(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4364790号
(P4364790)
(45)発行日　平成21年11月18日(2009.11.18)　(24)登録日　平成21年8月28日(2009.8.28)

(51)Int.Cl.　F I
G06F 12/00 (2006.01)　G06F 12/00　510B

請求項の数 21　(全 40 頁)

(21)出願番号	特願2004-504132 (P2004-504132)	(73)特許権者	504421497
(86)(22)出願日	平成15年5月12日 (2003.5.12)		イノパス・ソフトウェアー・インコーポレーテッド
(65)公表番号	特表2005-525641 (P2005-525641A)		アメリカ合衆国カリフォルニア州９４０８６－１１０５，サニーベール，イースト・カリビアン・ドライブ　４００
(43)公表日	平成17年8月25日 (2005.8.25)	(74)代理人	100089705
(86)国際出願番号	PCT/US2003/014903		弁理士　社本　一夫
(87)国際公開番号	WO2003/096211	(74)代理人	100076691
(87)国際公開日	平成15年11月20日 (2003.11.20)		弁理士　増井　忠弐
審査請求日	平成18年4月25日 (2006.4.25)	(74)代理人	100075270
(31)優先権主張番号	10/146,545		弁理士　小林　泰
(32)優先日	平成14年5月13日 (2002.5.13)	(74)代理人	100080137
(33)優先権主張国	米国(US)		弁理士　千葉　昭男

最終頁に続く

(54)【発明の名称】バイト・レベルのファイル相違検出および更新アルゴリズム

(57)【特許請求の範囲】

【請求項１】

バイト・レベルのファイル相違検出を実行する、コンピュータにより実行される方法であって、

前記コンピュータが、２つの受けたバイト・ストリームまたはそのサブストリーム間における最も長い共通サブストリング（ＬＣＳ）を識別するステップであって、前記バイト・ストリームは、原ファイルに対応する原バイト・ストリームと、前記原ファイルの新ファイル・バージョンに対応する新バイト・ストリームとを含む、ステップと、

前記コンピュータが、前記２つの受けたバイト・ストリームまたはそのサブストリームの各々をサブストリームに分割するステップであって、第１サブストリームが前記それぞれのＬＣＳの接頭部であり、第２サブストリームが前記それぞれのＬＣＳの接尾部である、ステップと、

前記コンピュータが、前記サブストリームのサイズが予め指定したサイズ未満となるまで、またはＬＣＳが存在しなくなるまで、前記のＬＣＳを識別するステップ及び前記のサブストリームに分割するステップを、前記サブストリームに対し繰り返すステップと、

前記コンピュータが、前記原バイト・ストリームにおけるサブストリームと前記新バイト・ストリームにおける対応するサブストリームとの間におけるバイト・レベルの相違を判定するステップであって、前記原バイト・ストリームのサブストリームと前記新バイト・ストリームの対応するサブストリームとの間の編集距離を計算することを含み、前記編集距離は、前記原バイト・ストリームのサブストリームから前記新バイト・ストリームの

対応するサブストリームに変換する最小数の編集動作により表され、前記予め指定したサイズは、前記編集距離の計算に関係する、ステップと、
前記コンピュータが、計算した前記編集距離を表す前記編集動作の情報を含む動作アレイを生成するステップと、
前記コンピュータが、前記動作アレイに含まれた前記編集動作の情報を含むデルタ・ファイルを生成するステップと、
を備えた方法。
【請求項２】
請求項１記載の方法において、前記編集動作は、置換動作、挿入動作、削除動作、一致動作を含み、
前記デルタ・ファイルを生成するステップは、前記動作アレイに含まれた前記編集動作のうち、一致動作以外の動作を前記デルタ・ファイルに書き込むことを含み、
前記デルタ・ファイルは、連接した一連のレコードを含み、各レコードは、前記動作アレイに含まれた前記置換動作、挿入動作、削除動作のうちの１つに関連した動作コードと、該関連した１つの動作に関係する情報を格納した、
方法。
【請求項３】
請求項２記載の方法において、前記動作アレイを生成するステップは、
生成された前記動作アレイに含まれる前記編集距離を表す前記の置換動作、挿入動作、削除動作、一致動作に対し合併またはインデックス化を行うステップ、
を含む方法。
【請求項４】
請求項１記載の方法において、前記デルタ・ファイルのレコードは、前記情報として、バイト・ストリームの変更位置を識別する相対的開始アドレスを含むこと、を特徴とする方法。
【請求項５】
請求項２記載の方法において、前記デルタ・ファイルを生成するステップは、前記デルタ・ファイルへの書き込みにおいて可変長整数フォーマットを使用することを含み、該可変長整数フォーマットは、１バイトの内の１ビットを用いて、後続のバイトを整数の一部として読み取るか否かについて示す、方法。
【請求項６】
請求項１記載の方法において、前記デルタ・ファイルのレコードは、更に、前記情報として、前記動作コードを適用する前記新ファイルの相対的開始アドレス、および前記原ファイルに対する挿入または置換のデータを含む、を特徴とする方法。
【請求項７】
請求項１記載の方法であって、更に、
前記コンピュータが、前記デルタ・ファイルを圧縮するステップと、
前記コンピュータが、誤り訂正コード化を前記デルタ・ファイルに適用するステップと、
を含む、方法。
【請求項８】
請求項１記載の方法であって、更に、
前記コンピュータが、前記２つのバイト・ストリームまたはサブストリームがＬＣＳの識別に関連する最大指定サイズよりも長いとき、前記２つのバイト・ストリームまたはサブストリームの各々を半分ずつに分割するステップ、
を含む方法。
【請求項９】
請求項１記載の方法であって、前記動作アレイを生成するステップは、更に、
前記コンピュータが、２つのバイト・ストリーム間または２つのサブストリームの間にＬＣＳがない場合、前記新ファイルに対応する前記バイト・ストリームまたはサブストリ

ームの長さが前記原ファイルに対応する前記バイト・ストリームまたはサブストリームの長さ未満であるとき、置換動作および削除動作を前記動作アレイに書き込むステップ、
を含む方法。
【請求項１０】
請求項１記載の方法であって、前記動作アレイを生成するステップは、更に、
前記コンピュータが、２つのバイト・ストリーム間または２つのサブストリームの間にＬＣＳがない場合、前記新ファイルに対応する前記バイト・ストリームまたはサブストリームの長さが前記原ファイルに対応する前記バイト・ストリームまたはサブストリームの長さよりも長いとき、置換動作および挿入動作を前記動作アレイに書き込むステップ、
を含む方法。
【請求項１１】
請求項３記載の方法において、前記合併またはインデックス化を行うステップは、更に、
前記コンピュータが、前記動作アレイにおいて一致動作の短いシーケンスを突き止めるステップであって、前記短いシーケンスが３つ以下の連続する一致動作を含む、ステップと、
前記コンピュータが、前記短いシーケンスの一致動作を置換動作と置換するステップと、
を含む方法。
【請求項１２】
請求項３記載のステップにおいて、前記合併またはインデックス化を行うステップは、更に、
前記コンピュータが、前記動作アレイにおいて一致動作以外の動作のシーケンスを突き止めるステップと、
前記コンピュータが、前記突き止めたシーケンスを、前記動作の種類および前記シーケンスの長さを指定する単一の動作と置換するステップ、
を含む方法。
【請求項１３】
請求項３記載の方法であって、前記合併またはインデックス化を行うステップは、更に、
前記コンピュータが、挿入、削除、および置換の動作の少なくとも１つの組み合わせを含むシーケンスを突き止めるステップと、
前記コンピュータが、前記シーケンスを、単一の可変長置換動作と置換するステップと、
を含む方法。
【請求項１４】
請求項３記載の方法において、前記合併またはインデックス化を行うステップは、更に、
前記コンピュータが、等しい数の挿入および削除の動作を含むシーケンスを突き止めるステップと、
前記コンピュータが、前記シーケンスを、単一の固定長置換動作で置換するステップと、
を含む方法。
【請求項１５】
請求項３記載の方法において、前記合併またはインデックス化を行うステップは、更に、
前記コンピュータが、あるパターンで繰り返す動作単位を突き止めるステップであって、前記単位は、置換動作の固定長シーケンスと、他の種類の動作の固定長シーケンスとを含む、ステップと、
前記コンピュータが、前記繰り返す単位を、前記置換動作シーケンスの長さ、前記単位

の長さ、および前記単位が前記パターンで繰り返す回数を指定する単一の動作と置換するステップと、
を含む方法。
【請求項１６】
請求項３記載の方法において、前記合併またはインデックス化を行うステップは、更に、
前記コンピュータが、前記動作アレイに適当な動作をハッシュ・テーブルに入力するステップと、
前記コンピュータが、オフセット・パラメータを含む単一のコピー動作を用いるステップであって、前記コピー動作が、前記ハッシュ・テーブルの情報を用いて、前記動作アレイ内に動作を挿入する、ステップと、
を含む方法。
【請求項１７】
請求項１記載の方法において、前記予め指定したサイズは、約４０９６バイトである、方法。
【請求項１８】
バイト・レベルのファイル相違検出および更新を実行する方法であって、
第１のコンピュータ・システムが請求項１から１７のいずれかに記載の方法を実行するステップと、
前記第１コンピュータ・システムが、前記デルタ・ファイルを第２のコンピュータ・システムに転送するステップと、
前記第２コンピュータ・システムが、前記原ファイルと前記デルタ・ファイルを用いて、前記新ファイルのコピーを生成するステップと、
前記第２コンピュータ・システムが、前記新ファイルの前記コピーを用いて、前記原ファイルを更新するステップと、
を備えた方法。
【請求項１９】
バイト・レベルのファイル相違検出を実行するように構成された、コンピュータに実現される装置であって、
２つのバイト・ストリームまたはそのサブストリームを受ける手段であって、前記２つのバイト・ストリームは原ファイルに対応する原バイト・ストリームと、新ファイルに対応する新バイト・ストリームとを含む、ステップと、
前記２つの受信したバイト・ストリームまたはそのサブストリーム間における最も長い共通サブストリング（ＬＣＳ）を判定する手段と、
前記２つの受信したバイト・ストリームまたはそのサブストリームの各々をサブストリームに分割する手段であって、第１サブストリームが前記それぞれのＬＣＳの接頭部であり、第２サブストリームが前記それぞれのＬＣＳの接尾部である、手段と、
前記サブストリームのサイズが予め指定したサイズ未満となるまで、前記ＬＣＳを判定する手段及び前記サブストリームに分割する手段に前記サブストリームを供給する手段と、
前記原バイト・ストリームにおけるサブストリームと前記新バイト・ストリームにおける対応するサブストリームとの間でバイト・レベルの相違を判定する手段であって、前記原バイト・ストリームのサブストリームと前記新バイト・ストリームの対応するサブストリームと間の編集距離を計算することを含み、前記編集距離は、前記原バイト・ストリームのサブストリームから前記新バイト・ストリームの対応するサブストリームに変換する最小数の編集動作により表され、前記予め指定したサイズは、前記編集距離の計算に関係する、手段と、
計算した前記編集距離を表す前記編集動作の情報を含む動作アレイを生成する手段と、
前記動作アレイに含まれる編集動作を合併することにより前記動作アレイを最適化する手段と、

可変長整数フォーマットを用いて、前記最適化した動作アレイに含まれた前記編集動作の情報を含むデルタ・ファイルを生成する手段と、
を備えた装置。
【請求項２０】
バイト・レベルのファイル相違検出および更新を実行するように構成された、第２のコンピュータ・システムで実現される装置であって、
第１のコンピュータ・システムに実現される請求項１９記載の装置からのデルタ・ファイルを受ける手段と、
前記原ファイルと前記デルタ・ファイルを用いて、前記新ファイルのコピーを生成する手段と、
前記コピーを用いて前記原ファイルを更新する手段と、
を備えた装置。
【請求項２１】
実行可能命令を含むコンピュータ読み取り可能記憶媒体であって、処理システムにおいて実行されると、バイト・レベルのファイル相違検出を実行する方法を行い、前記方法が、
受けた２つのバイト・ストリームまたはそのサブストリーム間における最も長い共通サブストリング（ＬＣＳ）を判定するステップであって、前記バイト・ストリームが、原ファイルに対応する原バイト・ストリームと、前記原ファイルの新ファイル・バージョンに対応する新バイト・ストリームとを含む、ステップと、
前記受けた２つのバイト・ストリームまたはそのサブストリームの各々をサブストリームに分割するステップであって、第１サブストリームが前記それぞれのＬＣＳの接頭部であり、第２サブストリームが前記それぞれのＬＣＳの接尾部である、ステップと、
前記サブストリームのサイズが予め指定したサイズ未満になるまで、前記ＬＣＳを判定するステップ及び前記のサブストリームに分割するステップを、前記サブストリームに対し繰り返すステップと、
前記原バイト・ストリームにおけるサブストリームと前記新バイト・ストリームにおける対応するサブストリームの各々の間でバイト・レベルの相違を判定するステップであって、前記原バイト・ストリームのサブストリームと前記新バイト・ストリームの対応するサブストリームとの間の編集距離を計算することを含み、前記編集距離は、前記原バイト・ストリームのサブストリームから前記新バイト・ストリームの対応するサブストリームに変換する最小数の編集動作により表され、前記予め指定したサイズは、前記編集距離の計算に関係する、ステップと、
前記計算した前記編集距離を表す前記編集動作の情報を含む動作アレイを生成するステップと、
可変長整数フォーマットを用いて、前記動作アレイに含まれた前記編集動作の情報を含むデルタ・ファイルを生成するステップと、
を含む、コンピュータ読み取り可能記憶媒体。
【発明の詳細な説明】
【技術分野】
【０００１】
ここに開示する実施形態は、相違ファイル(difference file)を用いたソフトウェア管理のような、ファイル更新に関する。
【背景技術】
【０００２】
プロセッサ即ち中央演算装置（ＣＰＵ）上で実行してある機能性を与えるソフトウェアは、時と共に移り変わっていくことが多い。この移り変わりは、ソフトウェア・ファイルにおけるバグ、即ち、エラーを訂正したり、発展する技術に合わせたり、新たな特徴を追加する必要性から生ずる場合もある。即ち、移動ワイヤレス・デバイス上にホストされている埋め込みソフトウェア・コンポーネントは、多数のソフトウェア・バグを含んでいる

ことが多く、その訂正が必要となるのである。

【０００３】

ソフトウェアには、人間が読み取り可能なＡＳＣＩＩ（American Standard Code for Information Interchange）プレーン・テキスト・ファイルまたは二進コードの形式の１つ以上のファイルが含まれる。ＵＮＩＸプラットフォームまたはパーソナル・コンピュータ（ＰＣ）は、多数のソフトウェア・コンポーネントを含み、そのソフトウェア・コンポーネントの各々は、対応するオペレーティング・システム（ＯＳ）が支援するファイル・システムによって独立して管理され、更新される。ＵＮＩＸプラットフォームまたはＰＣ上で処理されるソフトウェア・ファイルまたはソフトウェア・コンポーネントを更新するために用いられる情報は、インターネットを通じて転送することができ、あるいはフロッピ・ディスク、コンパクト・ディスク・リード・オンリ・メモリ（ＣＤ－ＲＯＭ）、またはコンパクト・フラッシュ・カードのような二次記憶媒体からロードすることができる。

【０００４】

対照的に、移動ワイヤレス・デバイスでは、リアルタイム・オペレーティング・システム（ＲＴＯＳ）が用いられるのが通例であり、その場合全てのソフトウェア・コンポーネントが単一の大きなファイルとして連携されている。更に、これらの移動ワイヤレス・デバイスでは、ファイル・システムの対応は得られないのが通例である。加えて、前述の単一の大きなファイルは、無線、赤外線、またはシリアル・リンクのような低速通信リンクを用いて、デバイスに予めロードしておく、即ち、埋め込んでおく必要がある。

【０００５】

低速通信リンクを通じて移動ワイヤレス・デバイスの大きなファイルを更新する場合の障害には、更新したファイルをデバイスに配信するのに伴う時間、帯域幅、およびコストが含まれる。このような大型ファイルの配布には、顧客の視点からは、望ましくない程に長い時間がかかる可能性があり、ファイル提供側の視点からは、大量のサーバ資源を消費する可能性がある。無線リンクのような信頼できない通信リンクを通じた大型ファイルの配信も、通信異常の率を高める虞れがあり、デバイス内部に大容量のワーキング・メモリ、例えば、ランダム・アクセス・メモリ（ＲＡＭ）が必要となる場合もある。

【０００６】

大型ファイルを移動デバイスに配信する問題に対する既存の解決手段の１つに、圧縮の使用が含まれる。多数の既存の圧縮アルゴリズムが一般に用いられている。これらのアルゴリズムは、通例では、元のサイズの約４０％までのファイル圧縮に対応するが、圧縮したファイルであっても、遅く、費用がかかる狭帯域通信リンクを通じてデバイスにダウンロードするには、未だ余りに大き過ぎることが多い。

【０００７】

ファイルを更新するための別の典型的な解決手段では、相違検出プログラム(difference program)を用いて、改訂したファイルが原ファイルとどのように異なるかに関する記述を生成する。ＵＮＩＸまたはＰＣプラットフォーム上で実行しこのような相違データを生成する相違検出プログラムは、一般に入手可能である。大型二進ファイルに対応する相違データ・ファイルのサイズは、改訂したファイル・サイズまたは原ファイル・サイズの１０％から２０％程に小さくすることができる。しかしながら、圧縮の場合と同様、これらの相違検出プログラムを用いて生成した相違ファイルは、関連する通信プロトコルによる転送には大き過ぎる場合が多い。加えて、当技術分野において公知の相違検出プログラムは、相違ファイルを発生するために長い計算時間を要し、更に原ファイルの更新を実行するには移動通信デバイス内に大量のワーキング・メモリを必要とするという点で効率的ではない。

【発明を実施するための最良の形態】

【０００８】

図面では、同一の参照番号は、同一または実質的に同様のエレメントまたは行為を識別する。いずれの識別のエレメントまたは行為の論述も容易に識別するために、参照番号における１つまたは複数の上位桁は、当該エレメントを最初に紹介した図の番号を示すこと

とする（例えば、エレメント１０６は、最初に図１に関して紹介され論じられている）。
【０００９】
以下で特に記載しない限り、図に示す種々のブロックおよび構造の構成および動作は、従来の設計のものである。その結果、このようなブロックについてはここで更に詳細に説明する必要はない。何故なら、当業者はそれらを理解しているからである。そのような更なる詳細は、簡略化のため、そして本発明の詳細な説明を曖昧にしないために省略する。図面に必要ないかなる修正も、ここに行う詳細な説明に基づけば、当業者には容易に行うことができる。
【００１０】
これより、バイト・レベルのファイル相違検出および更新（ＦＤＵ）アルゴリズムについて詳細に説明する。ここでは、このアルゴリズムのことをＦＤＵアルゴリズムと呼ぶ。図１は、一実施形態のＦＤＵアルゴリズムによって行われるファイルの相違検出および更新を示すブロック図である。ＦＤＵアルゴリズムは、相違検出コンポーネントと、更新コンポーネントとを含む。相違検出コンポーネントは、ここではファイル相違検出アルゴリズムと呼び、第１コンピュータ・システムにおいてある電子ファイルの元のバージョンおよび新バージョンから相違ファイルを生成する。更新コンポーネントは、ここでは、ファイル更新アルゴリズムと呼び、相違ファイルおよび原ファイルのホストされたコピーを用いて、第２コンピュータ・システム上で、新ファイルのコピーを生成する。以下の説明では、本発明の実施形態の完全な理解を得て、それを実施可能にする説明を行うために、多数の具体的な詳細を紹介する。しかしながら、当業者は、１つ以上のこれら具体的な詳細がなくても、または他のコンポーネント、システムなどを用いても、本発明は実施可能であることは認めよう。他の場合には、本発明の形態を曖昧にするのを避けるために、周知の構造や動作は示さず詳細な説明もしないこととする。
【００１１】
図１を参照すると、第１コンピュータ・システム１０２および第２コンピュータ・システム１０４は、通信経路１０６を通じて通信する。これらのコンピュータ・システム１０２および１０４は、一緒に動作する計算デバイスのあらゆる集合体を含む。これは、当技術分野では周知である。コンピュータ・システム１０２および１０４はまた、より大きなコンピュータ・システム内のコンポーネントも含む。通信経路１０６は、コンピュータ・システム１０２および１０４間でファイルを伝達または転送するためのあらゆる媒体を含む。したがって、この経路１０６は、ワイヤレス接続、ワイヤード接続、混合ワイヤレス／ワイヤード接続を含む。また、通信経路１０６は、ローカル・エリア・ネットワーク（ＬＡＮ）、都市区域ネットワーク（ＭＡＮ：metropolitan area network)、ワイド・エリア・ネットワーク（ＷＡＮ）、企業固有のネットワーク、オフィス間またはバックエンド・ネットワーク、およびインターネットを含むネットワークに対する結合部即ち接続部も含む。更に、通信経路１０６は、フロッピ・ディスク、ハード・ディスク・ドライブ、およびＣＤ－ＲＯＭディスクのような、着脱可能な固定媒体、ならびに電話線、バス、および電子メール・メッセージを含む。
【００１２】
第１通信システム１０２は、元の、即ち、旧バージョン１１０および新バージョン１１２の電子ファイルを受信する。新ファイル１１２は、一般には、原ファイル１１０の更新または改訂バージョンであるが、そのように限定される訳ではない。電子ファイル１１０および１１２に含まれるソフトウェア・ファイルには、ダイナミック・リンク・ライブラリ・ファイル、共有オブジェクト・ファイル、埋め込みソフトウェア・コンポーネント（ＥＢＳＣ）、ファームウェア・ファイル、実行可能ファイル、１６進データ・ファイルを含むデータ・ファイル、システム・コンフィギュレーション・ファイル、および個人使用データを含むファイルが含まれるが、これらに限定される訳ではない。いずれの形式のファイルでもバイト・ストリームと見なすことができるので、以後文脈によってはファイルのことをバイト・ストリームと記載することもできるものとする。
【００１３】

ファイル相違検出アルゴリズム１１４は、新ファイル１１２を受信し、それを原ファイル１１０と比較し、比較したファイル間におけるバイト・レベルの相違を計算する。これについては以下で説明する。また、ファイル相違検出アルゴリズム１１４は、原ファイル１１０および新ファイル１１２の前処理も行い、ファイルの相違を計算するのに先だって、ファイルのサイズを縮小することもできる。ファイル相違検出アルゴリズム１１４は、比較の間に、相違ファイル１１６を生成する。ここでは、これをデルタ・ファイルと呼ぶ。

【００１４】

デルタ・ファイル１１６の内容は、新ファイルおよび原ファイル間のバイト・レベルの相違を効率的に表す。デルタ・ファイル１１６は、関連するファイルの新バージョン即ち現バージョンとそのファイルの以前のバージョンとの間の相違を表す、置換および／または挿入動作の実際のデータと共に、メタデータを含む。これについては以下で説明する。ファイル相違検出アルゴリズム１１４は、最小数のバイトおよび以下で説明する既定のフォーマットまたはプロトコルを用いて、デルタ・ファイル１１６の中に、原ファイル１１０と新ファイル１１２との間のあらゆる相違を提示することによって、空間的に最適化したデルタ・ファイルを供給する。

【００１５】

デルタ・ファイル１１６は、通信経路１０６を通じて、別のコンピュータ・システムに転送または送信される。転送の前に、当技術分野では公知の圧縮技法を用いて、デルタ・ファイル１１６を圧縮することもできるが、そのように限定される訳ではない。受信側のコンピュータ・システム上にホストされている、ファイル更新アルゴリズム１１８は、デルタ・ファイル１１６を、ホストされている原ファイル１１０と共に用いて、新ファイル１１２のコピーを生成即ち作成する。この新ファイル１１２のコピーは、次に、改訂または更新の対象である、クライアント・デバイス上にホストされている原ファイル１１０を更新する際に用いられる。この更新プロセスの完了時には、第２コンピュータ・システム上に現在格納されている新ファイルは、第１コンピュータ・システムにおいて受信した新ファイルと同一となる。

【００１６】

原ファイルと新ファイルとの間の相違は、通例では、新ファイル自体よりも小さいので、新たなバイト・ストリーム全体の代わりに、相違分を送信し格納すれば、格納および送信において大幅な節約となる。これは、通例では遅くしかも費用がかかる可能性が高い接続部、例えば、ワイヤレスまたはセルラ接続部を通じて更新するプログラムをホストする移動電子デバイスにとっては、特に重要である。ここに記載するデルタ・ファイルは、新ファイルよりも９９％まで小さいフォーマットで、新ファイルと原ファイルとの間の相違を提示し、この縮小したファイル・サイズによって、典型的な相違検出技法に対して多数の改良が得られる。１つの改良は、デルタ・ファイルをクライアント・デバイスに送信する際に必要な帯域幅の縮小である。即ち、ファイルが小さい程、転送に必要な帯域幅が小さくて済むことを意味する。また、ファイルが小さい程、送信時間も短くて済み、したがって、ファイル転送が中断される確率が低下し、同時に受信ファイルにおける送信エラーも減少する。ファイルのセキュリティに関して、新ファイル全体の代わりにデルタ・ファイルを転送すれば、新ファイルの略奪が解消する。これらの改善は全て、顧客満足度を高めるものである。

【００１７】

図２は、一実施形態の下におけるデルタ・ファイルの生成のフロー図である。動作が開始するのは、第１処理システムにおいて新ファイルおよび原ファイルを受信した（２０２）ときである。新ファイルと原ファイルとの間で前処理動作を実行して（２０４）、２つのファイルの内容間における共通セグメントおよび単純なパターンを識別する。前処理アルゴリズムは、識別した共通セグメントおよびパターンを用いて、新ファイルおよび原ファイルのサイズを縮小する。このように、この前処理は、ファイルの共通セグメントについて差分計算を行う必要性をなくすことにより、差分計算の効率を高めるのである。

【００１８】
前処理に続いて、新ファイルおよび原ファイルとの間でバイト・レベルの相違を計算する（２０６）。計算した相違は、動作アレイを用いてコード化する（２０８）。合併動作を動作アレイに適応して、デルタ・ファイルの作成２１０におけるオーバーヘッドを削減する。デルタ・ファイルを生成するには、可変長整数フォーマットを用いて動作アレイを１つのファイルに書き込むことにより、ファイル間の相違を定義するために必要なバイト数を最少に減らす（２１２）。この可変長整数フォーマットについては、以下で更に説明する。

【００１９】
次に、可能であれば、デルタ・ファイルを最適化して、ファイル・サイズを更に縮小する（２１４）。この最適化は、圧縮およびチェックサム・ビットまたはバイトの加算を含み、デルタ・ファイルのクライアント・デバイスへの送信に続いて、エラーを検出できるようにすることを含む。最適化したデルタ・ファイルを、次に、出力として供給する（２１６）。一実施形態のデルタ・ファイルは、原ファイルのコピーをホストする他のコンピュータ・システムへの今後の転送または送信のために格納しておく。

【００２０】
前述のように、前処理動作は、新ファイルと原ファイルとの内容の間で実行し、２つのファイルの内容間にある共通のセグメントおよび単純なパターンを識別する。共通セグメントおよび単純パターンの知識を用いて、比較を行うバイト数を削減することによって、全体的な性能の向上(gain)を図る。例えば、新ファイルを用いてソフトウェアのバグを訂正したり、新たな機構を追加したり、または既存の機構を強化するときのように、新ファイルは原ファイルとはわずかしか相違しないことが多い。したがって、ファイルの部分集合に対する相違検出動作と共に、単純になった前処理動作を実行するために必要な蓄積時間は、新ファイルおよび原ファイルの内容全体に対して相違検出動作を実行するために必要な時間よりも遥かに短いので、全体的な性能向上が実現するのである。これは、特に、単一の挿入または削除のみを伴う場合に言えることである。したがって、前処理は、全体的な相違計算時間を短縮し、性能向上をもたらす。デルタ・ファイル生成のプロセスの各々については、以下で詳しく説明する。

【００２１】
図３は、一実施形態の下において、ファイル間の比較を行う前にファイルを前処理する方法のフロー図である。この前処理は、挿入内容を決定し、削除内容を決定し、挿入および削除内容の連接または特殊な場合を決定し、これらの決定に関するファイルに情報を書き込む動作を含む。

【００２２】
挿入内容の決定（３０６）に関して、そして新バイト・ストリームおよび原バイト・ストリームの受信（３０２）に続いて、一実施形態の前処理アルゴリズムは、原バイト・ストリームおよび新バイト・ストリームの最も長い共通接頭部および最も長い共通接尾部を識別する（３０４）。図４を参照すると、次に比較を行い、原バイト・ストリーム４０２が新バイト・ストリーム４０４の接頭部４０６か否か判定を行う。そうであれば、相違は一挿入４０８である。挿入メタデータおよび挿入内容またはデータをデルタ・ファイルに書き込む。図５を参照すると、再度比較を行い、原バイト・ストリーム５０２が新バイト・ストリーム５０４の接尾部であるか否か判定を行う。そうであれば、相違は１つの挿入５０８であり、挿入メタデータおよび内容をデータ・ファイルに書き込む。

【００２３】
削除内容を決定する（３０８）には、原バイト・ストリームおよび新バイト・ストリームの最も長い共通接頭部および最も長い共通接尾部を識別することから、動作を再度実行する。図６を参照すると、次に比較を行って、新バイト・ストリーム６０４が原バイト・ストリーム６９２の接頭部６０６か否か判定を行う。そうであれば、相違は１つの削除６０８となり、削除メタデータをファイルに書き込む。更に、図７を参照すると、比較を行い、新バイト・ストリーム７０４が原バイト・ストリーム７０２の接尾部７０６か否か判

定を行う。そうであれば、相違は１つの削除７０８となり、削除メタデータをファイルに書き込む。

【００２４】

連接、挿入および削除の特殊な場合に関して、一実施形態の前処理アルゴリズムは、開始すると、原バイト・ストリームおよび新バイト・ストリームで識別した最も長い共通接頭部および接尾部を用いる。図８を参照すると、原バイト・ストリーム８０２が最長共通接頭部８０６と最長共通接尾部８０８との連接である場合、新バイト・ストリーム８０４には、接頭部８０６と接尾部８０８との間に挿入８１０がある。この識別した挿入のメタデータおよび内容を双方ともファイルに書き込む。図９を参照すると、新バイト・ストリーム９０４が最長共通接頭部９０６と最長共通接尾部９０８との連接である場合、原バイト・ストリーム９０２には、接頭部９０６と接尾部９０８との間に削除９１０しかなく、削除メタデータをデータ・ファイルに書き込む。

【００２５】

挿入および削除内容を判定するための、原バイト・ストリームおよび新バイト・ストリームに対する上述の動作に応答して、前処理アルゴリズムは、相違に関する追加情報を戻す（３１０）。この情報には、２つのバイト・ストリームにおいてどこから相違が開始するかについての情報、最長共通接頭部および最長共通接尾部以外の原バイト・ストリームの２つのサブストリームのサイズが含まれるが、これらに限定されるのではない。

【００２６】

一実施形態の前処理の一例として、原バイト・ストリーム

ａｂｃａｔｓｒｓｔ

および新バイト・ストリーム

ａｂｄａｉｌｙｒｓｔ

について検討する。

【００２７】

最長の共通接頭部「ａｂ」および最長共通接尾部「ｒｓｔ」を削除することによって、ファイル相違検出アルゴリズムによる処理のために残っている２つのサブストリームは、それぞれ、「ｃａｔｓ」および「ｄａｉｌｙ」に縮小する。

【００２８】

前処理動作に続いて、相違検出アルゴリズムは、比較したバイト・ストリーム間の編集距離(edit distance)を計算する。２つのバイト・ストリーム間の編集距離とは、D. Gusfieldが"Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology" （ストリング、ツリー、およびシーケンスに関するアルゴリズム：コンピュータ・サイエンスおよび計算生物学）(「 Gusfieldの引例」) cambridge (1997)において記載しているように、原バイト・ストリームを新バイト・ストリームに変換するために必要な最小数の編集動作のことである。一実施形態の編集動作は、挿入、削除、および代替即ち置換を含む。原バイト・ストリームを新バイト・ストリームに変換するために用いられる編集動作は、記号「Ｉ」（１バイト挿入）、「Ｄ」（１バイト削除）、「Ｒ」（１バイト置換）、および「Ｍ」（１バイト一致）を含む動作アレイを用いて表現する。これより、一例を説明する。

【００２９】

この例では、原バイト・ストリームは「vintner」であり、新バイト・ストリームは「writers」であり、編集距離の計算の間に生成した、関連動作アレイは「RIMDMDMMI」となる。この動作は、以下のように表される。

【００３０】

【表１】

R	I	M	D	M	D	M	M	I
v		i	n	t	n	e	r	
w	r	i		t		e	r	s

【００３１】

動作アレイを原バイト・ストリームに適用して新バイト・ストリームを生成する際、原バイト・ストリームの最初のバイト「ｖ」を新バイト・ストリームにおける新バイト「ｗ」と置換することから開始する。原バイト・ストリームに対して行う２番目の動作は、新バイト「ｒ」の挿入である。動作アレイの３番目のエレメントは一致（Ｍ）を示しているので、元のバイト「ｉ」を新バイト・ストリームにコピーする。４番目の動作は削除（Ｄ）であり、新バイト・ストリームにおけるバイト「ｎ」が削除されることになる。５番目の動作は、もう１つの一致（Ｍ）であるので、元のバイト「ｔ」を新バイト・ストリームにコピーする。６番目の動作はもう１つの削除（Ｄ）であり、新バイト・ストリームにおいてバイト「ｎ」が削除される。７番目の動作はもう１つの一致（Ｍ）であるので、元のバイト「ｅ」を新バイト・ストリームにコピーする。８番目の動作は、更に別の一致（Ｍ）であるので、元のバイト「ｒ」を新バイト・ストリームにコピーする。原バイト・ストリームに対して行われる９番目の動作は、新バイト「ｓ」の新バイト・ストリームへの挿入（Ｉ）である。

【００３２】

一実施形態の下において動作アレイを生成するファイル相違検出アルゴリズムは、当技術分野では周知の２つのアルゴリズムに基づいている。これらのアルゴリズムの一方は、D.S. Hirschbergによる"A linear space algorithm for computing maximal common subsequences,"（最大共通部分列を計算するための線形空間アルゴリズム） Comm. ACM 18,6 (1975) 341-343に記載されている、Hirschbergの線形空間最適整合アルゴリズム(linear-space optimal alignment algorithm)である。他方のアルゴリズムは、E. Ukkonenによる"On-line construction of suffix tree" （接尾ツリーのオンライン構築）（「Ukkonen引例」）algorithmica 14(3), 249-260 (1995)に記載されている、Ukkonenの線形時間接尾ツリー・アルゴリズムである。

【００３３】

従来の動的プログラミング計画では、Ｏ（ｎｍ）時間およびＯ（ｎｍ）空間が必要であり、ｎおよびｍは、それぞれ、原バイト・ストリームおよび新バイト・ストリームの長さである。したがって、長さが約３２ｋバイトである場合、動的プログラミング・テーブルをメモリ内に構築するためには、Ｏ（１Ｇ）バイトのメモリが必要となる。しかし、バイト・ストリーム相違計算問題における資源の制限には、時間および空間双方が含まれることは、極ありふれたことである。Daniel Hirchbergは、多くの動的プログラム問題に有効な実用的な空間縮小方法を開発した。Hirschbergのアルゴリズムは、必要な空間をＯ（ｎｍ）からＯ（ｍｉｎ（ｎ，ｍ））に縮小するが、最悪時間境界(worst case time bound)は２倍になるだけである。バイト・ストリームのサイズが約１Ｍバイトであるとすると、メモリにはＯ（１Ｍ）バイトの作業空間だけがあればよく、T. Cormen、C.Leiserson、およびR. Rivestが"Introduction to Algorithms"（アルゴリズムの手引き）、 The MIT Press (1990)において記載しているように、この分割統治方式の再帰レベルは、約$\log_2$（１Ｍ）＝２０である。これは、利用可能なメモリがＯ（１Ｇ）までであり、デフォルトのスタック・サイズがＯ（１Ｍ）である、現行の計算環境には適当である。

【００３４】

接尾ツリー、または位置ツリー(position tree)は、Gusfield引例に記載されているように、可能な全ての接尾部を明示的に列挙することによって、ストリングの内部構造を暴露するデータ構造である。一実施形態のファイル相違検出アルゴリズムとの関連で最も重要で興味深い構造は、ストリームにおける最長繰り返しサブストリングである。更に具体

的には、これは、２つの所与のバイト・ストリーム間における最も長い共通のサブストリング（ＬＣＳ）である。例えば、「ａｂｃ」は、「ａｘａｂａｂｃｄｉｊ」および「ａｂｉａｂｃ」間のＬＣＳである。線形時間アルゴリズムについて、Ukkonenの引例において、あらゆる所与のストリングについても接尾ツリーを構成することについて記載されている。しかしながら、Ukkonenのアルゴリズムを実施する場合、種々の考慮点が伴う。

【００３５】

最初の考慮点は、バイト・ストリームは大抵二進ファイルであるので、殆どの場合、０から２５５までの文字全てがバイト・ストリーム内に現れるという事実と関係する。このため、アルゴリズムで用いられている特殊なマーカに特別な注意を払う必要がある。特殊なマーカとは、バイト・ストリームには出てこない文字と定義する。これに対処する１つの方法では、この特殊マーカが出現すると想定される場所を用いて、アルゴリズムを修正する。

【００３６】

第２の考慮点に関して、Ukkonenのアルゴリズムを用いる目的は、２つの所与のバイト・ストリーム間でＬＣＳを見出すことである。したがって、一般化した接尾ツリーを構成する際、２つの所与のバイト・ストリームの連接のための接尾ツリーを、これらを終端するための特殊マーカを加えて、構築しなければならない。

【００３７】

Ukkonenのアルゴリズムの使用に伴う第３の考慮点に取り組む際、接尾ツリーのレベルは、所与のバイト・ストリームのサイズおよび構造に左右される。再帰的に接尾ツリーを横断してツリーを構築し、主要情報を収集し、あるいはツリーを開放すると、大きなスタック・サイズ（例えば、６４ＭＢ）を確保していても、プロセス・スタックを使い果たす可能性がある。したがって、再帰の代わりに"while loop"を用いて、再帰ツリー横断関数(recursive tree traversing function)をデコードする。

【００３８】

第４の考慮点は、Gusfield引例に記載されている、エッジ・ラベル圧縮方式(edge-label compression scheme)に関与する。この圧縮方式は、線形空間の複雑性(linear space complexity)を達成する際に使用が推奨されるが、それ以外では、更に大きなバイト・ストリームの処理のために、利用可能なシステム・メモリを使い果たす虞れがある。

【００３９】

前述のように、Hirschbergのアルゴリズムは、線形空間および二次時間(quadratic time)における原バイト・ストリームと新バイト・ストリームとの間で最適な編集距離を計算するために用いることができる。ストリーム・サイズが小さい場合、大抵の場合数キロバイト未満であるが、最適編集距離は一般に妥当な時間量で計算することができる。しかしながら、ストリーム・サイズが数千バイト以上となると、Hirschbergのアルゴリズムが最適編集距離を計算するのに要する時間は非常に長く（数時間、数日、そして場合によっては数カ月）、実用にならない。新バイト・ストリームは通例原バイト・ストリームの修正であるので、２つのバイト・ストリームには多くの共通部分がある。これらバイト・ストリームの共通部分を素早く識別すれば、ここに記載するアルゴリズムは、より小さなバイト・ストリームに関する相違の計算に集中することによって、効率的かつ迅速に動作することが可能となる。

【００４０】

図１０は、一実施形態の下におけるファイル相違検出アルゴリズムのフロー図１０００である。ファイル相違検出アルゴリズムは、２つの入力バイト・ストリーム間の編集距離を計算し、その結果を生成して動作アレイに保存する。一実施形態のファイル相違検出アルゴリズムは、原バイト・ストリームと新バイト・ストリームとの間でＬＣＳを識別する。何故なら、ＬＣＳは、原バイト・ストリームにおいて修正されていない最も大きな部分であるからである。これは、Ukkonenのアルゴリズムを用いれば、線形時間および線形空間で行うことができる。次いで、分割統治手法を適用する。一旦ＬＣＳがわかったなら、旧バイト・ストリームおよび新バイト・ストリームを４つの小さなバイト・ストリームに

分割する。この方式は、旧バイト・ストリームおよび新バイト・ストリームが、Hirschbergのアルゴリズムを呼び出せるサイズ未満になるまで、再帰的に適用し、呼び出せる時点になったら、Hirshbergのアルゴリズムを呼び出して、バイト・ストリーム間の編集距離を計算する。結果は、グローバル動作アレイに記録される。

【００４１】

図１０の記号の使用には、Hirschbergのアルゴリズムを呼び出すことができる、バイト・ストリームの最少バイト数、例えば、４０９６を表す参照文字「Ｌ」が含まれている。参照文字「M」は、相違計算を許容時間内、即ち、約２時間未満程度で終了することができる入力バイト・ストリームにおける最大バイト数を表す。許容時間は、予め指定する。入力バイト・ストリームがMバイトを超過した場合、ファイル相違検出アルゴリズムはそれを複数の部分に分解する。これらの部分は、一実施形態の下では半分ずつであるが、そのように限定される訳ではない。サイズMは数十メガバイトとしてもよい。

【００４２】

記号の使用についての説明を続ける。o_{1h}およびo_{2h}は、原バイト・ストリーム、即ち、バイト・ストリームｏの前半および後半をそれぞれ表し、一方n_{1h}およびn_{2h}は、新バイト・ストリーム、即ち、バイト・ストリームｎの前半および後半をそれぞれ表す。したがって、これらの表現は、次の式を満たす。

【００４３】

【数１】

$$o_{1h}+o_{2h}=o$$

$$n_{1h}+n_{2h}=n$$

【００４４】

文字o_pおよびo_sは、バイト・ストリームｏの接頭部および接尾部をそれぞれ表し、次の式を満たす。

【００４５】

【数２】

$$o_p+LCS+o_s=o$$

【００４６】

文字n_pおよびn_sは、バイト・ストリームｎの接頭部および接尾部をそれぞれ表し、次の式を満たす。

【００４７】

【数３】

$$n_p+LCS+n_s=o$$

【００４８】

加えて、「長さ（ｏ）」は、バイト・ストリームｏの長さを表す。

以下の疑似コードは、ファイル相違検出アルゴリズムによって用いられるものの一例の上位記述として提示する。この例では、「ｏ」は原バイト・ストリームを表し、「ｎ」は新バイト・ストリームを表し、「ｏｐ＿ａｒｒａｙ」は動作アレイを表す。

【００４９】

【表２】

```
Procedure GETDIFF(o, n, op_array)
    Begin
        If length(o) and length(n) are zero, return;
        If length(o) is zero, write I's into op_array length(n) times, return;
        If length(n) is zero, write D's into op_array length(o) times, return;
        If length(o) < L and length(n) < L, call Algorithm Hirschberg, return;
        If length(o) > M and length(n) > M,
            cut o into the first half o1h and the second half o2h;
            cut n into the first half n1h and the second half n2h;
            GETDIFF(o1h, n1h, op_array);
            GETDIFF(o2h, n2h, op_array);
            return;
        Call Algorithm Ukkonen to get the LCS;
        If no LCS exists and length(o) < length(n),
            write R's and I's into op_array, where the number of
            'R' is length(o), the number of 'I' length(n)-length(o), return;
        else if no LCS exists and length(o) >= length(n),
            write R's and D's into op_array, where the number of
            'R' is length(n), the number of 'D' length(o)-length(n), return;
        GETDIFF(op, np, op_array);
        Write M's into op_array length(LCS) times;
        GETDIFF(os, ns, op_array);
    End
```

【００５０】

図１０を参照すると、動作は、ブロック１００２において開始し、ファイル相違検出アルゴリズムが原バイト・ストリームおよび新バイト・ストリームを受信する。ブロック１００４において、ファイル相違検出アルゴリズムは、両入力バイト・ストリームの長さが０に等しいか否か判定を行う。長さが０に等しい場合、この手順は戻る。

【００５１】

両入力バイト・ストリームの少なくとも一方が空でない場合、ファイル相違検出アルゴリズムは、バイト・ストリームの長さを別個に評価し、ブロック１００６において原バイト・ストリームの長さを評価し、ブロック１００８において新バイト・ストリームの長さを評価する。原バイト・ストリームの長さが０に等しいとき、これは新バイト・ストリームが収容する情報が全て新規であることを示す。したがって、ブロック１１０６において、ファイル相違検出アルゴリズムは、挿入動作（「Ｉ」）を動作アレイlength(n) timesに書き込む。次いで、手順は戻る。

【００５２】

ブロック１００８において、動作は、旧バイト・ストリームは０でないが、新バイト・ストリームは空であるという状況を評価する。新バイト・ストリームの長さが０に等しいとき、これは、原バイト・ストリームのデータを全て削除することを示す。したがって、ブロック１１０８において、ファイル相違検出アルゴリズムは、削除動作（「Ｄ」）を動作アレイ length(o) times に書き込む。次いで、手順は戻る。

【００５３】

両入力バイト・ストリームの長さが０よりも大きい場合、動作はブロック１０１０に進み、ファイル相違検出アルゴリズムは、両バイト・ストリームの長さを評価する。各入力バイト・ストリームの長さが、Hirschbergアルゴリズムを呼び出すことができるバイト・ストリームの最大バイト数であるＬ未満の場合、ブロック１１１０においてHirschbergアルゴリズムをコールし、２つのバイト・ストリーム間の編集距離を計算する。次いで、手順は戻る。

【００５４】

入力バイト・ストリームの少なくとも一方の長さがＬよりも大きい場合、動作はブロッ

ク１０１２に進む。ここで、ファイル相違検出アルゴリズムは、各入力バイト・ストリームの長さは、相違計算が予め指定した時間即ち許容時間内に終了することができる入力バイト・ストリームの最大バイト数であるMよりも大きいか否か判定を行う。各入力バイト・ストリームの長さがMを超過している場合、ブロック１１１２において各バイト・ストリームを、ほぼ等しいサイズに半分ずつ分割する。動作は続いて、ブロック１１１３および１１１４において、バイト・ストリーム・セグメントの各集合o_{1h}／n_{1h}およびo_{2h}／n_{2h}を、それぞれ、入力バイト・ストリームとしてファイル相違検出アルゴリズムに入力する。

【００５５】

入力バイト・ストリームの長さが、Hirschbergアルゴリズムを呼び出すことができる最大長よりも大きく、相違計算を許容時間内に終了できる最大長よりも短い場合、ステップ１０１４において、ファイル相違検出アルゴリズムは線形時間ルーチンをコールし、バイト・ストリーム間でＬＣＳを識別する。一実施形態の線形時間ルーチンは、Ukkonenのアルゴリズムであるが、本実施形態はそのように限定される訳ではない。

【００５６】

ブロック１０１６において、２つのバイト・ストリーム間にＬＣＳが存在しない場合、ブロック１１１６において２つのバイト・ストリームの長さを比較する。新バイト・ストリームの長さが原バイト・ストリームのそれを上回る場合、ブロック１１１８において置換（「Ｒ」）および挿入（「Ｉ」）動作を動作アレイ内に適宜書き込み、手順は戻る。原バイト・ストリームの長さが新バイト・ストリームのそれを上回る場合、ブロック１１２０において置換（「Ｒ」）および削除（「Ｄ」）動作を動作アレイ内に適宜書き込み、手順は戻る。

【００５７】

ブロック１０１６において、２つのバイト・ストリーム間でＬＣＳを識別した場合、図１１を参照して、ブロック１０１８において、２つのバイト・サブストリームを識別して、更に評価を行う。一方のサブストリーム（o_p）は原バイト・ストリーム１１０２のＬＣＳに対する接頭バイトを含み、他方のサブストリーム（n_p）は新バイト・ストリーム１１０４のＬＣＳに対する接頭部を含む。動作はブロック１００２に戻り、これら２つのサブストリーム（o_pおよびn_p）を、評価のためのファイル相違検出アルゴリズムへの入力として供給する。

【００５８】

ブロック１０２０において、一致（「M」）動作を動作アレイ length(LCS) timesに書き込む。更に、ブロック１０２２において、２つの追加のバイト・サブストリームを識別して、更に評価を行う。一方のサブストリーム（o_s）は、原バイト・ストリーム１１０２のＬＣＳに対する接尾バイトを含み、他方のサブストリーム（n_s）は、新バイト・ストリーム１１０４のＬＣＳに対する接尾バイトを含む。動作は、ブロック１００２に戻り、ここでこれら２つの追加サブストリーム（o_sおよびn_s）を入力としてファイル相違検出アルゴリズムに供給し、評価する。

【００５９】

ファイル相違検出アルゴリズムのフロー図１０００は、再帰的に動くように示されている。バイト・ストリームの近似的なサイズがＯ（１Ｍ）である場合、再帰レベルは$\log_2$（１Ｍ）＝２０となる。したがって、プロセス・スタックを外れて動作することなく、再帰的に書き込むのは安全である。

【００６０】

一実施形態のファイル相違検出アルゴリズムの空間、時間、および再帰レベル分析についてここで説明する。空間分析では、HirschbergおよびUkkonenのアルゴリズム双方が線形空間を用いることを注記しておく。一実施形態のファイル相違検出アルゴリズムは、再帰的にHirschbergおよびUkkonenのアルゴリズムをコールするので、全空間複雑度は入力バイト・ストリームのサイズに対して線形である。

【００６１】

時間の分析に関して、Ｌは、Hirschbergのアルゴリズムを呼び出すことができるバイト・ストリームの最大サイズであり、ｎは原入力バイト・ストリームであると仮定する。最悪の状況の１つでは、ファイル相違検出アルゴリズムがHirschbergアルゴリズムをＯ（ｎ／Ｌ）回コールし、Ukkonenのアルゴリズムを$O(\log_2 n)$回コールする。したがって、ファイル相違検出アルゴリズムのルーチンの実行のための総時間は、

【００６２】

【数４】

$$O(n/L) * O(L*L) + O(\log_2 n) * O(n) = O(n*L)$$

【００６３】

以下である。

再帰分析については、ファイル相違検出アルゴリズムのツリー横断関数を、「while ループ」を用いて実施するとよい。約１Ｍバイトを収容するバイト・ストリームでは、ファイル相違検出アルゴリズムの再帰レベルは約２０である。Ｌが、Hirschbergのアルゴリズムを呼び出すことができるバイト・ストリームの最大サイズとすると、Hirshbergアルゴリズムの再帰レベルは、$\log_2 L$となる。Ｌが１Ｍよりも小さい場合、総再帰レベルは約４０となる。したがって、殆どのＵＮＩＸおよびWindowsプラットフォーム上で用いられているＯ（１Ｍ）のデフォルト・スタック・サイズは、ファイル相違検出アルゴリズムの実行に十分対応して余りある。

【００６４】

前述の分析は、ファイル相違検出アルゴリズムが、デフォルト・スタック・サイズを用いて、線形空間において編集距離を計算することを示した。計算時間は、Ｏ（ｎ＊Ｌ）である。Ｌ＝ｎの場合、ファイル相違検出アルゴリズムはHirschbergアルゴリズムに縮退し、最適編集距離を計算する。Ｌが非常に小さい（例えば、１０２４）場合、ファイル相違検出アルゴリズムは線形時間で実行し、疑似最適編集距離を計算する。このように、ファイル相違検出アルゴリズムは、実際では、当技術分野において以前から知られている相違検出技法よりも優れている。

【００６５】

原バイト・ストリームを新バイト・ストリームに変換する際に用いる動作アレイを生成した後、プロセスは、動作アレイからデルタ・ファイルを生成し始める。デルタ・ファイルのサイズは、以下で説明するが、できるだけ小さい方がよい。一実施形態のデルタ・ファイルは、一連のレコードから成り、各レコードは、メタデータを、必要な場合には内容データと共に含む。したがって、メタデータは、ソートのオーバーヘッドであり、このオーバーヘッドは、相違検出プロセスの効率を高めるためには、できるだけ減少させなければならない。この減少を達成するためには、以下の七合併およびインデックス化（Ｍ＆Ｉ）技法(seven merging and indexing technique)を一実施形態において適用し、全メタデータ・サイズを縮小する。何故なら、動作アレイはバイト・レベル動作を指定するだけであるからである。

【００６６】

第１Ｍ＆Ｉ技法は、一致動作の探索において動作アレイを走査する。単一の一致動作が突き止められた場合、これらを代替動作によって置換する。何故なら、一致動作は代替動作の特殊な場合であるからである。この代替は、一致動作が連続して３つ以下の一致動作グループに対しても実行する。３つの動作のシーケンス長を用いる。何故なら、メタデータ・シーケンスの最少長さは３バイトであるからである。このように、１組のメタデータは、数回の動作の代わりにこの部分における動作を記録するには十分である。一例として、第１Ｍ＆Ｉ技法の適用前における動作アレイは次の通りである。

【００６７】

【表３】

ＩＩＩMMMDDMMMMMMMRMMD

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

【００６８】

このアレイは、３動作以下のシーケンス長を有する２つの一致動作シーケンスを含む。即ち、アレイの位置４から６における第１シーケンス、およびアレイの位置１６および１７における第２シーケンスである。第１技法の適用後、これらのシーケンスは、置換動作シーケンスによって置換され、動作アレイは次のようになる。

【００６９】

【表４】

ＩＩＩRRRDDMMMMMMMRRRD

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

【００７０】

第２Ｍ＆Ｉ技法は、動作アレイから同じ動作のシーケンスを探索する。この探索は、一致（「Ｍ」）動作以外の全ての動作を探す。これらのシーケンスが突き止められると、これらは、単一の大きな動作に合併される。これは、この動作を実行すべきバイト数を示す長さと共に、実行する動作を指定する。例えば、この技法の適用前の動作例は、次の通りとする。

【００７１】

【表５】

MMMMDDDDDDDDDDDDMMMM

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

【００７２】

このアレイは、２つの異なる位置動作シーケンスの間に、１２個の削除動作のシーケンスを含む。この技法は、１２個の削除動作のシーケンスを、動作

【００７３】

【表６】

deletion_with_length

【００７４】

で置換する。

したがって、この技法の適用後、動作アレイは次のようになる。

【００７５】

【表７】

MMMMdeletion_with_lengthMMMM

1 2 3 4 17 18 19 20

【００７６】

ここで、「length」は、１２に等しいと指定する。

第３Ｍ＆Ｉ技法は、シーケンスの中に一致（「Ｍ」）動作がない場合、挿入（「Ｉ」）動作、削除（「Ｄ」）動作、および代替（「Ｒ」）動作のシーケンスを合併して１つの動作を形成する。この技法は、２つの変形を含む。第１の変形は、挿入動作の数が削除動作の数と等しいときに用いられ、その場合、合併した動作は、指定の長さを有する固定長置換動作となる。それ以外の場合、第２の変形、即ち、２つの指定長を有する可変長置換が適用され、その場合動作は、次のように指定される。

【００７７】

【表８】
variable_length_replace

【００７８】
第１指定長は、識別したシーケンス（置き換える長さ）を置換するシーケンスの長さを与え、第２指定長は、置き換えるシーケンスの長さ（置き換えられる長さ）を与える。例えば、この技法を適用する前の動作アレイは次の通りである。
【００７９】
【表９】
MMMMRI DDDDDDDDDDDDMMMMM

【００８０】
このアレイは、シーケンス「ＲＩＤＤＤＤＤＤＤＤＤＤＤＤ」を含有する。このシーケンスは、第３Ｍ＆Ｉ技法の適用基準を満たす。何故なら、これは置換、挿入、および削除動作のシーケンスであるからである。ここでは第２の変形を適用する。何故なら、挿入動作の数（１つ）が削除動作の数（１２）と等しくないからである。第２変形の適用後では、動作アレイは次のようになる。
【００８１】
【表１０】
MMMMvariable_length_replaceMMMMM

【００８２】
ここで、置き換える長さは２（１バイト置き換え動作および１バイト挿入動作の総数）であり、置き換えられる長さは１３（１バイトの置き換え動作および１バイトの削除動作の総数）である。
【００８３】
置換動作の固定長シーケンスが、動作、例えば、一致動作の固定長シーケンスに１回よりも多く起こるパターンを利用するために、第４Ｍ＆Ｉ技法を設ける。新たな動作自体の名称は、間隔の長さおよび繰り返し回数を示す２つのパラメータと関連する、置換の長さを示す。この技法は、１間隔における固定長置換が３０バイト以下であるときに最も適している。動作アレイは、例えば、この技法の適用前では、以下の通りである。
【００８４】
【表１１】
MMMMMRRMMMMRRMMMMRRMMMMII

【００８５】
このアレイは、３回繰り返す
【００８６】
【表１２】
RRMMMM

【００８７】
というシーケンスを含むことがわかる。このシーケンスは、置換動作の２バイト・シーケンスを一致動作と共に含む。
したがって、この技法の適用後では、動作アレイは次のようになる。
【００８８】
【表１３】
MMMMM2_byte_fixed_length_repeat_replaceMII

【００８９】

ここで、
【００９０】
【表１４】

2_byte_fixed_length_repeat_replace

【００９１】
は、
【００９２】
【表１５】

RRMMMMRRMMMMRRMMMM

【００９３】
と交代し、「長さ」を６（各置換／一致シーケンスの全長）と指定し、一方「繰り返し」は３（各置換／一致シーケンスは３回繰り返される）となる。
多くの場合、置換動作の固定長シーケンスは、一致動作の固定長シーケンスの間で発生する。第５技法は、このパターンを利用する新たな動作を導入する。新動作の名称は、置換長と、間隔長および繰り返し回数を指定する２つのパラメータとを共に与える。一例についてこれより説明する。ここで、動作アレイを次の通りとする。
【００９４】
【表１６】

MMMMRRMMMMRRMMMMRRMMMMRRII

【００９５】
このアレイは、
【００９６】
【表１７】

RR

【００９７】
というシーケンスを含み、一致動作の固定長シーケンスの間に４回繰り返される。この技法を適用すると次の動作アレイが得られる。
【００９８】
【表１８】

MMMM2_byte_fixed_length_int_repeat_replaceII

【００９９】
ここで、新たな動作
【０１００】
【表１９】

2_byte_fixed_length_int_repeat_replace

【０１０１】
は、次のシーケンスと置き換わる。
【０１０２】
【表２０】

RRMMMMRRMMMMRRMMMMRR

【０１０３】
この例における間隔「長」は、６（シーケンス「ＲＲＭＭＭＭ」の長さ）に等しく、一方間隔「繰り返し」は、４（シーケンス「ＲＲ」が４回繰り返す）と指定される。

第６Ｍ＆Ｉ技法は、処理した全ての動作をデータ構造、例えば、ハッシュ・テーブルに入力することを含む。動作アレイにおいて以前に発生した動作に、動作アレイにおいて遭遇した場合、新たな動作を導入し、以前に遭遇した動作を新たな場所に挿入するように動作する。

【０１０４】

新たな動作は、

【０１０５】

【表２１】

exact_copy_index_to

【０１０６】

となり、コピーの位置を示すオフセット・パラメータを伴う。

バイト・ストリームが実行可能な場合、１つの命令、挿入または削除によって、後続の命令のアドレスが一定値だけずれることになる。第７Ｍ＆Ｉ技法は、このパターンを利用する新たな動作を導入する。新たな動作は、次の通りであり、

【０１０７】

【表２２】

address_repeat_shift

【０１０８】

この動作の繰り返し回数および間隔長は、メタデータとして動作と関連付けられ、一方一定値は実データのままである。

通信環境では、転送プロセスにおける効率を最大に高め、ファイル転送または送信に伴うエラーを最少に抑えるには、デバイスに送信するデータ・ファイルのサイズはできるだけ小さい方がよい。したがって、動作アレイの合併およびインデックス化に続いて、一実施形態のファイル相違検出アルゴリズムは、動作アレイをデルタ・ファイルに書き込む際における動作アレイの冗長情報を削減することによって、デルタ・ファイルのサイズを極力小さくする。この冗長情報の削減を行うには、可変長整数フォーマットを用いて、動作アレイをデルタ・ファイルに書き込むが、実施形態はそのように限定される訳ではない。可変長整数フォーマットに加えて、絶対開始アドレスの代わりに相対開始アドレスを動作アレイにおいて用い、バイト・ストリームに対する変更が行われる場所を示す。

【０１０９】

一実施形態の可変長整数フォーマットの説明は、整数値をデルタ・ファイルにいかにして格納するかについての説明から開始する。符号なし整数を表すバイト・ストリームの長さは、整数値によって異なる。０と１２７との間の値の整数は、１バイトを使って表すことができる。整数値が１２８と１６，３８３との間である場合、整数は２バイトで表すことができる。１６，３８４と２，０９７，１５１との間の整数値は、３バイトで表される。２，０９７，１５１よりも大きい値を有する整数は、４バイト以上を必要とする。

【０１１０】

格納の目的のために、一実施形態の整数値を以下の形式の文字列に変換する。

【０１１１】

【表２３】

CIIIIIIICIIIIIII…CIIIIIII

1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8　　1 2 3 4 5 6 7 8

【０１１２】

ここで、「ＣＩＩＩＩＩＩＩ」は、１バイトの８ビットを表す。１バイトの最初のビット（「Ｃ」ビット）を連続ビットと呼ぶ。連続ビットは、現在の整数の一部として、読み出す関連のビットに、更に別のバイトが続くか否かを示す。したがって、Ｃが「１」の場合、続くバイトは、現整数の一部として読み取られなければならない。Ｃが「０」の場合、

整数は現在のバイトで終了する。このように、符号なし整数を表すこのストリング・フォーマットを、可変長整数フォーマットと呼ぶ。

【０１１３】

通信システムを参照して先に説明したように、デバイス間で送信するファイル、例えば、デルタ・ファイルのサイズは、できるだけ小さい方がいい。１０００個の整数があり、その値は０から１２７までの間であり、デルタ・ファイルに格納すると仮定する。各整数に４バイトを消費する場合、４０００バイトが必要となる。殆どの場合、ここに提供するアルゴリズムによって用いられる整数は、オフセット、パターン繰り返し回数、動作長、および、バイト・ストリームの相違についてのその他の特性を記述する。したがって、これらの整数の値は、通例、１６，３８３未満である。つまり、ここに提示する可変長整数フォーマットを用いれば、ファイル相違検出アルゴリズムが用いる典型的な整数を格納するには、平均で約１０００バイトあれば十分である。その結果得られる約３０００バイトの節約は、整数値には係わらず、各符号なし整数を表すための４バイトの保存が無駄であることを意味する。したがって、ここに提示する可変長整数フォーマットを用いると、デルタ・ファイルにおける整数の格納に必要な空間のサイズが５０％ないし７５％縮小する。

【０１１４】

図１２は、一実施形態の下におけるデルタ・ファイル１２００のブロック図である。デルタ・ファイルのフォーマットは、一般に、一連のレコード１２０２～１２０８の連接を含む。

【０１１５】

図１３は、図１２の実施形態の下における、デルタ・ファイル・レコード１３００のブロック図である。図１４は、図１３の実施形態の下における、動作記述を含む表である。

図１３および図１４を参照すると、この例では、各レコードのフォーマットは、次のようになっている。

【０１１６】

【表２４】

opcode＋relative_start_addr＋[other_meta_data]＋[real_data]

【０１１７】

「＋」符号は、掲示したフィールドの連接を表す。「オペコード」レコード１３０２は、動作コードを与える１バイトである。

「relative_start_addr」レコード１３０４は、相対的開始アドレスを与え、このアドレスで現在の変更が発生する。このレコード１３０４が最初のレコードである場合、「relative_start_add」レコード１３０４は、新バイト・ストリームの先頭に対するオフセットを表し、それ以外の場合、現在の絶対開始アドレスと最後のレコードにおける絶対開始アドレスとの間の差を表す。この方式を用いると、現在の変更に対する絶対開始アドレスの計算が容易となる。その利点は、大きな整数の代わりに、小さな整数がデルタ・ファイルに格納されることである。したがって、可変長整数フォーマットを用いることによって、デルタ・ファイルが数千のレコードを含む場合、数千バイトの節約となる。

【０１１８】

「other_meta_data」レコード１３０６は、動作コードに応じて、以下のフィールド、operating_len１３０６ａ、operated_len１３０６ｂ、repeat_times１３０６ｃ、interval_length１３０６ｄ、およびdelta_offset１３０６ｅのいずれかを表すが、そのように限定される訳ではない。「real_data」レコード１３０８は、オペコード「address_repeat_shift」に対するシフト値、あるいは実際の置換または挿入データである。角括弧「［］」は、含まれるフィールドが任意であり、動作コードによって異なることを示す。

【０１１９】

２５６個までの動作コードを定義することができるが、バイト・ストリーム間で遭遇した相違のパターンを検討し、一実施形態では次の動作コードを導入する。図１５Ａ、図１

５Ｂ、図１５Ｃおよび図１５Ｄは、動作コードと、一実施形態の下で導入した、関連するフィールドとの摘要表を示す。

【０１２０】

動作コードの各々の後にレコード・フォーマットを示し、手短に説明する。

INSERTIONオペコード：これは挿入動作である。レコード・フォーマットは次の通りである。

【０１２１】

【表２５】

opcode + relative_start_addr + operating_len + real_data

【０１２２】

オペコードはINSERTIONであり、一方「operating_len」および「real_data」フィールドは、挿入長および挿入データを与える。「operating_len」フィールドは、前述の可変長整数フォーマットである。

【０１２３】

operating_lenが１０以下の場合、以下のオペコードを導入して、レコード毎に１バイト節約する。

【０１２４】

【表２６】

ONE_BYTE_INSERTION,
TWO_BYTE_INSERTION,
THR_BYTE_INSERTION,
FOU_BYTE_INSERTION,
FIV_BYTE_INSERTION,
SIX_BYTE_INSERTION,
SEV_BYTE_INSERTION,
EIG_BYTE_INSERTION,
NIN_BYTE_INSERTION,
TEN_BYTE_INSERTION.

【０１２５】

これらは小規模挿入動作（１０バイト以下）である。レコード・フォーマットは、

【０１２６】

【表２７】

opcode + relative_start_addr + real_data

【０１２７】

となる。挿入長はオペコードによって暗示される。

DELETIONオペコード：これは、削除動作である。レコード・フォーマットは、次の通りである。

【０１２８】

【表２８】

opcode + relative_start_addr + operated_len

【０１２９】

オペコードはDELETIONであり、一方可変長整数フォーマットにおける「operated_length」フィールドは削除の長さを示す。operating_lenが１０以下の場合、以下のオペコードを導入して、レコード毎に１バイト節約する。

【０１３０】

【表２９】

ONE_BYTE_DELETION,
TWO_BYTE_DELETION,
THR_BYTE_DELETION,
FOU_BYTE_DELETION,
FIV_BYTE_DELETION,
SIX_BYTE_DELETION,
SEV_BYTE_DELETION,
EIG_BYTE_DELETION,
NIN_BYTE_DELETION,
TEN_BYTE_DELETION.

【０１３１】

これらは、小規模な削除動作（１０バイト以下）である。レコード・フォーマットは、

【０１３２】

【表３０】

opcode + relative_start_addr

【０１３３】

となる。削除長は、オペコードによって暗示される。

REPLACEMENTオペコード：これは、同じ長さの置換動作である。レコード・フォーマットは、次の通りである。

【０１３４】

【表３１】

opcode + relative_start_addr + operating_len + real_data

【０１３５】

オペコードはREPLACEMENTであり、「operating_len」および「real_data」フィールドは置換の長さおよび置換データを示す。「operating_len」フィールドは、可変長整数フォーマットにある。operating_lenが２０以下の場合、以下のオペコードを導入して、レコード毎に１バイト節約する。

【０１３６】

【表３２】

ONE_BYTE_REPLACEMENT,
TWO_BYTE_REPLACEMENT,
THR_BYTE_REPLACEMENT,
FOU_BYTE_REPLACEMENT,
FIV_BYTE_REPLACEMENT,
SIX_BYTE_REPLACEMENT,
SEV_BYTE_REPLACEMENT,
EIG_BYTE_REPLACEMENT,
NIN_BYTE_REPLACEMENT,
TEN_BYTE_REPLACEMENT;
T11_BYTE REPLACEMENT;
T12_BYTE REPLACEMENT;
T13_BYTE REPLACEMENT;
T14_BYTE REPLACEMENT;
T15_BYTE REPLACEMENT;
T16_BYTE REPLACEMENT;
T17_BYTE REPLACEMENT;
T18_BYTE REPLACEMENT;
T19_BYTE REPLACEMENT;
T20_BYTE REPLACEMENT.

【０１３７】

これらは小規模の置換動作（２０バイト以下）である。レコード・フォーマットは、

【０１３８】

【表３３】

opcode＋relative_start_addr＋real_data

【０１３９】

となる。置換長は、オペコードによって暗示される。

VARIABLE_LEN_REPLACEMENTオペコード：これは可変長置換動作である。レコード・フォーマットは次の通りである。

【０１４０】

オペコードはVARIABLE_LEN_REPLACEMENTであり、一方

【０１４１】

【表３４】

opcode＋relative_start_addr＋operating_len＋operated_len＋real_data

【０１４２】

「operating_len」、「operated_len」および「real_data」フィールドは、それぞれ、置き換える長さ、置き換えられる長さ、および実際の置換データを与える。「operating_len」および「operated_len」フィールドは、可変長整数フォーマットである。

EXACT_COPY_INDEX_TOオペコード：これは、現在の動作がバイト・ストリーム内のどこかで以前に行われたことを示す。レコード・フォーマットは次の通りである。

【０１４３】

【表３５】

opcode＋relative_start_addr＋delta_offset

【０１４４】

オペコードはEXACT_COPY_INDEX_TOであり、可変長整数フォーマットにおける「delta_offset」フィールドは、コピーした動作のレコードがあるデルタ・ファイルにおけるオフセットである。

WHOLE_REPLACEMENTオペコード：これは全体置換動作である。レコード・フォーマットは次の通りである。

【０１４５】

【表３６】

opcode＋real_data

【０１４６】

オペコードは、WHOLE_REPLACEMENTである。「relative_start_addr」フィールドはここでは現れない。何故なら、これはデフォルトで「０」の値になっているからである。「real_data」フィールドは、新バイト・ストリームにおける全てのバイトである。

ADDRESS_REPEAT_SHIFTオペコード：これは、多重１バイト・アドレス・シフト動作(multi-one-byte address shift operation)である。レコード・フォーマットは次の通りである。

【０１４７】

【表３７】

opcode＋relative_start_addr＋repeat_times＋interval_len＋real_data

【０１４８】

オペコードはADDRESS_REPEAT_SHIFTであり、一方「repeat_times」および「interval_len」フィールドは、繰り返し回数および間隔長を与える。real_dataは実際のアドレス・シフトを含み、１バイト文字がシフト値を知らせる。「repeat_times」および「interval_len」フィールドは、可変長整数フォーマットである。

SAME_ADDRESS_REPEAT_SHIFTオペコード：これは、多重１バイト・アドレス・シフト動作であるが、シフト値は、以前のADDRESS_REPEAT_SHIFTレコードによって暗示される。したがって、以前のレコードと比較して、１バイトの節約となる。レコード・フォーマットは次の通りである。

【０１４９】

【表３８】

opcode＋relative_start_addr＋repeat_times＋interval_len

【０１５０】

オペコードはSAME_ADDRESS_REPEAT_SHIFTであり、一方「repeat_time」および「interval_len」フィールドは、繰り返し回数および間隔長を与える。「repeat_times」および「interval_len」フィールドは、可変長整数フォーマットである。

他の全オペコード：他のオペコードは、動作アレイ、例えば、固定長間隔で繰り返し現れる短い長さの動作シーケンスにおいて見出される動作のパターンを利用する。このようなシーケンスの一例は、

【０１５１】

【表３９】

RRMMMMRRMMMMRRMMMM

【０１５２】

を含む。このパターンに用いられるオペコードは、

【０１５３】

【表４０】

TWO_BYTE_FIX_LEN_REPEAT_REPLACE

【０１５４】

であり、置換動作の２バイト固定長繰り返しシーケンス（「ＲＲ」）を示す。

別の例は、

【０１５５】

【表４１】

RRMMMMRRMMMMRRI

【０１５６】

というシーケンスである。このパターンに用いられるオペコードは、

【０１５７】

【表４２】

TWO_BYTE_FIX_LEN_REPEAT_REPLACE_END_NO_M

【０１５８】

であり、置換動作の２バイト固定長繰り返しシーケンス（「ＲＲ」）を示し、この場合、シーケンスは、一致動作では終わらない。これら他の動作コードおよびこれらの例を参照すると、各間隔において、置換長は、３２バイト以下である。レコード・フォーマットは次の通りである。

【０１５９】

【表４３】

opcode＋relative_start_addr＋repeat_times＋interval_len＋real_data

【０１６０】

「repeat_times」および「interval_len」フィールドは、繰り返し回数および間隔長を与える。「real_data」フィールドは、各間隔における置換データの連接である。「repea

t_times」および「interval_len」フィールドは、前述の可変長整数フォーマットである。

【０１６１】

先に詳細に説明したような、２つの入力バイト・ストリームからのデルタ・ファイル生成を例示するために、以下に一例を示す。この例では、２つの入力バイト・ストリームは、旧バイト・ストリームおよび新バイト・ストリームを含む。旧バイト・ストリームは、次の通りである。

【０１６２】

【表４４】

555551212121delet121212replaced141414141414fixlenreplace999999999

【０１６３】

また、新バイト・ストリームは、次の通りである。

【０１６４】

【表４５】

55555insertion1212121212121REPLACING141414141414FIXLENREPLACE

999999999

【０１６５】

旧バイト・ストリームおよび新バイト・ストリームは、前述のように、前処理されており、ファイル相違検出アルゴリズムに入力される旧バイト・ストリームを生成する。これは、次の通りである。

【０１６６】

【表４６】

121212delet121212replaced141414141414fixlenreplace

【０１６７】

そして、ファイル相違検出アルゴリズムに入力される新バイト・ストリームも生成され、これは、

【０１６８】

【表４７】

insertion1212121212121REPLACING141414141414FIXLENREPLACE

【０１６９】

これら２つのバイト・ストリームに対応する動作アレイは、次の通りである。

【０１７０】

【表４８】

IIIIIIIIMMMMMMMDDDDDMMMMMMMIRRRRRRRRRMMMMMMMMMMM

MMMRRRRRRRRRRRR

【０１７１】

デルタ・ファイルの生成は、先に詳細に説明したように行われ、その結果得られたデルタ・ファイルは４つのレコードを含む。最初のデルタ・ファイル・レコードは、次の通りである。

【０１７２】

【表４９】

opcode NIN_BYTE_INSERTION
relative_start_addr 5
real_data insertion

【０１７３】
この最初のレコードは１１バイトを含む。２番目のデルタ・ファイル・レコードは、次の通りであり、
【０１７４】
【表５０】
opcode FIV_BYTE_DELETION
relative_start_addr 6

【０１７５】
２バイトを含む。３番目のデルタ・ファイル・レコードは、次の通りである。
【０１７６】
【表５１】
opcode VARIABLE_LEN_REPLACEMENT
relative_start_addr 6
operating_len 9
operated_len 8
real_data REPLACING

【０１７７】
この３番目のレコードは１３バイトを含む。４番目のデルタ・ファイル・レコードは、次の通りである。
【０１７８】
【表５２】
opcode T13_BYTE_REPLACEMENT
relative_start_addr 23
real_data FIXLENREPLACE

【０１７９】
この４番目のデルタ・ファイル・レコードは１５バイトを含む。したがって、比較前におけるこのデルタ・ファイルの全体的なサイズは、４つのデルタ・ファイル・レコードの各々の総和、即ち、４１バイトである。
【０１８０】
デルタ・ファイルの生成に続いて、一実施形態では、追加の動作を実行して、他のシステムへのデルタ・ファイルの転送効率を更に高めることもできる。例えば、圧縮を用いると、デルタ・ファイルのサイズを更に縮小して送信することができる。関連する伸張モジュールは、プロセッサを用い資源が限られたデバイス上にホストされている場合もあるので、用いる圧縮アルゴリズムは、予め割り当てられたメモリ上でできるだけ速く実行するアルゴリズムを用いて伸張可能なファイルを供給するものでなければならない。即ち、伸張アルゴリズムは、メモリを動的に割り当てない方がよい。圧縮の結果常にデルタ・ファイルのサイズが更に縮小するとは限らないが、デルタ・ファイルは、指定されたバイトを用いて、いつファイルを圧縮するか示す。
【０１８１】
一実施形態のファイル相違検出アルゴリズムは、チェックサム誤り訂正コード化のような、何らかの機構を用いて、送信するデルタ・ファイルのあらゆる転化(corruption)を検出するが、そのように限定される訳ではない。単純なチェックサム方式の１つは、次の通りである。チェックサムの長さは１バイトであり、チェックサムをデルタ・ファイルの終端に置く。チェックサムは、最後のバイトを除いてデルタ・ファイルの全バイトの和を表すバイトの２の補数である。このようにして、誤りのないデルタ・ファイルにおける全てのファイルを加算すると、０に等しい結果が得られる。
【０１８２】
生成に続いて、一実施形態のデルタ・ファイルを格納しておき、後に別のコンピュータ

・システムにダウンロードし、当該他のコンピュータ・システム上にホストされている原ファイルのコピーを更新する際に使用する。図１を参照すると、一旦何らかの伝送媒体１０６を通じて受信側コンピュータ・システム１０４においてデルタ・ファイル１１６を受信したなら、ファイル更新アルゴリズム１１８をコールする。

【０１８３】

図１６は、一実施形態の下において、対応する原ファイルおよびデルタ・ファイルを用いて新ファイルを生成するフロー図である。ブロック１６０２において、原ファイルおよびデルタ・ファイルを受信すると、ファイル更新アルゴリズムは、ブロック１６０４において、チェックサムのような基本的技法を用いて、デルタ・ファイルの転化または誤りをチェックする。デルタ・ファイルが転化している場合、ブロック１６１４において出力エラー・メッセージを生成し、動作は戻る。

【０１８４】

デルタ・ファイルに誤りがない場合、ファイル更新アルゴリズムは、ブロック１６０６において、デルタ・ファイルが圧縮されているか否か判定を行う。この判定を行うには、デルタ・ファイルの１つまたは複数の指定バイトを用いる。デルタ・ファイルが圧縮されており、圧縮方法がわかっている場合、ファイル更新アルゴリズムはデルタ・ファイルを伸張する。伸張に続いて、ファイル更新アルゴリズムは、ブロック１６１６において、伸張が成功であったか否か判定を行う。伸張が成功せずに終了した場合、ブロック１６１４において出力エラー・メッセージを生成し、動作は戻る。伸張が成功して終了した場合、ブロック１６０８においてデルタ・ファイルの読み取りを開始する。

【０１８５】

デルタ・ファイルからの読み取りが開始すると、ブロック１６０８において、ファイル更新アルゴリズムは、既定のデルタ・ファイル・フォーマットをプロトコルとして用い、デルタ・ファイル内に内在する相違を解析し理解して、新バージョンのファイルを１００％の精度で作成する。デルタ・ファイルの各レコードを読み出す毎に、ファイル更新アルゴリズムは、ブロック１６１０において、デルタ・ファイルから読み出すために追加のレコードが使用可能か否か判定を行う。追加のレコードが使用可能でありそして読み出す場合、ファイル更新アルゴリズムは、ブロック１６１８において、レコードのオペコードおよび関連するフィールドの有効性をチェックする。オペコードまたは追加フィールドのいずれかが無効と見なされた場合、または未知のオペコードのような、何らかの誤りが更新プロセス中に発生した場合、ブロック１６１４において出力エラー・メッセージを生成して、更新失敗を知らせ、動作は戻る。

【０１８６】

オペコードおよび関連するフィールドが有効である場合、ファイル更新アルゴリズムは、ブロック１６２０において新ファイルの対応する部分を生成し、ブロック１６１０において、デルタ・ファイルの残りのレコードを読み出す。デルタ・ファイルから読み出すレコードが既に残っていない場合、ブロック１６１２において新ファイルを供給し、動作は戻る。

【０１８７】

一例として、ファイル更新アルゴリズムの記述を以下に示す。

【０１８８】

【表５３】

```
Procedure GETUPDATE(o_ptr, n_ptr, delta_file)
    Begin
        If delta_file is corrupted, return the error;
        If delta_file is compressed, do de-compressing;
        Set delta_offset = 0;
        While (delta_offset < delta_file_length) {
            Get opcode;
            Based on the opcode, do the corresponding operation
                including adjusting delta_offset, o_ptr, n_ptr;
            If any error encounters, return the error;
        }
        Output new version file;
    End
```

【０１８９】

このファイル更新アルゴリズムの例では、o_ptrは、旧バージョンのファイルが位置する場所を指し示すポインタを表し、一方n_ptrは、新バージョンのファイルを保存しようとする記憶位置を指し示すポインタを表す。delta_fileは、何らかの伝送媒体を通じて受信したデルタ・ファイルを表す。delta_file_lengthは、圧縮されない場合のデルタ・ファイルの長さ、または圧縮されたデルタ・ファイルの圧縮されていない場合の長さを表す。

【０１９０】

一実施形態のファイル更新アルゴリズムは、予め割り当てたメモリ上で動作する。更に、ファイル更新アルゴリズムは、資源が限られているいずれの計算機上でも非常に高速に機能する。

【０１９１】

一例として、デルタ・ファイルを受信し使用する計算機は、更新を必要とする、対応のソフトウェア・アプリケーションをホストするクライアント・デバイスとすることができ、例えば、セルラ電話機、移動電子デバイス、移動通信デバイス、パーソナル・ディジタル・アシスタント、およびその他のプロセッサ・ベースのデバイスがある。この対応は、キャリアやデバイス製造業者に電子ファイル内容およびアプリケーションを彼らのワイヤレス・インフラストラクチャを通じて効率的に配布させることによって、ファームウェアから埋め込みアプリケーションまでの範囲に及ぶあらゆる移動デバイスのソフトウェアに与えられる。

【０１９２】

ＦＤＵアルゴリズムの使用によって効果が得られるシステムの一例には、ワイヤード・シリアル接続を用いてデルタ・ファイルを、ファイル相違検出アルゴリズムをホストするデバイスから、ファイル更新アルゴリズムをホストするデバイスに転送するシステムが含まれる。これらのシステムは、転送レートが遅いのが通例である（現在、約１９２ｋバイト／秒）。転送レートが遅いので、デルタ・ファイルのサイズ縮小は、転送時間短縮を実現する１つの方法である。したがって、これらシリアル転送システムにおけるＦＤＵアルゴリズムの使用は、ユーザの満足度を高めることになる。

【０１９３】

ＦＤＵアルゴリズムの使用によって効果が得られるシステムの別の例には、無線通信を用いてデルタ・ファイルを、ファイル相違検出アルゴリズムをホストするデバイスから、ファイル更新アルゴリズムをホストするデバイスに転送するシステムが含まれる。ワイヤレス接続に伴う信頼性の低さという欠点があるが、これらのシステムも転送レートは低い（現在、約９．６ないし６４ｋバイト／秒）。これらのシステムにおいて縮小したデルタ・ファイルの使用により、様々な利点が得られる。例えば、ファイル・サイズ縮小の結果、デルタ・ファイルの転送時間が短縮する。転送時間の短縮によって、一方ではデバイス・ユーザの時間が節約され、他方ではエラーがデルタ・ファイルに混入する機会が減るこ

とによって、システムの信頼性が向上する。また、セルラ通信出は、転送時間短縮の結果、通例分毎にサービスに対して課金される消費者のコスト節約となる。

【０１９４】

別の利点として、デルタ・ファイルの縮小によって、デルタ・ファイルをクライアント・デバイスに転送するために必要な帯域幅が減少する。帯域幅の減少によって、割り当てられたチャネルを通じて対応できるクライアント・デバイスの数が増加する。転送時間の短縮に関しては、これによっても、ワイヤレス・サービス提供側にとって、運営コストの削減となる。

【０１９５】

前述のアルゴリズムは、電子ファイルを更新するシステムと共に用い、電子ファイルを更新するシステムは、バイト・レベルのファイル相違検出および更新（ＦＤＵ）アルゴリズムを備えている。ＦＤＵアルゴリズムのファイル相違検出アルゴリズムは、第１デバイス上にホストされており、ファイル相違検出アルゴリズムは、原ファイルおよび新ファイルに対応するバイト・ストリームを含む、２つのバイト・ストリームを受信する。ここで、新ファイルは、原ファイルの更新および改訂バージョンを含む。次いで、２つのバイト・ストリーム間の最長共通サブストリング（ＬＣＳ）を判定し、２つのバイト・ストリームの各々をサブストリームに分割する。第１サブストリームはそれぞれのＬＣＳの接頭部であり、第２サブストリームはそれぞれのＬＣＳの接尾部である。サブストリームのサイズが予め指定したサイズ未満になるまで、再帰的にＬＣＳを判定し、各サブストリームを分割し、対応するサブストリームの各々の間でバイト・レベルの相違を判定し、１組の動作コードおよび可変長整数フォーマットを用いることによってデルタ・ファイルを構築し、デルタ・ファイルにおける冗長データを除外し、判定したバイト・レベルの相違をコード化した動作アレイを含むデルタ・ファイルを生成する。この生成には、動作アレイの動作を合併してメタデータ量を削減することを含む。次いで、デルタ・ファイルを、少なくとも１つの結合部を通じて第２デバイスに転送する。

【０１９６】

ＦＤＵアルゴリズムのファイル更新アルゴリズムは、システムの第２デバイス上にホストされており、ファイル更新アルゴリズムは、少なくとも１つの結合部を介して第１デバイスからデルタ・ファイルを受信し、デルタ・ファイルおよびホストされている原ファイルを用いて新ファイルのコピーを生成し、そのコピーを用いて第２デバイスにおける原ファイルを更新する。

【０１９７】

一実施形態の第１デバイスは、プロセッサ・ベースのデバイスであり、第２デバイス上で実行するソフトウェアの少なくとも１つの供給元によってアクセス可能である。

一実施形態の第２デバイスは、パーソナル・コンピュータ、携帯用計算機、セルラ電話機、携帯用通信デバイス、およびパーソナル・ディジタル・アシスタントの中から選択した、少なくとも１つのプロセッサを用いるデバイスである。

【０１９８】

一実施形態のシステムのデバイス間の結合部は、ワイヤレス接続、ワイヤード接続、混合ワイヤレス／ワイヤード接続、およびローカル・エリア・ネットワーク（ＬＡＮ）、都市区域ネットワーク（ＭＡＮ）、ワイド・エリア・ネットワーク（ＷＡＮ）、企業固有のネットワーク(proprietary network)、バックエンド・ネットワーク(backend network)、インターネットを含むネットワークへの結合部、ならびにフロッピ・ディスク、ハード・ディスク・ドライブ、およびＤＣ－ＲＯＭディスクを含む着脱可能固定媒体、そして電話回線、バス、および電子メール・メッセージから選択する。

【０１９９】

原ファイルおよび新ファイルは、ダイナミック・リンク・ライブラリ・ファイル、共有オブジェクト・ファイル、埋め込みソフトウェア・コンポーネント（ＥＢＳＣ）、ファームウェア・ファイル、実行可能ファイル、１６進データ・ファイル、システム・コンフィギュレーション・ファイル、および個人使用データを含むファイルを含むデータ・ファイ

ルを含む。

【０２００】

一実施形態のシステムは、更に、第１および第２デバイス間に結合されている通信サービス提供側の少なくとも１つのサーバを備えており、少なくとも１つのサーバは、第１デバイスからデルタ・ファイルを受信し、デルタ・ファイルの受信に応答して第２デバイスにメッセージを生成する。このメッセージは、第２デバイスのユーザに、第２デバイス上にホストされている少なくとも１つの電子ファイルに対する更新の利用可能性について知らせる。サーバは、更に、ユーザからの更新要求を受信し、要求に応答してデルタ・ファイルを第２デバイスに供給する。

【０２０１】

前述のファイル相違検出および更新アルゴリズムは、電子ファイルを更新する装置と共に用いる。この装置は、バイト・レベルのファイル相違検出および更新を実行するように構成された少なくとも１つのプロセッサを含む。この装置は、原ファイルに対応するバイト・ストリームと新ファイルに対応するバイト・ストリームとを含む２つのバイト・ストリームを受信する手段と、２つの受信したバイト・ストリーム間において最も長い共通サブストリング（ＬＣＳ）を判定する手段と、２つの受信したバイト・ストリームの各々をサブストリームに分割する手段であって、第１サブストリームはそれぞれのＬＣＳの接頭部であり、第２サブストリームはそれぞれのＬＣＳの接尾部である、手段と、サブストリームのサイズが予め指定したサイズ未満となるまで、再帰的にＬＣＳを判定し、各サブストリームを分割する手段と、サブストリームの各々の間のバイト・レベルの相違を判定する手段であって、バイト・レベルの相違を原ファイルと新ファイルとの間で判定する、手段と、判定したバイト・レベルの相違をコード化する動作アレイを生成する手段と、可変長整数フォーマットを用いて、最適化動作アレイを書き込むことによって、デルタ・ファイルを生成する手段とを含むが、これらに限定されるのではない。

【０２０２】

前述のファイル相違検出および更新アルゴリズムは、電子ファイルを更新する装置と共に用いる。この装置は、バイト・レベルのファイル相違検出および更新を実行するように構成された少なくとも１つの処理デバイスを含む。この装置は、デルタ・ファイルを受信する手段と、デルタ・ファイルを用いて新ファイルのコピーを生成する手段と、第２デバイスにおいてコピーを用いて原ファイルを更新する手段とを備えているが、これらに限定されるのではない。デルタ・ファイルの生成は、原ファイルに対応するバイト・ストリームと、原ファイルの新バージョンに対応するバイト・ストリームとを含む２つのバイト・ストリーム間における最も長い共通サブストリング（ＬＣＳ）を判定し、２つのバイト・ストリームの各々をサブストリームに分割し、第１サブストリームがそれぞれのＬＣＳの接頭部であり、第２サブストリームがそれぞれのＬＣＳの接尾部であり、サブストリームのサイズが予め指定したサイズ未満となるまで、再帰的にＬＣＳを判定し各サブストリームを分割し、対応するサブストリームの各々の間でバイト・レベルの相違を判定し、動作アレイにおいてバイト・レベルの相違をコード化することによってデルタ・ファイルを生成し、動作アレイの少なくとも１つの動作を合併してメタデータの量を削減し、可変長整数フォーマットを用いて動作アレイを書き込むことから成るが、このように限定されるのではない。

【０２０３】

前述のファイル相違検出および更新アルゴリズムは、バイト・レベルのファイル相違検出を実行する方法を備えている。一実施形態の方法は、２つの受信したバイト・ストリーム間における最も長い共通サブストリング（ＬＣＳ）を識別するステップであって、バイト・ストリームが原ファイルに対応するバイト・ストリームと、原ファイルの新ファイル・バージョンに対応するバイト・ストリームとを含む、ステップと、２つの受信したバイト・ストリームの各々をサブストリームに分割するステップであって、第１サブストリームがそれぞれのＬＣＳの接頭部であり、第２サブストリームがそれぞれのＬＣＳの接尾部である、ステップと、サブストリームのサイズが予め指定したサイズ未満となるまで、ま

たはＬＣＳがなくなるまで、再帰的にＬＣＳを判定し各サブストリームを分割するステップと、対応するサブストリームの各々の間でバイト・レベルの相違を判定するステップと、判定したバイト・レベルの相違をコード化する動作アレイを生成するステップであって、生成において、動作アレイの動作を合併してメタデータの量を削減することを含む、ステップと、動作アレイをデルタ・ファイルに書き込むステップであって、書き込みにおいて、動作アレイの動作を合併してメタデータの量を削減し、１組の動作コードおよび可変長整数フォーマットを用いることによってデルタ・ファイルを構築するプロトコルを用いて、デルタ・ファイルにおける冗長データを排除する、ステップとから成るが、そのように限定されるのではない。

【０２０４】

一実施形態では、動作アレイを書き込むステップは、更に、相対的開始アドレスを用いて、バイト・ストリームの変更位置を識別するステップを含む。

一実施形態では、可変長整数フォーマットは、１バイトの内の１ビットを用い、後続のバイトを整数の一部として読み取るか否かについて示す。

【０２０５】

一実施形態では、デルタ・ファイル・フォーマットは、動作コード、動作コードを適用する新ファイルの相対的開始アドレス、および必要な場合に原ファイルに対する挿入または置換のデータから選択した一連のレコードの連接を備えている。

【０２０６】

前述のファイル相違検出および更新アルゴリズムは、バイト・レベルのファイル相違検出を実行する方法を備え、更に、デルタ・ファイルを圧縮するステップと、誤り訂正コード化をデルタ・ファイルに適用するステップとを備えている。

【０２０７】

前述のファイル相違検出および更新アルゴリズムは、バイト・レベルのファイル相違検出を実行する方法を備え、更に、バイト・ストリームの受信時に、２つのバイト・ストリームの各々を半分に分割するステップと、バイト・ストリームが最大指定サイズよりも長いか否か判定するステップとを備えている。

【０２０８】

前述のファイル相違検出および更新アルゴリズムは、バイト・レベルのファイル相違検出を実行する方法を備え、更に、２つのバイト・ストリーム間または２つのサブストリーム間にＬＣＳがない場合、新ファイルに対応するバイト・ストリームの長さが原ファイルに対応するバイト・ストリームの長さよりも短いとき、置換動作および削除動作を動作アレイの適当な位置に書き込むステップを備えている。

【０２０９】

前述のファイル相違検出および更新アルゴリズムは、バイト・レベルのファイル相違検出を実行する方法を備え、更に、２つのバイト・ストリーム間または２つのサブストリーム間にＬＣＳがない場合、新ファイルに対応するバイト・ストリームの長さが原ファイルに対応するバイト・ストリームの長さよりも長いとき、置換動作および挿入動作を動作アレイの適当な位置に書き込むステップを備えている。

【０２１０】

前述のファイル相違検出および更新アルゴリズムは、バイト・レベルのファイル相違検出を実行する方法を備えており、合併する際に、更に、動作アレイにおける一致動作の短いシーケンスを突き止めるステップであって、短いシーケンスが３つ未満の連続する一致動作を含む、ステップと、短いシーケンスの一致動作を、代替動作と置換するステップとを備えている。

【０２１１】

前述のファイル相違検出および更新アルゴリズムは、バイト・レベルのファイル相違検出を実行する方法を備えており、合併する際に、更に、動作アレイにおける一致動作以外の動作のシーケンスを突き止めるステップと、突き止めたシーケンスを、動作の種類およびシーケンスの長さを指定する単一の動作で置換するステップとを備えている。

【０２１２】
前述のファイル相違検出および更新アルゴリズムは、バイト・レベルのファイル相違検出を実行する方法を備えており、合併する際に、更に、挿入、削除、および代替動作の少なくとも１つの組み合わせを含むシーケンスを突き止めるステップと、このシーケンスを単一の可変長置換動作と置換するステップとを備えている。
【０２１３】
前述のファイル相違検出および更新アルゴリズムは、バイト・レベルのファイル相違検出を実行する方法を備えており、合併する際に、等しい数の挿入および削除動作を含むシーケンスを突き止めるステップと、このシーケンスを単一の固定長置換動作と置換するステップとを備えている。
【０２１４】
前述のファイル相違検出および更新アルゴリズムは、バイト・レベルのファイル相違検出を実行する方法を備えており、合併する際に、更に、あるパターンで繰り返す動作単位を突き止めるステップであって、この単位が置換動作の固定長シーケンスと、別の種類の動作の固定長シーケンスとを含む、ステップと、置換動作シーケンスの長さ、単位の長さ、および単位がパターンを繰り返す回数を指定する単一動作と、繰り返し単位を置換するステップとを備えている。
【０２１５】
前述のファイル相違検出および更新アルゴリズムは、バイト・レベルのファイル相違検出を実行する方法を備えており、合併する際に、更に、動作アレイに適当な動作をハッシュ・テーブルに入力するステップと、オフセット・パラメータを含む単一のコピー動作を用い、コピー動作によって、ハッシュ・テーブルの情報を用いて、動作アレイ内の適当な位置に動作を挿入するステップとを備えている。
【０２１６】
前述のファイル相違検出および更新アルゴリズムは、バイト・レベルのファイル相違検出を実行する方法を備えており、前述の予め指定したサイズは、約４０９６バイトであるが、実施形態はそのように限定されているのではない。
【０２１７】
前述のファイル相違検出および更新アルゴリズムは、電子ファイル間で相違ファイルを生成する方法を備えている。一実施形態では、この相違ファイルを生成する方法は、原バイト・ストリームと新バイト・ストリームとを含むバイト・ストリームを受信するステップと、バイト・ストリームの各々のサイズを判定するステップと、バイト・ストリームの各々が第１サイズである場合、線形時間分析を用いてバイト・ストリーム間における最も長い共通サブストリング（ＬＣＳ）を計算するステップと、バイト・ストリームの各々から少なくとも１つの接頭バイト・ストリーム集合を生成するステップであって、各接頭バイト・ストリーム集合はＬＣＳに先立つバイトを含む、ステップと、バイト・ストリームの各々から少なくとも１つの接尾バイト・ストリーム集合を生成するステップであって、各接尾バイト・ストリーム集合はＬＣＳの後ろに続くバイトを含む、ステップと、生成した接頭バイト・ストリーム集合のサイズが第２サイズ未満となるまで、再帰的にサイズを判定し、ＬＣＳを計算し、生成した接頭バイト・ストリーム集合および接尾バイト・ストリーム集合の各々に対して少なくとも１つの接頭バイト・ストリーム集合および少なくとも１つの接尾バイト・ストリーム集合を生成するステップと、生成した接頭バイト・ストリーム集合の各々のバイト間、および生成した接尾バイト・ストリーム集合の各々のバイト間の編集距離を判定するステップであって、編集距離は、原バイト・ストリームを新バイト・ストリームに変換するための最少数の編集動作である、ステップと、バイト・ストリーム間の編集距離の情報を含む動作アレイから成るデルタ・ファイルを生成するステップとを備えているが、そのように限定されるのではない。
【０２１８】
一実施形態では、相違ファイルを生成する方法は、更に、デルタ・ファイルのサイズを縮小するステップを備えており、その際、少なくとも１つの合併およびインデックス化動

作を動作アレイに適用してメタデータの量を削減し、可変長整数フォーマットを用いて動作アレイを書き込み、動作アレイの冗長情報を削減する。

【０２１９】

前述のファイル相違検出および更新アルゴリズムは、バイト・レベルでファイル相違検出および更新を実行する方法を備えている。一実施形態では、この方法は、第１システムにおいて２つのバイト・ストリームを受信するステップであって、２つのバイト・ストリームが、原ファイルおよび新ファイル双方に対応するバイト・ストリームを含む、ステップと、２つのバイト・ストリーム間における最も長い共通サブストリング（ＬＣＳ）を判定し、２つのバイト・ストリームの各々をサブストリームに分割するステップであって、第１サブストリームはそれぞれのＬＣＳの接頭部であり、第２サブストリームはそれぞれのＬＣＳの接尾部である、ステップと、サブストリームのサイズが予め指定したサイズ未満となるまで、再帰的にＬＣＳを判定し各サブストリームを分割し、対応するサブストリームの各々の間におけるバイト・レベルの相違を判定するステップと、判定したバイト・レベルの相違をコード化した動作アレイを含むデルタ・ファイルを生成するステップであって、その生成において、動作アレイの少なくとも１つの動作を合併してメタデータの量を削減し、可変長整数フォーマットを用いて動作アレイを書き込む、ステップと、デルタ・データを第２システムに転送するステップと、デルタ・ファイルを用いて、第２システムにおいて新ファイルのコピーを生成するステップと、第２システムにおいて新ファイルのコピーを用いて原ファイルを更新するステップとを備えているが、このように限定されるのではない。

【０２２０】

前述のファイル相違検出および更新は、実行可能命令を含むコンピュータ読み取り可能媒体を含み、処理システムにおいて実行すると、バイト・レベルのファイル相違検出および更新を実行し、その際に、２つの受信したバイト・ストリーム間における最も長い共通のサブストリング（ＬＣＳ）を判定し、バイト・ストリームが原ファイルに対応するバイト・ストリームと、原ファイルの新ファイル・バージョンに対応するバイト・ストリームとを含み、２つの受信したバイト・ストリームの各々をサブストリームに分割し、第１サブストリームはそれぞれのＬＣＳの接頭部であり、第２サブストリームはそれぞれのＬＣＳの接尾部であり、サブストリームのサイズが予め指定したサイズ未満になるまで再帰的にＬＣＳを判定し各サブストリームを分割し、対応するサブストリームの各々の間でバイト・レベルの相違を判定し、判定したバイト・レベルの相違をコード化する動作アレイを生成し、生成する際に動作アレイの少なくとも１つの動作を合併してメタデータ量を削減することを含み、可変長整数フォーマットを用いて動作アレイをデルタ・ファイルに書き込む。

【０２２１】

文脈から特に明確に必要としない限り、明細書および特許請求の範囲全域において、「備える」、「備えている」等の言葉は、排他的または網羅的意味ではなく、包含的意味で解釈するものとする。即ち、「含むが、限定されない」という意味である。単数または複数を用いる単語も、それぞれ、複数および単数を含むものとする。加えて、「ここでは」、「以下で」という単語、および同様な意義の単語は、本願において用いられる場合、本願全体を指すのであり、本願のいずれの特定部分を指すのではないとする。

【０２２２】

本発明の例示した実施例についてのこれまでの説明は、それで全てであることも、開示した正確な形態に本発明を限定することも意図していない。本発明の具体的な実施形態および例を例示の目的のためにここで記載したが、種々の等価な変更も本発明の範囲内で可能であることは、当業者は認めよう。ここに提示した本発明の教示は、他の処理システムや通信システムにも適用することができ、前述のＦＤＵアルゴリズムのみに適用できるのではない。

【０２２３】

前述した種々の実施形態のエレメントおよび作用を組み合わせれば、更に別の実施形態

も得ることができる。これらおよびその他の変更は、前述の詳細な説明に鑑みれば、容易に行うことができる。

【０２２４】

先の引例は全て、ここで引用したことにより、本願にも含まれるものとする。必要であれば、本発明の形態を変更して前述の種々の特許および出願のシステム、機能および概念を採用すれば、本発明の更に別の実施形態を得ることができる。

【０２２５】

一般に、特許請求の範囲においては、用いられる用語は、明細書に開示される具体的な実施形態および特許請求の範囲に本発明を限定するように解釈するのではなく、特許請求の範囲の下で動作してファイル相違検出方法を実施するあらゆる処理システムを含むように解釈するのが当然である。したがって、本発明は、本開示に限定されるのではなく、むしろ発明の範囲は特許請求の範囲によって全体的に決定されるものとする。

【０２２６】

本発明のある種の形態をある請求項の形式で提示するが、本願の発明者は、本発明の種々の形態をいかなる数の請求項の形式でも想定している。例えば、本発明の一形態のみを、コンピュータ読み取り可能媒体に具体化するものとして明記するが、他の形態も同様にコンピュータ読み取り可能媒体に具体化することができる。したがって、本発明は、本願を出願した後でも、更に別の請求項を追加し、本発明の別の形態に対する、このような更に別の請求項を追求する権利を保有するものとする。

【図面の簡単な説明】

【０２２７】

【図１】図１は、一実施形態の下における、バイト・レベルのファイル相違検出および更新（ＦＤＵ）アルゴリズムによって行われるファイル相違検出および更新を示すブロック図である。

【図２】図２は、図１の実施形態の下におけるデルタ・ファイルの生成のフロー図である。

【図３】図３は、ＦＤＵアルゴリズムの一実施形態の下において、ファイル間の比較を行う前にファイルを前処理する方法のフロー図である。

【図４】図４は、一実施形態の下における新たなファイル・接尾部の識別した挿入データを示す。

【図５】図５は、一実施形態の下における新たなファイル接頭部の識別した削除データを示す。

【図６】図６は、一実施形態の下における原ファイル・接尾部の識別した削除データを示す。

【図７】図７は、一実施形態の下における原ファイル接頭部の識別した削除データを示す。

【図８】図８は、一実施形態の下における新たなファイルの識別した挿入データを示す。

【図９】図９は、一実施形態の下における原ファイルの識別した削除データを示す。

【図１０】図１０は、図１の実施形態の下におけるバイト・レベルのファイル相違検出アルゴリズムのフロー図である。

【図１１】図１１は、図１０の実施形態の下における、識別した最も長い共通サブストリング（ＬＣＳ）および関連する接頭部および接尾部を示す。

【図１２】図１２は、一実施形態の下における、一連のレコードの連接を含むデルタ・ファイルのブロック図である。

【図１３】図１３は、図１２の実施形態の下における、デルタ・ファイル・レコードのブロック図である。

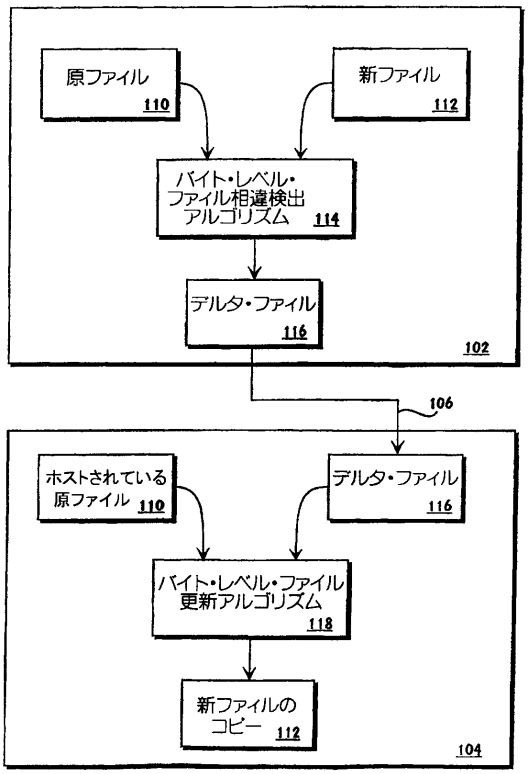
【図１４】図１４は、図１３の実施形態の下における、動作記述を含む表である。

【図１５Ａ】図１５Ａは、一実施形態の下で導入した動作コードの摘要表を、関連するフィールドと共に示す。

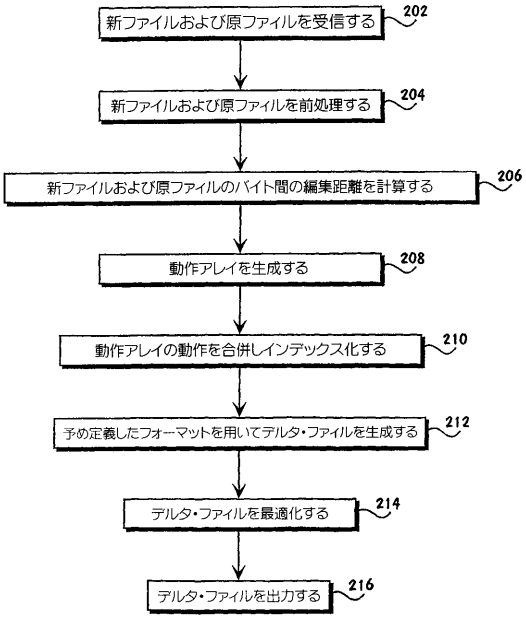
【図１５Ｂ】図１５Ｂは、一実施形態の下で導入した動作コードの摘要表を、関連するフ

ィールドと共に示す。
【図１５Ｃ】図１５Ｃは、一実施形態の下で導入した動作コードの摘要表を、関連するフィールドと共に示す。
【図１５Ｄ】図１５Ｄは、一実施形態の下で導入した動作コードの摘要表を、関連するフィールドと共に示す。
【図１５Ｅ】図１５Ｅは、一実施形態の下で導入した動作コードの摘要表を、関連するフィールドと共に示す。
【図１６】図１６は、一実施形態の下におけるバイト・レベルのファイル更新アルゴリズムのフロー図である。

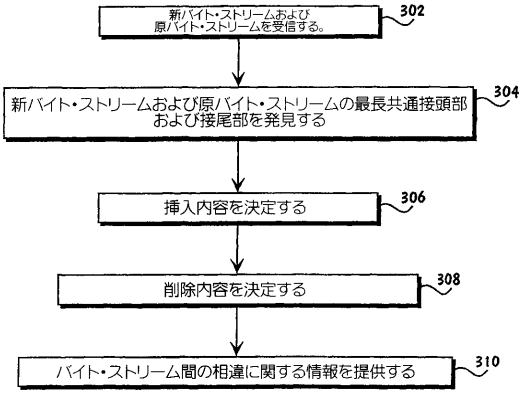
【図１】



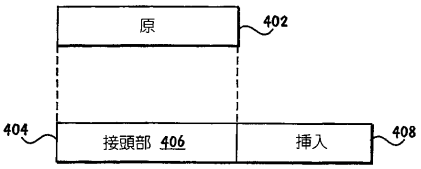
【図２】



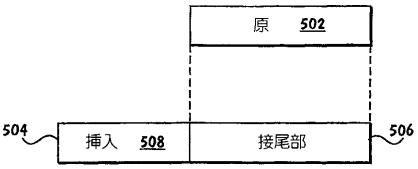
【図３】



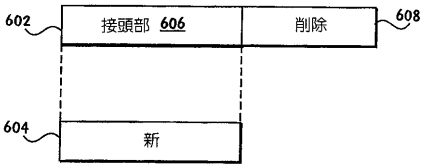
【図４】



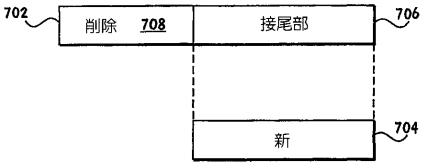
【図５】



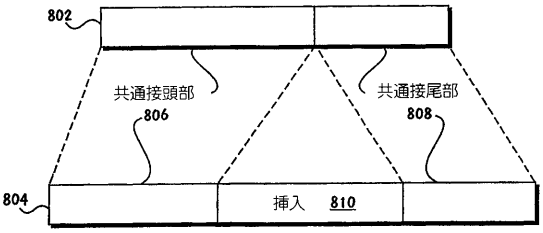
【図６】



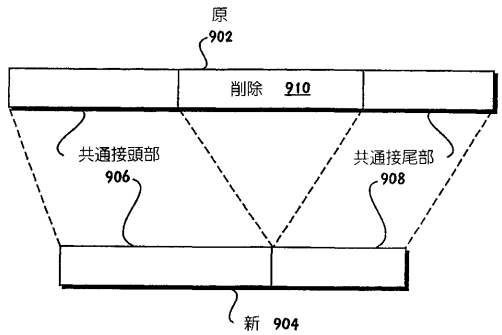
【図７】



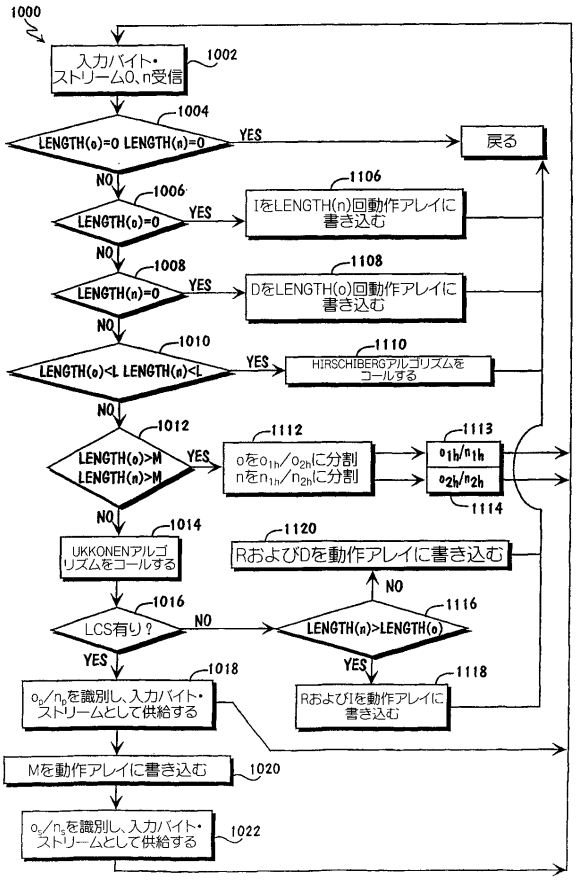
【図８】



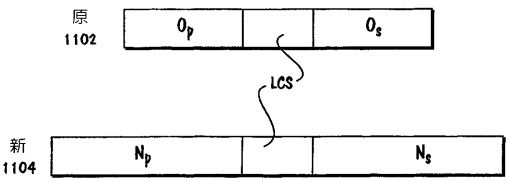
【図９】



【図１０】



【図１１】



【図１２】

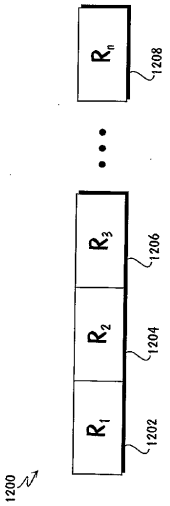
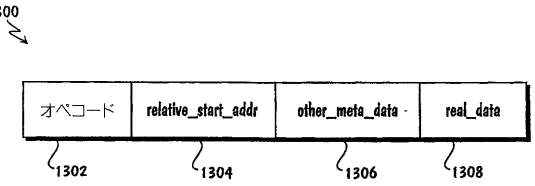


FIG. 12

【図１３】



【図１４】

1302	オペコード	挿入、削除のような動作の意味を記述する1つの文字
1304	relative_start_addr.	新/ハイト・ストリームにおける現変更と真重は変更との間のオフセット
1306a	operating_len	ある旧コンテンツと置き換わる新コンテンツの長さ
1306b	operated_len	ある新コンテンツにより置き換えられる旧コンテンツの長さ
1306c	repeat_times	パターンの出現回数
1306d	interval_len	パターンが1回繰り返す間隔の長さ
1306e	data_offset	現動作のコピーが得られるデルタ・ファイルのオフセット
1308	real_data	オペコードがADDRESS_REPEAT_SHIFTの場合、置換内容、またはバイト・シフト値

【図１５Ａ】

オペコード	relative_start_addr	operating_len	operated_len	repeat_times	interval_len	real_data
INSERTION	✓	✓				✓
DELETION	✓		✓			
REPLACEMENT	✓	✓	✓			✓
VARIABLE_LEN_REPLACEMENT	✓	✓	✓			✓
ADDRESS_REPEAT_SHIFT	✓	✓		✓	✓	✓
SAME_ADDRESS_REPEAT_SHIFT	✓			✓	✓	
ONE_BYTE_DELETION	✓					
TWO_BYTE_DELETION	✓					
THR_BYTE_DELETION	✓					
FOU_BYTE_DELETION	✓					
FIV_BYTE_DELETION	✓					
SIX_BYTE_DELETION	✓					
SEV_BYTE_DELETION	✓					
EIG_BYTE_DELETION	✓					
NIN_BYTE_DELETION	✓					
TEN_BYTE_DELETION	✓					
ONE_BYTE_INSERTION	✓					✓
TWO_BYTE_INSERTION	✓					✓
THR_BYTE_INSERTION	✓					✓
FOU_BYTE_INSERTION	✓					✓
FIV_BYTE_INSERTION	✓					✓
SIX_BYTE_INSERTION	✓					✓
SEV_BYTE_INSERTION	✓					✓
EIG_BYTE_INSERTION	✓					✓
NIN_BYTE_INSERTION	✓					✓
TEN_BYTE_INSERTION	✓					✓

【図１５Ｂ】

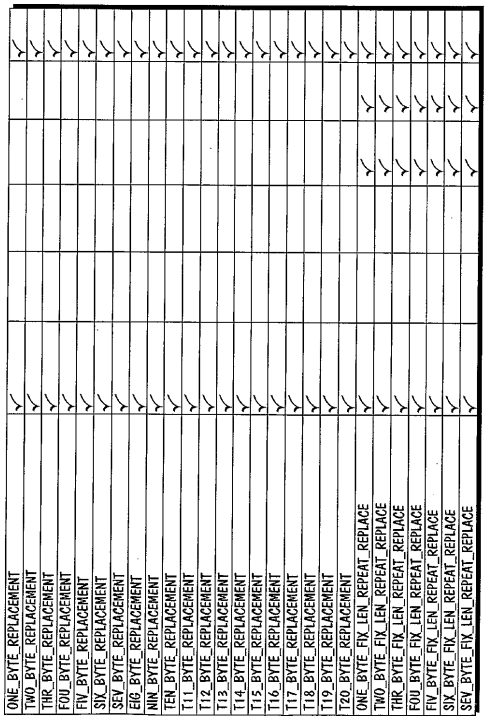


FIG. 15B

【図１５Ｃ】

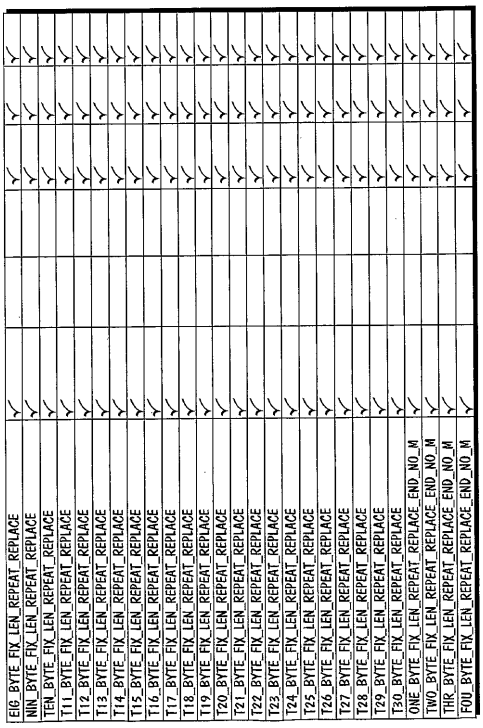
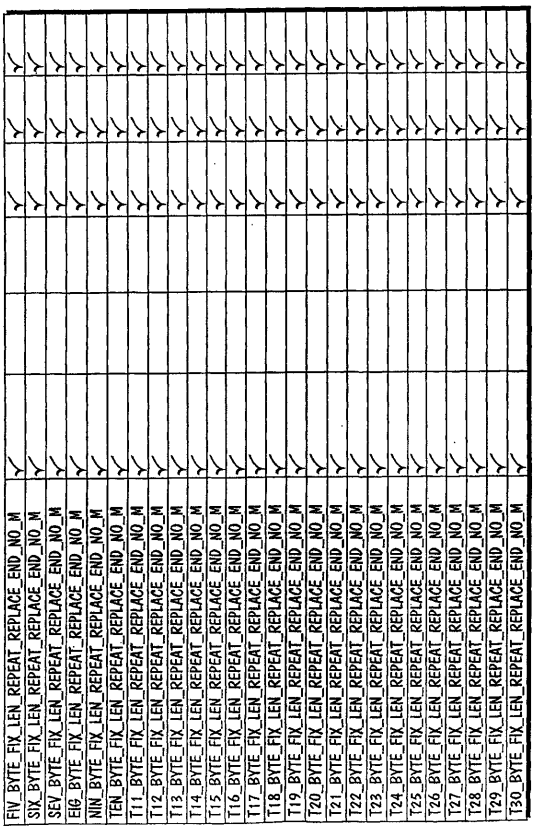


FIG. 15C

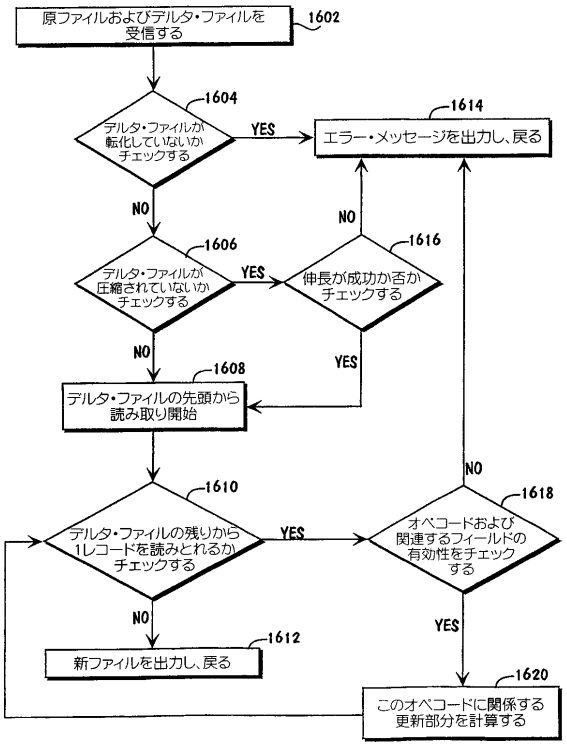
【図１５Ｄ】



【図１５Ｅ】

オペコード	relative_start_addr	delta_offset	real_data
EXACT_COPY_INDEX_TO	✓	✓	
WHOLE_REPLACEMENT			✓

【図１６】



フロントページの続き

(74)代理人 100096013
弁理士 富田 博行
(74)代理人 100120112
弁理士 中西 基晴
(72)発明者 グ，ジンシェン
アメリカ合衆国カリフォルニア州９４４０３，サン・マテオ，ウエスト・サーティーセブンス・アベニュー １８２
(72)発明者 ペン，リョーシェン
アメリカ合衆国カリフォルニア州９５００２，アルビソ，ゴールド・ストリート ２１３０，スウィート ２００

審査官 田川 泰宏

(56)参考文献 結城 浩，Ｅｎｊｏｙ Ｐｅｒｌ Ｐｒｏｇｒａｍｍｉｎｇ モジュールを活用しよう 第6回，Ｃ ＭＡＧＡＺＩＮＥ，日本，ソフトバンクパブリッシング株式会社，２００１年 7月 1日，第13巻,第7号，p.90-97
津田 伸秀，アプリケーション作成の決め手 プログラミングＴｉｐｓ大全，Ｃ ＭＡＧＡＺＩＮＥ，日本，ソフトバンクパブリッシング，１９９９年１０月 1日，第11巻,第10号，p.44-55

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 12/00