(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2004-514968
(P2004-514968A)
(43)公表日　平成16年5月20日(2004.5.20)

(51)Int.Cl.[7]	F I		テーマコード（参考）
G06F 12/00	G06F	12/00　501B	5B082

審査請求 未請求　予備審査請求 有　（全 76 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)
(85)翻訳文提出日	平成15年4月28日(2003.4.28)
(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)
(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)

(71)出願人　502300428
アヴァマー　テクノロジーズ　インコーポレイテッド
アメリカ合衆国，カリフォルニア州，アーヴァイン，テクノロジー　ドライヴ　１エー
(74)代理人　100094318
弁理士　山田　行一
(74)代理人　100104282
弁理士　鈴木　康仁
(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

Fターム(参考)　5B082 CA11　CA18

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)【要約】
共通シーケンスといったデジタルシーケンスでブレークポイントを決める「スティッキーバイト」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めるシステムおよび方法を確認できる。スティッキーバイトくくり出しは、最適な共通性に近いところを通常もたらす、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。これは、ローリングハッシュ合計、および本願で開示されている典型的な実施例における、決定論的にデータシーケンスに部をセットする閾値関数を用いることによって、達成される。ローリングハッシュおよび閾値関数は、最小限度の計算を必要とするようできている。この低オーバーヘッドは、くくり出しエンジン、またはデータセットにわたって後続の同期を選択する他のアプリケーションに提示するためにデータシーケンスを迅速に分割することを可能にする。

【特許請求の範囲】
【請求項１】
デジタル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シーケンスの一部にハッシュ関数を実行するステップと、
前記ハッシュ関数によって生成されたハッシュ値を第１の予め決められた数パターンに対してモニタするステップと、
前記第１の予め決められた数パターンが発生した場合、前記デジタル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、
を備える方法。
【請求項２】
前記ハッシュ関数を実行するステップは、ローリングハッシュ関数手段によって実行される、請求項１記載の方法。
【請求項３】
前記第１の予め決められた数パターンは、ビットパターンである、請求項１記載の方法。
【請求項４】
前記ハッシュ関数を実行するステップは、構造上効率的なｎビットハッシュ関数手段によって実行される、請求項１記載の方法。
【請求項５】
前記構造上効率的なｎビットハッシュ関数は、３２ビットハッシュ関数を備える、請求項４記載の方法。
【請求項６】
前記第１の予め決められた数パターンは、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７】
前記連続ビットシーケンスは複数の末端ビットを備える、請求項６記載の方法。
【請求項８】
前記複数の末端ビットは１１ビットを備える、請求項７記載の方法。
【請求項９】
前記１１ビットは「０」である、請求項７記載の方法。
【請求項１０】
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と、
前記閾値制限が達成された場合、前記モニタステップに対して少なくとも第２の予め決められた数パターンを確立するステップと、
前記第２の予め決められた数パターンが発生した場合、代わりに前記デジタル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１１】
前記第２の予め決められた数パターンは、ビットパターンである、請求項１０記載の方法。
【請求項１２】
前記第２の予め決められた数パターンは、前記第１の予め決められた数パターンのサブセットである、請求項１１記載の方法。
【請求項１３】
前記第１の予め決められた数パターンは連続した１１ビットシーケンスを備え、前記第２の予め決められた数パターンは前記１１ビットのうちの１０を備える、請求項１２記載の方法。
【請求項１４】
前記第１の予め決められた数パターンは１１の連続した「０」を備え、前記第２の予め決められた数パターンは１０の連続した「０」を備える、請求項１３記載の方法。
【請求項１５】

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と、
前記デジタルシーケンスに前記ブレークポイントをマーク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１６】
前記デジタルシーケンスに前記ブレークポイントをマークする確率を上げる前記ステップは、少なくとも所望のチャンクサイズの関数である、請求項１５記載の方法。
【請求項１７】
前記デジタルシーケンスに前記ブレークポイントをマーク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現在の部分の長さの関数である、請求項１５記載の方法。
【請求項１８】
前記デジタルシーケンスに前記ブレークポイントをマーク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の予め決められた数パターンを利用するステップと、
前記第２の予め決められた数パターンが発生した場合、代わりに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１５記載の方法。
【請求項１９】
第１のデジタルシーケンスで第１のブレークポイントを決める方法であって、
前記第１のデジタルシーケンスの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の予め決められた数パターンを得るまで、前記第１のデジタルシーケンスの開始位置から始まっている前記第１のデジタルシーケンスの前記サブセットグループにハッシュ関数を実行するステップと、
前記ハッシュ値で前記第１の予め決められた数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２０】
前記数パターンはビットパターンを備える、請求項１９記載の方法。
【請求項２１】
前記ハッシュ関数を実行するステップは、ローリングハッシュ関数手段によって実行される、請求項１９記載の方法。
【請求項２２】
前記ハッシュ値で前記第１の予め決められた数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のデジタルシーケンスのサブセットグループにハッシュ関数を更に実行するステップと、
前記ハッシュ値で前記第１の予め決められた数パターンを再び得た場合に、前記第１のデジタル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１９記載の方法。
【請求項２３】
前記ハッシュ関数を更に実行するステップは、ローリングハッシュ関数手段によって実行される、請求項２２記載の方法。
【請求項２４】
前記ハッシュ値で第２の予め決められた数パターンを決めるステップと、
確立された閾値制限を達成するまで、前記第１のデジタルシーケンスの前記サブセットグループに前記ハッシュ関数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の予め決められた数パターンを得た場合、代わりに前記第１のブレークポイントをマークするステップと、
を更に備える、請求項１９記載の方法。
【請求項２５】

前記ハッシュ値で前記第１の予め決められた数パターンを得るまで、第２のデジタルシーケンスの開始位置から始まっている前記サブセットグループにハッシュ関数を実行するステップと、
前記ハッシュ値で前記第１の予め決められた数パターンを得た場合に、前記第２のデジタルシーケンスに第２のブレークポイントをマークするステップと、
前記第１のブレークポイントで予め決められたハッシュ値を前記第２のブレークポイントで予め決められたハッシュ値と比較するステップと、
前記開始位置から前記第１のブレークポイントまでの前記第１のデジタルシーケンスの対応部分を、前記開始位置から前記第２のブレークポイントまでの前記第２のデジタルシーケンスの対応部分と同等視するステップと、
を更に備える、請求項１９記載の方法。
【請求項２６】
コンピュータープログラム製品であって、前記コンピュータープログラム製品は、
デジタルシーケンスが分割されるよう中に含まれるコンピュータ可読コードを有するコンピュータ可使用媒体を備え、前記コンピュータ可使用媒体が、
コンピュータに、少なくとも一部の前記デジタルシーケンスに対してハッシュ関数を実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
コンピュータに、第１の予め決められた数パターンに対する前記ハッシュ関数によって生成されるハッシュ値のモニタを行わせるよう構成されたコンピュータ可読プログラムコード・デバイスと、
前記第１の予め決められた数パターンが発生した場合に、コンピュータに、前記デジタルシーケンスへのブレークポイントのマークを行わせるよう構成されたコンピュータ可読プログラムコード・デバイスと、
を備える、コンピュータープログラム製品。
【請求項２７】
コンピュータに前記ハッシュ関数を実行させるよう構成された前記コンピュータ可読プログラムコード・デバイスは、ローリングハッシュ関数手段によって実行される、請求項２６記載のコンピュータープログラム製品。
【請求項２８】
前記第１の予め決められた数パターンはビットパターンである、請求項２６記載のコンピュータープログラム製品。
【請求項２９】
コンピュータに前記ハッシュ関数を実行させるよう構成された前記コンピュータ可読プログラムコード・デバイスは、構造上効率的なｎビットハッシュ関数手段によって実行される、請求項２６記載のコンピュータープログラム製品。
【請求項３０】
前記構造上効率的なｎビットハッシュ関数は、３２ビットハッシュ関数を備える、請求項２９記載のコンピュータープログラム製品。
【請求項３１】
前記第１の予め決められた数パターンは、連続ビットシーケンスを備える、請求項２６記載のコンピュータープログラム製品。
【請求項３２】
前記連続ビットシーケンスは複数の末端ビットを備える、請求項３１記載のコンピュータープログラム製品。
【請求項３３】
前記複数の末端ビットは１１ビットを備える、請求項３２記載のコンピュータープログラム製品。
【請求項３４】
前記１１ビットは「０」である、請求項３２記載のコンピュータープログラム製品。
【請求項３５】

コンピュータに、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を実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
前記閾値制限が達成された場合に、コンピュータに、前記モニタステップに対して少なくとも第２の予め決められた数パターンを確立するステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
前記第２の予め決められた数パターンが発生した場合に、コンピュータに、代わりに前記デジタル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を実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
を更に備える、請求項２６記載のコンピュータープログラム製品。
【請求項３６】
前記第２の予め決められた数パターンはビットパターンである、請求項３５記載のコンピュータープログラム製品。
【請求項３７】
前記第２の予め決められた数パターンは、前記第１の予め決められた数パターンのサブセットである、請求項３６記載のコンピュータープログラム製品。
【請求項３８】
前記第１の予め決められた数パターンは連続した１１ビットシーケンスを備え、前記第２の予め決められた数パターンは前記１１ビットのうちの１０を備える、請求項３７記載のコンピュータープログラム製品。
【請求項３９】
前記第１の予め決められた数パターンは１１の連続した「０」を備え、前記第２の予め決められた数パターンは１０の「０」を備える、請求項３８記載のコンピュータープログラム製品。
【請求項４０】
コンピュータに、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を実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
コンピュータに、前記デジタルシーケンスに前記ブレークポイントをマークする確率を上げるステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
を更に備える、請求項２６記載のコンピュータープログラム製品。
【請求項４１】
コンピュータに、前記デジタルシーケンスに前記ブレークポイントをマークする前記確率を上げるステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスは、少なくとも所望のチャンクサイズの関数である、請求項４０記載のコンピュータープログラム製品。
【請求項４２】
コンピュータに、前記デジタルシーケンスに前記ブレークポイントをマークする前記確率を上げるステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスは、少なくとも前記デジタルシーケンスの現在の部分の長さの関数である、請求項４０記載のコンピュータープログラム製品。
【請求項４３】
コンピュータに、前記デジタルシーケンスに前記ブレークポイントをマークする前記確率を上げるステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスは、
コンピュータに、前記ハッシュ値をモニタする前記ステップに第２の予め決められた数パターンを利用するステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
前記第２の予め決められた数パターンが発生した場合に、コンピュータに、代わりに前記ブレークポイントをマークするステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、

によって実行される、請求項４０記載のコンピュータープログラム製品。
【請求項４４】
コンピュータープログラム製品であって、前記コンピュータープログラム製品は、
第１のデジタルシーケンスで第１のブレークポイントが決定されるよう中に含まれるコンピュータ可読コードを有するコンピュータ可使用媒体を備え、前記コンピュータ可使用媒体が、
コンピュータに、前記第１のデジタルシーケンスのサブセットグループを決めるステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
ハッシュ値で第１の予め決められた数パターンを得るまで、コンピュータに、前記第１のデジタルシーケンスの開始位置から始まっている前記第１のデジタルシーケンスの前記サブセットグループにハッシュ関数を実行するステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
前記ハッシュ値で前記第１の予め決められた数パターンを得た場合に、コンピュータに、前記第１のブレークポイントをマークするステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
を備える、コンピュータープログラム製品。
【請求項４５】
前記数パターンはビットパターンを備える、請求項４４記載のコンピュータープログラム製品。
【請求項４６】
コンピュータに前記ハッシュ関数を実行させるよう構成された前記コンピュータ可読プログラムコード・デバイスは、ローリングハッシュ関数手段によって実行される、請求項４４記載のコンピュータープログラム製品。
【請求項４７】
前記ハッシュ値で前記第１の予め決められた数パターンを再び得るまで、コンピュータに、前記第１のブレークポイントから前記第１のデジタルシーケンスのサブセットグループにハッシュ関数を更に実行するステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
前記ハッシュ値で前記第１の予め決められた数パターンを再び得た場合に、コンピュータに、前記第１のデジタル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を実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
を更に備える、請求項４４記載のコンピュータープログラム製品。
【請求項４８】
コンピュータに前記ハッシュ関数を実行させるよう構成された前記コンピュータ可読プログラムコード・デバイスは、ローリングハッシュ関数手段によって実行される、請求項４７記載のコンピュータープログラム製品。
【請求項４９】
コンピュータに、前記ハッシュ値で第２の予め決められた数パターンを決めるステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
確立された閾値制限を達成するまで、コンピュータに、前記第１のデジタルシーケンスの前記サブセットグループに前記ハッシュ関数を実行するステップを継続するステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
前記ハッシュ値で前記第２の予め決められた数パターンを得た場合、コンピュータに、代わりに前記第１のブレークポイントをマークするステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
を更に備える、請求項４４記載のコンピュータープログラム製品。
【請求項５０】
前記ハッシュ値で前記第１の予め決められた数パターンを得るまで、コンピュータに、第２のデジタルシーケンスの開始位置から始まっている前記サブセットグループにハッシュ関数を実行するステップを実行させるよう構成されたコンピュータ可読プログラムコード

・デバイスと、
前記ハッシュ値で前記第１の予め決められた数パターンを得た場合に、コンピュータに、前記第２のデジタルシーケンスに第２のブレークポイントをマークするステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
コンピュータに、前記第１のブレークポイントで予め決められたハッシュ値を前記第２のブレークポイントで予め決められたハッシュ値と比較するステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
コンピュータに、前記開始位置から前記第１のブレークポイントまでの前記第１のデジタルシーケンスの対応部分を、前記開始位置から前記第２のブレークポイントまでの前記第２のデジタルシーケンスの対応部分と同等視するステップを実行させるよう構成されたコンピュータ可読プログラムコード・デバイスと、
を更に備える、請求項４４記載のコンピュータープログラム製品。
【請求項５１】
前記デジタルシーケンスに前記ブレークポイントをマークする前記確率を上げるステップは、前記シーケンスの一部のコンテンツ部分の関数である、請求項１５記載の方法。
【請求項５２】
前記ハッシュ関数は、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５３】
前記ハッシュ関数は、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５４】
前記ハッシュ関数は、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－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１０

００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) 〉 thres
hold )
 {
       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。

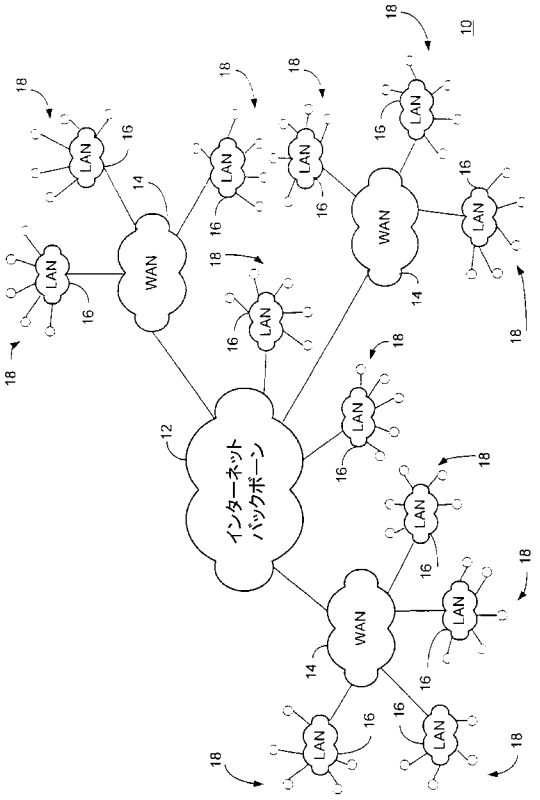
【図１３】

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。

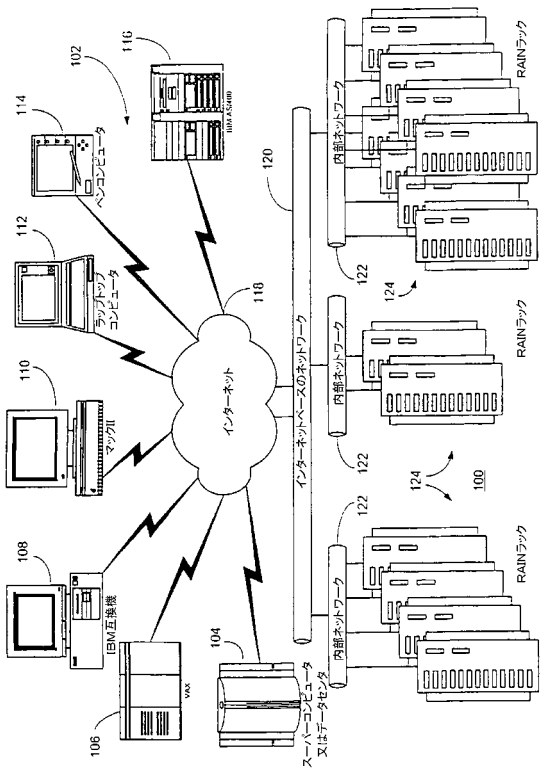
【図１４】

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。

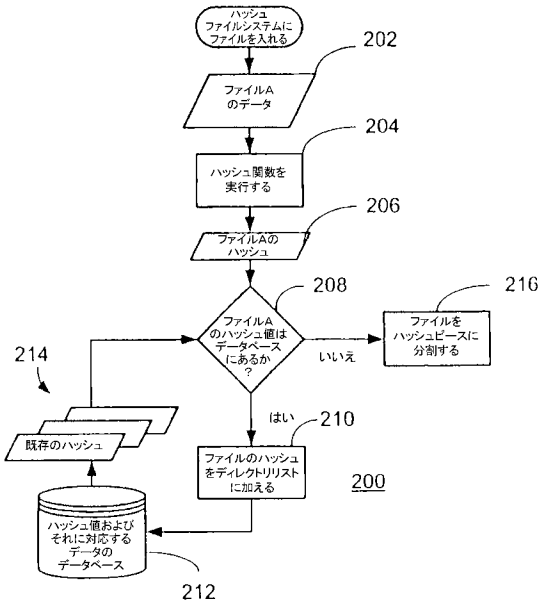
【図１】



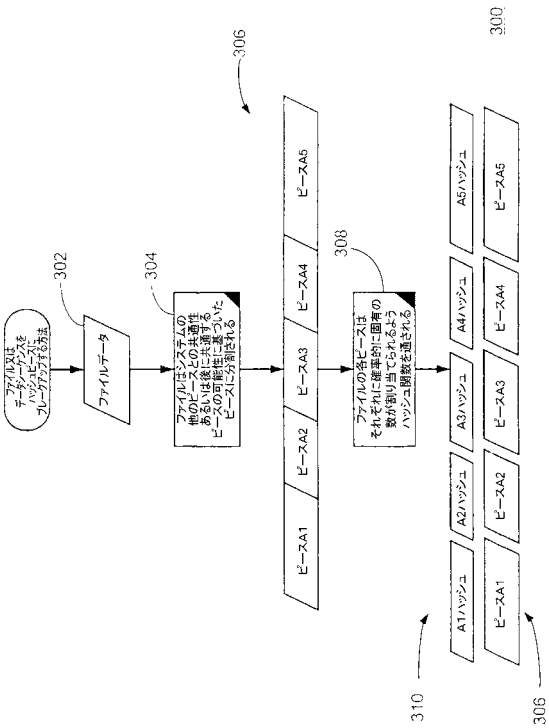
【図２】



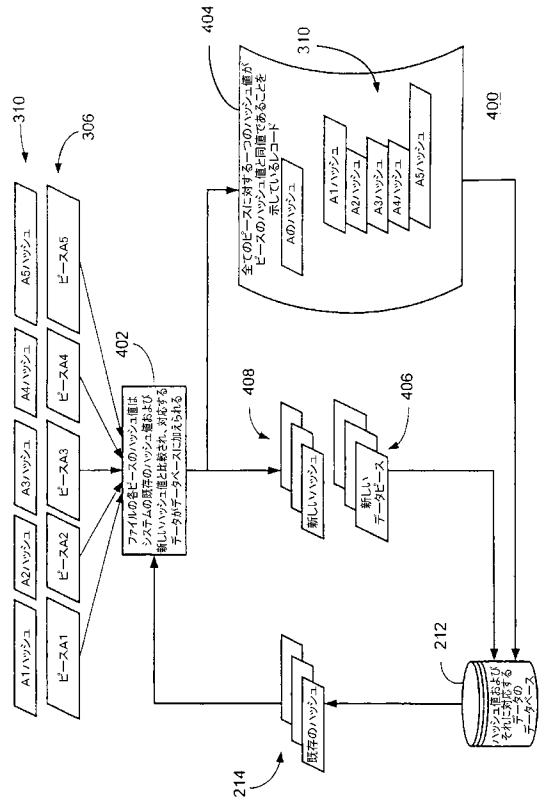
【図３】


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
はい
210
ファイルのハッシュ
をディレクトリスト
に加える
200
214
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
212

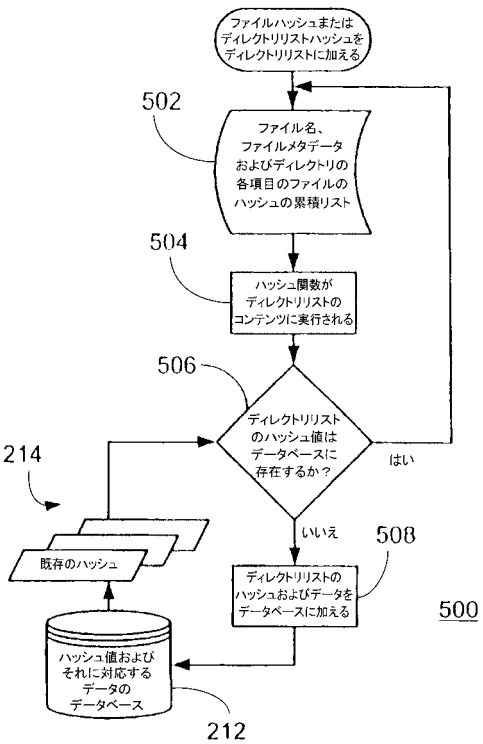
【図４】


300
306
302
304
308
ファイルは
データベースを
ハッシュピースを
フレームアップする方法
ファイルデータ
ファイルは、システムの
他のピースとの共通性
あるいは後に共通する
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
それぞれに個別的に個有の
数が割り当てられるよう
ハッシュ関数を通される
ピースA1
ピースA2
ピースA3
ピースA4
ピースA5
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

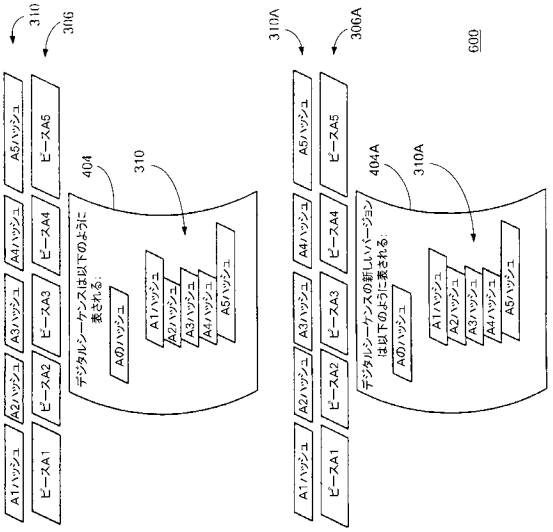
【図５】


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
A0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
データピース
212
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
既存のハッシュ
214

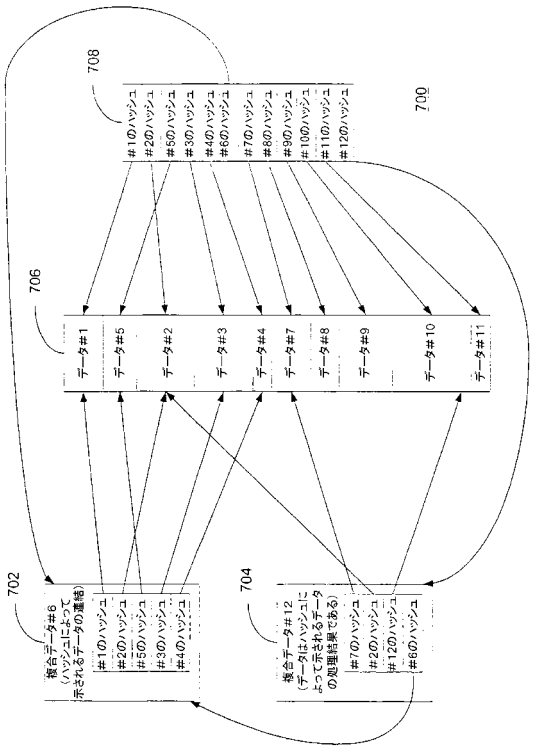
【図６】


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

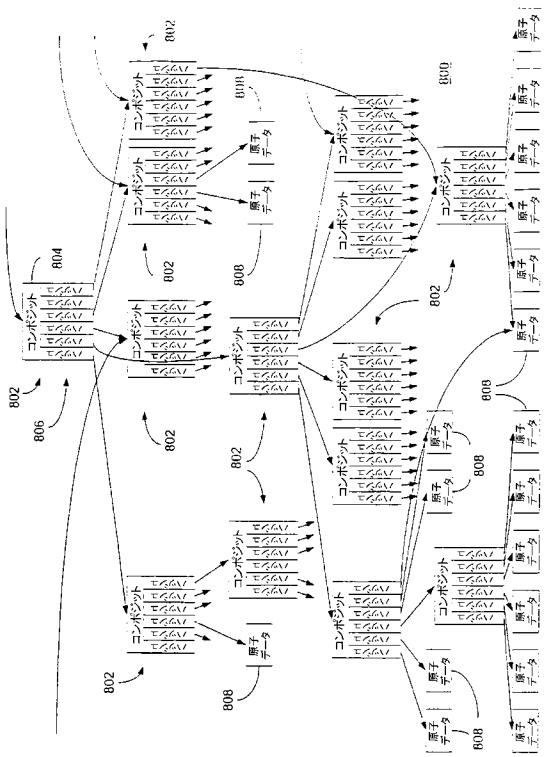
【図７】


310
306
310A
306A
600
404
404A
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のハッシュ
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。

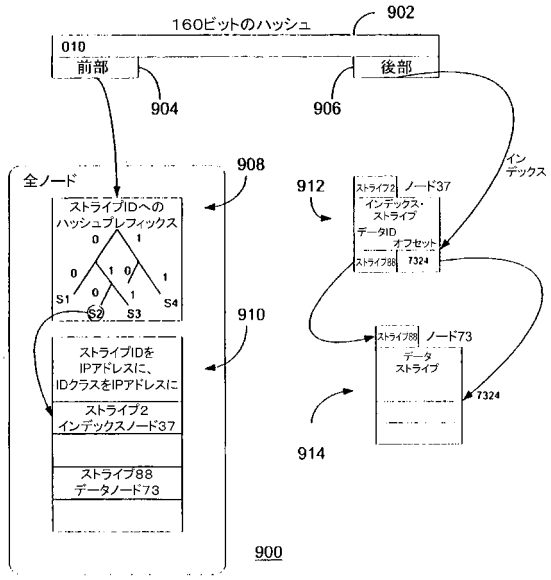
【図８】


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
#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
#4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
（ハッシュによって示されるデータの種類）
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処理結果である）

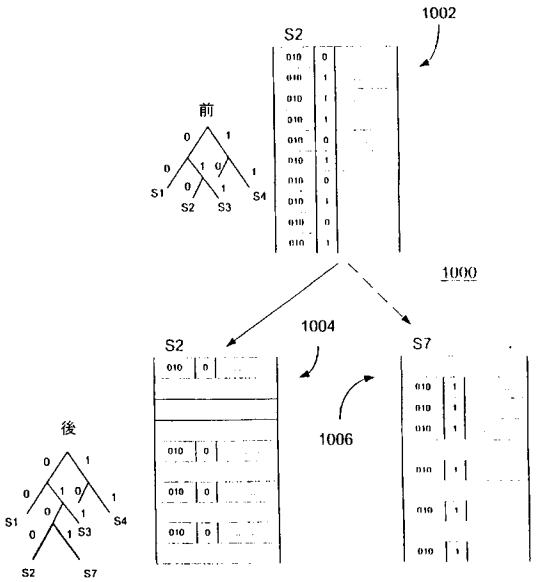
【図９】


802
804
806
808
810
コンポジット
エレメント
原子データ

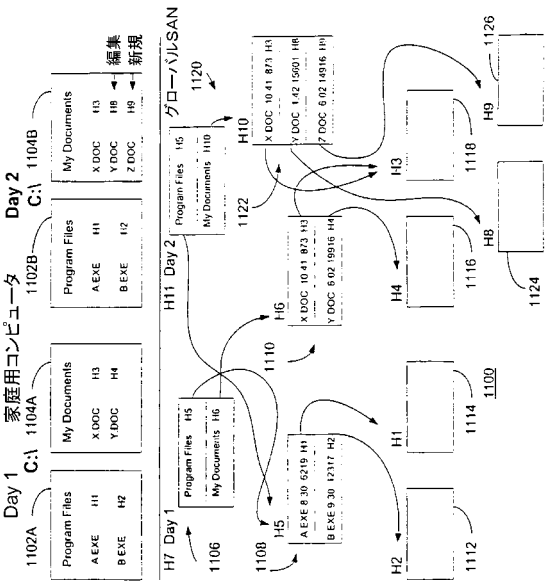
【図１０】


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
インデックス・ストライプ
データID
オフセット
データストライプ
7324
S1
S2
S3
S4

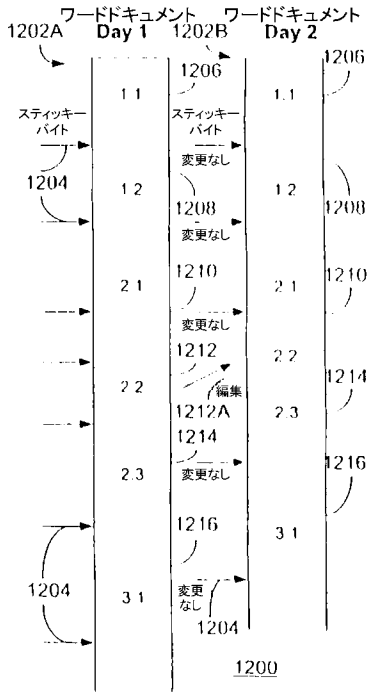
【図１１】



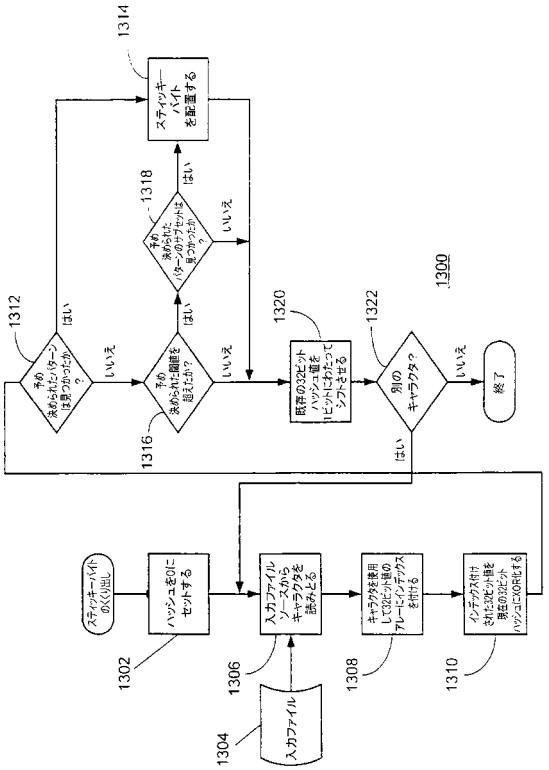
【図１２】



【図１３】



【図１４】



【国際公開パンフレット】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
10 May 2002 (10.05.2002)

PCT

(10) International Publication Number
WO 02/37689 A1

(51) International Patent Classification⁷: H03M 7/30, H04L 23/00, G06F 7/00, 7/06, 7/22

(21) International Application Number: PCT/US01/31306

(22) International Filing Date: 4 October 2001 (04.10.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/245,920 6 November 2000 (06.11.2000) US
09/777,149 5 February 2001 (05.02.2001) US

(71) Applicant *(for all designated States except US)*: **AVAMAR TECHNOLOGIES, INC.** [US/US]; 1A Technology Drive, Irvine, CA 92618 (US).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: **MOULTON, Gregory, Hagan** [US/US]; 6 Bayberry Way, Irvine, CA 92612 (US).

(74) Agents: **BURTON, Carol, W.** et al.; Hogan & Hartson LLP, Suite 1500, 1200 17th Street, Denver, CO 80202 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 02/37689 A1

(54) Title: SYSTEM FOR IDENTIFYING COMMON DIGITAL SEQUENCES

(57) Abstract: A system and method for unorchestrated determination of data sequences using "sticky byte" factoring to determine breakpoints in digital sequences such that common sequences can be identified. Sticky byte factoring provides an efficient method of dividing a data set into pieces that generally yields near optimal commonality. This is effectuated by employing a rolling hashsum and, in an exemplary embodiment disclosed herein, a threshold function to deterministically set divisions in a sequence of data. Both the rolling hash and the threshold function are designed to require minimal computation. This low overhead makes it possible to rapidly partition a data sequence for presentation to a factoring engine or other applications that prefer subsequent synchronization across the data set.

SYSTEM FOR IDENTIFYING COMMON DIGITAL SEQUENCES

COPYRIGHT NOTICE/PERMISSION

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document of the patent disclosure as it appears in the United States Patent and Trademark Office patent file or records, but otherwise, reserves all copyright rights whatsoever. The following notice applies to the software and data and described below, inclusive of the drawing figures where applicable: Copyright © 2000, Undoo Technologies.

BACKGROUND OF THE INVENTION

The present invention relates, in general, to the field of systems and methods for the unorchestrated determination of data sequences using "sticky byte" factoring to determine breakpoints in digital sequences. More particularly, the present invention relates to an efficient and effective method of dividing a data set into pieces that generally yields near optimal commonality.

Modern computer systems hold vast quantities of data - on the order of a billion, billion bytes in aggregate. Incredibly, this volume tends to quadruple each year and even the most impressive advances in computer mass storage architectures cannot keep pace.

The data maintained in most computer mass storage systems has been observed to have the following interesting characteristics: 1) it is almost never random and is, in fact, highly redundant; 2) the number of unique sequences in this data sums to a very small fraction of the storage space it actually occupies; 3) a considerable amount of effort is required in attempting to manage this volume of data, with much of that being involved in the identification an removal of redundancies (i.e.

duplicate files, old versions of files, purging logs, archiving etc.); and 4) large amounts of capital resources are dedicated to making unnecessary copies, saving those copies to local media and the like.

A system that factored redundant copies would reduce the number of storage volumes otherwise needed by orders of magnitude. However, a system that factors large volumes of data into their common sequences must employ a method by which to determine those sequences. Conventional methods that attempt to compare one data sequence to another typically suffer from extreme computational complexity and these methods can, therefore, only be employed to factor relatively small data sets. Factoring larger data sets is generally only done using simplistic methods such as using arbitrary fixed sizes. These methods factor poorly under many circumstances and the efficient factoring of large data sets has long been a persistent and heretofore intractable problem in the field of computer science.

SUMMARY OF THE INVENTION

Disclosed herein is a system and method for unorchestrated determination of data sequences using "sticky byte" factoring to determine breakpoints in digital sequences such that common sequences can be identified. Sticky byte factoring provides an efficient method of dividing a data set into pieces that generally yields near optimal commonality. As disclosed herein, this may be effectuated by employing a hash function with periodic reset of the hash value or, in a preferred embodiment, a rolling hashsum. Further, in the particular exemplary embodiment disclosed herein, a threshold function is utilized to deterministically set divisions in a digital or numeric sequence, such as a sequence of data. Both the rolling hash and the threshold function are designed to require minimal computation. This low overhead makes it possible to rapidly partition a data sequence for presentation to a factoring engine or other applications that prefer subsequent synchronization across the entire data set.

Among the significant advantages of the system and method disclosed herein is that its calculation requires neither communication nor comparisons (like conventional factoring systems) to perform well. This is particularly true in a distributed environment where, while conventional systems require communication to compare one sequence to another, the system and method of the present invention can be performed in isolation using only the sequence being then considered.

In operation, the system and method of the present invention provides a fully automated means for dividing a sequence of numbers (e.g. bytes in a file) such that common elements may be found on multiple related and unrelated computer systems without the need for communication between the computers and without regard to the data content of the files. Broadly, what is disclosed herein is a system and method for a data processing system which includes a fully automated means to partition a sequence of numeric elements (i.e. a sequence of bytes) so that common sequences may be found without the need for searching, comparing, communicating or coordinating with other processing elements in the operation of finding those sequences. The system and method of the present invention produces "sticky byte" points that partition numeric sequences with a distribution that produces subsequences of the type and size desired to optimize commonality between partitions.

BRIEF DESCRIPTION OF THE DRAWINGS

The aforementioned and other features and objects of the present invention and the manner of attaining them will become more apparent and the invention itself will be best understood by reference to the following description of a preferred embodiment taken in conjunction with the accompanying drawings, wherein:

Fig. 1 is a high level illustration of a representative networked computer environment in which the system and method of the present invention may be implemented;

Fig. 2 is a more detailed conceptual representation of a possible operating environment for utilization of the system and method of the present invention wherein files maintained on any number of computers or data centers may be stored in a decentralized computer system through an Internet connection to a number of Redundant Arrays of Independent Nodes ("RAIN") racks located, for example, at geographically diverse locations;

Fig. 3 is logic flow chart depicting the steps in the entry of a computer file into the hash file system of the present invention wherein the hash value for the file is checked against hash values for files previously maintained in a set, database;

Fig. 4 is a further logic flow chart depicting the steps in the breakup of a file or other data sequence into hashed pieces resulting in the production of a number of data pieces as well as corresponding probabilistically unique hash values for each piece;

Fig. 5 is another logic flow chart depicting the comparison of the hash values for each piece of a file to existing hash values in the set or database, the production of records showing the equivalence of a single hash value for all file pieces with the hash values of the various pieces and whereupon new data pieces and corresponding new hash values are added to the set;

Fig. 6 is yet another logic flow chart illustrating the steps in the comparison of file hash or directory list hash values to existing directory list hash values and the addition of new file or directory list hash values to the set directory list;

Fig. 7 is a comparison of the pieces of a representative computer file with their corresponding hash values both before and after editing of a particular piece of the exemplary file;

Fig. 8 is a conceptual representation of the fact that composite data which may be derived by means of the system and method of the present invention is effectively the same as the data represented explicitly but may instead be created by a "recipe" such as the concatenation of data represented by its corresponding hashes or the result of a function using the data represented by the hashes;

Fig. 9 is another conceptual representation of how the hash file system and method of the present invention my be utilized to organize data to optimize the reutilization of redundant sequences through the use of hash values as pointers to the data they represent and wherein data may be represented either as explicit byte sequences (atomic data) or as groups of sequences (composites);

Fig. 10 is a simplified diagram illustrative of a hash file system address translation function for an exemplary 160 bit hash value;

Fig. 11 is a simplified exemplary illustration of an index stripe splitting function for use with the system and method of the present invention;

Fig. 12 is a simplified illustration of the overall functionality of the system and method of the present invention for use in the backup of data for a representative home computer having a number of program and document files on Day 1 and wherein one of the document files is edited on Day 2 together with the addition of a third document file;

Fig. 13 illustrates the comparison of various pieces of a particular document file marked by a number of "sticky bytes" both before and following editing wherein one of the pieces is thereby changed while other pieces remain the same; and

Fig. 14 is a representative flow chart for an exemplary sticky byte factoring process in accordance with the present invention.

DESCRIPTION OF A REPRESENTATIVE EMBODIMENT

The present invention is illustrated and described in terms of a distributed computing environment such as an enterprise computing system using public communication channels such as the Internet. However, an important feature of the present invention is that it is readily scaled upwardly and downwardly to meet the needs of a particular application. Accordingly, unless specified to the contrary the present invention is applicable to significantly larger, more complex network environments as well as small network environments such as conventional LAN systems.

With reference now to Fig. 1, the present invention may be utilized in conjunction with a novel data storage system on a network 10. In this figure, an exemplary internetwork environment 10 may include the Internet which comprises a global internetwork formed by logical and physical connection between multiple wide area networks ("WANs") 14 and local area networks ("LANs") 16. An Internet backbone 12 represents the main lines and routers that carry the bulk of the data traffic. The backbone 12 is formed by the largest networks in the system that are operated by major Internet service providers ("ISPs") such as GTE, MCI, Sprint, UUNet, and America Online, for example. While single connection lines are used to conveniently illustrate WANs 14 and LANs 16 connections to the Internet backbone 12, it should be understood that in reality, multi-path, routable physical connections exist between multiple WANs 14 and LANs 16. This makes internetwork 10 robust when faced with single or multiple failure points.

A "network" comprises a system of general purpose, usually switched, physical connections that enable logical connections between processes operating on nodes 18. The physical connections

implemented by a network are typically independent of the logical connections that are established between processes using the network. In this manner, a heterogeneous set of processes ranging from file transfer, mail transfer, and the like can use the same physical network. Conversely, the network can be formed from a heterogeneous set of physical network technologies that are invisible to the logically connected processes using the network. Because the logical connection between processes implemented by a network is independent of the physical connection, internetworks are readily scaled to a virtually unlimited number of nodes over long distances.

In contrast, internal data pathways such as a system bus, peripheral component interconnect ("PCI") bus, Intelligent Drive Electronics ("IDE") bus, small computer system interface ("SCSI") bus, and the like define physical connections that implement special-purpose connections within a computer system. These connections implement physical connections between physical devices as opposed to logical connections between processes. These physical connections are characterized by limited distance between components, limited number of devices that can be coupled to the connection, and constrained format of devices that can be connected over the connection.

In a particular implementation of the present invention, storage devices may be placed at nodes 18. The storage at any node 18 may comprise a single hard drive, or may comprise a managed storage system such as a conventional RAID device having multiple hard drives configured as a single logical volume. Significantly, the present invention manages redundancy operations across nodes, as opposed to within nodes, so that the specific configuration of the storage within any given node is less relevant.

Optionally, one or more of the nodes 18 may implement storage allocation management ("SAM") processes that manage data storage across nodes 18 in a distributed, collaborative fashion. SAM processes

preferably operate with little or no centralized control for the system as whole. SAM processes provide data distribution across nodes 18 and implement recovery in a fault-tolerant fashion across network nodes 18 in a manner similar to paradigms found in RAID storage subsystems.

However, because SAM processes operate across nodes rather than within a single node or within a single computer, they allow for greater fault tolerance and greater levels of storage efficiency than conventional RAID systems. For example, SAM processes can recover even when a network node 18, LAN 16, or WAN 14 becomes unavailable. Moreover, even when a portion of the Internet backbone 12 becomes unavailable through failure or congestion the SAM processes can recover using data distributed on nodes 18 that remain accessible. In this manner, the present invention leverages the robust nature of internetworks to provide unprecedented availability, reliability, fault tolerance and robustness.

With reference additionally now to Fig. 2, a more detailed conceptual view of an exemplary network computing environment in which the present invention is implemented is depicted. The Internetwork 10 of the preceding figure (or Internet 118 in this figure) enables an interconnected network 100 of a heterogeneous set of computing devices and mechanisms 102 ranging from a supercomputer or data center 104 to a hand-held or pen-based device 114. While such devices have disparate data storage needs, they share an ability to retrieve data via network 100 and operate on that data within their own resources. Disparate computing devices 102 including mainframe computers (e.g., VAX station 106 and IBM AS/400 station 116) as well as personal computer or workstation class devices such as IBM compatible device 108, Macintosh device 110 and laptop computer 112 are readily interconnected via internetwork 10 and network 100. Although not illustrated, mobile and other wireless devices may be coupled to the internetwork 10.

Internet-based network 120 comprises a set of logical connections, some of which are made through Internet 118, between a plurality of internal networks 122. Conceptually, Internet-based network 120 is akin to a WAN 14 (Fig. 1) in that it enables logical connections between geographically distant nodes. Internet-based networks 120 may be implemented using the Internet 118 or other public and private WAN technologies including leased lines, Fibre Channel, frame relay, and the like.

Similarly, internal networks 122 are conceptually akin to LANs 16 (Fig. 1) in that they enable logical connections across more limited stance than WAN 14. Internal networks 122 may be implemented using various LAN technologies including Ethernet, Fiber Distributed Data Interface ("FDDI"), Token Ring, Appletalk, Fibre Channel, and the like.

Each internal network 122 connects one or more redundant arrays of independent nodes (RAIN) elements 124 to implement RAIN nodes 18 (Fig. 1). Each RAIN element 124 comprises a processor, memory, and one or more mass storage devices such as hard disks. RAIN elements 124 also include hard disk controllers that may be conventional IDE or SCSI controllers, or may be managing controllers such as RAID controllers. RAIN elements 124 may be physically dispersed or co-located in one or more racks sharing resources such as cooling and power. Each node 18 (Fig. 1) is independent of other nodes 18 in that failure or unavailability of one node 18 does not affect availability of other nodes 18, and data stored on one node 18 may be reconstructed from data stored on other nodes 18.

In a particular exemplary implementation, the RAIN elements 124 may comprise computers using commodity components such as Intel-based microprocessors mounted on a motherboard supporting a PCI bus and 256 megabytes of random access memory ("RAM") housed in a conventional AT or ATX case. SCSI or IDE controllers may be

implemented on the motherboard and/or by expansion cards connected to the PCI bus. Where the controllers are implemented only on the motherboard, a PCI expansion bus may be optionally used. In a particular implementation, the motherboard may implement two mastering EIDE channels and a PCI expansion card which is used to implement two additional mastering EIDE channels so that each RAIN element 124 includes up to four or more EIDE hard disks. In the particular implementation, each hard disk may comprise an 80 gigabyte hard disk for a total storage capacity of 320 gigabytes or more per RAIN element. The hard disk capacity and configuration within RAIN elements 124 can be readily increased or decreased to meet the needs of a particular application. The casing also houses supporting mechanisms such as power supplies and cooling devices (not shown).

Each RAIN element 124 executes an operating system. In a particular implementation, a UNIX or UNIX variant operating system such as Linux may be used. It is contemplated, however, that other operating systems including DOS, Microsoft Windows, Apple Macintosh OS, OS/2, Microsoft Windows NT and the like may be equivalently substituted with predictable changes in performance. The operating system chosen forms a platform for executing application software and processes, and implements a file system for accessing mass storage via the hard disk controller(s). Various application software and processes can be implemented on each RAIN element 124 to provide network connectivity via a network interface using appropriate network protocols such as User Datagram Protocol ("UDP"), Transmission Control Protocol (TCP), Internet Protocol (IP) and the like.

With reference additionally now to Fig. 3, a logic flow chart is shown depicting the steps in the entry of a computer file into the hash file system of the present invention and wherein the hash value for the file is checked against hash values for files previously maintained in a set or database. Any digital sequence could also be entered into the

hash file system of the present invention in much the same way, but the current example wherein the digital sequence entered consists of a computer file is instructive.

The process 200 begins by entry of a computer file data 202 (e.g. "File A") into the hash file system ("HFS") of the present invention upon which a hash function is performed at step 204. The data 206 representing the hash of File A is then compared to the contents of a set containing hash file values at decision step 208. If the data 206 is already in the set, then the file's hash value is added to a hash recipe at step 210. This hash recipe consists of the data and associated structures needed to reconstruct a file, directory, volume, or entire system depending on the class of computer file data entered into the system. The contents of the set 212 comprising hash values and corresponding data is provided in the form of existing hash values 214 for the comparison operation of decision step 208. On the other hand, if the hash value for File A is not currently in the set, the file is broken into hashed pieces (as will be more fully described hereinafter) at step 216.

With reference additionally now to Fig. 4, a further logic flow chart is provided depicting the steps in the process 300 for breakup of a digital sequence (e.g. a file or other data sequence) into hashed pieces. This process 300 ultimately results in the production of a number of data pieces as well as corresponding probabilistically unique hash values for each piece.

The file data 302 is divided into pieces based on commonality with other pieces in the system or the likelihood of pieces being found to be in common in the future at step 304. The results of the operation of step 304 upon the file data 302 is, in the representative example shown, the production of four file pieces 306 denominated A1 through A5 inclusively.

Each of the file pieces 306 is then operated on at step 308 by placing it through individual hash function operations to assign a

probabilistically unique number to each of the pieces 306 A1 through A5. The result of the operation at step 308 is that each of the pieces 306 (A1 through A5) has an associated, probabilistically unique hash value 310 (shown as A1 Hash through A5 Hash respectively). The file division process of step 304 is described in greater detail hereinafter in conjunction with the unique "sticky byte" operation also disclosed herein.

With reference additionally now to Fig. 5, another logic flow chart is shown depicting a comparison process 400 for the hash values 310 of each piece 306 of the file to those of existing hash values 214 maintained in the set 212. Particularly, at step 402, the hash values 310 for each piece 306 of the file are compared to existing hash values 214 and new hash values 408 and corresponding new data pieces 406 are added to the set 212. In this way, hash values 408 not previously present in the set 212 are added together with their associated data pieces 406. The process 400 also results in the production of records 404 showing the equivalence of a single hash value for all file pieces with the hash values 310 of the various pieces 306.

With reference additionally now to Fig. 6, yet another logic flow chart is shown illustrating a process 500 for the comparison of file hash or directory list hash values to existing directory list hash values and the addition of new file or directory list hash values to the set directory list. The process 500 operates on stored data 502 which comprises an accumulated list of file names, file meta-data (e.g. date, time, file length, file type etc.) and the file's hash value for each item in a directory. At step 504, the hash function is run upon the contents of the directory list. Decision step 506 is operative to determine whether or not the hash value for the directory list is in the set 212 of existing hash values 214. If it is, then the process 500 returns to add another file hash or directory list hash to a directory list. Alternatively, if the hash

value for the directory list is not already in the set 212, the hash value and data for the directory list are added to the set 212 at step 508.

With reference additionally now to Fig. 7, a comparison 600 of the pieces 306 of a representative computer file (i.e. "File A") with their corresponding hash values 310 is shown both before and after editing of a particular piece of the exemplary file. In this example, the record 404 contains the hash value of File A as well as the hash values 310 of each of the pieces of the file A1 through A5. A representative edit or modification of the File A may produce a change in the data for piece A2 (now represented by A2-b) of the file pieces 306A along with a corresponding change in the hash value A2-b of the hash values 310A. The edited file piece now produces an updated record 404A that includes the modified hash value of File A and the modified hash value of piece A2-b.

With reference additionally now to Fig. 8, a conceptual representation 700 is shown illustrative of the fact that composite data (such as composite data 702 and 704) derived by means of the system and method of the present invention, is effectively the same as the data 706 represented explicitly but is instead created by a "recipe", or formula. In the example shown, this recipe includes the concatenation of data represented by its corresponding hashes 708 or the result of a function using the data represented by the hashes. The data blocks 706 may be variable length quantities as shown and the hash values 708 are derived from their associated data blocks. As previously stated, the hash values 708 are a probabilistically unique identification of the corresponding data pieces but truly unique identifications can be used instead or intermixed therewith. It should also be noted that the composite data 702, 704 can also reference other composite data many levels deep while the hash values 708 for the composite data can be derived from the value of the data the recipe creates or the hash value of the recipe itself.

With reference additionally now to Fig. 9, another conceptual representation 800 is shown of how a hash file system and method may be utilized to organize data 802 to optimize the reutilization of redundant sequences through the use of hash values 806 as pointers to the data they represent and wherein data 802 may be represented either as explicit byte sequences (atomic data) 808 or as groups of sequences (composites) 804.

The representation 800 illustrates the tremendous commonality of recipes and data that gets reused at every level. The basic structure of the hash file system of the present invention is essentially that of a "tree" or "bush" wherein the hash values 806 are used instead of conventional pointers. The hash values 806 are used in the recipes to point to the data or another hash value that could also itself be a recipe. In essence then, recipes can point to other recipes that point to still other recipes that ultimately point to some specific data that may, itself, point to other recipes that point to even more data, eventually getting down to nothing but data.

With reference additionally now to Fig. 10, a simplified diagram 900 is shown illustrative of a hash file system address translation function for an exemplary 160 bit hash value 902. The hash value 902 includes a data structure comprising a front portion 904 and a back portion 906 as shown and the diagram 900 illustrates a particular "0 (1)" operation that is used for enabling the use of the hash value 902 to go to the location of the particular node in the system that contains the corresponding data.

The diagram 900 illustrates how the front portion 904 of the hash value 902 data structure may be used to indicate the hash prefix to stripe identification ("ID") 908 and how that is, in turn, utilized to map the stripe ID to IP address and the ID class to IP address 910. In this example, the "S2" indicates stripe 2 of index Node 37 912. The index stripe 912 of Node 37 then indicates stripe 88 of data Node 73

indicated by the reference numeral 914. In operation then, a portion of the hash value 902 itself may be used to indicate which node in the system contains the relevant data, another portion of the hash value 902 may be used to indicate which stripe of data at that particular node and yet another portion of the hash value 902 to indicate where within that stripe the data resides. Through this three step process, it can rapidly be determined if the data represented by the hash value 902 is already present in the system.

With reference additionally now to Fig. 11, a simplified exemplary illustration of an index stripe splitting function 1000 is shown for use with the system and method of the present invention. In this illustration, an exemplary function 1000 is shown that may be used to effectively split a stripe 1002 (S2) into two stripes 1004 (S2) and 1006 (S7) should one stripe become too full. In this example, the odd entries have been moved to stripe 1006 (S7) while the even ones remain in stripe 1004. This function 1000 is one example of how stripe entries may be handled as the overall system grows in size and complexity.

With reference additionally now to Fig. 12, a simplified illustration 1100 of the overall functionality of the system and method of the present invention is shown for use, for example, in the backup of data for a representative home computer having a number of program and document files 1102A and 1104A on Day 1 and wherein the program files 1102B remain the same on Day 2 while one of the document files 1104B is edited on Day 2 (Y.doc) together with the addition of a third document file (Z.doc).

The illustration 1100 shows the details of how a computer file system may be broken into pieces and then listed as a series of recipes on a global data protection network ("gDPN") to reconstruct the original data from the pieces. This very small computer system is shown in the form of a "snapshot" on "Day 1" and then subsequently on "Day 2". On "Day 1", the "program files H5" and "my documents H6" are illustrated

by numeral 1106, with the former being represented by a recipe 1108 wherein a first executable file is represented by a hash value H1 1114 and a second represented by a hash value H2 1112. The document files are represented by hash value H6 1110 with the first document being represented by hash value H3 1118 and the second by hash value H4 1116. Thereafter on "Day 2", the "program files H5" and "my documents" H10 indicated by numeral 1120 show that the "program files H5" have not changed, but the "my document H10" have. H10 indicated by numeral 1122 shows the "X.doc" is still represented by hash value H3 1118 while "Y.doc" is now represented by hash value H8 at number 1124. New document file "Z.doc" is now represented by hash value H9 at numeral 1126.

In this example, it can be seen that on Day 2, some of the files have changed, while others have not. In the files that have changed, some of the pieces of them have not changed while other pieces have. Through the use of the hash file system of the present invention, a "snap shot" of the computer system can be made on Day 1 (producing the necessary recipes for reconstruction of the computer files as they exist then) and then on Day 2 through the reuse of some of the previous day's recipes together with the reformulation of others and the addition of new ones to describe the system at that time. In this manner, the files that together constitute the computer system may be recreated in their entirety at any point in time on Day 1 and Day 2 for which a snapshot was taken, as well as from snapshots taken on any subsequent day. Thus any version of a computer file committed to the hash file system of the current invention can be retrieved from the system at any time after it has been initially committed.

With reference additionally now to Fig. 13, a comparison 1200 of various pieces of a particular document file marked by a number of "sticky bytes" 1204 is shown both before (Day 1 1202A) and following

editing (Day 2 1202B) wherein one of the pieces is thereby changed while other pieces remain the same.

For example, on Day 1, file 1202A comprises variable length pieces 1206 (1.1), 1208 (1.2), 1210 (2.1), 1212 (2.), 1214 (2.3) and 1216 (3.1). On Day 2, pieces 1206, 1208, 1210, 1214 and 1216 remain the same (thus having the same hash values) while piece 1212 has now been edited to produce piece 1212A (thus having a differing hash value).

With reference additionally now to Fig. 14, a representative sticky byte (or sticky point) factoring process 1300 is illustrated for possible use in the implementation of the present invention. The process 1300 begins by setting the hash value to "0" at step 1302 to initialize the process.

A data object 1304, comprising the contents of an input computer file, is acted upon at step 1306 wherein a character from the input file source is read. At step 1308, the character read at step 1306 is utilized to index an array of 32 bit values (this size array is described for purposes of example only). Thereafter, at step 1310, the indexed 32 bit value found at step 1308 is exclusive OR'd ("XOR'd") into the current 32 bit hash value.

At decision step 1312, if the predetermined pattern is found (e.g. a selected number of least significant bit "0's"), then the sticky byte is placed in the input file at that point at step 1314. On the other hand, if the predetermined pattern is not found at decision step 1312, then, at decision step 1316, a determination is made as to whether a predetermined threshold number of characters in the input file (having been operated on by the rolling hash function of process 1300, as will be more fully described hereinafter) has been exceeded. If the predetermined threshold number has been exceeded, then the process 1300 proceeds to decision step 1318 to see if some subset number of the predetermined pattern (e.g. a smaller selected number of least

significant bit "0's") being searched for in decision step 1312 has been found. If so, then the sticky byte is placed at step 1314.

Alternatively, if at decision step 1316 the predetermined threshold has not been exceeded, the process 1300 proceeds to step 1320 wherein the existing 32 bit hash value is shifted over one bit position (either "right" or "left"). At decision step 1322, if there is still another character to be operated upon by the process 1300, a next character in the input file source is read at step 1306. If at decision step 1318, the subset of the predetermined pattern is not found, or at step 1314 the sticky byte has been placed, the process proceeds to step 1320 as previously described.

Data sticky bytes (or "sticky points") are a unique, fully automated way to sub-divide computer files such that common elements may be found on multiple related and unrelated computers without the need for communication between the computers. The means by which data sticky points are found is completely mathematical in nature and performs equally well regardless of the data content of the files. Through the use of a hash file system, all data objects may be indexed, stored and retrieved using, for example, but not limited to an industry standard checksum such as: MD4, MD5, SHA, or SHA-1. In operation, if two files have the same checksum, it may be considered to be highly likely that they are the same file. Using the system and method disclosed herein, data sticky points may be produced with a standard mathematical distribution and with standard deviations that are a small percentage of the target size.

A data sticky point is a statistically infrequent arrangement of n bytes. In this case, an example is given with 32 bits because of its ease in implementation for current 32-bit oriented microprocessor technology. While the hashing function utilized to implement the hash file system requires a moderately complex computation, it is well within the capability of present day computer systems. Hashing functions are

inherently probabilistic and any hashing function can produce the same results for two different data objects. However, the system and method herein disclosed mitigates this problem by using well known and researched hashing functions that reduce the probability of collision down to levels acceptable for reliable use (i.e. one chance in a trillion trillion), far less than the error rates otherwise tolerated in conventional computer hardware operation.

For purposes of more fully explaining the sticky byte factoring system of the present invention, the following definitions pertain:

Rolling Hash:

A rolling hash function preserves the essential nature of a normal hash function but is designed to have limited memory of its input values. Specifically it is a hash function with these properties:

1. It has a fixed or variable length window (sequence length).

2. It produces the same value given the same window of data; that is, it is deterministic. Ideally the hashsums produced uniformly span the entire range of legal values.

3. Its hashsum is unaffected by the data either before or after the window.

In a particular implementation of the present invention, a 32-bit rolling hash function may be used. Its general operation is: 1) shift the existing 32-bit hash value over one bit (either left or right); 2) read a character from the input file source; 3) use that character to index an array of 32-bit values; and 4) XOR the indexed 32-bit value into the current 32-bit hash. The operation then repeats.

The rolling hash value then remains a 32-bit value and all 32 bits are affected by the XOR operation. In the shifting phase, one of the bits is moved out of the rolling hash "window" leaving the remaining 31

bits moved over one place but otherwise unchanged. The effect of this is to move the window over one unit.

It should be noted that a 64-bit (or other size) rolling hashes may be used although the additional computational effort may not be required in the determination of "sticky bytes" since only a small number of bits are generally used by many applications, e.g. some number of the least significant "0"'s. For a 32-bit number, the maximum number of zeros is, of course, 32, which would occur only once every 4 billion characters on average - assuming the function utilized produces well distributed numbers. Four billion characters is approximately four gigabytes of data; a large "chunk". Using 64-bit hash values would aid in producing even larger chunk sizes, but since the particular implementation of the present invention herein disclosed uses about a 2K chunk size, the full range of a 32-bit rolling hash is seldom required.

Consider the following C language example, wherein: "f" is an array of bytes, "i" is the index into that array, and "hash" is the hashsum being computed. A simple rolling hash might be written as:

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

This hash can be improved by including a second array "scramble" indexed by the input byte values (0 through 255) which produces large randomized integer values:

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

This example of a rolling hash function produces fairly uniform numbers across the range of 32 bit values.

Threshold Function:

A threshold function performs a calculation to determine whether a value is above or below a given level. This action produces discontinuous results, placing some values above and others below the

threshold. It may optionally perform some transformation on its input value. As an example:

threshold_value = (hash - 1) ^ hash;

or:

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

The system and method of the present invention for sticky byte factoring advantageously partitions data sets into sequences that promote commonality. The ensuing example, is illustrative of a preferred implementation utilizing a 32-bit rolling hash together with a threshold function which may be carried out particularly well with modern 32-bit microprocessors.

A rolling hash of 32 bits is generated using the byte array "f" as the input sequence to be partitioned where:

1. f[i] = is the i-th byte of the byte sequence contained in "f".

2. "Scramble" is a 256-element array of 32-bit integers chosen to "churn" the bits in the hashsum created. These integers are typically chosen to have the characteristic that their combined exclusive OR ("XOR") equals zero, meaning they have a combined binary equality of one and zero bits.

3. The "^" operator is the exclusive-or function.

4. The length_of_byte_sequence is the length of the sequence being partitioned.

5. The function "output_sticky_point" is called with the index of the partitioning element.

6. "threshold" is a value chosen to produce sequences of the desired length. That is, the larger the value, the longer the sequences produced.

Example 1:

int hash = 0; //initial value of hashsum is zero.

```
        int sticky_bits = 0;
        int last_sticky_point = 0;
        for( int i=O; i < length_of_byte_sequence; i++ ) {
//For each byte in the sequence of "f", "hash" //represents the rolling
hash of the file.
        hash = (hash >> 1) | scramble[f[i]];
//sticky_bits is a non-uniform value with the //characteristic that larger
values are produced less //frequently.
        sticky_bits = (hash - 1) ^ hash;
//This calculation determines whether the current byte //should be
considered the end of the partition.
        if( sticky_bits > threshold )
{
        output_sticky_point( i );
//"last_sticky_point" remembers the index of the //previous partition for
(optional) use in determining //the existing partition's length as a factor
in the //threshold calculation.
        last_sticky_point = i;
        }
}
```

The system and method of the present invention steps sequentially through a sequence of values and calculates a rolling hashsum value. That hashsum value at index "i" is dependent only on the bytes at indexes i-31 through i. In the case of i being less than 31, the hashsum reflects the values for the bytes between 0 and i. Assuming an input text in "f" that uses a large range of byte values and a well chosen set of randomizing values present in "scramble", the hashsum will produce a well-distributed range of values, that largely and uniformly spans the desired 32-bit numeric range. While it should be noted that some byte sequences do not produce well-distributed numbers, byte sequences having this behavior should be uncommon for typical input texts.

The "sticky_bits" value is calculated using the current hashsum. This value is designed to have a highly non-uniform distribution and to produce numbers that span the entire range of 32-bit values, with larger values being produced in inverse proportion to their magnitude as illustrated in the following Table 1:

Table 1

Sticky Byte Value	% of Sequences w/Value
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.

Without further modification, this particular example demonstrates the statistical property of having sequence lengths with a standard deviation that are 95% of their mean. In order to compensate for this, the "sticky_bits" value can be combined with the length of the current sequence to produce partitions more evenly spaced. In this regard, "sticky_weighting" is a factor that is used to adjust the weight of the "sticky_bits" value versus the length of the current partition.

sticky_weighting + (i-last_sticky_point)

Example 2:

```
int hash = 0; //initial value of hashsum is zero.
int sticky_bits = 0;
int last_sticky_point = 0;
for( int i=O; i < length_of_byte_sequence; i++ ) {
//For each byte in the sequence of "f", "hash" //represents the rolling
hash of the file.
hash = (hash >> 1) | scramble[f[i]];
//sticky_bits is a non-uniform value with the //characteristic that larger
values are produced less //frequently.
sticky_bits = (hash - 1) ^ hash;
//This calculation determines whether the current byte //should be
considered the end of the partition.
if( sticky_bits + sticky_weighting*(i-last_sticky_point) > threshold
)
{
output_sticky_point( i );
//"last-sticky-point" remembers the index of the //previous partition for
optional use in determining //the existing partition's length as a factor in
the //threshold calculation.
last_sticky_point = i;
}
}
```

In this particular embodiment of the system and method of the present invention, an adjustment has been made to produce more consistent partition sizes. This is effectuated by essentially increasing the "pressure" on the threshold to create a partition as the potential partition size increases. It should be noted that any number of a variety of methods for effectuating this end might be employed and the foregoing example is intended to illustrate but one.

As will be appreciated, the system and method of the present invention for unorchestrated determination of data sequences disclosed herein provides an efficient and readily effectuated means to factor large volumes of data into their common sequences using modern computer processors. Unlike conventional factoring techniques, it requires no sequence comparisons, communication, or historical record of previous actions in order to establish commonality. Further, the system and method of the present invention is essentially immune to the type of data being partitioned and it performs consistently on text files, binary files, set images, audio and video clips, still images and the like.

The sticky byte factoring technique disclosed herein also advantageously creates partitions that tend to identify commonality even when that commonality lies in variable locations within a sequence; for example, while the difference between two versions of a particular document file might be only minor, sticky byte factoring nevertheless produces a high commonality between the factored documents despite the insertion or deletion of characters. Moreover, the system and method of the present invention creates partitions, or breakpoints, that tend to identify commonality in data that "slides", or changes its absolute location.

In essence, the system and method of the present invention effectively solves the problem of how to locate common data sequences quickly and efficiently. Further, it can be used to search for alternative encodings of a sequence of data that would have a higher likelihood of being present in a system designed to store information based on such a partitioning scheme. The sticky byte factoring technique of the present invention performs particularly well when searching for common sequences in typical computer file systems and produces much higher compression ratios for some test suites that even the best known compression algorithms, many of which exploit commonality factoring as their fundamental file size reduction technique.

Although as used herein, the term "Internet infrastructure" encompasses a variety of hardware and software mechanisms, the term primarily refers to routers, router software, and physical links between these routers that function to transport data packets from one network node to another. As also used herein, a "digital sequence" may comprise, without limitation, computer program files, computer applications, data files, network packets, streaming data such as multimedia (including audio and video), telemetry data and any other form of data which can be represented by a digital or numeric sequence.

While there have been described above the principles of the present invention in conjunction with specific exemplary sticky byte factoring techniques and computer systems, it is to be clearly understood that the foregoing description is made only by way of example and not as a limitation to the scope of the invention. Particularly, it is recognized that the teachings of the foregoing disclosure will suggest other modifications to those persons skilled in the relevant art. Such modifications may involve other features which are already known per se and which may be used instead of or in addition to features already described herein. Although claims have been formulated in this application to particular combinations of features, it should be understood that the scope of the disclosure herein also includes any novel feature or any novel combination of features disclosed either explicitly or implicitly or any generalization or modification thereof which would be apparent to persons skilled in the relevant art, whether or not such relates to the same invention as presently claimed in any claim and whether or not it mitigates any or all of the same technical problems as confronted by the present invention. The applicants hereby reserve the right to formulate new claims to such features and/or combinations of such features during the prosecution of the present application or of any further application derived therefrom.

CLAIMS

What is claimed is:

1. A method for partitioning a digital sequence comprising:

performing a hash function on at least a portion of said digital sequence;

monitoring hash values produced by said hash function for a first predetermined numeric pattern; and

marking a breakpoint in said digital sequence when said first predetermined numeric pattern occurs.

2. The method of claim 1 wherein said step of performing said hash function is carried out by means of a rolling hash function.

3. The method of claim 1 wherein said first predetermined numeric pattern is a bit pattern.

4. The method of claim 1 wherein said step of performing said hash function is carried out by means of an architecturally efficient n-bit hash function.

5. The method of claim 4 wherein said architecturally efficient n-bit hash function comprises a 32-bit hash function.

6. The method of claim 1 wherein said first predetermined numeric pattern comprises a consecutive sequence of bits.

7. The method of claim 6 wherein said consecutive sequence of bits comprises a plurality of endmost bits.

8. The method of claim 7 wherein said plurality of endmost bits comprises eleven bits.

9. The method of claim 7 wherein said eleven bits are "0".

10. The method of claim 1 further comprising:

determining a threshold restriction for said step of monitoring said hash values;

establishing at least a second predetermined numeric pattern for said step of monitoring when said threshold restriction is met; and

alternatively marking said breakpoint in said digital sequence when said second predetermined numeric pattern occurs.

11. The method of claim 10 wherein said second predetermined numeric pattern is a bit pattern.

12. The method of claim 11 wherein said second predetermined numeric pattern is a subset of said first predetermined numeric pattern.

13. The method of claim 12 wherein said first predetermined numeric pattern comprises a consecutive sequence of eleven bits and said second predetermined numeric pattern comprises ten of said eleven bits.

14. The method of claim 13 wherein said first predetermined numeric pattern comprises eleven consecutive "0"'s and said second predetermined numeric pattern comprises ten consecutive "0"'s.

15. The method of claim 1 further comprising:

determining a threshold restriction for said step of monitoring said hash values; and

increasing a probability of said marking of said breakpoint in said digital sequence.

16. The method of claim 15 wherein said step of increasing said probability of said marking of said breakpoint in said digital sequence is a function of at least a desired chunk size.

17. The method of claim 15 wherein said step of increasing said probability of said marking of said breakpoint in said digital

sequence is a function of at least a length of a current portion of said digital sequence.

18. The method of claim 15 wherein said step of increasing said probability of said marking of said breakpoint in said digital sequence is carried out by the step of:

utilizing a second predetermined numeric pattern for said step of monitory said hash values; and

alternatively marking said breakpoint when said second predetermined numeric pattern occurs.

19. A method for determining a first breakpoint in a first digital sequence comprising:

determining a subset group of said first digital sequence;

performing a hash function on said subset group of said first digital sequence beginning at a starting position in said first digital sequence until a first predetermined numeric pattern in said hash value is obtained; and

marking said first breakpoint when said first predetermined numeric pattern in said hash value is obtained.

20. The method of claim 19 wherein said numeric pattern comprises a bit pattern.

21. The method of claim 19 wherein said step of performing a hash function is carried out by means of a rolling hash function.

22. The method of claim 19 further comprising the steps of:

further performing a hash function on a subset group of said first digital sequence from said first breakpoint until said first predetermined numeric pattern in said hash value is again obtained; and

marking another breakpoint in said first digital sequence when said first predetermined numeric pattern in said hash value is again obtained.

23. The method of claim 22 wherein said step of further performing said hash function is carried out by means of a rolling hash function.

24. The method of claim 19 further comprising the steps of:

determining a second predetermined numeric pattern in said hash value;

continuing said step of performing said hash function on said subset group of said first digital sequence until an established threshold restriction has been met; and

alternatively marking said first breakpoint when said second predetermined numeric pattern in said hash value is obtained.

25. The method of claim 19 further comprising the steps of:

performing a hash function on said subset group beginning at a starting position in a second digital sequence until said first predetermined numeric pattern in said hash value is obtained;

marking a second breakpoint in said second digital sequence when said first predetermined numeric pattern in said hash value is obtained;

comparing said predetermined hash value at said first breakpoint with that at said second breakpoint; and

equating a corresponding portion of said first digital sequence from said starting point to said first breakpoint with a corresponding portion of said second digital sequence from said starting position to said second breakpoint.

26. A computer program product comprising:

a computer usable medium having computer readable code embodied therein for partitioning a digital sequence comprising:

computer readable program code devices configured to cause a computer to effect performing a hash function on at least a portion of said digital sequence;

computer readable program code devices configured to cause a computer to effect monitoring hash values produced by said hash function for a first predetermined numeric pattern; and

computer readable program code devices configured to cause a computer to effect marking a breakpoint in said digital sequence when said first predetermined numeric pattern occurs.

27. The computer program product of claim 26 wherein said computer readable program code devices configured to cause a computer to effect performing said hash function is carried out by means of a rolling hash function.

28. The computer program product of claim 26 wherein said first predetermined numeric pattern is a bit pattern.

29. The computer program product of claim 26 wherein said computer readable program code devices configured to cause a computer to effect performing said hash function is carried out by means of an architecturally efficient n-bit hash function.

30. The computer program product of claim 29 wherein said architecturally efficient n-bit hash function comprises a 32-bit hash function.

31. The computer program product of claim 26 wherein said first predetermined numeric pattern comprises a consecutive sequence of bits.

32. The computer program product of claim 31 wherein said consecutive sequence of bits comprises a plurality of endmost bits.

33. The computer program product of claim 32 wherein said plurality of endmost bits comprises eleven bits.

34. The computer program product of claim 32 wherein said eleven bits are "0".

35. The computer program product of claim 26 further comprising:

computer readable program code devices configured to cause a computer to effect determining a threshold restriction for said step of monitoring said hash values;

computer readable program code devices configured to cause a computer to effect establishing at least a second predetermined numeric pattern for said step of monitoring when said threshold restriction is met; and

computer readable program code devices configured to cause a computer to effect alternatively marking said breakpoint in said digital sequence when said second predetermined numeric pattern occurs.

36. The computer program product of claim 35 wherein said second predetermined numeric pattern is a bit pattern.

37. The computer program product of claim 36 wherein said second predetermined numeric pattern is a subset of said first predetermined numeric pattern.

38. The computer program product of claim 37 wherein said first predetermined numeric pattern comprises a consecutive sequence of eleven bits and said second predetermined numeric pattern comprises ten of said eleven bits.

39. The computer program product of claim 38 wherein said first predetermined numeric pattern comprises eleven consecutive "0"'s and said second predetermined numeric pattern comprises ten "0"'s.

40. The computer program product of claim 26 further comprising:

computer readable program code devices configured to cause a computer to effect determining a threshold restriction for said step of monitoring said hash values; and

computer readable program code devices configured to cause a computer to effect increasing a probability of said marking of said breakpoint in said digital sequence.

41. The computer program product of claim 40 wherein said computer readable program code devices configured to cause a computer to effect increasing said probability of said marking of said breakpoint in said digital sequence is a function of at least a desired chunk size.

42. The computer program product of claim 40 wherein said computer readable program code devices configured to cause a computer to effect increasing said probability of said marking of said breakpoint in said digital sequence is a function of at least a length of a current portion of said digital sequence.

43. The computer program product of claim 40 wherein said computer readable program code devices configured to cause a computer to effect increasing said probability of said marking of said breakpoint in said digital sequence is carried out by:

computer readable program code devices configured to cause a computer to effect utilizing a second predetermined numeric pattern for said step of monitory said hash values; and

computer readable program code devices configured to cause a computer to effect alternatively marking said breakpoint when said second predetermined numeric pattern occurs.

44. A computer program product comprising:

a computer usable medium having computer readable code embodied therein for determining a first breakpoint in a first digital sequence comprising:

computer readable program code devices configured to cause a computer to effect determining a subset group of said first digital sequence;

computer readable program code devices configured to cause a computer to effect performing a hash function on said subset group of said first digital sequence beginning at a starting position in said first digital sequence until a first predetermined numeric pattern in said hash value is obtained; and

computer readable program code devices configured to cause a computer to effect marking said first breakpoint when said first predetermined numeric pattern in said hash value is obtained.

45. The computer program product of claim 44 wherein said numeric pattern comprises a bit pattern.

46. The computer program product of claim 44 wherein said computer readable program code devices configured to cause a computer to effect performing a hash function is carried out by means of a rolling hash function.

47. The computer program product of claim 44 further comprising:

computer readable program code devices configured to cause a computer to effect further performing a hash function on a subset group of said first digital sequence from said first breakpoint until said first predetermined numeric pattern in said hash value is again obtained; and

computer readable program code devices configured to cause a computer to effect marking another breakpoint in said first digital

sequence when said first predetermined numeric pattern in said hash value is again obtained.

48. The computer program product of claim 47 wherein said computer readable program code devices configured to cause a computer to effect further performing said hash function is carried out by means of a rolling hash function.

49. The computer program product of claim 44 further comprising:

computer readable program code devices configured to cause a computer to effect determining a second predetermined numeric pattern in said hash value;

computer readable program code devices configured to cause a computer to effect continuing said step of performing said hash function on said subset group of said first digital sequence until an established threshold restriction has been met; and

computer readable program code devices configured to cause a computer to effect alternatively marking said first breakpoint when said second predetermined numeric pattern in said hash value is obtained.

50. The computer program product of claim 44 further comprising:

computer readable program code devices configured to cause a computer to effect performing a hash function on said subset group beginning at a starting position in a second digital sequence until said first predetermined numeric pattern in said hash value is obtained;

computer readable program code devices configured to cause a computer to effect marking a second breakpoint in said second digital sequence when said first predetermined numeric pattern in said hash value is obtained;

computer readable program code devices configured to cause a computer to effect comparing said predetermined hash value at said first breakpoint with that at said second breakpoint; and

computer readable program code devices configured to cause a computer to effect equating a corresponding portion of said first digital sequence from said starting point to said first breakpoint with a corresponding portion of said second digital sequence from said starting position to said second breakpoint.

51. The method of claim 15 wherein said step of increasing said probability of said marking of said breakpoint in said digital sequence is a function of some content portion of said sequence.

52. The method of claim 1 wherein said hash function is selected to produce preferential hashsums that do not uniformly cover the range of possible output values.

53. The method of claim 1 wherein said hash function is selected or modified dynamically during the execution of said hash function.

54. The method of claim 1 wherein said hash function includes the relative or absolute value of the current location as an input value in said hash function.

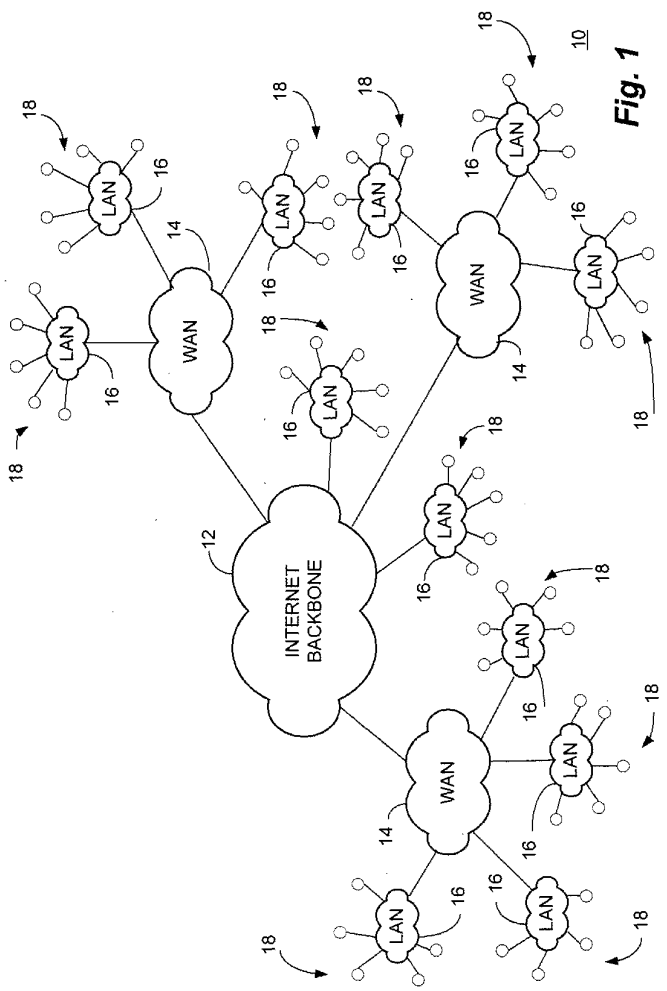


Fig. 1

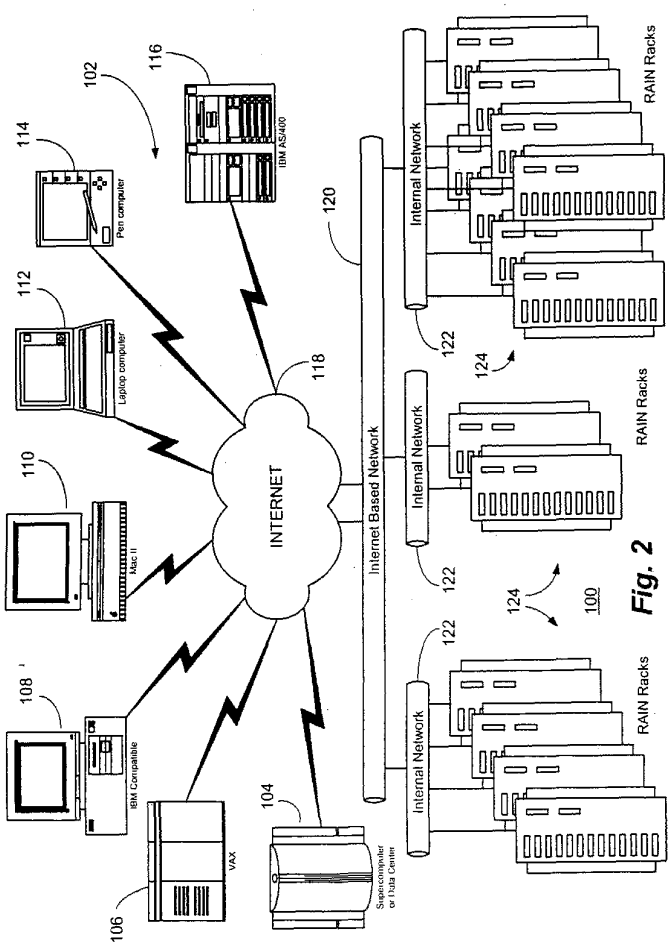

102
116
IBM AS/400
114
Pen computer
112
Laptop computer
110
Mac II
108
IBM Compatible
106
VAX
104
Supercomputer or Data Center
118
INTERNET
120
Internet Based Network
Internal Network
Internal Network
Internal Network
122
122
122
124
124
RAIN Racks
RAIN Racks
RAIN Racks
100

Fig. 2

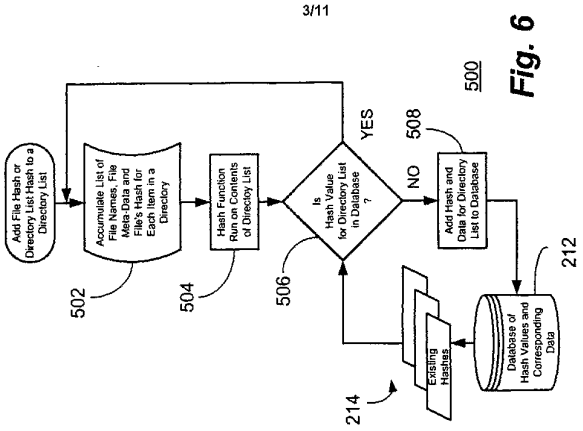

Add File Hash or Directory List Hash to a Directory List
Accumulate List of File Names, File Attributes, and Hashes or Files Hash for Each Item in a Directory
Hash Function Run on Contents of Directory List
Is Hash Value for Directory List in Database ?
YES
NO
Add Hash and Data for Directory List to Database
Database of Hash Values and Corresponding Data
Existing Hashes
500
502
504
506
508
212
214

Fig. 6

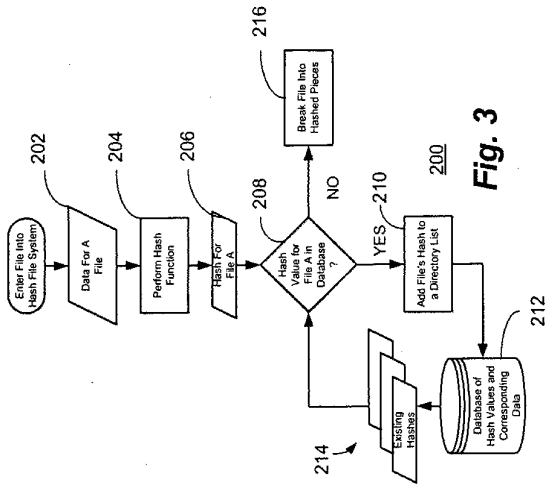

Enter File into Hash File System
Data For A File
Perform Hash Function
Hash for File A
Hash Value for File A in Database ?
NO
YES
Break File Into Hashed Pieces
Add File's Hash to a Directory List
Database of Hash Values and Corresponding Data
Existing Hashes
200
202
204
206
208
210
212
214
216

Fig. 3

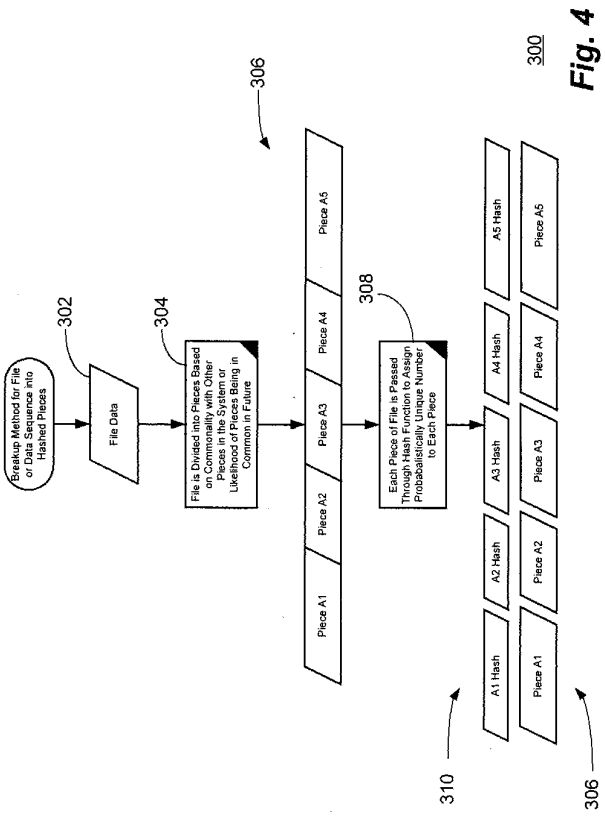

Breakup Method for File or Data Sequence into Hashed Pieces
File Data
File is Divided into Pieces Based on Commonality with Other Pieces in the System or Likelihood of Pieces Being in Common in Future
Each Piece of File is Passed Through Hash Function to Assign Probabalistically Unique Number to Each Piece
Piece A1
Piece A2
Piece A3
Piece A4
Piece A5
A1 Hash
A2 Hash
A3 Hash
A4 Hash
A5 Hash
302
304
306
308
310
300

Fig. 4

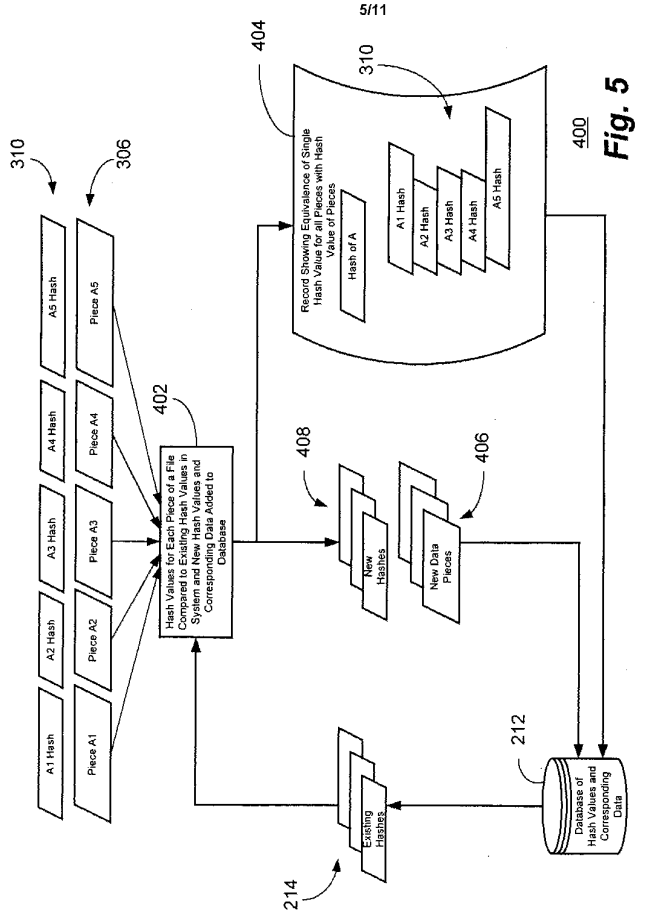

310
306
A5 Hash
A4 Hash
A3 Hash
A2 Hash
A1 Hash
Piece A5
Piece A4
Piece A3
Piece A2
Piece A1
402
Hash Values for Each Piece of a File Compared to Existing Hash Values in System and New Hash Values and Corresponding Data Added to Database
404
Record Showing Equivalence of Single Hash Value for all Pieces with Hash Value of Pieces
Hash of A
310
A1 Hash
A2 Hash
A3 Hash
A4 Hash
A5 Hash
400
408
New Hashes
406
New Data Pieces
212
Database of Hash Values and Corresponding Data
Existing Hashes
214

Fig. 5

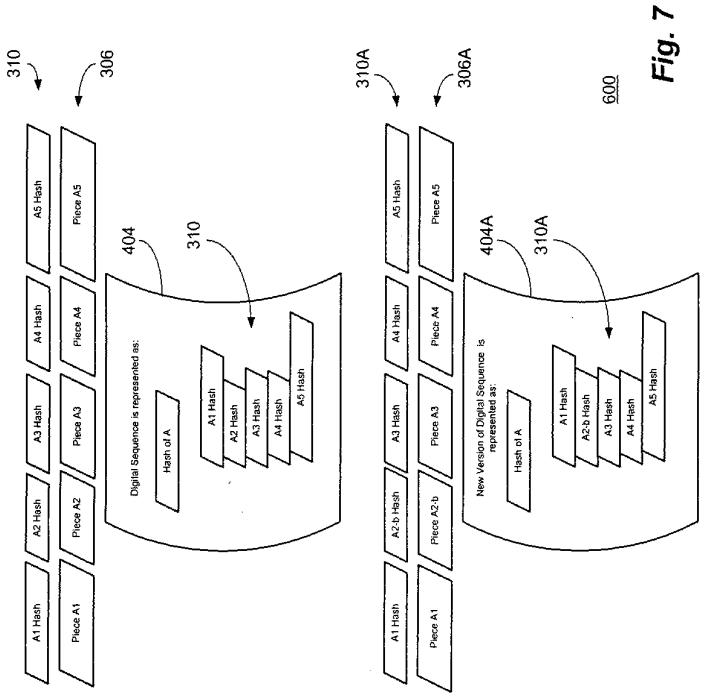

310
306
A1 Hash
A2 Hash
A3 Hash
A4 Hash
A5 Hash
Piece A1
Piece A2
Piece A3
Piece A4
Piece A5
404
310
Digital Sequence is represented as:
Hash of A
A1 Hash
A2 Hash
A3 Hash
A4 Hash
A5 Hash
310A
306A
A1 Hash
A2-b Hash
A3 Hash
A4 Hash
A5 Hash
Piece A1
Piece A2-b
Piece A3
Piece A4
Piece A5
404A
310A
New Version of Digital Sequence is represented as:
Hash of A
A1 Hash
A2-b Hash
A3 Hash
A4 Hash
A5 Hash
600

Fig. 7

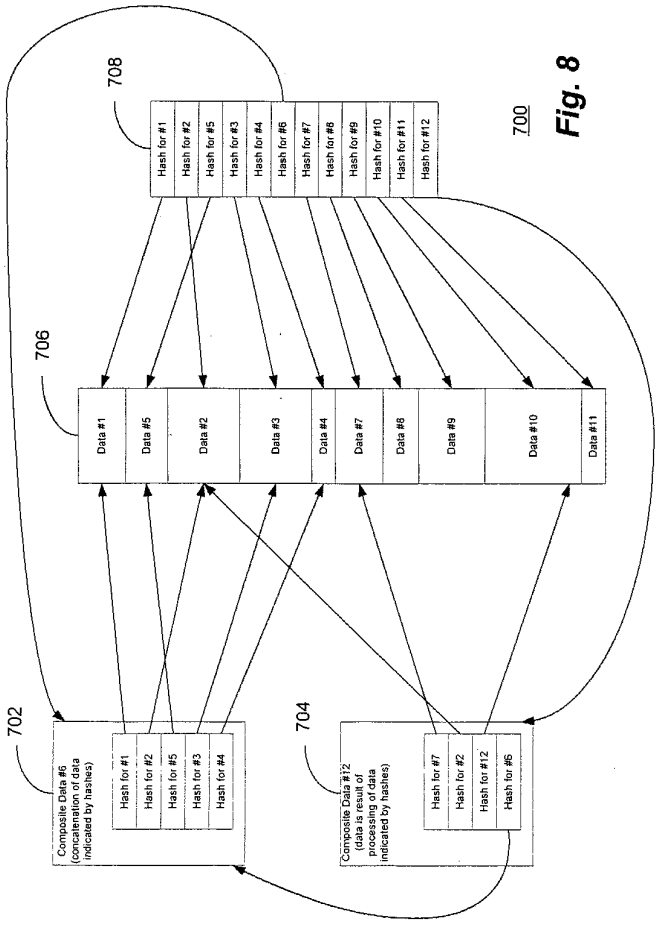

700
708
Hash for #1
Hash for #2
Hash for #5
Hash for #3
Hash for #4
Hash for #6
Hash for #7
Hash for #8
Hash for #9
Hash for #10
Hash for #11
Hash for #12
706
Data #1
Data #5
Data #2
Data #3
Data #4
Data #7
Data #8
Data #9
Data #10
Data #11
702
Composite Data #6
(concatenation of data
indicated by hashes)
Hash for #1
Hash for #2
Hash for #5
Hash for #3
Hash for #4
704
Composite Data #12
(data is result of
processing of data
indicated by hashes)
Hash for #7
Hash for #2
Hash for #12
Hash for #6

Fig. 8

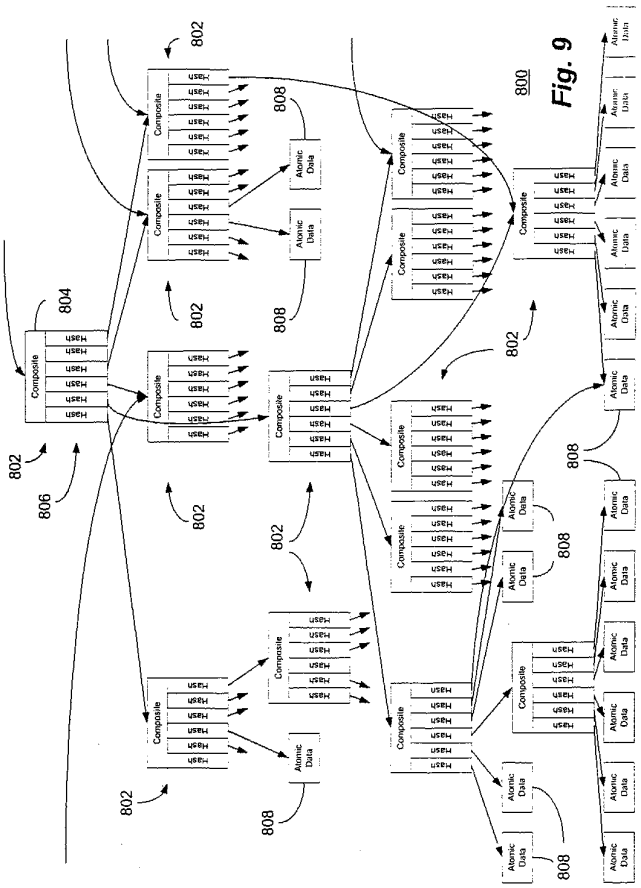


Fig. 9

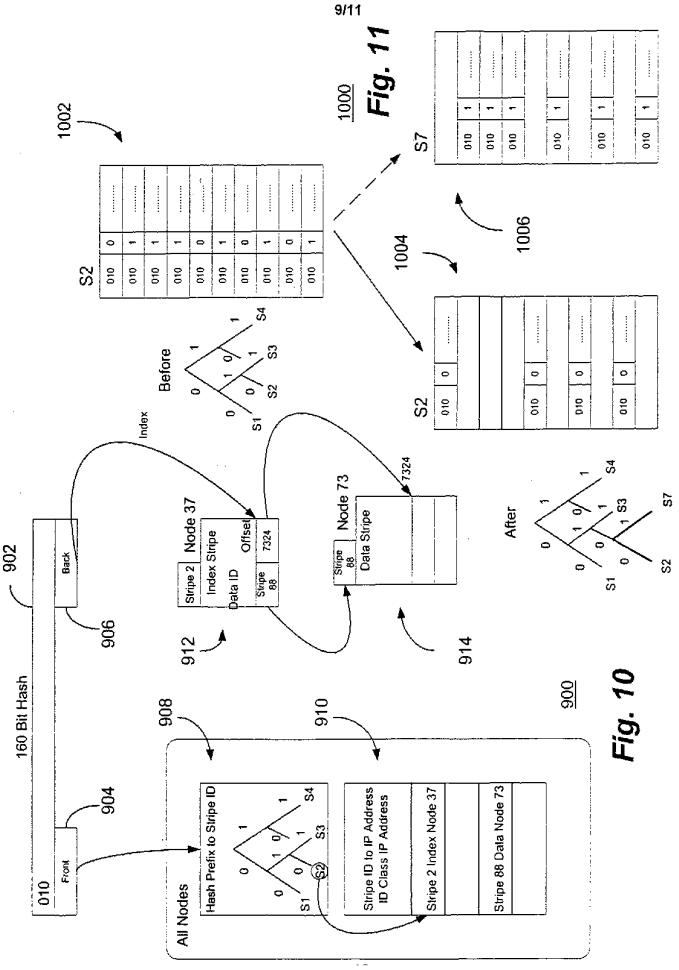
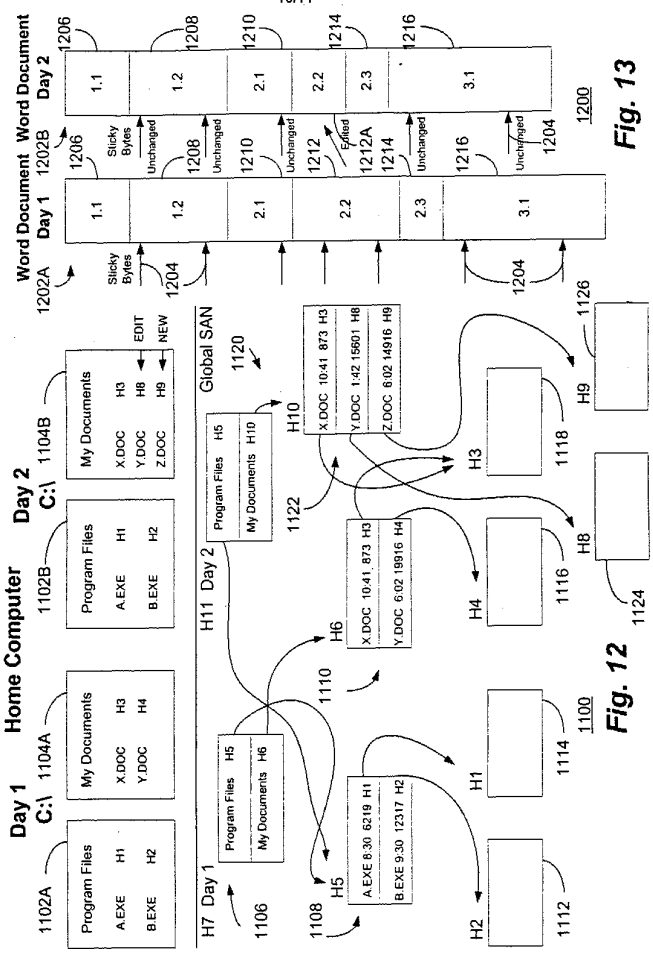

160 Bit Hash
902
Front
Back
010
904
906
Index
Node 37
Stripe 2
Index Stripe
Data ID
Offset
7324
Stripe 88
912
Node 73
Stripe 88
Data Stripe
7324
914
908
910
900
All Nodes
Hash Prefix to Stripe ID
Stripe ID to IP Address
ID Class IP Address
Stripe 2 Index Node 37
Stripe 88 Data Node 73
Before
After
S1
S2
S3
S4
S7

Fig. 10

1000
1002
1004
1006
S2
S7

Fig. 11


Word Document Day 1
Word Document Day 2
Sticky Bytes
Unchanged
Edited
1.1
1.2
2.1
2.2
2.3
3.1
1200
Fig. 13
Day 1 Home Computer
Day 2
C:\
Program Files
A.EXE H1
B.EXE H2
My Documents
X.DOC H3
Y.DOC H4
Z.DOC H9
EDIT
NEW
Global SAN
H7 Day 1
H11 Day 2
X.DOC 10:41 873 H3
Y.DOC 1:42 15601 H8
Z.DOC 6:02 14916 H9
A.EXE 8:30 6219 H1
B.EXE 9:30 12317 H2
X.DOC 10:41 873 H3
Y.DOC 6:02 19916 H4
1100
Fig. 12

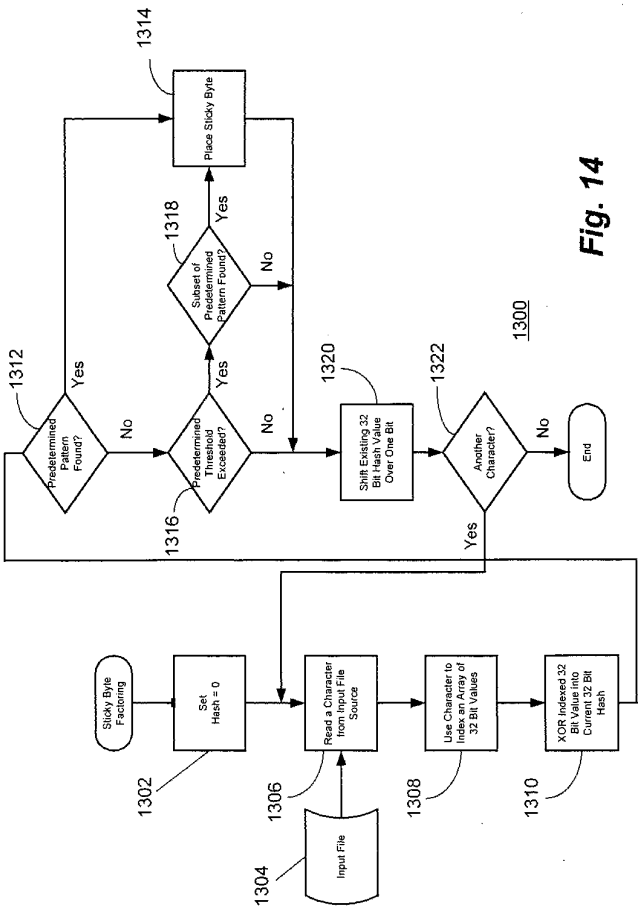

1314
Place Sticky Byte
1318
Subset of Predetermined Pattern Found?
Yes
No
1312
Predetermined Pattern Found?
Yes
No
1316
Predetermined Threshold Exceeded?
Yes
No
1320
Shift Existing 32 Bit Hash Value Over One Bit
1322
Another Character?
No
End
1300
Yes
Sticky Byte Factoring
Set Hash = 0
Read a Character from Input File Source
Use Character to Index an Array of 32 Bit Values
XOR Indexed 32 Bit Value into Current 32 Bit Hash
1302
1306
1308
1310
1304
Input File

Fig. 14

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/31306

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : H03M 7/30; H04L 23/00; G06F 7/00, 7/06, 7/22
US CL : 341/51,67
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED
Minimum documentation searched (classification system followed by classification symbols)
U.S. : 341/51,67

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X --- Y	US 5,990,810 A (WILLIAMS) 23 November 1999 (23.11.1999), column 3, lines 7-34; column 10, line 60 through column 16, line 53.	1, 3-20, 22, 24-26, 28-45, 47, 49-51 --- 2, 21, 23, 27, 46, 48, 52-54
X --- Y	WO 96/25801 A1 (WILLIAMS) 22 August 1996 (22.08.1996), pages 5-24, 32-44 and 47-49.	1, 3-20, 22, 24-26, 28-45, 47 --- 2, 21, 23, 27, 46, 48, 52-54
Y	TRIDGELL, A. Efficient Algorithms for Sorting and Synchronization, PhD Thesis, The Australian National University, April 2000, chapters 3-5.	2, 21, 23, 27, 46, 48
A	US 5,608,801 A (AIELLO et al) 04 March 1997 (04.03.1997), entire document.	1-54
Y	AHO, A.V. et al Data Structures and Algorithms, Reading:Addison-Wesley, 1983, chapter 4.	52-54

☒ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
30 November 2001 (30.11.2001)	20 DEC 2001
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703)305-3230	Authorized officer John E. Breene Telephone No. 703-305-3900

Form PCT/ISA/210 (second sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/31306

C. (Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X,E	US 2001/0037323 A1 (MOULTON et al) 01 November 2001 (01.11.2001), entire document.	1-54

Form PCT/ISA/210 (second sheet) (July 1998)

フロントページの続き

(81)指定国 AP(GH,GM,KE,LS,MW,MZ,SD,SL,SZ,TZ,UG,ZW),EA(AM,AZ,BY,KG,KZ,MD,RU,TJ,TM),EP(AT,BE,CH,CY,DE,DK,ES,FI,FR,GB,GR,IE,IT,LU,MC,NL,PT,SE,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AT,AU,AZ,BA,BB,BG,BR,BY,BZ,CA,CH,CN,CO,CR,CU,CZ,DE,DK,DM,DZ,EC,EE,ES,FI,GB,GD,GE,GH,GM,HR,HU,ID,IL,IN,IS,JP,KE,KG,KP,KR,KZ,LC,LK,LR,LS,LT,LU,LV,MA,MD,MG,MK,MN,MW,MX,MZ,NO,NZ,PH,PL,PT,RO,RU,SD,SE,SG,SI,SK,SL,TJ,TM,TR,TT,TZ,UA,UG,US,UZ,VN,YU,ZA,ZW

（特許庁注：以下のものは登録商標）
マッキントッシュ
ＥＴＨＥＲＮＥＴ
ＵＮＩＸ
Ｌｉｎｕｘ
ＯＳ／２