(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4263477号
(P4263477)
(45)発行日　平成21年5月13日(2009.5.13)　(24)登録日　平成21年2月20日(2009.2.20)

(51)Int.Cl.　F I
G06F 12/00 (2006.01)　G06F 12/00 501B

請求項の数 17 (全 22 頁)

(21)出願番号	特願2002-540319(P2002-540319)
(86)(22)出願日	平成13年10月4日(2001.10.4)
(65)公表番号	特表2004-514968(P2004-514968A)
(43)公表日	平成16年5月20日(2004.5.20)
(86)国際出願番号	PCT/US2001/031306
(87)国際公開番号	WO2002/037689
(87)国際公開日	平成14年5月10日(2002.5.10)
審査請求日	平成16年10月4日(2004.10.4)
(31)優先権主張番号	60/245,920
(32)優先日	平成12年11月6日(2000.11.6)
(33)優先権主張国	米国(US)
(31)優先権主張番号	09/777,149
(32)優先日	平成13年2月5日(2001.2.5)
(33)優先権主張国	米国(US)

(73)特許権者　503093224
イーエムシー　コーポレイション
ＥＭＣ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国　０１７４８　マサチューセッツ州　ホプキントン　サウス　ストリート　１７６
(74)代理人　100068755
弁理士　恩田　博宣
(74)代理人　100105957
弁理士　恩田　誠
(74)代理人　100094318
弁理士　山田　行一
(74)代理人　100104282
弁理士　鈴木　康仁

最終頁に続く

(54)【発明の名称】共通デジタルシーケンスを識別するシステム

(57)【特許請求の範囲】
【請求項１】
デジタルビットシーケンスを分割する方法であって、
少なくとも前記デジタルビットシーケンスの一部にハッシュ関数演算を実行するステップと、
前記ハッシュ関数演算によって生成されたハッシュ値を第１の予め決められたビットパターンに対してモニタするステップと、
前記第１の予め決められたビットパターンが発生した場合、前記デジタルビットシーケンスにブレークポイントをマークするステップとを備え、前記ハッシュ関数演算を実行する前記ステップは、
ローリングハッシュ関数を使用して前記デジタルビットシーケンスの一部を検索し、前記デジタルビットシーケンスの現在のパーティションの長さに基づいてローリングハッシュ値を調節して、前記現在のパーティションにおける前記ブレークポイントを生成する可能性を増大させることを含み、前記現在のパーティションの長さが増加する時に、前記現在のパーティションにおける前記ブレークポイントの生成の可能性が増大するようになる、方法。
【請求項２】
前記第１の予め決められたビットパターンは、連続ビットシーケンスを備える、請求項１記載の方法。
【請求項３】

前記ハッシュ値をモニタする前記ステップに対して閾値制限を決めるステップであって、前記モニタしたハッシュ値が閾値を越えるかどうかの計算を行うことを含む、閾値制限を決めるステップと、
前記閾値を越えた場合、第２の予め決められたビットパターンに対する、前記ハッシュ関数演算により生成されたハッシュ値をモニタするステップと、
前記第２の予め決められたビットパターンが発生した場合、前記デジタルビットシーケンスに前記ブレークポイントをマークするステップと、
を更に備える、請求項１記載の方法。
【請求項４】
前記ハッシュ値をモニタする前記ステップに対して閾値制限を決めるステップであって、前記モニタしたハッシュ値が閾値を越えるかどうかの計算を行うことを含む、閾値制限を決めるステップと、
前記モニタしたハッシュ値が閾値を越えた場合、前記モニタしたハッシュ値を変換して前記デジタルビットシーケンスに前記ブレークポイントを生成する確率を上げるステップと、
を更に備える、請求項１記載の方法。
【請求項５】
前記モニタしたハッシュ値を変換して前記デジタルビットシーケンスに前記ブレークポイントを生成する確率を上げる前記ステップは、少なくとも所望のパーティションサイズに応じて行われる、請求項４記載の方法。
【請求項６】
前記モニタしたハッシュ値を変換して前記デジタルビットシーケンスに前記ブレークポイントを生成する確率を上げる前記ステップは、少なくとも前記デジタルシーケンスの前記現在のパーティションの長さに応じて行われる、請求項４記載の方法。
【請求項７】
前記モニタしたハッシュ値を変換して前記デジタルビットシーケンスに前記ブレークポイントを生成する確率を上げる前記ステップは、
前記ハッシュ値をモニタする前記ステップに第２の予め決められたビットパターンを利用するステップと、
前記第２の予め決められたビットパターンが発生した場合、前記ブレークポイントをマークするステップと、
によって実行される、請求項４記載の方法。
【請求項８】
第１のデジタルビットシーケンスで第１のブレークポイントを決める方法であって、
前記第１のデジタルビットシーケンスのビットサブセットグループを決めるステップと、
ハッシュ値で第１の予め決められたビットパターンを得るまで、前記第１のデジタルビットシーケンスの開始位置から始まっている前記第１のデジタルビットシーケンスの前記ビットサブセットグループにハッシュ関数演算を実行してハッシュ値を生成するステップであって、前記第１のデジタルビットシーケンスの現在のパーティションの長さに基づいてハッシュ値が調節されて、前記現在のパーティションにおける前記第１のブレークポイントを生成する可能性が増大し、前記現在のパーティションの長さが増加する時に、前記現在のパーティションにおける前記第１のブレークポイントの生成の可能性が増大するようになる、ハッシュ値を生成するステップと、
前記ハッシュ値で前記第１の予め決められたビットパターンを得た場合に、前記第１のブレークポイントをマークするステップと、
を備える方法。
【請求項９】
前記ビットパターンはビットパターンを備える、請求項８記載の方法。
【請求項１０】

前記ハッシュ値で前記第１の予め決められたビットパターンを再び得るまで、前記第１のブレークポイントから前記第１のデジタルビットシーケンスのビットサブセットグループにハッシュ関数演算を更に実行するステップと、
前記ハッシュ値で前記第１の予め決められたビットパターンを再び得た場合に、前記第１のデジタルビットシーケンスに別のブレークポイントをマークするステップと、
を更に備える、請求項８記載の方法。
【請求項１１】
前記モニタしたハッシュ値が確立された閾値値を越えると判定されるまで、前記第１のデジタルビットシーケンスの前記ビットサブセットグループに前記ハッシュ関数演算を実行するステップを継続するステップと、
前記ハッシュ値で前記第２の予め決められたビット値が得られるまで、前記第１のデジタルビットシーケンスの前記ビットサブセットのグループのハッシュ関数演算を実行するステップと、
前記ハッシュ値で前記第２の予め決められたビット値が得られた場合、前記第１のブレークポイントをマークするステップと、
を更に備える、請求項８記載の方法。
【請求項１２】
コンピュータープログラムであって、前記コンピュータープログラムは、
プロセッサによって実行されたときに、コンピュータにデジタルビットシーケンスを分割する方法を行わせるコンピュータ読取可能な命令を含むコンピュータ読取可能なコードを備え、前記方法は、
少なくとも一部の前記デジタルビットシーケンスに対してハッシュ関数演算を実行するステップと、
第１の予め決められたビットパターンに対する前記ハッシュ関数演算によって生成されるハッシュ値をモニタするステップと、
前記第１の予め決められたビットパターンが発生した場合に、前記デジタルビットシーケンスにブレークポイントをマークするステップとを備え、前記ハッシュ関数演算を実行する前記ステップは、ローリングハッシュ関数を使用して前記デジタルビットシーケンスの一部を検索し、前記デジタルビットシーケンスの現在のパーティションの長さに基づいてローリングハッシュ値を調節して、前記現在のパーティションにおけるブレークポイントを生成する可能性を増大させることを含み、前記現在のパーティションの長さが増加する時に、前記現在のパーティションにおける前記ブレークポイントの生成の可能性が増大するようになる、コンピュータープログラム。
【請求項１３】
コンピュータープログラムであって、前記コンピュータープログラムは、
プロセッサによって実行されたときに、コンピュータに第１のデジタルビットシーケンスにおける第１のブレークポイントを決める方法を行わせるコンピュータ読取可能な命令を含むコンピュータ読取可能なコードを備え、前記方法は、
前記第１のデジタルビットシーケンスのビットサブセットグループを決めるステップと、
ハッシュ値で第１の予め決められたビットパターンを得て第１のブレークポイントを生成するまで、前記第１のデジタルビットシーケンスの開始位置から始まっている前記第１のデジタルビットシーケンスの前記ビットサブセットグループにハッシュ関数演算を実行してハッシュ値を生成するステップであって、前記第１のデジタルビットシーケンスの現在のパーティションの長さに基づいてハッシュ値を調節して、前記現在のパーティションにおける前記第１のブレークポイントを生成する可能性を増大させることを含み、前記現在のパーティションの長さが増加する時に、前記現在のパーティションにおける前記第１のブレークポイントの生成の可能性が増大するようになる、ハッシュ値を生成するステップと、
前記ハッシュ値で前記第１の予め決められたビットパターンを得た場合に、前記第１

のブレークポイントをマークするステップと、
を備える、コンピュータープログラム。
【請求項１４】
前記モニタしたハッシュ値を変換して前記デジタルビットシーケンスに前記ブレークポイントを生成する前記確率を上げる前記ステップは、前記デジタルビットシーケンスに含まれるコンテンツに応じて行われる、請求項４記載の方法。
【請求項１５】
前記ハッシュ関数演算は、可能な出力値の範囲を均一にカバーしない優先ハッシュ合計を生じるよう選択される、請求項１記載の方法。
【請求項１６】
前記ハッシュ関数演算は、その実行中にダイナミックに選択されるか或いは変更される、請求項１記載の方法。
【請求項１７】
前記ハッシュ関数演算は、該ハッシュ関数への入力値として現在のロケーションの相対値または絶対値を含む、請求項１記載の方法。
【発明の詳細な説明】
【０００１】
【著作権に関する通知／許可】
この特許文献の開示の一部は、著作権保護の対象となる材料を含んでよい。著作権所有者は、米国登録商標特許庁の特許ファイルまたはレコードに記載されているような特許の開示の特許文献が誰によってファクシミリ再生されても異論はないが、それ以外においては、いかなるものに対しても全ての著作権の権利を確保する。以下の通知はソフトウェア、データおよび該当する場合には図面を含む以下の説明に適用される。Copyright 2000, Undoo Technologies

【０００２】
【発明の背景】
本発明は、一般的に、デジタルシーケンスのブレークポイントを求める「スティッキーバイト」のくくり出しを用いて、データシーケンスを組織化しないで決定するシステムおよび方法の分野に関する。より詳細には、本発明は、最適に近い共通性を一般的に生ずる、データセットをピースに分割する効率的かつ有効な方法に関する。
【０００３】
現代のコンピュータシステムは、総計すると何十億桁ものバイトになる莫大な量のデータを保持している。信じられないことに、この量は年ごとに４倍になる傾向があり、コンピュータ大容量記憶アーキテクチャの最もみごとな進歩でさえも追いつくことができない。
【０００４】
大部分のコンピュータ大容量記憶システムで保持されるデータは、以下の興味深い特徴を有すると見られている：１）それはほとんど全くランダムでなく、事実上非常に冗長である；２）このデータに特有のシーケンスの数は、実際にデータが占める記憶空間のなかで非常に低い割合となる。３）かなりの量の労力がこの量のデータの管理を試みるのに必要とされ、その多くが冗長の除去（すなわち複製ファイル、ファイルの古いバージョン、パージログ、アーカイブ処理等）の識別に関係している。４）大量の資本的資源が、不必要なコピーを作成し、それらコピーを局所型媒体に保存することなどに捧げられる。
【０００５】
冗長コピーくくり出しシステムは、別な方法では何桁も必要とされる記憶量の数を減らすであろう。しかしながら、大容量データをそれらの共通のシーケンスにくくり出すシステムでは、シーケンスを求める方法を使用しなければならない。1つのデータシーケンスを別のシーケンスと比較することを試みる従来の方法は、一般的に、計算の極度な複雑性を被ることになり、したがって、これらの方法は比較的小さいデータセットをくくり出すことにのみ使用可能である。より大きなデータセットのくくり出しは、一般的に、任意の固

定サイズを用いるといった単純な方法を用いて成されるだけである。これらの方法は、多くの環境下で十分なくくり出しができず、大きいデータセットの効率的なくくり出しは、コンピュータ科学分野において今まで長きにわたり持続的に厄介な問題であった。

【０００６】

【発明の概要】

共通シーケンスを識別できるように、デジタルシーケンスのブレークポイントを求める「スティッキーバイト」のくくり出しを用いて、データシーケンスを組織化しないで決定するシステムおよび方法が、本願に開示されている。スティッキーバイトのくくり出しは、最適に近い共通性を一般的に生ずる、データセットをピースに分割する効率的な方法を提供する。本願で開示されるように、これは、ハッシュ値の定期的なリセット、或いは好適な実施例においてはローリングハッシュ合計、を有するハッシュ関数を用いることによって実現されてよい。さらに、本願で開示されている特に典型的な実施例において、閾値関数は、デジタルまたは数のシーケンス（例えばデータのシーケンス）に区分を決定的に設定するのに利用される。ローリングハッシュおよび閾値関数の双方は、最小限の計算が必要とされるようできている。この低オーバーヘッドは、くくり出しエンジンまたは全データセットにわたる後続の同期を好む他のアプリケーションに対する表現のデータシーケンスを速やかに分割することを可能にする。

【０００７】

本願で開示されるシステムおよび方法の重要な利点の内の一つは、計算が良好に機能するための（従来のくくり出しているシステムのような）通信も比較も必要としないことである。これは分散環境において特に当てはまり、従来のシステムが1つのシーケンスを別のシーケンスと比較するのに通信を必要とするのに対して、本発明のシステムおよび方法は、次に考えられるシーケンスだけを用いて隔離状態で実行可能である。

【０００８】

オペレーションにおいて、コンピュータ間の通信の必要がなく、またファイルのデータコンテンツに関係なく、多重関連および無関連のコンピュータシステム上で共通要素を見つけることができるように、本発明のシステムおよび方法は、数のシーケンス（例えばファイルのバイト）を分割する完全自動手段を提供する。概括的にいえば、共通シーケンスを、それらシーケンスを見つけるオペレーションにおける検索、比較、通信、または他の処理要素との調整の必要なく見つけることができるように、数要素のシーケンス（すなわちバイトシーケンス）を分割する完全自動手段を含むデータ処理システムに関するシステムおよび方法が、本願に開示されている。本発明のシステムおよび方法は、パーティション間で共通性を最適化するのに求められるタイプおよびサイズのシーケンスを生ずる分布を有する数シーケンスを分割する「スティッキーバイト」ポイントを生成する。

【０００９】

添付の図と共に取り入れられた好適な実施例の以下の説明を参照することによって、本発明の上述ならびに他の特徴および目的、またそれらを達成する方法がより明瞭になり、本発明は最大限理解されるであろう。

【００１０】

【代表的な実施例の説明】

本発明は、インターネットのような公衆通信回線を使っている企業コンピューティングシステムのような分散コンピューティング環境に関して、図示され説明される。しかしながら、本発明の重要な特徴は、特定のアプリケーションのニーズに合うよう上方および下方に容易に基準化されるということである。したがって、相反するものに対して指示がない限り、本発明は従来のLANシステムのような小さなネットワーク環境と同様に相当に大きくかつ複雑なネットワーク環境にも適用できる。

【００１１】

図1に関して、本発明はネットワーク10上の新規のデータ記憶システムとともに利用できる。この図において、典型的なインターネットワーク環境10は、多重広域ネットワーク（「WAN」）14とローカルエリアネットワーク（「LAN」）16間の論理および物理接続によっ

て形成されるグローバルインターネットワークを備えるインターネットを含んでよい。インターネットバックボーン12は、データトラフィックのバルクを運ぶメインラインおよびルーターを表す。バックボーン12は、例えばGTE、MCI、Sprint、UUNetおよびAmerica Onlineといった主要なインターネットサービスプロバイダ（「ISP」）によって管理されるシステムの最大ネットワークによって形成される。単独接続ラインはインターネットバックボーン12へのWAN 14およびLAN 16の接続を好都合に図示するのに用いられているが、現実には、複数パスでルート可能な物理接続が多重WAN 14とLAN 16間に存在することは理解されるべきである。これは、単一または多重故障ポイントに直面した際にインターネットワーク10を強力にする。

【００１２】

「ネットワーク」は汎用のシステムを備え、通常、ノード18で作動するプロセス間の論理接続を可能にする物理接続を切り替える。ネットワークによって実現される物理接続は、一般的に、ネットワークを使っているプロセス間で確立される論理接続から独立している。このように、ファイル転送、メール転送などに及ぶ異種のプロセスセットが、同じ物理的なネットワークを使うことができる。逆に言えば、ネットワークは、ネットワークを用いて論理的に接続されているプロセスには見えない異質の物理的なネットワーク技術のセットから形成可能である。ネットワークによって実現されるプロセス間の論理接続が物理接続から独立しているので、インターネットワークは長距離にわたる仮想的に無制限のノード数に容易に基準化される。

【００１３】

対照的に、システムバス、周辺コンポーネント相互接続（「PCI」）バス、インテリジェントドライブエレクトロニクス（「IDE」）バス、小型コンピューターシステムインターフェース（「SCSI」）バスなどといった内部データ経路は、コンピュータシステム内で特殊目的接続を実現する物理接続を画成する。これらの接続は、プロセス間の論理接続に対してフィジカルデバイス間の物理接続を実現する。これらの物理接続は、コンポーネント、該接続に結合できる限定された数のデバイス、および該接続を介して接続できる条件付きフォーマットのデバイス間の限られた距離によって特徴づけられている。

【００１４】

本発明の特定のインプリメンテーションで、記憶装置はノード18に配置されてよい。あらゆるノード18のストレージが、一つのハードディスクを備えてよく、或いは一つの論理ボリュームとして構成される多重ハードディスクを有する従来のRAIDデバイスのような管理された記憶システムを備えてもよい。重要なこととして、本発明は、ノード内で行なうのとは対照的に、ノードの全域にわたって冗長オペレーションを管理するので、任意のノード内のストレージの特定の構成はあまり重要ではない。

【００１５】

オプションとして、ノード18のうちの1つ以上が、分散型かつ共同的なやり方でノード18にわたってデータストレージを管理するストレージ割付け管理（「SAM」）プロセスを実現してもよい。SAMプロセスは、全体として、システムを集中制御しないか或いはほとんどしないように機能するのが望ましい。SAMプロセスは、ノード18全域にわたってデータ分布を提供し、RAID記憶サブシステムで見つけられるパラダイムと同様の方法で、ネットワークノード18全域にわたって故障を許容するやり方でリカバリを実現する。

【００１６】

しかしながら、SAMプロセスは、一つのノード内または一つのコンピュータ内ではなくノードの全域にわたって機能するので、該プロセスは従来のRAIDシステムよりも大きな故障許容とストレージ効率レベルを可能にする。たとえば、SAMプロセスは、ネットワークノード18、LAN 16またはWAN 14が利用できなくなった場合でさえも回復可能である。さらに、一部のインターネットバックボーン12が故障または輻輳によって利用できなくなった場合でさえも、SAMプロセスはアクセス可能なままであるノード18で分配されるデータを用いて回復可能である。このように、本発明はインターネットワークの強力な性質に影響を及ぼし、先例のない可用性、信頼性、障害の許容範囲および堅固性を提供する。

【００１７】
図2に関して、本発明が実現される典型的なネットワークコンピューティング環境のより詳細な概念図が表される。先行図のインターネットワーク10（またはこの図でのインターネット118）は、スーパーコンピュータ即ちデータセンター104からハンドヘルド即ちペンベースのデバイス114の範囲に及ぶ、異質のコンピューティング装置およびメカニズム102のセットの相互接続ネットワーク100を可能にする。このようなデバイスは異なるデータストレージのニーズを有しているが、それらはネットワーク100を介してデータを検索する能力を共有しており、それら自身のリソース内のデータに作用する。IBM互換機デバイス108、マッキントッシュデバイス110およびラップトップコンピューター112といったパーソナルコンピュータまたはワークステーションクラスデバイスと同様にメインフレームコンピュータ（例えばVAXステーション106およびIBM AS/400ステーション116）を含む異種のコンピューティング装置102は、インターネットワーク10およびネットワーク100を介して容易に相互接続する。図示されてはいないが、移動式および他のワイヤレスデバイスをインターネットワーク10に結合してもよい。
【００１８】
インターネットベースのネットワーク120は一組の論理接続を備えており、その幾つかはインターネット118を介して複数の内部ネットワーク122間に作られる。概念的に、インターネットベースのネットワーク120はWAN 14（図1）と同種であり、それは地理的に遠いノード間での論理接続を可能にしている。インターネットベースのネットワーク120は、インターネット118、または専用回線、Fibre Channelなどを含む他の公共および私用のWAN技術を用いて実現されてもよい。
【００１９】
同様に、内部ネットワーク122は概念的にLAN 16（図1）と同種であり、それはWAN 14よりも限られたスタンスにわたって論理接続を可能にしている。内部ネットワーク122は、 Ethernet、Fiber Distributed Data Interface（「FDDI」）、Token Ring、AppleTalk、Fibre Channel、などを含むさまざまなLAN技術を用いて実現されてもよい。
【００２０】
各内部ネットワーク122は、独立したノード（RAIN）エレメント124の一つ以上の冗長配列を接続し、RAINノード18（図1）を実現する。各RAINエレメント124は、プロセッサ、メモリおよび一つ以上の大容量記憶装置（例えばハードディスク）を備える。RAINエレメント124はまた、従来のIDE またはSCSIコントローラであってよく、RAIDコントローラのような管理コントローラであってもよいハードディスクコントローラを含む。RAINエレメント124は、物理的に分散してよく、冷却および電力といったリソースを共有している一つ以上のラックで同じ位置に配置されてもよい。各ノード18（図1）は他のノード18から独立しており、1つのノード18の故障または不稼働が他のノード18の可用性に影響を及ぼさず、1つのノード18に格納されているデータは他のノード18に格納されているデータから復元可能である。
【００２１】
特定の典型的なインプリメンテーションにおいて、RAINエレメント124は、PCIバスを支持するマザーボードおよび従来のATまたはATXケースに収容される256メガバイトのランダムアクセスメモリー（「RAM」）上に取り付けられるインテルベースのマイクロプロセッサーのような商品コンポーネントを用いたコンピュータを備えてもよい。SCSIまたはIDEコントローラは、マザーボード上でおよび／またはPCIバスに接続された拡張カードによって実現されてよい。コントローラがマザーボード上のみで実現される場合、PCI拡張バスがオプションとして使われてもよい。特定のインプリメンテーションにおいて、マザーボードは、各RAINエレメント124が最高4つ以上のEIDEハードディスクを含むよう、2つのマスタリングEIDEチャネルおよび2つの付加マスタリングEIDEチャネルを実現するのに用いられるPCI拡張カードを実現できる。特定のインプリメンテーションにおいて、各ハードディスクは、RAINエレメントにつき320ギガバイト以上の総記憶容量の80ギガバイトハードディスクを備えてよい。RAINエレメント124内のハードディスク容量および構成は、特

定のアプリケーションのニーズを満たすよう容易に増大または低減できる。外被もまた、電源および冷却デバイス（図示せず）といった支持メカニズムを収容する。

【００２２】

各RAINエレメント124は、オペレーティングシステムを実行する。特定のインプリメンテーションにおいて、UNIXまたはLinuxのようなUnixバリアントオペレーティングシステムが、使われてよい。しかしながら、 DOS、マイクロソフトウインドウズ、アップルマッキントッシュOS、OS/2、マイクロソフトウインドウズNTなどを含む他のオペレーティングシステムを、パフォーマンスにおける予測可能な変更を行なうことで同等に置換え可能であることが企図される。選択されたオペレーティングシステムは、アプリケーションソフトウェアおよびプロセスを実行するプラットホームを形成し、ハードディスクコントローラを介して大容量記憶域にアクセスするファイルシステムを実現する。様々なアプリケーションソフトウェアおよびプロセスが各RAINエレメント124上で実現され、適切なネットワークプロトコル（例えばユーザーデータグラムプロトコル（「UDP」）、伝送制御プロトコル（TCP）、インターネットプロトコル（IP）など）を用いたネットワークインターフェースを介して、ネットワークの接続性を提供できる。

【００２３】

図3に関しては、ロジックフローチャートが、本発明のハッシュファイルシステムにコンピュータファイルをエントリするステップを表すために示されており、ファイルのハッシュ値が、セットまたはデータベースに予め維持されるファイルのハッシュ値と照合される。いかなるデジタルシーケンスもまた、全く同様に本発明のハッシュファイルシステムに入力できるが、入力されるデジタルシーケンスがコンピュータファイルから成る本例は教訓的である。

【００２４】

プロセス200は、コンピューターファイルデータ202（例えば「ファイルA」）を本発明のハッシュファイルシステム（「HF」）に入力することから始まり、ハッシュ関数がステップ204で実行される。次いで、ファイルAのハッシュを表すデータ206が判断ステップ208でハッシュファイル値を含むセットのコンテンツと比較される。データ206が既にセットにある場合、ファイルのハッシュ値はステップ210でハッシュレシピに加えられる。このハッシュレシピは、データ、並びにシステムに入力されたコンピューターファイルデータのクラスに従って、ファイル、ディレクトリ、ボリュームまたは全体システムを復元するのに必要な関連する構成から成る。ハッシュ値を含み、データに対応しているセット212のコンテンツは、判断ステップ208の比較演算のために既存のハッシュ値214の形で提供される。一方、ファイルAのハッシュ値がその時点でセットに存在しない場合、ファイルは、ステップ216で（以下でより完全に説明されるように）ハッシュピースに分割される。

【００２５】

図4に関しては、更なるロジックフローチャートが、デジタルシーケンス（例えばファイルまたは他のデータシーケンス）をブレークアップしてハッシュピースにするプロセス300でのステップを表すために提供される。このプロセス300は、多数のデータピースと同様にそれに対応している確率的に固有のハッシュ値を各ピースに対して最終的に生成することになる。

【００２６】

ファイルデータ302は、システムの他のピースとの共通性あるいは後にステップ304で共通していることが明らかになるピースの可能性に基づいたピースに分割される。ファイルデータ302に関するステップ304のオペレーションの結果は、代表的な例において、A1～A5と名づけられた4つのファイルピース306を生成する。

【００２７】

ファイルピース306の各々は、次いで、A1～A5のピース306の各々に確率的に固有の数が割り当てられるよう、個々のハッシュ関数オペレーションを通して各々を配置することによってステップ308で操作される。ステップ308のオペレーションの結果、ピース306（A1～A5）の各々は、関連づけられた、確率的に固有のハッシュ値310（それぞれA1ハッシュ～A5

ハッシュとして示される）を有することになる。ステップ304のファイル分割プロセスは、ここで開示されている固有の「スティッキーバイト」オペレーションとともに、更に詳細に以下で説明される。

【００２８】

更に図5に関しては、別のロジックフローチャートが、ファイルの各ピース306のハッシュ値310を、セット212で維持されている既存のハッシュ値214と比較するプロセス400を表すために示される。特に、ステップ402で、ファイルの各ピース306のハッシュ値310は既存のハッシュ値214および新しいハッシュ値408と比較され、対応する新しいデータピース406がセット212に加えられる。このように、セット212に予め存在していないハッシュ値408が、それらに関連づけられたデータピース406と共に加えられる。プロセス400はまた、全てのファイルピースに対する一つのハッシュ値が様々なピース306のハッシュ値310と同値であることを示しているレコード404を生成することになる。

【００２９】

更に図6は、ファイルハッシュまたはディレクトリリストハッシュ値を既存のディレクトリリストハッシュ値と比較し、新しいファイルまたはディレクトリリストハッシュ値をセットディレクトリリストに加えるプロセス500を図示している別のロジックフローチャートを示す。プロセス500は、ファイル名、ファイルメタデータ（例えば日付、時間、ファイルの長さ、ファイルタイプ等）およびディレクトリの各項目のファイルのハッシュ値の累積リストを備える記憶データ502上で機能する。ステップ504で、ハッシュ関数はディレクトリリストのコンテンツに実行される。判断ステップ506は、ディレクトリリストのハッシュ値が既存のハッシュ値214のセット212であるかどうかを判断するよう機能する。判断が肯定である場合、プロセス500は、別のファイルハッシュまたはディレクトリリストハッシュをディレクトリリストに加えるために戻る。これに対して、ディレクトリリストのハッシュ値がセット212にすでにない場合、ディレクトリリストのハッシュ値およびデータはステップ508でセット212に加えられる。

【００３０】

更に図7に関しては、代表的なコンピュータファイル（すなわち「ファイルA」）のピース306とそれらに対応するハッシュ値310の比較600が、典型的なファイルの特定ピースの編集の前後双方で示される。この例では、レコード404はファイルA1～A5のピースの各々のハッシュ値310と同様にファイルAのハッシュ値を含む。ファイルAの代表的な編集または変更は、306Aのファイルピースのピース A2（A2-bによって表される）のデータの変化と共にハッシュ値310Aのハッシュ値A2-bの対応する変化を生成可能である。編集されたファイルピースは、ファイルAの修正ハッシュ値およびピースA2-bの修正ハッシュ値を含む更新済みレコード404Aを生成する。

【００３１】

更に図8に関しては、本発明のシステムおよび方法によって導き出される複合データ（例えば複合データ702および704）は明示的に表されるデータ706と実質的に同じものであるが、「レシピ」すなわち式によって作成されるという事実を図示している概念図700が示される。図示される例において、このレシピは、対応するハッシュ708によって表されるデータの連結またはハッシュによって表されるデータを用いた関数の結果を含む。データブロック706は示されるように可変長数量であってよく、ハッシュ値708はそれに関連づけられたデータブロックから導き出される。既に述べたように、ハッシュ値708は対応するデータピースの確率的に固有のＩＤであるが、全く固有のＩＤがその代わりに使われるか、或いは確率的に固有のＩＤと混在させることもできる。複合データ702、704はまた、多くのレベルが深い他の複合データを参照でき、一方、複合データのハッシュ値708はレシピが生成するデータの値またはレシピ自体のハッシュ値から導き出されることが可能であることもまた注目されるべきである。

【００３２】

更に図9に関しては、ハッシュファイルシステムおよび方法がデータ802を編成し、それらが表すデータへのポインタとしてハッシュ値806を使用することによって冗長シーケンス

の再利用を最適化するのにどのように利用できるかを別の概念図800で示していて、データ802は、明確なバイトシーケンス808（原子データ）か或いはシーケンスのグループ（コンポジット）804として表されてよい。

【００３３】

図800は、レシピおよびあらゆるレベルで再利用されるデータの強大な共通性を図示する。本発明のハッシュファイルシステムの基本構成は、本質的に、ハッシュ値806が従来のポインタの代わりに使われる「ツリー」または「ブッシュ」である。ハッシュ値806はレシピで使われて、データまたはそれ自体がレシピでありえる別のハッシュ値を示している。したがって、本質において、レシピは他のレシピを示し、他のレシピはさらに他のレシピを示し、最終的に幾つかの特定データを示し、その特定データ自体は更に多くのデータを示す他のレシピを示してよく、最後にはデータだけにかかる。

【００３４】

更に図10に関しては、典型的な160ビットのハッシュ値902のハッシュファイルシステムアドレス変換機能の実例となる簡略図900が示される。ハッシュ値902は図示されるように前部904および後部906を備えるデータ構造を含み、図900は対応するデータを含むシステムにおいて特定のノードのロケーションに行くためにハッシュ値902の使用を可能にするよう用いられる特定の「0（1）」オペレーションを図示する。

【００３５】

図900は、ハッシュ値902のデータ構造の前部904がストライプ識別（「ID」）908へのハッシュ・プレフィックスを示すのにどのように用いられることができるか、また、次に、ストライプIDをIPアドレスにマップし、IDクラスをIPアドレス910にマップするのにどのように利用されるかを図示する。この例では、「S2」はインデックスノード37 912のストライプ2を示す。ノード37のインデックス・ストライプ912は、次いで、リファレンス番号914によって示されるデータノード73のストライプ88を示す。次いで、オペレーションにおいて、ハッシュ値902自体がシステムのどのノードが関連したデータを含むのかを示すのに使用でき、ハッシュ値902の別の部分はどのデータストライプがその特定のノードにあるのかを示すのに使用でき、ハッシュ値902の更に別の部分はそのストライプ内のどこにデータが存在するのかを示すのに使用できる。この３ステッププロセスを通して、ハッシュ値902によって表されるデータがシステムに既に存在するかどうかを迅速に判断することができる。

【００３６】

更に図11に関しては、本発明のシステムおよび方法で使用されるインデックス・ストライプ分割機能1000の簡略的な典型図が示される。この図において、典型的な機能1000は、ストライプ1002（S2）を1つのストライプが充分にいっぱいになるように2つのストライプ1004（S2）と1006（S7）に効果的に分割するのに使用可能であることが示されている。この例では、奇数のエントリはストライプ1006（S7）に動かされ、一方偶数のエントリはストライプ1004に残る。この機能1000は、システム全体のサイズが大きくなり複雑さが増すにつれて、ストライプエントリをどう処理することができるかを示す1つの例である。

【００３７】

更に図12に関しては、本発明のシステムおよび方法の全体機能の簡略図1100は、例えば、Day1に多数のプログラムおよびドキュメントファイル1102Aおよび1104Aを有する代表的な家庭用コンピュータのデータのバックアップに用いられているのを示しており、プログラムファイル1102BはDay2に同じものを残し、一方、ドキュメントファイル1104Bのうちの1つはDay2で編集され（Y.doc）、第３のドキュメントファイル（Z.doc）が加えられる。

【００３８】

図1100は、コンピュータファイルシステムがピースに分けられ、次いでピースからオリジナルデータを復元するためにグローバルデータ保護回路（「gDPN」）上に一連のレシピとしてリストされることができる方法の詳細を示す。この非常に小さいコンピュータシステムは、「スナップショット」の形で「Day1」で、次いでその後「Day2」で示される。「Day1」上で「program files H5」および「my documents H6」が数字1106で図示され、前者

はレシピ1108によって表されており、第１の実行可能ファイルはハッシュ値H1 1114で表され、第２の実行可能ファイルはハッシュ値H2 1112で表されている。ドキュメントファイルはハッシュ値H6 1110によって表され、第１のドキュメントはハッシュ値H3 1118で表され、第２のドキュメントファイルはハッシュ値H4 1116によって表されている。その後、「Day2」で数字1120によって示される「program files H5」および「my documents H10」は、「program files H5」は変更されていないが、「my document H10」は変更されたことを示している。数字1122によって示されるH10は、「X.doc」が依然としてハッシュ値H3 1118で表されていることを示し、一方「Y.doc」は今では数字1124においてハッシュ値H8で表されていることを示している。新しいドキュメントファイル「Z.doc」は、数字1126においてハッシュ値H9で表される。

【００３９】

この例では、Day2で幾つかのファイルが変更され、他は変更されていないことがわかる。変更されたファイルにおいて、それらのうちの幾つかのピースは変更されておらず、他のピースは変更されている。本発明のハッシュファイルシステムの使用を通して、コンピュータシステムの「スナップショット」がDay1で作成可能であり（次に存在できるようコンピュータファイルの再生に必要なレシピを生成する）、次いでDay2で以前の日のレシピの幾つかの再使用、他のレシピの再定式化、ならびに新規のレシピの追加によって、その時点でのシステムが記述される。このように、コンピュータシステムを共に構成するファイルは、スナップショットがとられたDay1およびDay2のいかなる時点でもその全体において、並びにいかなる後続日にとられたスナップショットからでも、再現可能である。したがって、本発明のハッシュファイルシステムに委ねられたコンピュータファイルのいかなるバージョンも、初めに委ねられた後であればいつでも、システムから検索できる。

【００４０】

更に図13に関しては、多数の「スティッキーバイト」1204によってマークされる特定のドキュメントファイルの様々なピースの比較1200が、編集前（Day1 1202A）および編集後（Day2 1202B）の双方で示されており、それによってピースのうちの一つが変更され、他のピースは同じままである。

【００４１】

たとえば、Day1に、ファイル1202Aは、可変長ピース1206（1.1）、1208（1.2）、1210（2.1）、1212（2.）、1214（2.3）および1216（3.1）を備える。Day2では、ピース1206、1208、1210、1214および1216は同じままであり（したがって同じハッシュ値を有する）、一方、ピース1212は編集されてピース1212Aを生成する（したがって異なるハッシュ値を有する）。

【００４２】

更に図14を参照すると、本発明のインプリメンテーションで使用できる代表的なスティッキーバイト（またはスティッキーポイント）くくり出しプロセス1300が示されている。プロセス1300は、ステップ1302でハッシュ値を「0」にセットして、プロセスを初期化することから始まる。

【００４３】

入力コンピュータファイルのコンテンツを備えるデータオブジェクト1304は、入力ファイルソースからのキャラクタを読みとるステップ1306で実行される。ステップ1308において、ステップ1306で読みとられたキャラクタは、32ビット値のアレイ（このサイズのアレイは、例として記載されているだけである）にインデックスを付けるために利用される。その後、ステップ1310で、ステップ1308で見つかったインデックスを付けられた32ビット値は、現在の32ビットハッシュ値に排他的ＯＲ化（「ＸＯＲ化」）される。

【００４４】

判断ステップ1312で、予め決められたパターンが見つかった場合（例えば、最下位ビット「０」の選択された数）、スティッキーバイトはその時点でステップ1314において入力ファイルに配置される。一方、予め決められたパターンが判断ステップ1312で見つからなかった場合、判断ステップ1316において、入力ファイルでの予め決められたキャラクタの閾

値数（以下でより完全に説明するように、プロセス1300のローリングハッシュ関数によって操作されている）を超えたかどうかによって判断が下される。予め決められた閾値数を超えた場合、判断ステップ1312で検索される予め決められたパターン（例えば最下位ビット「0」のより小さい選択数）の幾つかのサブセット数が見つかったかどうかを確かめるために、プロセス1300は判断ステップ1318に進む。そうであった場合、スティッキーバイトはステップ1314に置かれる。

【００４５】

あるいは、判断ステップ1316において予め決められた閾値を超えなかった場合、プロセス1300は、既存の32ビットハッシュ値を１ビットポジションにわたってシフトさせる（「右」か「左」に）ステップ1320に進む。判断ステップ1322において、プロセス1300が作用するさらに別のキャラクタが存在する場合、入力ファイルソースの次のキャラクタはステップ1306で読みとられる。判断ステップ1318で予め決められたパターンのサブセットが見つからなかった場合、あるいは、ステップ1314でスティッキーバイトが配置された場合、上述したように、プロセスはステップ1320に進む。

【００４６】

データ・スティッキーバイト（または「スティッキーポイント」）は、共通ブロック要素がコンピュータ間で通信する必要なく複数の関連したおよび無関連のコンピュータで見つかるようコンピュータファイルをサブ分割するユニークな完全自動化の方法である。スティッキーポイントを見つける手段は、本質的に完全に数学的であり、ファイルのデータコンテンツに関係なく均等かつ適切に実行される。ハッシュファイルシステムの使用を通して、全てのデータオブジェクトはインデックスを付けられ、格納され、例えば、限定されるものではないが、MD4、MD5、SHAまたはSHA-1といった業界標準チェックサムを用いて検索されてよい。オペレーションにおいて、2つのファイルが同じチェックサムを有する場合、それらが同一ファイルであることは非常に可能性が高いと考えられ得る。本願で開示されているシステムおよび方法を用いると、データスティッキーポイントは、標準的な数理分布およびターゲットサイズが僅かな割合である標準偏差で生成できる。

【００４７】

データスティッキーポイントは、統計学的には稀なｎバイトの配列である。この場合、現在の３２ビットオリエンテッドマイクロプロセッサ技術でのインプリメンテーションにおいて容易であるという理由から、例においては３２ビットが使われている。ハッシュファイルシステムを実行するために利用されるハッシュ関数は適度に複雑な計算を必要とするが、それは現代コンピュータシステムの機能の範囲内では問題ない。ハッシュ関数は本質的に確率的であり、いかなるハッシュ関数でも２つの異なるデータオブジェクトに対して同じ結果をもたらすことができる。しかしながら、本願に開示されているシステムおよび方法は、従来のコンピュータハードウェアオペレーションでは容認されているエラー率よりはるかに低い、信頼できる使用のために許容できるレベル（すなわち何兆分の１の可能性）まで衝突の確率を減らす、周知でありかつ研究されたハッシュ関数を用いて、この問題を緩和している。

【００４８】

本発明のスティッキーバイトくくり出しシステムをより完全に説明するために、以下の定義を付随させる。

【００４９】

ローリングハッシュ

ローリングハッシュ関数は、標準ハッシュ関数の本質的な性質を維持しているが、その入力値の限られたメモリを有するようにデザインされている。具体的には、ハッシュ関数は以下の特性を有する：

１．固定長または可変長ウィンドウ（シーケンスの長さ）を有する；

２．同一のデータウィンドウを与えられた同一値を生じる；

すなわち、決定性がある。理想的には、生成されたハッシュ合計は、正当な値の全ての範囲に均一にまたがっている；

３．そのハッシュ合計は、ウィンドウの前か或いは後のデータに影響されない。

【００５０】

本発明の特定のインプリメンテーションで、３２ビットローリングハッシュ関数が使われてよい。その一般的なオペレーションは、以下の通りである： １）既存の３２ビットハッシュ値を１ビットにわたって（左か右に）シフトさせる； ２）入力ファイルソースからのキャラクタを読みとる； ３）そのキャラクタを用いて、３２ビット値のアレイにインデックスを付ける；および ４）インデックス付き３２ビット値を現在の３２ビットハッシュにＸＯＲ化する。次いで、オペレーションを繰り返す。

【００５１】

ローリングハッシュ値は次いで３２ビット値のままであり、全ての３２ビットがＸＯＲ演算の影響を受ける。シフト段階において、ビットのうちの一つが、残りの３１ビットから離れてローリングハッシュ「ウィンドウ」から移動し、残された３１ビットは１つの場所にわたって移動するが、それ以外に変化はない。この結果、ウィンドウは1つのユニットにわたって移動する。

【００５２】

少数のビットのみが、例えば最下位「０」の幾つかの数といった多くのアプリケーションで一般に使われるので、更なる計算の労力が、「スティッキーバイト」の決定において要求されはしないが、６４ビット（または他のサイズ）のローリングハッシュが使われてよい点に留意する必要がある。 利用される関数が適切に分散された数を生じると仮定した場合、３２のビット数に対して、ゼロの最大数はもちろん３２であり、平均して４０億のキャラクタ毎に一回のみ発生する。４０億キャラクタはおよそ４ギガバイトのデータであり、大きな「チャンク」である。６４ビットハッシュ値を用いることは更に大きなチャンクサイズを生成する際の助けになるが、本願に開示されている本発明の特定のインプリメンテーションは約２Ｋのチャンクサイズを使用しているので、３２ビットローリングハッシュの全範囲を必要とすることはほとんどない。

【００５３】

以下のＣ言語例について考えてみる。ここで、「f」はバイトアレイであり、「i」はそのアレイへのインデックスであり、「hash」は計算されるハッシュ合計である。単純なローリングハッシュは以下のように書くことができる：

```
hash = (hash << 1) | f[i];
```

【００５４】

このハッシュは、大きくランダム化された整数値を生成する、入力バイト値（０～２５５）によってインデックス付けされる第２のアレイ「スクランブル」を含むことによって、改善可能である：

```
hash = (hash >> 1) | scramble[f[i]];
```

【００５５】

このローリングハッシュ関数の例は、３２ビット値の範囲にわたってかなり均一な数をもたらす。

【００５６】

閾値関数

閾値関数は、或る値が任意のレベルより上か下かを求める計算を実行する。 このアクションは不連続な結果を生じ、一部の値は閾値の上に、また他の値は閾値の下になる。閾値関数は、オプションとして、その入力値に対して或る変換を実行することもできる。例として：

```
threshold_value = (hash - 1) ^ hash;
```

あるいは：

```
threshold_value = ((hash - 1) ^ hash) + length;
```

【００５７】

スティッキーバイトのくくり出しに関する本発明のシステムおよび方法は、有利にデータセットを共通性を助長するシーケンスに分割する。続いての例は、現代の32ビットマイクロプロセッサで特に適切に実行できる閾値関数と共に３２ビットローリングハッシュを利用した、好適なインプリメンテーションの実例である。

【００５８】

３２ビットのローリングハッシュは、分割される入力シーケンスとしてバイトアレイ「f」を用いて生成される。ここで：

１．f[i] は、「f」に含まれるバイトシーケンスのｉ番目のバイトである；

２．「スクランブル」は、生成されたハッシュ合計でビットが「かき回される」ように選択された３２ビット整数の２５６要素アレイである。これらの整数は、それらの複合型排他的論理和（「ＸＯＲ」）がゼロに等しくなる（該整数が複合型の１と０ビットのバイナリ同等を有することを意味する）特徴を有するように、一般的に選択される；

３．「^」オペレータは、排他的論理和関数である；

４．length_of_byte_sequence は、分割されるシーケンスの長さである；

５．関数「output_sticky_point」は、分割要素インデックスと共に呼び出される；

６．「閾値」は、所望の長さのシーケンスが生成されるよう選択される値である。すなわち、値が大きくなればなるほど、より長いシーケンスが生成される。

【００５９】

例１：

```
int hash= 0; // ハッシュ合計の初期値はゼロである；
int sticky_bits = 0;
int last_sticky_point = 0;
for( int i=0; i < length_of_byte_sequence; i++ ) {
// シーケンス「f」の各バイトに対して、「hash」はファイルのローリングハッシュを表す；
hash = (hash >> 1) | scramble[f[i]];
// sticky_bitsは、より大きな値がより低い頻度で生成されるという特徴を有する非均一な値である；
sticky_bits = (hash - 1) ^ hash;
// この計算は、現在のバイトをパーティションの最後と考える必要があるかどうかを判断する；
if( sticky_bits > threshold )
 {
output_sticky_point(i);
// 「last_sticky_point」は、既存のパーティションの長さを閾値計算の要素として求めるのに（オプションとして）用いられる前のパーティションのインデックスを覚えている；
last_sticky_point = i;
}
}
```

【００６０】

本発明のシステムおよび方法は、値のシーケンスを通して順次に進み、ローリングハッシュ合計値を計算する。インデックス「i」におけるそのハッシュ合計値は、インデックスｉ-31～ｉにおけるバイトにのみ依存している。31未満のｉの場合、ハッシュ合計は０～ｉの間のバイトに対する値を反映する。広範なバイト値を用いる「f」の入力テキストおよび適切に選択されたランダム化値のセットが「スクランブル」に存在すると仮定した場合、ハッシュ合計は適切に分散された値域を生じ、所望の３２ビット数範囲に広くかつ均一にまたがる。一部のバイトシーケンスが適切に分散された数を生じない点に留意する必要があるが、この動作を有するバイトシーケンスは典型的な入力テキストに関しては一般的でない。

【００６１】
「sticky_bits」値は、現在のハッシュ合計を使って計算される。この値は、非常に非均一な分散を有するよう、また３２ビット値の全範囲にまたがる数を生成するようにできていて、以下の表１で示されるように、より大きな値はそれらの大きさに反比例して生成される：
【表１】

スティッキーバイト値	値を有するシーケンスの%
1	50.00000
3	25.00000
7	12.50000
15	6.25000
31	3.12500
63	1.56250
127	0.78125
255	0.39062
511	0.19531
1023	0.09766
2047	0.04883
4095	0.02441
8191	0.01221
16383	0.00610
32767	0.00305
65535	0.00153
etc.	etc.

【００６２】
更なる変更なしで、この特定の例は、平均の９５％となる、標準偏差を含むシーケンスの長さを有する統計プロパティを実証する。これを補正するために、「sticky_bits」値は、現在のシーケンスの長さと組み合わされ、より均一に間隔をあけられたパーティションを生成することができる。この点に関しては、「sticky_weighting」は、「sticky_bits」値の重み対現在のパーティションの長さを調節するのに用いられるファクタである：
sticky_weighting + (i-last_sticky_point)
【００６３】
例２：

```
int hash= 0; // ハッシュ合計の初期値はゼロである；
int sticky_bits = 0;
int last_sticky_point = 0;
for( int i=0; i < length_of_byte_sequence; i++ ) {
// シーケンス「f」の各バイトに対して、「hash」はファイルのローリングハッシュを表す；
hash = (hash >> 1) | scramble[f[i]];
// sticky_bitsは、より大きな値がより低い頻度で生成されるという特徴を有する非均一な値である；
```

```
sticky_bits = (hash - 1) ^ hash;
// この計算は、現在のバイトをパーティションの最後と考える必要があるかどうかを判断する；
if( sticky_bits + sticky_weighting * (i-last_sticky_point) > threshold )
 {
output_sticky_point(i);
// 「last_sticky_point」は、既存のパーティションの長さを閾値計算の要素として求めるのにオプションとして用いられる前のパーティションのインデックスを覚えている；
last_sticky_point = i;
}
}
```

【００６４】
本発明のシステムおよび方法のこの特定の実施例において、調整は、より一貫したパーティションサイズを作り出すために成されてきた。これは、潜在的なパーティションサイズが増加するのにつれて、閾値への「圧力」を本質的に増大させてパーティションを作成することによって、達成される。これを達成する様々な方法が際限なくあるが、前述の例は一つの実例を示すことを意図している点に留意されたい。
【００６５】
理解されるように、本願で開示されているデータシーケンスを組織化しないで決定する本発明のシステムおよび方法は、現代のコンピュータプロセッサを用いて大容積のデータをそれらの共通シーケンスにくくり出す、効率的かつ容易な達成手段を提供する。従来のくくり出し技術と異なり、それは、共通性を確立するために、シーケンス比較、コミュニケーションまたは前のアクションの履歴記録を必要としない。 更に、本発明のシステムおよび方法は、分割されるデータのタイプの影響を本質的に受けず、テキストファイル、バイナリファイル、セット画像、オーディオおよびビデオクリップ、静止画像等に一貫して実行する。
【００６６】
本願に開示されているスティッキーバイトくくり出し技術はまた、共通性がシーケンス内の可変ロケーションにある場合でも、その共通性を識別するのに役立つパーティションを有利に作成する；例えば、特定のドキュメントファイルの２つのバージョン間の差が小さなものだけだったにしても、スティッキーバイトくくり出しは、キャラクタの挿入または削除にもかかわらず、くくり出されたドキュメント間に高い共通性をもたらす。更に、本発明のシステムおよび方法は、「スライドする」か又は絶対位置を変更するデータで、共通性を識別するのに役立つパーティションまたはブレークポイントを生成する。
【００６７】
本質において、本発明のシステムおよび方法は、共通データシーケンスを迅速かつ効率的に見つける方法における問題点を効果的に解決する。更に、それは、前述の分割方式に基づく情報を格納するようにできているシステムに高い可能性で存在しているデータシーケンスの別のコード化を検索するのに使用することができる。本発明のスティッキーバイトくくり出し技術は、典型的なコンピュータファイルシステムで共通シーケンスを検索する場合に特に適切に機能し、最も有名な圧縮アルゴリズムをも含み、多くが基本ファイルサイズ低減技術として共通性くくり出しを利用する幾つかのテスト組合せに対して非常に高い圧縮比をもたらす。
【００６８】
本願で用いられるように、用語「インターネットインフラ」は様々なハードウェアおよびソフトウェアメカニズムを含んでいるが、該用語は、主に、一つのネットワークノードから別のネットワークノードにデータパケットを移動させる機能を有するルーター、ルーターソフトウェアおよびこれらのルーター間の物理リンクを指す。また、本願で用いられるように、「デジタルシーケンス」はコンピュータプログラムファイル、コンピュータアプリケーション、データファイル、ネットワークパケット、マルチメディア（オーディオお

よびビデオを含む）といったストリーミングデータ、テレメトリーデータ、およびデジタルまたは数字シーケンスによって表されることができる他のあらゆるデータフォームを含むことができるが、これに限定されるものではない。

【００６９】

特定の例示的なスティッキーバイトくくり出し技術およびコンピュータシステムに関連して本発明の原理を以上で説明してきたが、前述の説明は例としてのみ成されており、本発明の範囲を限定するものではないことは明白に理解されたであろう。特に、前述の開示内容の教示が他の変形実施例を当業者に示唆していることは理解されたであろう。このような変形実施例は、それ自体が既に知られており、本願で既に説明された特徴の代わりにまたはそれに加えて使用可能な他の特徴を含んでもよい。特許請求の範囲は本出願において特定の特徴の組合せに対して明確に述べているが、本願における開示の範囲はまた、特許請求の範囲と同一の発明に関連があるにせよ、また本発明が直面したものと同じ技術的な問題の全てを軽減するにせよ、当業者には明らかな明示的または暗示的に開示されている特徴のあらゆる新規な特徴または組合わせ、またはそれを一般化した例または変形実施例を含むことを理解されたい。出願人は、これによって本出願のまたはそこから導き出される更なる出願の全ての手続き中に、前記の特徴および／または前記の特徴の組合せを新しい特許請求の範囲に発展させる権利を確保する。

【図面の簡単な説明】

【図１】　本発明のシステムおよび方法を実施可能な代表的なネットワーク化されたコンピュータ環境の高レベル図解である。

【図２】　本発明のシステムおよび方法のユーティライゼーションに対して可能な操作環境のより詳細な概念図であり、ここで、任意の数のコンピュータまたはデータセンターで維持されるファイルは、インターネット接続を介した、例えば地理的に多様なロケーションに位置する多くの独立ノードの冗長アレイ（「RAIN」）ラックに対する分散型コンピュータシステムに格納されてよい。

【図３】　本発明のハッシュファイルシステムにコンピュータファイルをエントリするステップを表すロジックフローチャートであり、ここで、ファイルのハッシュ値は、セット（データベース）で予め維持されるファイルのハッシュ値とチェックされる。

【図４】　ファイルまたは他のデータシーケンスをハッシュされたピースにブレークアップするステップを表す更なるロジックフローチャートであり、多くのデータピースを生成させるのに加えて、各ピースに対して確率的に固有のハッシュ値を対応させる。

【図５】　ファイルの各ピースのハッシュ値とセットまたはデータベースの既存のハッシュ値との比較を表す別のロジックフローチャートであり、レコードの生成は、全ファイルピースに対して単一のハッシュ値と様々なピースのハッシュ値との等価を示し、そこで、新しいデータピースおよびそれに対応する新しいハッシュ値がセットに加えられる。

【図６】　ファイルハッシュまたはディレクトリリストハッシュ値を既存のディレクトリリストハッシュ値と比較して、新しいファイルまたはディレクトリリストハッシュ値をセットディレクトリリストに加えるステップを示す、更に別のロジックフローチャートである。

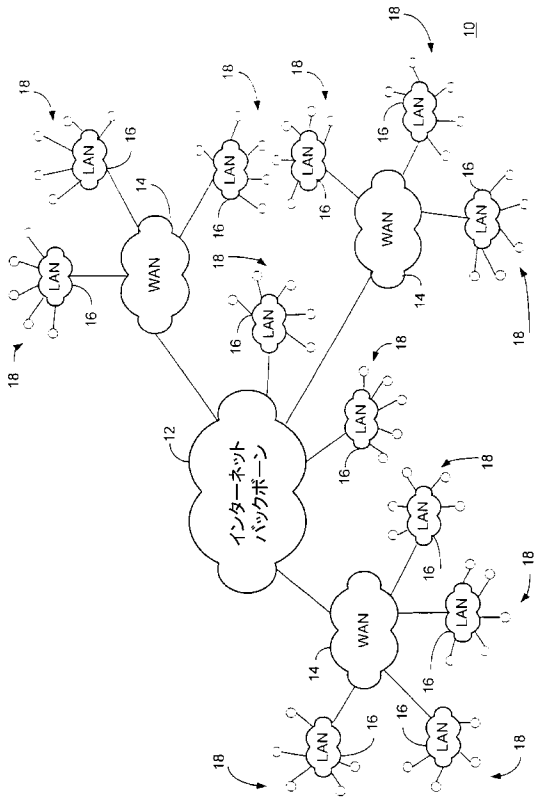
【図７】　典型的なファイルの特定のピースの編集の前後に、代表的なコンピュータファイルのピースをそれに対応するハッシュ値と比較している。

【図８】　本発明のシステムおよび方法によって導き出されることができる複合データが明示的に表されるデータと事実上同一ではあるが、「レシピ」（例えばその対応しているハッシュによって表されるデータの連結またはハッシュによって表されるデータを用いた関数の結果）によって、その代わりに作成されてもよいという事実の概念図である。

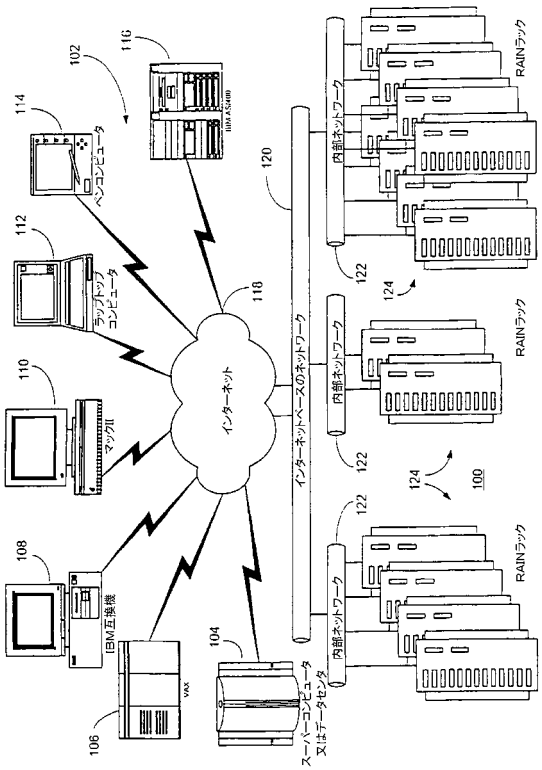
【図９】　本発明のハッシュファイルシステムおよび方法が、ハッシュ値をそれらが表すデータへのポインターとして使用することによって、冗長シーケンスの再利用を最適化するためにデータを編成するのにどのように利用可能であるかを表す別の概念図であり、ここで、データは、明確なバイトシーケンス（原子データ）またはシーケンスのグループ（複合物）として表されてよい。

【図１０】　典型的な160ビットハッシュ値のハッシュファイルシステムアドレス変換関数を示す簡略図である。
【図１１】　本発明のシステムおよび方法で用いられるインデックス・ストライプ分割関数の典型的な簡略図である。
【図１２】　Day 1に多くのプログラムおよびドキュメントファイルを有している典型的な家庭用コンピュータのデータのバックアップに用いられる本発明のシステムおよび方法の全体的な機能を示す簡略図であり、ドキュメントファイルの一つが第３のドキュメントファイルを加えると共にDay 2で編集される。
【図１３】　多くの「スティッキーバイト」によってマークされる特定のドキュメントファイルの様々なピースの編集の前後両方での比較を示し、ピースのうちの一方はそれによって変更され、他方のピースは同じままである。
【図１４】　本発明による典型的なスティッキーバイトくくり出しプロセスの代表的なフローチャートである。

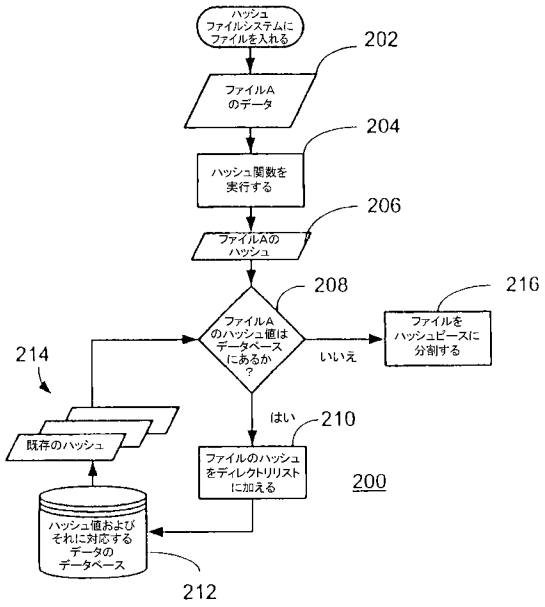
【図１】



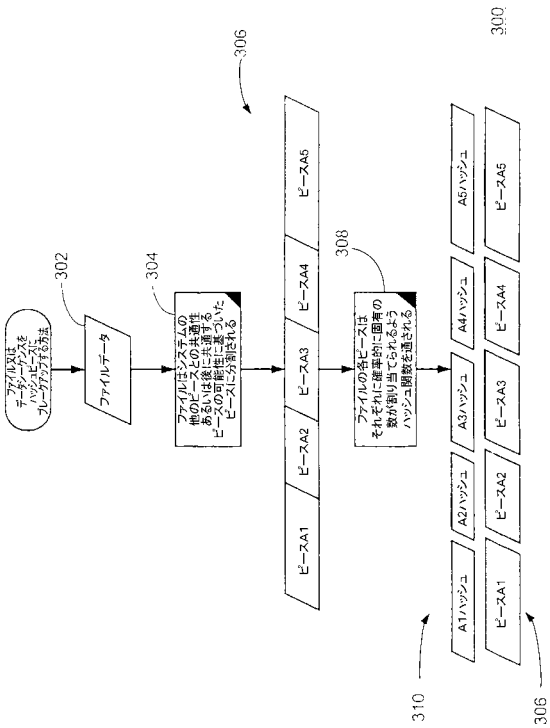
【図２】



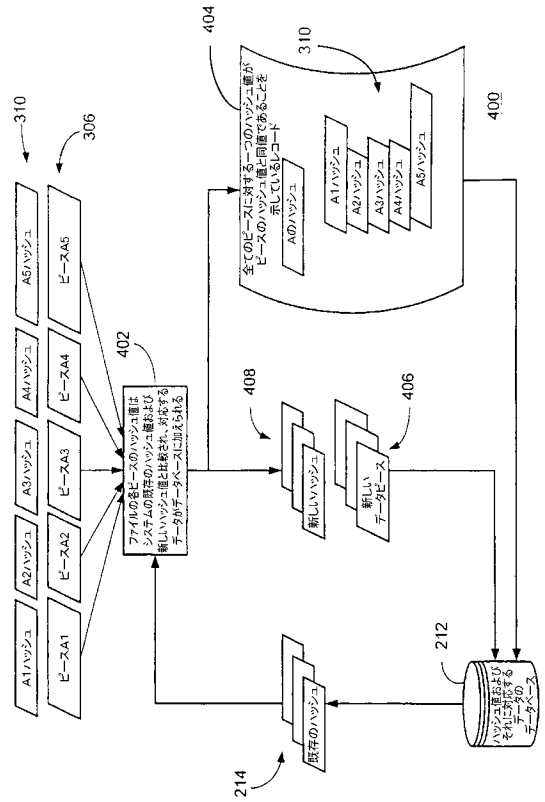
【図３】


ハッシュ
ファイルシステムに
ファイルを入れる
202
ファイルA
のデータ
204
ハッシュ関数を
実行する
206
ファイルAの
ハッシュ
208
ファイルA
のハッシュ値は
データベース
にあるか
？
いいえ
216
ファイルを
ハッシュピースに
分割する
214
はい
210
既存のハッシュ
ファイルのハッシュ
をディレクトリリスト
に加える
200
ハッシュ値および
それに対応する
データの
データベース
212

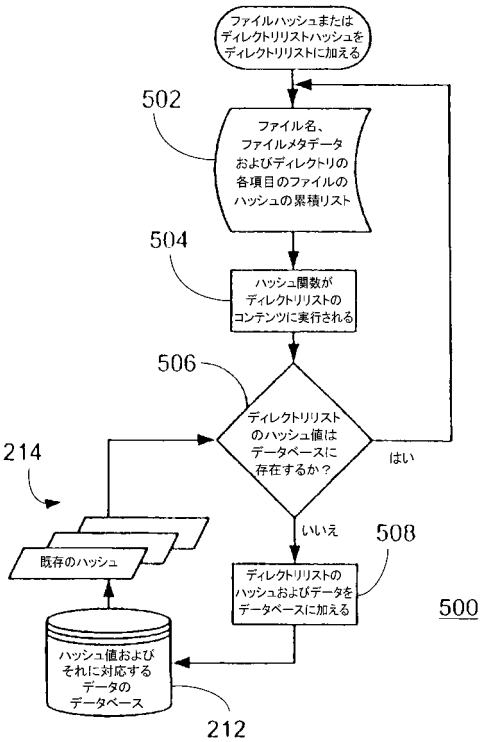
【図４】


300
306
302
304
308
ファイル又は
データベースを
ハッシュピースに
ブレークアップする方法
ファイルデータ
ファイルシステムの
他のピースとの
あるいは後に適用する
ピースの可能性に基づいた
ピースに分割される
ファイルの各ピースは
それぞれに確率的に固有の
数が割り当てられるような
ハッシュ関数を通される
ピースA5
ピースA4
ピースA3
ピースA2
ピースA1
A5ハッシュ
A4ハッシュ
A3ハッシュ
A2ハッシュ
A1ハッシュ
310
306

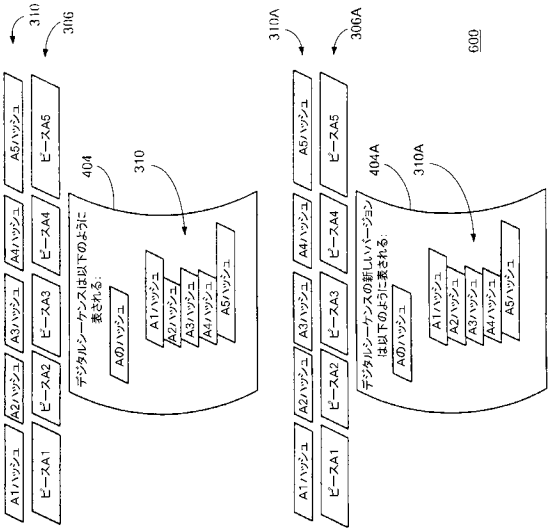
【図５】


404
310
400
全てのピースに対する一つのハッシュ値が
ピースのハッシュ値と同値であることを
示しているレコード
Aのハッシュ
A1ハッシュ
A2ハッシュ
A3ハッシュ
A4ハッシュ
A5ハッシュ
310
306
A5ハッシュ
A4ハッシュ
A3ハッシュ
A2ハッシュ
A1ハッシュ
ピースA5
ピースA4
ピースA3
ピースA2
ピースA1
402
ファイルの各ピースのハッシュ値は
システムの既存のハッシュ値および
新しいハッシュ値と比較され、対応する
データがデータベースに加えられる
408
406
新しいハッシュ
新しい
データベース
212
既存のハッシュ
ハッシュ値および
それに対応する
データの
データベース
214

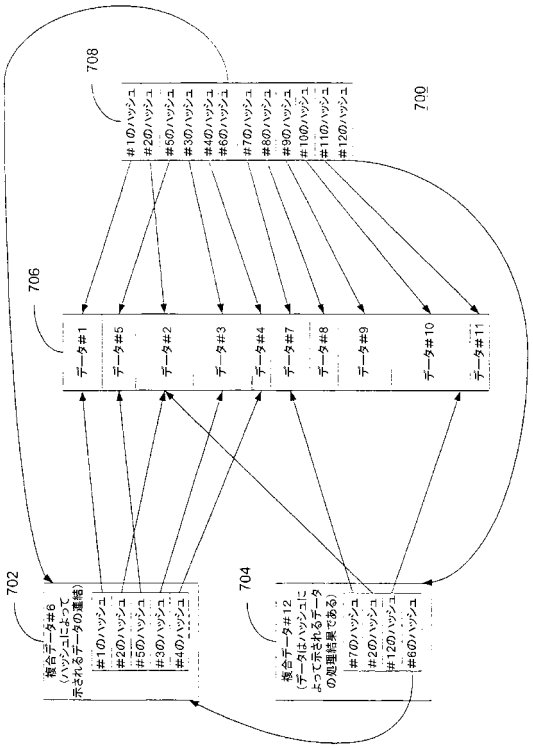
【図６】


ファイルハッシュまたは
ディレクトリリストハッシュを
ディレクトリリストに加える
502
ファイル名、
ファイルメタデータ
およびディレクトリの
各項目のファイルの
ハッシュの累積リスト
504
ハッシュ関数が
ディレクトリリストの
コンテンツに実行される
506
ディレクトリリスト
のハッシュ値は
データベースに
存在するか？
はい
いいえ
214
既存のハッシュ
508
ディレクトリリストの
ハッシュおよびデータを
データベースに加える
500
ハッシュ値および
それに対応する
データの
データベース
212

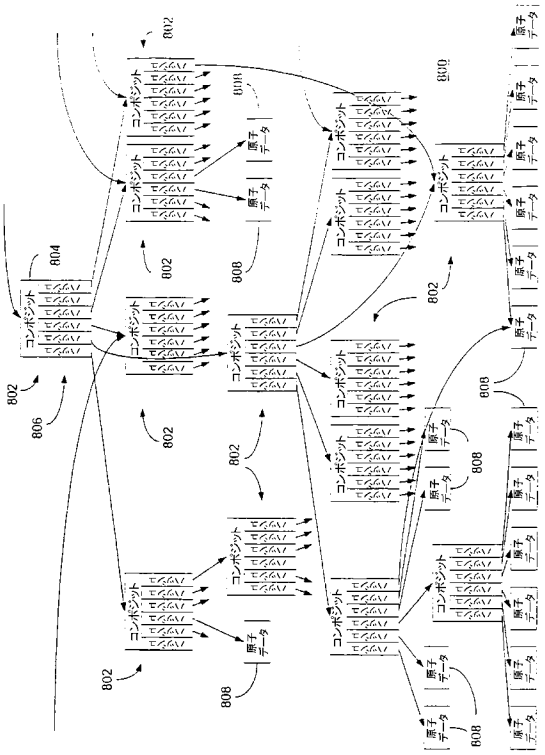
【図７】


310
306
310A
306A
600
404
404A
310A
A1ハッシュ
A2ハッシュ
A3ハッシュ
A4ハッシュ
A5ハッシュ
ビースA1
ビースA2
ビースA3
ビースA4
ビースA5
デジタルシーケンスは以下のように表される:
A0ハッシュ
デジタルシーケンスの新しいバージョンは以下のように表される:

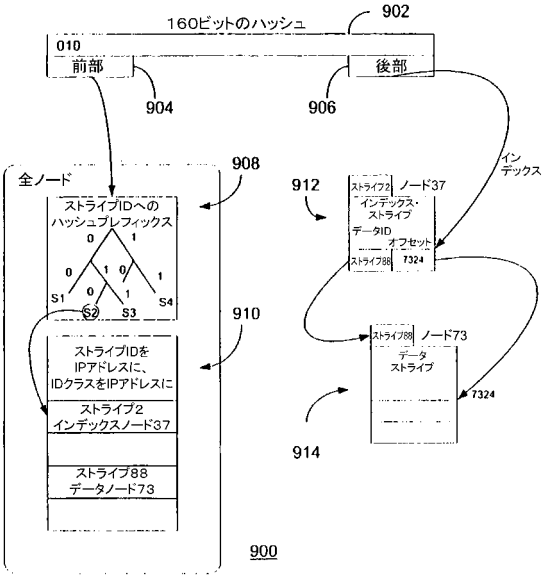
【図８】


700
702
704
706
708
#1のハッシュ
#2のハッシュ
#5のハッシュ
#3のハッシュ
#4のハッシュ
#6のハッシュ
#7のハッシュ
#8のハッシュ
#9のハッシュ
#10のハッシュ
#11のハッシュ
#12のハッシュ
データ#1
データ#5
データ#2
データ#3
データ#4
データ#7
データ#8
データ#9
データ#10
データ#11
複合データ#6
(データはハッシュによって示されるデータの連結)
複合データ#12
(データはハッシュによって示されるデータの連結結果である)

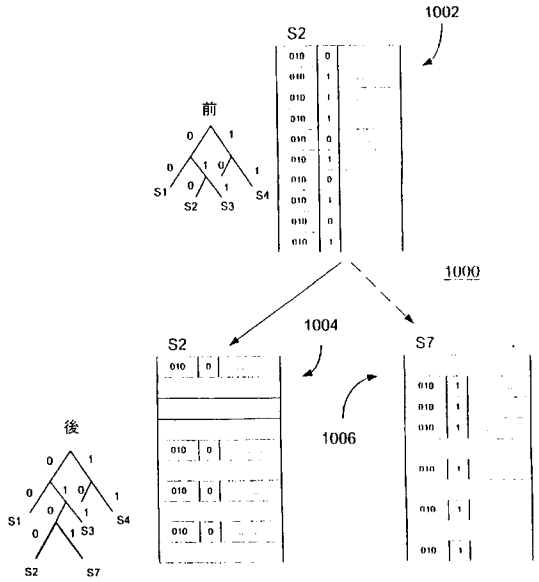
【図９】


802
804
806
808
810
コンポジット
原子データ

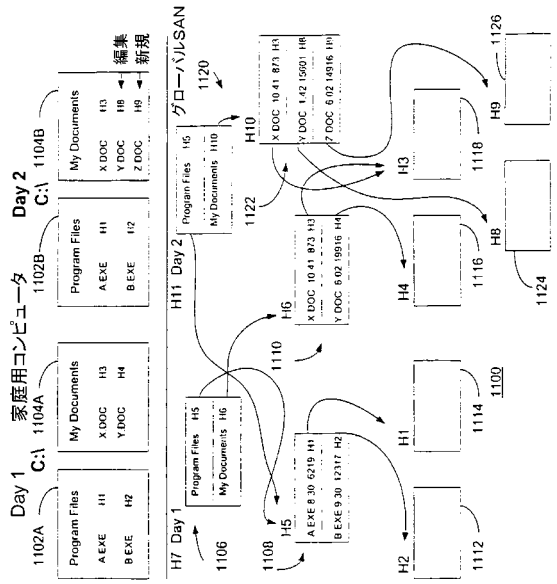
【図１０】


160ビットのハッシュ
902
010
前部
後部
904
906
全ノード
908
910
912
914
900
ストライプIDへのハッシュプレフィックス
ストライプIDをIPアドレスに、IDクラスをIPアドレスに
ストライプ2
インデックスノード37
ストライプ88
データノード73
インデックス
ノード37
ノード73
データストライプ
7324

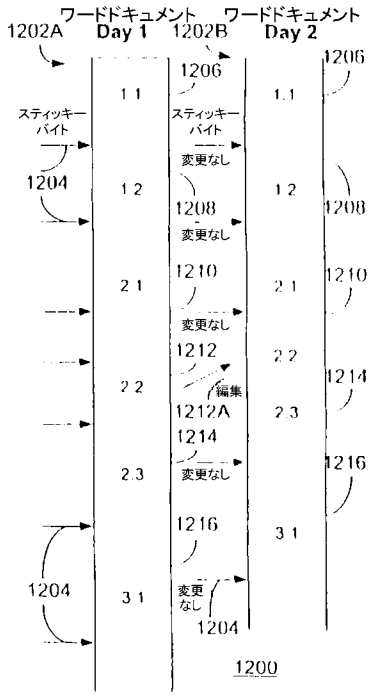
【図１１】



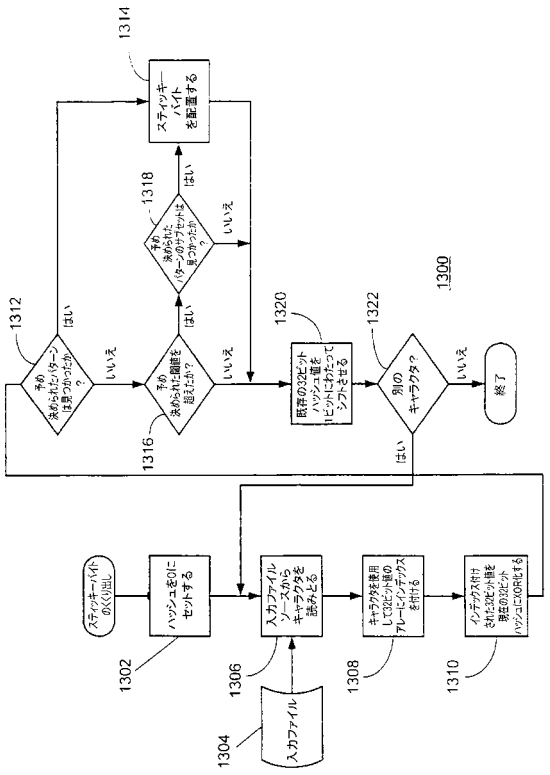
【図１２】



【図１３】



【図１４】



フロントページの続き

(72)発明者　モールトン，　グレゴリー，　ハーガン
アメリカ合衆国，　カリフォルニア州，　アーヴァイン，　ベイベリー　ウェイ　６

審査官　廣瀬　文雄

(56)参考文献　米国特許第０５９９０８１０（ＵＳ，Ａ）
特開平０９－１１４７１７（ＪＰ，Ａ）
特表２００３－５２４２４３（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F　12/00