。

たとえば、次の例を参照してください。

PoH シーケンス

インデックス	操作	出力 ハッシュ
1	sha256(「任意の ランダムな 開始値」)	ハッシュ1
2	sha256(ハッシュ1)	ハッシュ2
3	sha256(ハッシュ2)	ハッシュ3

ここで、hashN は実際のハッシュ出力を表します。

一度に、一部のシャッシュとインデックスを公開する必要があります。

たとえば、次の例を参照してください。

PoH シーケンス

インデックス	操作	出力 ハッシュ
1	sha256(「任意の ランダムな 開始値」)	ハッシュ1
200	sha256(ハッシュ199)	ハッシュ200
300	sha256(ハッシュ299)	ハッシュ300

選択されたハッシュ関数が衝突に強い場合、このハッシュのセットは、1 つのコンピューターススレッドで連続して計算できます。これは、実際に開始値からアルゴリズムを300回実行せずに、インデックス300のハッシュ値が何であるかを予測する方法が存在しないという事実から続きます。したがって、インデックス0とインデックス300の間でリアルタイムが渡されたことをデータ構造から推測することができます。

図2の例では、カウント510144806912でハッシュ62f51643c1が生成され、ハッシュc43d862d88がカウント510146904064で生成されました。前述のPoHアルゴリズムのプロパティに従って、カウント510144806912とカウント510146904064の間に渡されたリアルタイムを信頼できます。

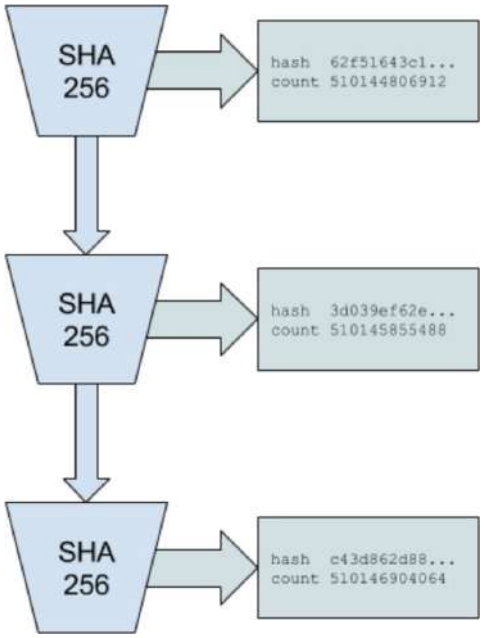


図2:歴史の証明シーケンス

4.2 イベントのタイムスタンプ

この一連のハッシュは、特定のハッシュインデックスが生成される前にデータが作成されたことを記録するためにも使用できます。'combine' 関数を使用して、現在のインデックスの現在のハッシュとデータを結合します。データは、任意のイベント データの暗号化された一意のハッシュにできます。結合関数は、データの単純な追加、または衝突抵抗ant である任意の操作です。挿入されます。

たとえば、次の例を参照してください。

PoH シーケンス

インデックス	操作	の出力 ハッシュ
1	sha256(「任意の ランダムな 開始 値」)	ハッシュ1
200	sha256(ハッシュ199)	ハッシュ200
300	sha256(ハッシュ299)	ハッシュ300

写真が撮影された、または任意のデジタルデータが作成されたなどの外部イベントが発生します。

データを含むPoHシーケンス

インデックス	操作	出力 ハッシュ
1	sha256(「任意の ランダムな 開始 値」)	ハッシュ1
200	sha256(ハッシュ199)	ハッシュ200
300	sha256(ハッシュ299)	ハッシュ300
336	sha256(追加(ハッシュ335、 写真 sha256))	ハッシュ336

Hash336 は、写真の hash335 およびsha256の追加されたバイナリ データから計算されます。写真のインデックスとsha256は、シーケンス出力の一部として記録されます。したがって、このシーケンスを検証するユーザーは、この変更を se quenceに再作成できます。検証は、引き続き並行して行うことができますが、セクション4.3で説明されています。

初期プロセスは依然として連続的であるため、シーケンスに入力された処理は、将来のハッシュ値が計算される前に発生している必要があることを知ることができます。

POH シーケンス

インデックス	操作	出力 ハッシュ
1	sha256(「任意の ランダムな 開始 値」)	ハッシュ1
200	sha256(ハッシュ199)	ハッシュ200
300	sha256(ハッシュ299)	ハッシュ300
336	sha256(追加(ハッシュ335、写真1 sha256))	ハッシュ336
400	sha256(ハッシュ399)	ハッシュ400
500	sha256(ハッシュ499)	ハッシュ500
600	sha256(追加(hash599、写真2 sha256))	ハッシュ600
700	sha256(ハッシュ699)	ハッシュ700

表1:2つのイベントを伴うPoHシーケンス

表1に示されるシーケンスでは、hash600より前に写真2が作成され、写真1はhash336より前に作成されました。データを一連の一連のハッシュに挿入すると、シーケンス内の後続のすべての値が変更されます。使用されるハッシュ関数が衝突に強く、データが追加されている限り、シーケンスにどのデータがすりおろされるかの事前知識に基づいて、将来のシーケンスを事前に計算することは計算不可能であるべきです。

シーケンスに混在するデータは、生データ自体、またはメタデータを伴うデータのハッシュです。

図3の例では、cfd40df8.を入力します。は、歴史の証明シーケンスに挿入されました。挿入された count は510145855488され、挿入された状態は 3d039eef3 です。将来のジェンエレートされたハッシュはすべて、この変化によってシーケンスに変更され、この変化は図中の色の変化によって示される。

このシーケンスを観察するすべてのノードは、すべてのイベントが挿入された順序を決定し、挿入間のリアルタイムを推定できます。

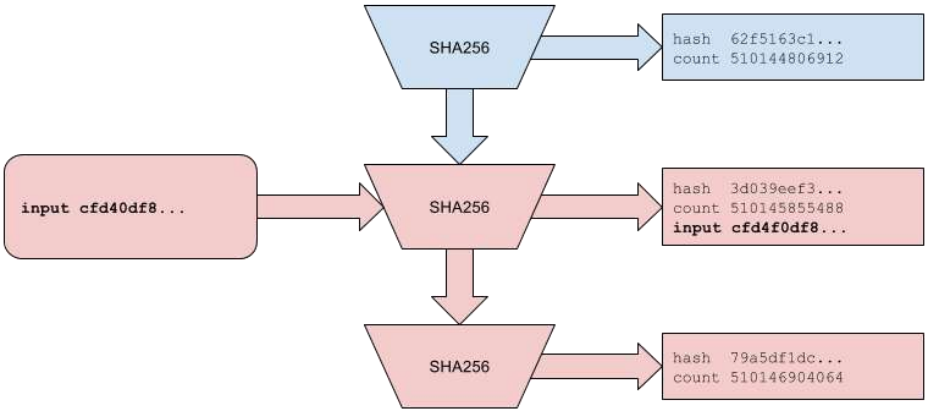


図3: 履歴証明へのデータの挿入

4.3 検証

シーケンスは、マルチコアコンピュータで、生成に要した時間よりも大幅に短い時間で正しいことを確認できます。

たとえば、次の例を
参照してください。

コア 1		
インデックス	データ	出力ハッシュ
200	sha256(ハッシュ199)	ハッシュ200
300	sha256(ハッシュ299)	ハッシュ300

コア 2		
インデックス	データ	出力ハッシュ
300	sha256(ハッシュ299)	ハッシュ300
400	sha256(ハッシュ399)	ハッシュ400

4000コアを持つ最新のGPU　のようないくつかの数のコアを考えると

、検証ツールはハッシュのシーケンスとそのインデックスを 4000 スライスに分割し、並行して各スライスが最初から正しいことを確認できます。スライス内の最後のハッシュにハッシュします。シーケンスを生成する予定の時間は、次のようになります。

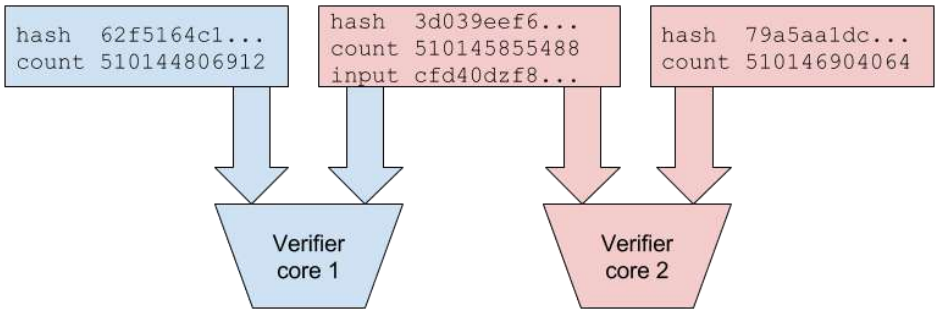


図4:複数のコアを使用した検証

$$\frac{\text{1 コアの1秒あたりのハッシュの合計数}}{}$$

シーケンスが正しいことを確認する予定時間は、次のようになります。

$$\frac{\text{合計のハッシュ数}}{\text{(コアあたりの 1 秒あたりのハッシェ *検証可能なコアの数)}}$$

図4の例では、各コアがシーケンスの各スライスを並列に検証できます。すべての入力文字列が出力に記録され、カウンターと追加された状態が付けられているため、検証者は各スライスを並列に複製できます。赤い色付きのハッシュは、データ挿入によってシーケンスが変更されたことを示します。

4.4 水平スケーリング

各ジェネレータから互いのジェネレータにシーケンス状態を混合することによって、複数の歴史実証gエネレータを同期させ、したがって、歴史証明ジェネレータの水平スケーリングを達成することが可能です。このスケーリングは、シャーディングなしで行われます。両方のジェネレータの出力は、システム内のイベントの全順序を再構築するために必要です。

ポージェネレータ A インデックス　ハッシュ　データ		
1	ハッシュ1a	
2	ハッシュ2a	ハッシュ1b
3	ハッシュ3a	
4	ハッシュ4a	

ポージェネレータ B インデックス　ハッシュ　データ		
1	ハッシュ1b	
2	ハッシュ2b	ハッシュ1a
3	ハッシュ3b	
4	ハッシュ4b	

ジェネレータ A と B を与えると、A は B (hash1b) からデータ パケットを受け取ります。 ジェネレータBの最後の状態と最後の状態ジェネレータを含む Bはジェネレータから観測される。ジェネレータ A の次の状態ハッシュは、次に de- ジェネレータB、sから状態をペンプ私たちはそのhash1bが起こったことを導き出すことができます いつか hash3aの前に。このプロパティは推移的な場合がありますので、3人のジーナー- アトール アール シンクロナイズド 通じて a´単`コモンジェネレータ *ある B C,* A と C の間の依存関係は、たとえそうではないにもかかわらずトレースできます。 彼のnchronized 直に。

ジェネレータを定期的に同期させることで、各ジェネレータは外部トラフィックの一部を処理できるため、システム全体が、ジェネレータ間のネットワーク遅延による真の時間精度を犠牲にして、より大量のイベントを処理できます。ハッシュ自体の値など、同期ウィンドウ内のイベントを順序付けする決定的関数を選択することで、グローバルな順序を実現できます。

図5では、2 つのジェネレータが出力状態を相互に挿入し、操作を記録します。色の変化は、ピアからのデータがシーケンスを mod- で整化したことを示します。各ストリームに混合された生成されたハッシュは太字で強調表示されます。

同期は推移的です。*ある B C*指定可能な注文があります の イベント 間 A そして C 通じて B.　　‘‘‘

このようにスケーリングすると、可用性が低下しまx す。10 1 gbps のコネク- 取り引き で 可用性 の 0.999 だろう 有る $0.999^{10} = 0.99$ 可用性。

4.5　一貫性

ユーザーは、生成された se-quence の一貫性を強制し、入力に有効と見なされるシーケンスの最後の観測出力を挿入することによって、攻撃に対して耐性を持たせると期待されます。

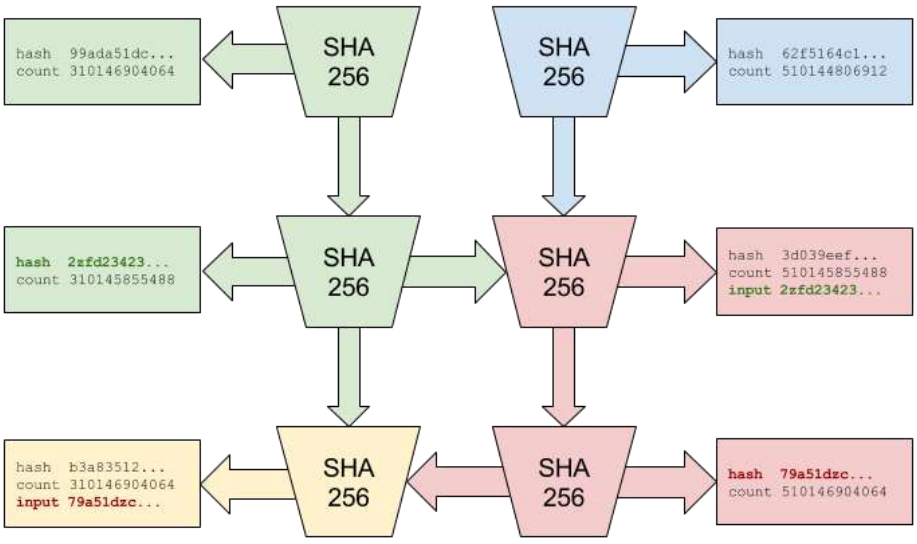


図5: 2 つのジェネレータの同期

インデックス	ポフ データ	シーケンスある力 ハッシュ
10		ハッシュ10a
20	イベン	ハッシュ20a

インデックス	ポフ データ	隠れたシーケンスB力 ハッシュ
10		ハッシュ10b
20	イベ	ハッシュ20b

悪意のあるPoH ジェネレータは、すべてのイベントに一度にアクセスできる場合、またはイベントが逆順に並んだ 2番目の隠しシーケンスを生成する可能性があります。 より高速な隠されたシーケンス。

この攻撃を防ぐには、クライアントが生成した各イベント自体に、有効なシーケンスと見なされるクライアントが確認した最新のハッシュを含める必要があります。したがって、クライアントが"Event1"データを作成する際には、最後に観察したハッシュを追加する必要があります。

PoH シーケンス A

インデックス	データ	出力 ハッシュ
10		ハッシュ10a
20	イベント1 = 追加(イベント1データ、ハッシュ10a)	ハッシュ20a
30	イベント2 = 追加(イベント2データ、ハッシュ20a)	ハッシュ30a
40	イベント3 = 追加(イベント3データ、ハッシュ30a)	ハッシュ40a

シーケンスが公開されると、Event3 は hash30a を参照し、このイベントの前のシーケンス内に存在しない場合、シーケンスのコンシューマーは無効なシーケンスを認識します。その後、部分的な順序変更攻撃は、クライアントがイベントを監視している間、およびイベントが入力されたときに生成される h 灰の数に制限されます。クライアントは、最後に観測されたハッシュと挿入されたハッシュの間の短いハッシュの順序が正しいと仮定しないソフトウェアを書くことができるはずです。

悪意のある PoH ジェネレータがクライアントの Event ハッシュを書き換えることを防ぐために、クライアントは、データだけでなく、イベント データと最後に観察されたハッシュの署名を送信できます。

PoH シーケンス A

インデックス	データ	出力 ハッシュ
10		ハッシュ10a
20	イベント1 = 符号(append(イベント1データ、ハッシュ10a) クライアント 秘密 キー)	ハッシュ 20a
30	イベント2 = 符号(追加(イベント2データ、ハッシュ20a) クライアント 秘密 キー)	ハッシュ 30a
40	イベント3 = 符号(追加(イベント3データ、ハッシュ30a) クライアント 秘密 キー)	ハッシ

ュ40a

このデータの検証には、署名の検証と、このデータの前のハッシュのシーケンスでハッシュの検索が必要です。
確かめる：

(署名、公開鍵、ハッシュ30a、イベント3データ)=
イベント3 検証(署名、公開キー、イベント3)
ルックアップ(ハッシュ30a、 ポックシーケンス)

図 6では、ユーザーが指定した入力はハッシュ0xdeadbeefに依存しています。
が挿入される前に生成されたシーケンスに存在します。 青

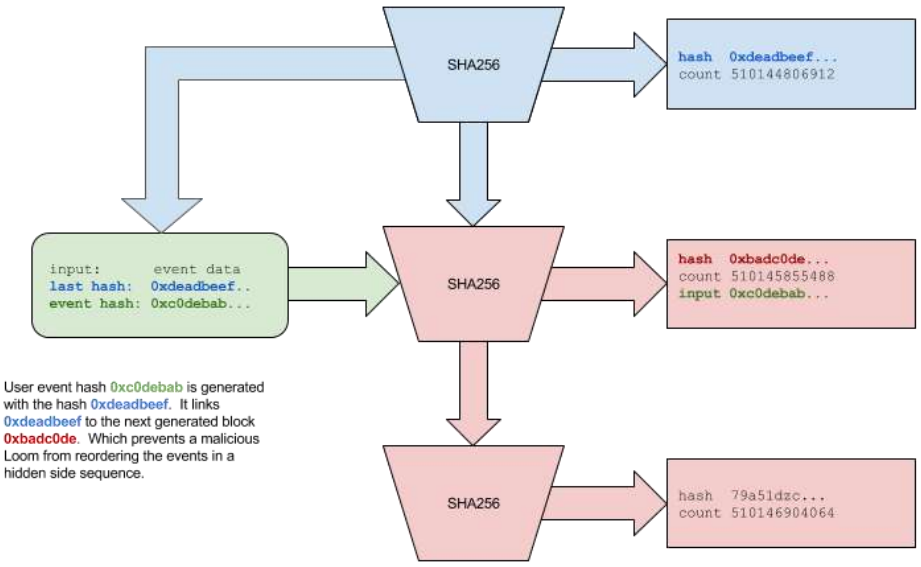


図6: バックリファレンスを使用した入力

左上の矢印は、クライアントが以前に生成されたハッシュを参照していることを示します。クライアント メッセージは、ハッシュ 0xdeadbeefを含むシーケンスでのみ有効です。シーケンスの赤い色は、シーケンスがクライアントデータによって変更されたことを示します。

4.6 オーバーヘッド

毎秒 4000 のハッハは 160 キロバイトのデータをさらに生成し、4000 コアと約 0.25 ~ 0.75 ミリ秒の検証に必要な GPU へのアクセスが必要になります。

4.7 攻撃

4.7.1 逆転

逆の順序を生成するには、攻撃者は 2 番目のイベントの後に悪質なシーケンスを開始する必要があります。この遅延により、悪意のないピアツー ピアノードが元の順序について通信できるようになります。

4.7.2 速度

複数のジェネレータを使用すると、攻撃に対する展開の抵抗力が高まります。 1つのジェネレータは、高帯域幅であり、そのシーケンスにミックスする多くのイベントを受信することができ、別のジェネレータは、定期的に高帯域幅のジェネレータと混合高速低帯域幅である可能性があります。

高速シーケンスは、攻撃者が逆にする必要があるデータのセカンダリ シーケンスを作成します。

4.7.3 長距離攻撃

長距離攻撃には、古い捨てられたクライアントの秘密鍵を取得し、改ざんされた元帳を生成する[10]。歴史の証明は、長距離攻撃に対していくつかの保護を提供します。古い秘密キーにアクセスする悪意のあるユーザーは、偽造しようとしている元のキーと同じくらい多くの ti meを取る履歴レコードを再作成する必要があります。この場合、ネットワークが現在使用しているプロセッサよりも高速なプロセッサにアクセスする必要があります。

さらに、単一の時間ソースを使用すると、より単純なレプリケーションの証明の構築が可能になります(詳細については、セクション6を参照)。ネットワークの署名が解除されるため、ネットワーク内のすべての参加者は、イベントの履歴記録に依存します。

PoRep と PoH を組み合わせて、偽造された元帳に対して空間と時間の両方を防御する必要があります。

5 ステークコンセンサスの証明

5.1 形容

このステークの証明のインスタンスは、歴史証明ジェネレータによって生成された現在のシーケンスの迅速な確認、次の歴史証明生成器の投票と選択、および誤った振る舞いの検証器を処罰するために設計されています。このアルゴリズムは、特定のタイムアウト内に参加しているすべてのノードに最終的に到着するメッセージに依存します。

5.2 用語

社債は、労働証明の資本費に相当します。鉱夫はハードウェアと電気を購入し、それを仕事の証明ブロックチェーンの単一の支店にコミットします。 債券は、検証者が取引を検証している間に担保としてコミットするコインです。

削減ステークシステムの証明で何も問題に対して提案された解決策 [7]。別のブランチの投票のp屋根が公開されると、そのブランチはバリデータの結合を破壊することができます。 これは、検証者が複数の支店を確認するのを阻止するために設計された経済的インセンティブです。

スーパーマジョリティ 超多数派は⅔によって重み付けされたバリデーターのrds 絆。過半数の投票は、ネットワークが到達したことを示します コンセンサス そして で 最少 ⅓rd の ザ ネットワーク だろう 有る いた 宛先 表決 このブランチが無効になる可能性があります。これはtを置くだろう彼は経済的 費用 の ひとつの 攻撃 で ⅓rd の ザ 市場 頭 の ザ 硬貨。

5.3 ボンディング

ボンディング取引は、コインの量を取り、ユーザーのアイデンティティの下でボンディングアカウントに移動します。 ボンディング口座のコインは使用できず、ユーザーがそれらを削除するまでアカウントに残る必要があります。ユーザーは、タイムアウトした古いコインのみを削除できます。債券は、現在の利害関係者の過半数がシーケンスを確認した後に有効です。

5.4 投票

歴史証明ジェネレーターは、事前定義された期間に状態の署名を公開することが期待されます。各結合 ID は、その署名を署名済みの状態に発行して確認する必要があります。投票は、ノーなしで、単純なイエス投票です。

結合された ID の過半数がタイムアウト内に投票した場合、この分岐は有効として受け入れられます。

5.5　ボンディング解除

N個の投票数が欠落すると、コインは古く、投票の対象ではなくなります。ユーザーは、非ボンディングトランザクションを発行して、それらを削除できます。

Nは、アクティブな投票に対する古い値の比率に基づく動的な値です。古い票の数が増えると N が増加します。大規模なネットワーク パーティションの場合、大きなブランチは、より小さいブランチより速く回復することができます。

5.6　選挙

PoH ジェネレータの故障が検出されると、新しい PoH ジェネレータの選択が行われます。最大の投票権を持つバリデータ、またはタイがある場合は最も高い公開キーアドレスが新しいPoHジェネレータとして選択されます。

新しいシーケンスでは、確認の過半数が必要です。 過半数の確認が使用可能になる前に新しいリーダーが失敗した場合は、次に高い検証コントロールが選択され、新しい確認のセットが必要です。投票を切り替えるには、バリデーターがより高いPoHシーケンスカウンターで投票する必要があり、新しい投票には切り替えたい票を含める必要があります。それ以外の場合、2番目の投票はスラッシュ可能になります。 投票の切り替えは、スーパーを持たない高さでのみ発生するように設計することが期待されています

過半数。

PoH ジェネレータが確立されると、トランザクション処理の作業を引き継ぐためにセカンダリを選択できます。セカンダリが存在する場合、プライマリ障害時に次のリーダーと見なされます。

プラットフォームは、セカンダリがプライマリになり、例外が検出された場合や事前に定義されたスケジュールに基づいて下位ランクジェネレータが昇格するように設計されています。

5.7　選挙 のトリガー

5.7.1　歴史の証明ジェネレータ

PoHの発電機は生成されたse-量子に署名する同一性と設計されている。フォークは、PoH ジェネレータの ID が侵害された場合にのみ発生します

。同じPoH ID で 2 つの異なる履歴レコードが公開されているため、フォークが検出されます。

5.7.2 ランタイム 例外

ハードウェア障害、b ug、または PoH ジェネレータの意図的なエラーにより、無効な状態が生成され、その状態の署名が公開される可能性があります。 ローカル検証コントロールの結果と一致しません。バリデータはゴシップを介して正しい署名を公開し、このイベントは新しい選挙を引き起こすでしょう。無効な状態を受け入れる検証者は、債券を切り取ります。

5.7.3 ネットワーク タイムアウト

ネットワークタイムアウトは、選挙をトリガーします。

5.8 削減

スラッシュは、バリデータが 2 つの個別のシーケンスを処理するときに発生します。悪意のある投票の証拠は、流通から結合コインを削除し、マイニングプールにそれらを追加します。

競合するシーケンスに関する以前の投票を含む投票は、悪意のある投票の証拠として適格ではありません。この投票は、債券をスラッシュする代わりに、競合するシーケンスに対する現在の投票を削除します。

また、PoH ジェネレータによって生成された無効なハッシュに対して投票が行われた場合にもスラッシュが発生します。ジェネレータは、無効な状態をランダムに生成し、セカンダリへのフォールバックをトリガーすると想定されます。

5.9 二次 選挙

二次および低ランクの証明of歴史発電機を提案し、承認することができる。提案は、一次発電機のシーケンスにキャストされます。提案にはタイムアウトが含まれていますが、タイムアウト前に投票の過半数で動議が承認された場合、セカンダリーが選出されると見なされ、dが予定通りに職務を引き継ぐ。プライマリは、メッセージを生成されたシーケンスに挿入して、ハンドオーバーが発生することを示すか、無効な状態を挿入してネットワークをセカンダリにフォールバックさせることによって、セカンダリへのソフトハンドオーバを行うことができます。

セカンダリが選出され、プライマリが失敗した場合、セカンダリ

は選挙中の最初のフォールバックとみなされます。

5.10　可用性

パーティションを扱うCAPシステムは、整合性またはAvail-能力を選択する必要があります。最終的には可用性が選択されますが、時間の客観的な尺度があるため、一貫性は合理的な人間のタイムアウトで選択されます。

ステークの検証者の証明は、彼らが取引の特定のセットに投票することを可能にするステークでコインの一定量をノックアップします。コインのロックは、他のトランザクションと同様にPoHストリームに入力される取引です。投票するには、PoS 検証者は、PoH 元帳内の特定の位置にすべてのトランザクションを処理した後に計算された状態のハッシュに署名する必要があります。この投票は、PoH ストリームへのトランザクションとしても入力されます。PoH元帳を見ると、各投票の間に経過した時間と、パーティションが発生した場合、各検証者が使用できない期間を推測できます。

宛先 対処する で パーティション で 合理的 人間 時間枠、 私たち提供 利用不可の検証者に対する動的なアプローチ。 の数が 検証 です 高い そして 上 ², ザ 不取り₃過程 缶 いる 殆ど。 ザ 数利用不可になる前に元帳に生成する必要があるハッシュの数が低い 検証 杭 です 十分 未賭け そして 彼らが アール いいえ 長い カウント 対して コンセンサス。
検証方法の数が² rds より少ないですが、1 より上の場合、非ステイクタイマーはより遅く、不足する前に生成される³ハッシュの数²が多くなる
検証者は、賭けが無い。 大きなパーティションでは、欠落している₂パーティションのように ¹ 又は もっとその の ザ 検証 ザ 不取り 過程です すごく すごく 遅い。 トランザクション ストリームに入力することができ、検証者は投票できますが、フル ²rds コンセンサス非常に大量のハッシュが続くまで達成されない 生成され、利用不可検証者は、未だに取り付けられていた。違い で 時間 対して a ネットワーク 宛先 取り戻す 活性 により 私達 として 顧客 の ザ ネットワーク 人間 時間枠 宛先 摘む a 分割 それ 私たち 欲しい 宛先 続ける 使用。

5.11　回復

で ザ 制 私たち 提供 ザ 台帳 缶 いる 十分 回復 差出人 任意 失敗。それ手段 誰でも で ザ 世界 缶 摘む 任意 ランダム 点 で ザ 台帳 そして新しく生成されたハッシュとトランザクションを追加して、有効なフォークを作成します。 すべての検証機がこのフォークから欠落してい

³

る場合、それは非常に長い時間がかかります 時間 対して 任意 余分な 絆 宛先 成る 有効 そして 対して これ 枝 宛先 達成する ²rds超多数派短所続きます。 したがって、使用可能な検証コントロールがゼロの完全な回復 だろう 要する a すごく 大きい 量 の ハッシュ 宛先 いる 追加 宛先 ザ 台帳

使用できないすべての検証者が解かれて初めて、新しい債券は元帳を検証することができます。

5.12 きっぱり

PoHは、ネットワークの検証者が過去に起こったことを、それらのイベントの時間をある程度確実に観察することを可能にします。 PoH ジェネレータはメッセージのストリームを生成しているので、すべての検証者は 500 ミリ秒以内に状態の署名を提出する必要があります。この数は、ネットワークの状態によってさらに減らすことができます。各検証がストリームに入力されるため、ネットワーク内の全員が、実際に投票を直接観察することなく、必要なタイムアウト内にすべての検証者が投票を送信したことを検証できます。

5.13 攻撃

5.13.1 コモンズの悲劇

PoS 検証は、PoH gen-エレータによって生成された状態ハッシュを確認するだけです。彼らが仕事をしないし、単にすべての生成された状態ハッシュを承認するための経済的なインセンティブがあります。この状態を避けるには、PoH gener-ator はランダムな間隔で無効なハッシュを注入する必要があります。このハッシュの有権者は、スラッシュする必要があります。ハッシュが生成されると、ネットワークはすぐにセカンダリ選出PoHジェネレータを昇格させる必要があります。

各検証方法は、小さなタイムアウト (たとえば 500 ミリ秒) 内に応答する必要があります。タイムアウトは、悪意のある虚たなる検証者が別の検証者の投票を観察し、十分に速くストリームに自分の票を得る可能性が低いほど低く設定する必要があります.

5.13.2 PoHジェネレータとの癒着

PoH ジェネレータと結託している検証者は、無効なハッシュがいつ生成され、投票しないかを事前に知っています。このシナリオは、より大きな検証のステークを持つ PoH ID と実際に変わりません。PoH ジェネレータは、状態ハッシュを生成するためにすべての作業を行う必要があります。

5.13.3 検閲

検閲またはサービス拒否が発生する可能性があり3ます。 1債券保有者のrd 新しい結合を持つシーケンスの検証を拒否します。プロトコルは防御することができます この形態の攻撃に対して、結合がどれくらいの速さになるかを動的に調整する いつも。 で ザ 出来事 の a 拒否 の サービス ザ 大きい 分割 遺言 いる 設計 ビザンチン債券保有者をフォークし、検閲する。大規模なネットワークは再び 覆う として ザ ビザンチン 絆 成る いつも で 時間。 ザ 小さい ビザンチン 分割 だろう じゃない いる 有能 宛先 動く フォワード 対して a 長い 時代 の 時間 。

アルゴリズムは follo ws として機能します。ネットワークの過半数は、新しいリーダーを選出します。その後、リーダーはビザンチンの債券保有者の参加を検閲するだろう。 歴史の証明ジェネレータは、時間の経過を証明するために、時間の経過を証明するために、より大きなネットワークが超多数派を持つように十分なビザンチン債が古くなるまで、シーケンスを生成し続ける必要があります。債券が失効するレートは、アクティブな債券の割合に基づいて動的に行われます。したがって、ネットワークのB yzantine少数派フォークは、過半数を回復するために大多数のフォークよりもはるかに長く待たなければなりません。超多数派が設立されると、ビザンチン債保有者を永久に処罰するためにスラッシングを使用することができます。

5.13.4 長距離攻撃

PoHは長距離攻撃に対する自然な防御を提供します。過去の任意の時点から元帳を回復するには、攻撃者がPoH ジェネレータの速度を上回ることによって有効な元帳を時間内に追い越す必要があります。

コンセンサスプロトコルは、攻撃に時間がかかり、有効な検証コントロールをすべて取り消す時間がかかるため、第2の防御層を提供します。また、元帳の履歴に利用可能時間のギャップも作成されます。同じ高さの 2 つの元帳を比較する場合、最大パーティションが最小の元帳を客観的に有効と見なすことができます。

5.13.5 ASIC 攻撃

ASIC攻撃には、このプロトコルが存在する2つの機会があります- パートの間に、そしてFinalityで不正行為のタイムアウト。

パーティション中の ASIC 攻撃では、債券が取り込まれるレートは非線形であり、パーティションが大きいネットワークの場合、そのレートは ASIC攻撃から予想される gainsの桁違いです。

ファイナリティ中の ASIC 攻撃の場合、この脆弱性により、結合されたステークを持つビザンチン検証者は、他のノードからの確認を待ち、協力 PoH ジェネレータと票を注入することができます。 PoH gener- ator は、より高速な ASIC を使用して、より短時間で 500ms 相当のハッシュを生成し、PoH 生成器とコラボレーションノード間のネットワーク通信を可能にします。しかし、PoH ジェネレータもバイザンチンである場合、障害を挿入する予定の場合に、ビザンチン生成器が正確なカウンタを通信しなかった理由はありません。このシナリオは、PoH ジェネレータと、同じ ID を共有し、1つの結合された杭を持ち、1セットのハードウェアのみを使用するすべての共同作業者と変わりません。

6 レプリケーションのストリーミングの証明

6.1 形容

ファイルコインは、レプリケーションの証明のバージョンを提案しました [6].このバージョンの目標は、歴史の証明で時間を追跡することによって有効にされているレプリケーションの証明の高速かつストリーミング検証を持つことです生成されたシーケンス。レプリケーションはコンセンサスアルゴリズムとしては使用されませんが、ブロックチェーンの履歴や状態を高可用性で保存するコストを考慮するのに便利なツールです。

6.2 アルゴリズム

図7に示すように CBC 暗号化は、入力データを XOR に対して以前に暗号化されたブロックを使用して、各データブロックを se-quence で暗号化します。

各レプリケーション idid は、生成された証明の履歴シーケンスのハッシュに署名することによってキーを生成します。これは、リプリケータ iden-tity と特定の歴史証明シーケンスにキーを結び付けます。特定のハッシュのみを選択できます。(ハッシュ選択のセクション6.5を参照)

データ・セットはブロックごとに完全に暗号化されています。次に、証明を生成するために、キーを使用して、各ブロックからランダムな 32 バイトを選択する擬似乱数ジェネレーターをシードします。

選択した PoH ハッシュが各スライスの先頭に付加されたマークル ハッシュが計算されます。

ルートがキーと共に公開され、生成された選択したハッシュが発行されます。レプリケーションノードは、別の証明を公開する必要があります。

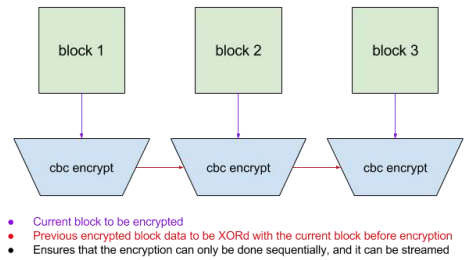


図7: 順次 CBC 暗号化

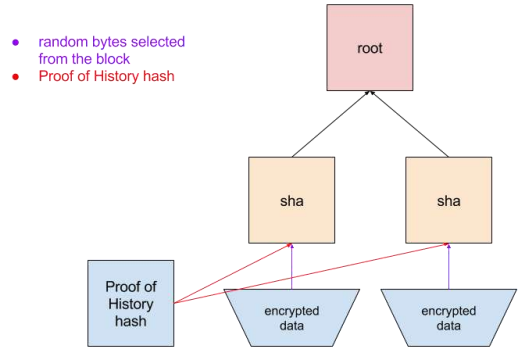


図8: レプリケーションの高速証明

彼らは歴史の証明ジェネレータによって生成されるようにNハッシュで、です 約[1] [2] データの暗号化にかかる時間。 の証明ヒストリジェネレータは、事前に定義された期間にレプリケーションの証明のための特定のハッシュを公開します。レプリケータ ノードは、証明を生成するために次に公開されたハッシュを選択する必要があります。ハッシュが署名され、ランダムスライスがブロックから選択され、マークルルートが作成されます。

N証明の期間が経過すると、データは新しいCBCキーで再暗号化されます。

6.3 検証

で N 色 各 コア 缶 川 暗号化 対して 各 同一性。 トータル 間 必須は 2 ですブロック *Ncores*前の暗号化ブロックが必要なため をクリックして次の 1 つを生成します。 各コアは、すべての 証拠 それ 派生 差出人 ザ 現在の 暗号化 ブロック。

証明を検証する合計時間は、暗号化にかかる時間と等しくなると予想されます。証明自体はブロックからランダムなバイトをほとんど消費しませんので、ハッシュするデータの量は暗号化されたブロックサイズよりかなり少ないです。同時に検証できるレプリケーション ID の数は、使用可能なコアの数と同じです。現代のGPUは3500以上を持っています

1 - [1]で、彼らに利用可能なコア [2] [3] CPU のクロック速度。

6.4 キー ローテーション

キーのローテーションを使用しない場合、同じ暗号化レプリケーションによって、複数の Proof of History シーケンスに対する安価な証明が生成されます。キーは定期的にローテーションされ、各レプリケーションは、一意の歴史証明に関連付けられた新しいキーで再暗号化されます。

回転は、CPU よりもコアあたりの速度が遅いGPU ハードウェア上のレプリケーション証明を検証するのが現実的であるほど低速である必要があります。

6.5 ハッシュ 選択

Proof of History ジェネレータは、レプリケーションの証明を暗号化するためにネットワーク全体で使用されるハッシュを公開し、バイト選択用

の擬似乱数ジェネレータとして使用します。 速い証拠で。

ハッシュ です 公開 で a 周期的 カウンター それ です 略 等しい 宛先 ザ 時間 それはかかる 暗号化する データ セット。 各レプリケーション 同一性 使用する必要があります 同じ

ハッシュを使用し、ハッシュの符号付き結果をバイト選択のシードとして使用するか、暗号化キーを使用します。

各レプリケータが証明を提供する必要がある期間は、暗号化時間よりも短くする必要があります。それ以外の場合、レプリケータは暗号化をストリーミングし、各証明のために削除できます。

悪意のあるジェネレーターは、このハッシュの前にシーケンスにデータを挿入して、特定のハッシュを生成する可能性があります。この攻撃については 5.13.2で詳しく説明しています。

6.6 証明 検証

[履歴の証明] ノードは、送信されたレプリケーション証明の証明を検証する必要はありません。レプリケータID によって送信された保留中および検証済みの証明の数を追跡することが期待されます。 レプリケータがネットワーク内の検証対象の大多数によって証明に署名できる場合、証明が検証されるはずです。

検証は p2p ゴシップネットワークを介してレプリケータによって収集され、ネットワーク内のバリデータの超大多数を含む 1 つのパケットとして送信されます。このパケットは、Historyの証明シーケンスによって生成される特定のハッシュ生成の前に、すべての証明を検証し、同時に複数のレプリケータ ID を含めることができます。

6.7 攻撃

6.7.1 スパム

悪意のあるユーザーは、多くのレプリケータ ID を作成し、不正な証拠を使用してネット作業をスパムする可能性があります。 より迅速な検証を容易にするために、ノードは検証を要求するときに、暗号化されたデータと全体のマークルツリーをネットワークの残りの部分に提供する必要があります。

このペーパーで設計されたレプリケーションの証明は、追加のスペースを取らないため、追加の証明を安価に検証することができます。しかし、各 ID は暗号化時間の 1 コアを消費します。レプリケーション ターゲットは、すぐに利用できるコアの最大サイズに設定する必要があります。3500+コアを搭載した現代のGPUが出荷されます。

6.7.2 部分的な 消去

レプリケータ ノードは、状態全体を格納しないように、データの一部を部分的に消去しようとする可能性があります。証明の数とランダム性 of

種はこの攻撃を困難にするはずです。

対して 例 a 利用者 入庫 1 テラバイト の データ 消去 a 単 バイト 差出人各 1 メガバイト ブロック。 A 単 証拠 それ サンプル 1 バイト アウト の 毎メガバイト だろう 有る a 可能性 の 衝突 で 任意 消去 バイト 1 (1 1/1, 000, 0000)^{1,000,000} = 0.63. 後 5 証拠 ザ 可能性 です 0.99.

6.7.3 PoHジェネレータとの癒着

符号付きハッシュは、サンプルのシードに使用される予定です。レプリケータが事前に特定のハッシュを選択できる場合、レプリケータはサンプリングされないすべてのバイトを消去できます。

Proof of Historyジェネレータと結託しているレプリケータID は、ランダムの定義済みハッシュの前にシーケンスの最後に特定のトランザクションを挿入する可能性があります。バイト選択が生成されます。十分なコアを持つ攻撃者は、レプリケータ ID よりも望ましいハッシュを生成する可能性があります。この攻撃は、単一のレプリケータIDにのみ利益をもたらします。すべての IDは、ECDSA (またはそれと同等)で暗号化された同じハッシュを使用する必要があるため、結果の署名は、各 Repli-Cator に対して一意です。アイデンティティ、および衝突耐性。単一のレプリケータ ID は、限界利益を持っています。

6.7.4 サービス拒否

レプリケータ ID を追加するコストは、ストレージのコストと等しくなると予想されます。すべてのレプリケータID を検証するために計算容量を追加するコストは、レプリケーションIDごとの CPU または GPU コアのコストと等しくなると予想されます。

これにより、多数の有効なレプリケータID を作成することにより、ネットワークに対するサービス拒否攻撃の機会が生じます。

この攻撃を制限するために、ネットワークに選択されたコンセンサス プロトコルは、レプリケーション ターゲットを選択し、ネットワークでの可用性、帯域幅、位置情報など、信頼性の高い特性を満たすレプリケーション証明を授与できます。

6.7.5 コモンズの悲劇

PoS検証ツールは、作業を行わずにPoRepを確認するだけで済む。経済的インセンティブは、仕事をするためにPoS検証者と並べられるべきである、

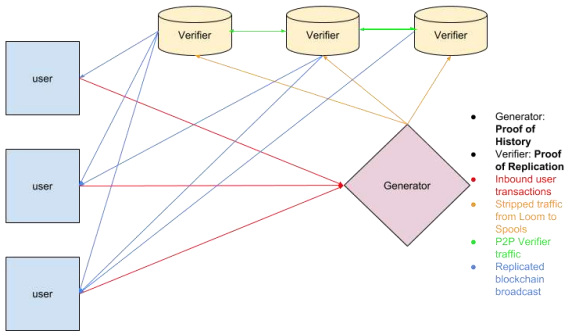


図9: システムアーキテクチャ

PoS 検証ツールと PoRepレプリケーションノード間でマイニング支払いを分割する場合と同様です。

このシナリオをさらに回避するために、PoRep検証者は、わずかな時間で誤った証明を提出できます。 偽データを生成した機能を提供することで、証明が誤っていることを証明できます。偽の証明を確認した PoS 検証ツールは、スラッシュされます。

7 システム アーキテクチャ

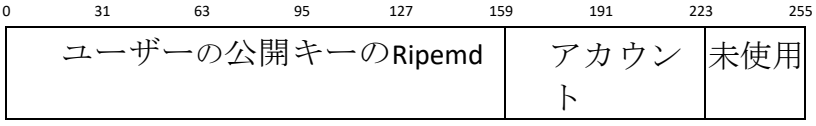
7.1 コンポーネント

7.1.1 リーダー、歴史の証明ジェネレータ

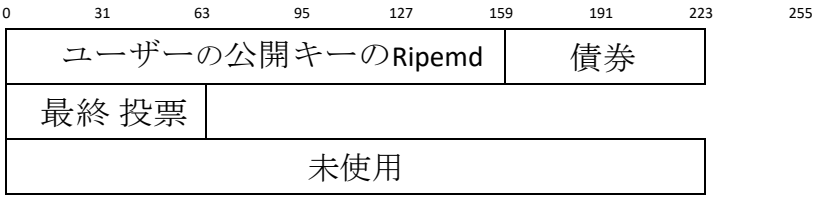
リーダーは選出された歴史証明ジェネレータです。任意のユーザトランザクションを消費し、システム内の一意なグローバル順序を保証するすべてのトランスアクの証シーケンスを出力します。各トランザクションのバッチの後、Leader は、その順序でトランザクションを実行した結果である状態の署名を出力します。この署名はリーダーの ID で署名されます。

7.1.2 状態

ユーザーアドレスによってインデックス付けされた、ナイーブなハッシュテーブル。各セルには、完全なユーザー アドレスと、この計算に必要なメモリが含まれています。たとえば、トランザクションテーブルには次のものが含まれます。



合計 32 バイトの場合。
ステーク債表の証明は、以下を含みます。



合計 64 バイトの場合。

7.1.3 検証方法、 状態 レプリケーション

検証ツールノードは、ブロックチェーン状態をレプリケートし、ブロックチェーン状態の高可用性を提供します。レプリケーションターゲットはコンセンサスアルゴリズムによって選択され、コンセンサスアルゴリズムのバリデータは、承認するレプリケーションの証明ノードを選択して投票します。 オフチェーン定義の基準に基づきます。

ネットワークは、最小の証明の証のボンドサイズと、ボンドごとの単一のレプリケータ ID の要件を使用して構成できます。

7.1.4 検証コントロール

これらのノードは、検証ツールからの帯域幅を消費しています。これらは仮想ノードであり、検証ツールまたはリーダーと同じマシン上で実行することも、このネットワーク用に構成されたコンセンサス アルゴリズムに固有の別のマシン上で実行することもできます。

7.2 ネットワーク の制限

リーダーは、受信ユーザーパケットを受け取り、可能な限り最も効率的な方法でパケットを注文し、ダウンストリームの検証ツールに公開される履歴の証明シーケンスにシーケンスすることが期待されます。効率は、

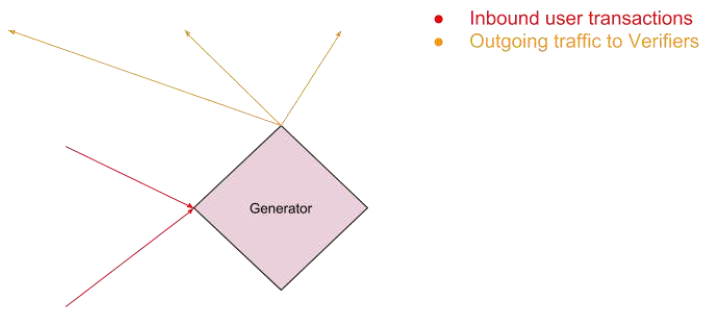
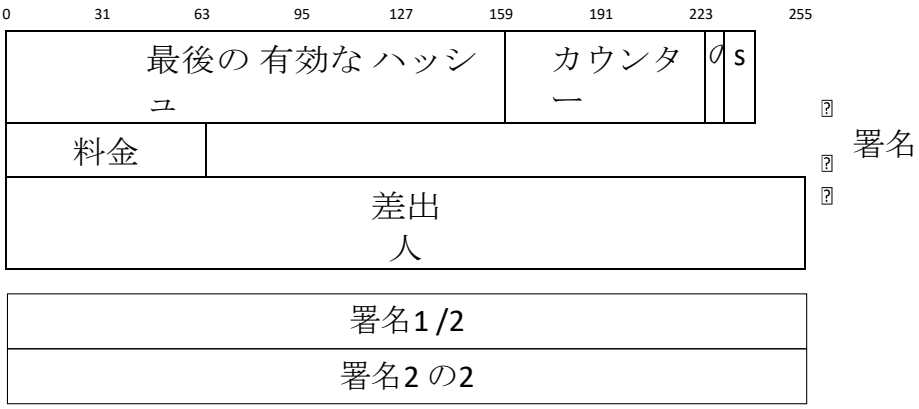


図10: ジェネレータのネットワークの制限

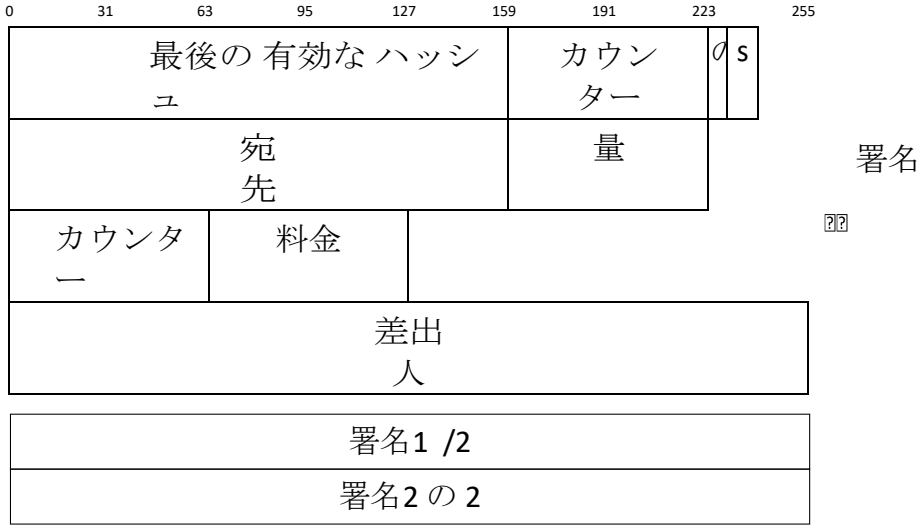
トランザクションのメモリアクセスパターンは、エラーを最小限に抑え、プリフェッチを最大化するようにトランザクションを順序付けます。

着信 パケット 形式:



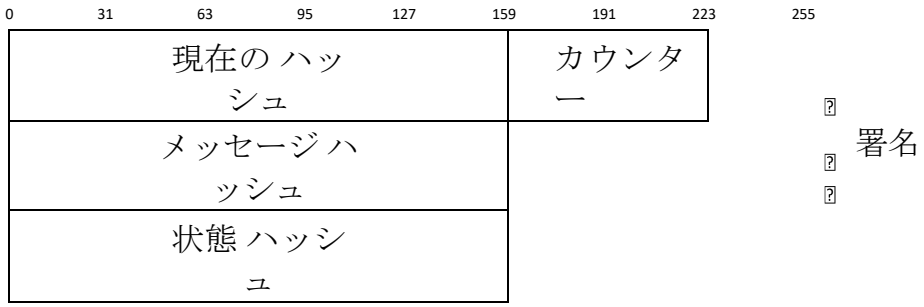
サイズ20 + 8 + 16 + 8 + 32 + 3232 = 148 バイト。

サポートできる最小ペイロードは、1 宛先の ac-count になります。ペイロード付き:



ペイロードの最小サイズ: 176 バイト

Proof of History シーケンス パケットには、現在のハッシュ、カウンタ、およびPoH シーケンスに追加されたすべての新しいメッセージのハッシュと、すべてのメッセージを処理した後の状態のシグネチャが含まれます。 このパケットは、N メッセージがブロードキャストされるたびに送信されます。
履歴パケットの証明:



署名1 /2
署名2 の2

出力パケットの最小サイズは132 バイトです。

1gbps ネットワーク接続でトランザクションの最大数

可能 です 1 ギガビット 対して 秒 / 176 バイト = 710k tps マックス。 幾 損失 1 4%イーサネット フレーミングが原因で発生します。 目標を超える予備容量 ネットワークの量を使用して、コードを作成することで可用性を向上させることができます。 アウトプット で リード= ソロモン コード そして ストライピング それ 宛先 ザ 利用できる 下流 検証。

7.3 計算限界

各トランザクションにはダイジェスト検証が必要です。この操作では、トランザクション メッセージ自体の外部のメモリを使用せず、独立して並列化できます。したがって、スループットは、システム上で使用できるコアの数によって制限されると予想されます。

GPUベースのECDSA検証サーバは、毎秒900k動作の実験結果を持っています[9].

7.4 メモリ の制限

32 バイト en-を持つ 50% の完全なハッシュテーブルとして状態の素朴な実装 各アカウントの試行は、理論的には640GBに100億のアカウントを適合させるだろう。 このテーブルへの安定状態のランダム アクセスは 1 で測定されます。.1 10^7 書き込みまたは 1 秒あたりの読み取り数。 基d 2 読み取りとトランザクションごとに 2 回の書き込み、メモリ スループット 缶 ハンドル 2.75m トランザクション 対して 秒。 これ は 測定 オン ひとつの アマゾン ウェブ サービス 1TB x1.16xlarge 例。

7.5 高性能スマートコントラクト

スマートコントラクトは、一般的なトランスアクションです。これらは、各ノードで実行され、状態を変更するプログラムです。この設計では、拡張されたBerkeleyパケットフィルタのバイトコードをスマートコントラクト言語と同じくらい高速かつ簡単に分析し、JITバイトコードを利用します。

その主な利点の1つは、ゼロコストの外部関数インタフェースです。組み込み関数、またはプラットフォームに直接実装される関数は、プログラムから呼び出すことができます。組み込み関数を呼び出すと、そのプログラムが中断され、高パーフォ rmance サーバー上で組

み込みがスケジュールされます。組み込み関数は、GPU 上で並列に実行するためにバッチ処理されます。

上の例では、2 つの異なるユーザー プログラムが同じ組み込み関数を呼び出します。各プログラムは、組み込みのバッチ実行が

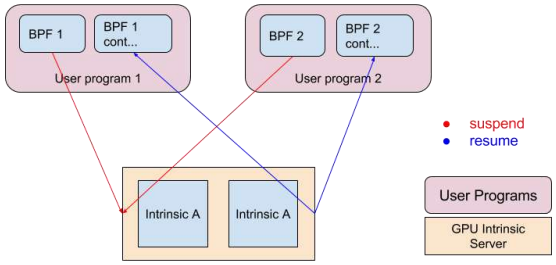


図11: BPF プログラムの実行

完成。 組み込みの例として、ECDSA 検証があります。 GPU で実行するためにこれらの呼び出しをバッチ処理すると、スループットが何千倍も向上します。

BPFバイトコードは、すべてのメモリに対して適切に定義されたコンテキストを持つため、このトランポリンはネイティブのオペレーティング システムのスレッド コンテキストの切り替えが不要です。
使用しています。

eBPFバックエンドは 2015 年から LLVM に組み込まれているため、LLVM フロントエンド言語を使用してスマート コントラクトを作成できます。それは2015年以来Linuxカーネルにあり、バイトコードの最初の反復は1992年以来存在しています。1回のパスで eBPF の正確性をチェックし、ランタイムとメモリを必要とすることを確認し、それを x86命令に変換できます。

参照

[1] リスコフ、時計の実用
http://www.dainf.cefetpr.br/ タクラ/SDII/実用的な使用時計.pdf

[2] グーグル スパナ トゥルータイム の一貫性
https://cloud.google.com/spanner/docs/true-time-external-consistency

[3] 注文オラクルとの合意の解決
http://www.inf.usi.ch/faculty/pedone/Paper/2002/2002EDCCb.pdf

[4] テンダーミント:鉱業なしのコンセンサス
https://tendermint.com/static/docs/tendermint.pdf

[5] ヘデラ:運営協議会と公共ハッシュグラフネットワーク
https://s3.amazonaws.com/hedera-hashgraph/hh-whitepaper-v1.0-180313.pdf

[6] ファイルコイン、レプリケーションの証明、
https://filecoin.io/proof-of-replication.pdf

[7] スラッシャー、ステークアルゴリズムのプネイティブ証明
https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof--ステークアルゴリット

[8] ビットシェアーズ委任ステークの証明
https://github.com/BitShares/bitshares/wiki/Delegated-Proof-of-Stake

[9] GPUアクセラレーションを備えた効率的な楕円曲線暗号署名サーバー
http://ieeexplore.ieee.org/document/7555336/

[10] キャスパーフレンドリーファイナリティガジェット
https://arxiv.org/pdf/1710.09437.pdf