(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第3751018号

(P3751018)

(45)発行日　平成18年3月1日(2006.3.1)　(24)登録日　平成17年12月16日(2005.12.16)

(51)Int.Cl.　F I

G06F 12/00 (2006.01)　G06F 12/00 520J

G06F 12/00 531M

G06F 12/00 545B

請求項の数 26　(全 41 頁)

(21)出願番号　特願平7-501999

(86)(22)出願日　平成6年6月2日(1994.6.2)

(65)公表番号　特表平8-511367

(43)公表日　平成8年11月26日(1996.11.26)

(86)国際出願番号　PCT/US1994/006320

(87)国際公開番号　WO1994/029807

(87)国際公開日　平成6年12月22日(1994.12.22)

審査請求日　平成12年9月6日(2000.9.6)

(31)優先権主張番号　071,643

(32)優先日　平成5年6月3日(1993.6.3)

(33)優先権主張国　米国(US)

(73)特許権者

ネットワーク・アプライアンス・インコーポレイテッド

アメリカ合衆国９４０８９カリフォルニア州　サニーベイル、イースト・ジャーバ・ドライブ４９５番

(74)代理人

弁理士　青山　葆

(74)代理人

弁理士　河宮　治

(74)代理人

弁理士　石野　正弘

最終頁に続く

(54)【発明の名称】ライトエニウエアファイルシステムレイアウト

(57)【特許請求の範囲】

【請求項１】

１以上の新規のまたは変更されたファイルのためのブロックがファイルシステムに書き込まれ次に上記のファイルシステムが上記のブロックを含まない第１整合点を上記のブロックを含む第２整合点へ移動する、任意の場所に書き込み可能なファイルシステムレイアウトにおいて、不揮発性記憶手段のブロックに記憶される上記のファイルシステムの複数の読み出し専用コピーを作成する方法であって、上記のファイルシステムは、上記の不揮発性記憶手段のブロックを同定するメタデータからなり、各整合点はブロックのツリーのルートであるルートｉノードによりルートされる１組の自己整合ブロックであり、上記の方法は、

不揮発性記憶手段に上記のファイルシステムの逐次的な状態のための上記メタデータを記憶するステップと、

上記のファイルシステムの上記の逐次的な状態の各々で上記のメタデータのコピーを作成するステップと、

上記のファイルシステムのそれぞれの状態での上記のメタデータの上記のコピーの各々について、上記のメタデータにおいて同定される上記の不揮発性記憶手段の上記のブロックを、上記のファイルシステムの読出し専用コピーからなるとしてマークするステップとからなる方法。

【請求項２】

請求項１に記載された方法において、

上記のファイルシステムの読出し専用コピーからなる上記のブロックにマークする上記のステップは、上記の不揮発性記憶手段のブロックごとに多重ビットを記録する記録手段に適当なエントリを配置するステップからなる方法。

【請求項３】

請求項２に記載された方法において、

上記の記録手段に記録される多重ビットが各ブロックについての多重ビットエントリからなるブロックマップからなる方法。

【請求項４】

請求項１に記載された方法において、上記のメタデータは、上記のファイルシステムからなるブロックの階層構造のツリーのポインタからなる方法。

【請求項５】

上記のメタデータが、上記のファイルシステムのファイルを表わす構造からなることを特徴とする請求項１に記載された方法。

【請求項６】

上記のファイルシステムのファイルを表わす上記の構造が、ｉノードからなることを特徴とする請求項５に記載された方法。

【請求項７】

請求項１に記載された方法において、さらに、

上記のファイルシステムの読出し専用コピーに属するものとしてマークされている上記のブロックの上書きを禁止するステップを有する方法。

【請求項８】

請求項１に記載された方法において、上記のファイルシステムの上記の読出し専用コピーがもはや必要でないとき、上記のファイルシステムの読出し専用コピーに属するものとしてマークされている上記のブロックのマークをはずすステップを有する方法。

【請求項９】

請求項１に記載された方法において、上記のファイルシステムの読出し専用コピーに属するものとしてマークされている上記の複数のブロックが上記のファイルシステムに補助的なデータを含み、上記の方法は、さらに、

上記の補助的データを上書きすることを許可するステップと、

許可しない場合、上記のファイルシステムの読出し専用コピーに属するものとしてマークされている上記のブロックの上書きを禁止するステップと

からなる方法。

【請求項１０】

上記の請求項９に記載された方法において、上記の補助的データがアクセス時間データからなる方法。

【請求項１１】

請求項１に記載された方法において、上記のメタデータは、上記のファイルシステムのファイルを表わす構造を参照するルート構造からなり、上記のメタデータの上記のコピーは、上記のルート構造のコピーからなる方法。

【請求項１２】

請求項１１に記載された方法において、上記のルート構造がブロックのツリーのルートであるルートｉノードからなる方法。

【請求項１３】

請求項１に記載された方法において、さらに、上記のファイルシステムの１以上の上記の読出し専用コピーを使用して、上記のファイルシステムの１以上の整合点からなる上記のブロックをバックアップするステップからなる方法。

【請求項１４】

１以上の新規のまたは変更されたファイルのためのブロックがファイルシステムに書き込まれ次に上記のファイルシステムが上記のブロックを含まない第１整合点を上記のブロックを含む第２整合点へ移動する、任意の場所に書き込み可能なファイルシステムレイアウ

トにおいて整合点を生成する整合点作成方法において、上記の整合点は、ブロックのツリーのルートであるルートｉノードによりルートされる１組の自己整合ブロックであり、上記の整合点作成方法は、
上記のファイルシステムにおいて複数の変更されたブロックを指す複数のｉノードを、整合点にあるものとしてマークするステップ、
正規ファイルを不揮発性の記憶手段にフラッシュするステップ、
正規ファイルでない所定のファイルを上記の記憶手段にフラッシュするステップ、
ファイルシステム情報構造を記憶する複数のブロックの中の少なくとも一つのブロックを上記の記憶手段にフラッシュするステップ、および、
変更されたデータを含む新しいブロックを指し、かつ、上記の整合点の一部でなかったｉノードを、上述の各ステップを行うためのキューに入れるステップ
からなる整合点作成方法。
【請求項１５】
請求項１の方法において、前記の正規ファイルでない所定のファイルを上記の記憶手段にフラッシュする上記のステップは、さらに、
ブロックの割り当てを示すブロックマップファイル用のｉノードを、上記のファイルシステム用のｉノードテーブルを含むｉノードファイルに予めフラッシュするステップ、
上記のｉノードと上記のブロックマップファイルにあるすべての、変更されたデータを含む新しいブロックを指しかつ上記の整合点の一部でなかったブロックとに上記の記憶手段における空間を割当てるステップ、
上記のブロックマップファイル用の上記のｉノードを再びフラッシュするステップ、
上記のブロックマップファイルにある、上記の記憶手段上のブロックを示す複数のエントリを更新するステップ、および、
上記のブロックマップファイルと上記のｉノードファイルにある、変更されたデータを含む新しいブロックを指しかつ上記の整合点の一部でなかった全てのブロックを上記の記憶手段に書込むステップ
からなる方法。
【請求項１６】
１以上の新規のまたは変更されたファイルのためのブロックがファイルシステムに書き込まれ次に上記のファイルシステムが上記ブロックを含まない第１整合点を上記ブロックを含む第２整合点へ移動する、任意の場所に書き込み可能なファイルシステムレイアウトにおいて、不揮発性の記憶手段に記憶された上記のファイルシステムを逐次的な整合点で維持する方法であって、各々の整合点はブロックのツリーのルートであるルートｉノードによりルートされる１組の自己整合ブロックであり、上記のファイルシステムは、複数のデータブロックからなり、このデータブロックは、ファイルシステム情報構造を含む複数の正規ファイルデータのブロックと、上記ファイルシステムの上記の複数のデータブロックを参照する複数のメタデータファイルデータのブロックとからなり、上記のメタデータファイルデータは、第１整合点での上記のファイルシステムを記載するデータからなり、
上記の方法は、
正規ファイルデータとメタデータファイルの複数の変更されたブロックを記憶手段に維持するステップであって、この変更されたブロックが上記の第１の整合点から変更されたデータのブロックからなるステップと、
正規ファイルデータとメタデータファイルの上記の変更されたブロックを参照するメタデータファイルデータを汚れブロックと表わすステップであって、この汚れブロックは第２整合点に含まれるべきメタデータファイルデータのブロックからなるステップと、
メタデータファイルの上記汚れブロックにより参照されるメタデータファイルの上記の変更されたブロックを、上記の不揮発性の記憶手段のフリーのブロックにコピーするステップと、
上記の記憶手段に維持される上記のファイルシステム情報構造のコピーを変更して、メタデータファイルの上記の汚れブロックにより参照するステップと、

上記の変更されたファイルシステム情報構造を上記の記憶手段にコピーするステップと
からなる方法。
【請求項１７】
請求項１６に記載された方法において、メタファイルデータの上記のブロックはｉノードブロックの１以上のブロックとブロックマップファイルデータの１以上のブロックからなり、メタデータファイルの上記の変更されたブロックを、上記の不揮発性記憶手段のフリーのブロックにコピーする上記のステップは、
ブロックマップファイルデータの上記の１以上の変更されたブロックに割り当てられた上記の不揮発性記憶手段の上記の１以上の変更されたブロックを参照するため、ブロックマップファイルデータの上記の１以上の変更されたブロックを参照する上記のｉノードを更新するステップと、
上記の更新されたｉノードをｉノードファイルデータの上記のブロックにコピーするステップと、
ブロックマップファイルデータの上記の１以上のブロックを更新するステップと、
ブロックマップファイルデータの上記の１以上の更新されたブロックと、上記のｉノードファイルデータの上記のブロックを、上記の不揮発性記憶手段の上記の割り当てられたフリーのブロックに書き込むステップと
からなる方法。
【請求項１８】
１以上の新規のまたは変更されたファイルのためのブロックがファイルシステムに書き込まれ次に上記のファイルシステムが上記ブロックを含まない第１整合点を上記ブロックを含む第２整合点へ移動する、任意の場所に書き込み可能なファイルシステムレイアウトにおいて、不揮発性記憶手段のブロックに記憶されたデータのブロックからなる上記ファイルシステムを逐次的な整合点で維持する方法であって、上記の整合点は、ブロックのツリーのルートであるルートｉノードによりルートされるディスク上の自己整合ブロックの集合であり、
上記のファイルシステムの上記の第１整合点での上記のファイルシステムレイアウトを記述するファイルシステム情報構造である、上記の不揮発性記憶手段における第１整合点のための第１ファイルシステム情報構造を記憶するステップと、
上記の第１整合点から変更されている上記のファイルシステムのデータのブロックを、第２整合点の開始で、上記の不揮発性記憶手段のフリーのブロックに書き込むステップと、
上記の不揮発性記憶手段において、上記の第２整合点のための、上記のファイルシステムの上記の第２整合点での上記のファイルシステムレイアウトを記述する第２のファイルシステム情報構造を記憶するステップと
からなる方法。
【請求項１９】
請求項１８に記載された方法において、
上記の第１のファイルシステム情報構造を上記の不揮発性記憶手段のフリーのブロックに書き込む上記のステップは、上記の不揮発性記憶手段の第１と第２の位置に上記の第１ファイルシステム情報構造の第１と第２のコピーをそれぞれ記憶するステップからなり、
上記の第２のファイルシステム情報構造を上記の不揮発性記憶手段のフリーのブロックに書き込む上記のステップは、
上記の第１ファイルシステム情報構造の上記の第１のコピーを、上記の第２ファイルシステム情報構造の第１のコピーで上書きするステップと、
上記の第１ファイルシステム情報構造の上記の第２のコピーを、上記の第２ファイルシステム情報構造の第２のコピーで上書きするステップとからなる方法。
【請求項２０】
請求項１９に記載された方法において、
上記の不揮発性記憶手段の上記の第１と第２の位置は、上記の不揮発性記憶手段のあらかじめ決められた固定された位置である方法。

【請求項２１】
請求項１９に記載された方法において、
上記のファイルシステム情報構造の各コピーは、上記のファイルシステム情報構造の最新のバージョンを決定する手段と、上記のファイルシステム情報構造の正当性を決定する手段とを含み、上記の方法は、さらに、
システムの故障の後で、上記の不揮発性記憶手段の上記の第１と第２の位置から上記のファイルシステム情報構造の上記の第１と第２のコピーを読み出すステップと、
上記のファイルシステム情報構造の上記の第１と第２のコピーから、最新の正当なファイルシステム情報構造を決定するステップと
からなる方法。
【請求項２２】
１以上の新規のまたは変更されたファイルのためブロックがファイルシステムに書き込まれ次に上記ファイルシステムが上記ブロックを含まない第１整合点を上記ブロックを含む第２整合点へ移動する、任意の場所に書き込み可能なファイルシステムレイアウトにおいて、不揮発性の記憶手段に記憶されたデータの複数のブロックについての複数のデータを記録する方法であって、
上記の整合点は、ブロックのツリーのルートであるルートｉノードによりルートされるディスク上の自己整合ブロックの集合であり、
記憶手段のブロックごとの多重ビットを記録する所定のファイルを維持するステップと、
上記の記憶手段の上記の複数のブロックの各々のためのブロックごとの多重ビットを上記の所定のファイルに記憶するステップとからなり、
上記のブロックごとの上記の多重ビットの中の少なくとも１ビットは、アクティブなファイルシステムにおけるブロックであることを示し、上記の多重ビットの中の１以上のビットは、ファイルシステムの１以上の読み出し専用のコピーにおけるブロックであることを示す
方法。
【請求項２３】
１以上の新規のまたは変更されたファイルのためのブロックがファイルシステムに書き込まれ次に上記のファイルシステムが上記のブロックを含まない第１整合点を上記のブロックを含む第２整合点へ移動する。任意の場所に書き込み可能なファイルシステムレイアウトにおいて、不揮発性の記憶手段に記憶されたデータの複数のブロックについての複数のデータを記録する方法であって、
上記の整合点は、ブロックのツリーのルートであるルートｉノードによりルートされるディスク上の自己整合ブロックの集合であり、
上記の記憶手段のブロックごとの多重ビットを記録する所定のファイルを維持するステップと、
上記の所定のファイルにおいて、上記の記憶手段の上記の複数のブロックの各々のためブロックごとの多重ビットを記憶するステップとからなり、
上記の多重ビットのなかの少なくとも１ビットは、すでにデータを格納するために使用されていたブロックを使用する再使用可能性を示すビットである
方法。
【請求項２４】
１以上の新規のまたは変更されたファイルのためのブロックがファイルシステムに書き込まれ次に上記のファイルシステムが上記のブロックを含まない第１整合点を上記のブロックを含む第２整合点へ移動する、任意の場所に書き込み可能なファイルシステムレイアウトにおいて、不揮発性の記憶手段に記憶された複数のブロックについての複数のデータを記録する方法であって、
上記の整合点は、ブロックのツリーのルートであるルートｉノードによりルートされるディスク上の自己整合ブロックの集合であり、
上記の記憶手段のブロックごとの多重ビットを記録する所定のファイルを維持するステッ

プと、
上記の記録手段において、上記の記憶手段の上記の複数のブロックの各々についての多重ビットを記憶するステップとからなり、
上記の複数のブロックの各々について、ブロックごとの上記の多重ビットの中の１ビットは、アクティブなファイルシステムを構成するブロックであることを示し、１以上のビットはファイルシステムの１以上の読み出し専用コピーを構成するブロックであることを示し、
上記の多重ビットのなかの少なくとも１ビットは、すでにデータを格納するために使用されていたブロックを再び使用する可能性を示す方法。
【請求項２５】
少なくとも１個のプロセッサと、この少なくとも１個のプロセッサに結合される少なくとも１個のメモリとを備え、不揮発性の記憶手段に記憶されるデータの複数のブロックについて複数のデータを記録する装置であって、
１以上の新規のまたは変更されたファイルのためのブロックがファイルシステムに書き込まれ次に上記のファイルシステムが上記のブロックを含まない第１整合点を上記のブロックを含む第２整合点へ移動する、任意の場所に書き込み可能なファイルシステムレイアウトにおいて、する記録媒体であって、上記の整合点は、ブロックのツリーのルートであるルートｉノードによりルートされるディスク上の自己整合ブロックの集合であり、上記の記憶手段の上記の複数のブロックについての上記の複数のデータに対応して、上記の記憶手段のブロックごとの多重ビットを記録する記録機構を含み、上記の多重ビットのなかの少なくとも１ビットは、すでにデータを格納するために使用されていたブロックを再び使用する可能性を示す装置。
【請求項２６】
少なくとも１個のプロセッサと、この少なくとも１個のプロセッサに結合される少なくとも１個のメモリとを備え、不揮発性の記憶手段に記憶されるデータの複数のブロックについて複数のデータを記録する装置であって、
１以上の新規のまたは変更されたファイルのためのブロックがファイルシステムに書き込まれ次に上記のファイルシステムが上記のブロックを含まない第１整合点を上記のブロックを含む第２整合点へ移動する、任意の場所に書き込み可能なファイルシステムレイアウトにおいて、上記の整合点は、ブロックのツリーのルートであるルートｉノードによりルートされるディスク上の自己整合ブロックの集合であり、上記の記憶手段の上記の複数のブロックについての上記の複数のデータに対応して、上記の記憶手段のブロックごとの多重ビットを記録する記録機構を含み、上記のブロックの各々についてブロックごとの上記の多重ビットの１ビットは、アクティブなファイルシステムを構成するブロックであることを示し、１以上のビットはファイルシステムの１以上の読出し専用コピーを構成するブロックであることを示す装置。
【発明の詳細な説明】
発明の背景
１．発明の分野
本発明は、整合性のある（consistent）ファイルシステムを維持し、かつ、このファイルシステムの読出し専用コピーを作成する方法及びその装置の分野に関する。
２．背景技術
全てのファイルシステムは、システムが故障した場合でも、整合性を維持する必要がある。この目的のために、現在までにいくつかの異なる整合性維持技法が、従来技術において用いられて来た。
どのようなファイルサーバの管理においても、最も難しく時間のかかる問題点の１つは、ファイルデータのバックアップを作成することにある。伝統的な解決方法は、データをテープまたは他のオフライン媒体にコピーすることであった。あるファイルシステムでは、バックアップが完全に整合性があることを確認するために、バックアップ処理中に、ファイルサーバをオフラインにする必要がある。最近のバックアップにおける進歩により、迅

速にファイルシステムのクローン（clone）を作り（即ち、ディスク上にファイルシステムの読出し専用のコピーを作る従来技術の方法）、アクティブ（active）なファイルシステムからではなく、このクローンからバックアップを行うことができる。このタイプのファイルシステムでは、バックアップの間ファイルサーバをオンラインに保持できる。

ファイルシステムの整合性

従来技術によるファイルシステムが、Ｃｈｕｔａｎｉ等によるエピソードファイルシステム（The Episode File System）と題する論文（ＵＳＥＮＩＸ、１９９２年ウィンタ号、４３～５９頁）に開示されている。この論文は、メタデータ（即ち、ｉノードテーブル、ディレクトリ、ビットマップ、間接ブロック）等を用いるファイルシステムであるエピソードファイルシステムについて述べている。それは、スタンドアローンのファイルシステムとして、または、分散ファイルシステムとして使用できる。エピソードファイルシステムは、複数の別々のファイルシステム階層構造をサポートし、また、複数のファイルシステムについて、一括して「集合体」として言及している。特に、エピソードシステムは、除々にデータを変化させるため、各フアイルシステムのクローンを備える。エピソードシステムでは、各ロジカルファイルシステムは、「アノード（anode）テーブル」を含む。アノードテーブルは、バークレ・ファスト・ファイル・システム（Berkeley Fast File System）等のファイルシステムで用いられるｉノードテーブルに相当する。アノードテーブルは、２５２バイトの構造を有している。アノードは、エピソードファイルシステムのメタデータだけではなく、全ての使用者データを記憶するのに用いられる。アノードは、補助ファイルとディレクトリを含むファイルシステムのルートディレクトリを記述する。エピソードの各々のこの様なファイルシステムは、「ファイル集合」と呼ばれる。ファイル集合内の全データは、アノードテーブルをとおして反復し、各ファイルを順に処理することにより、探し出すことができる。エピソードシステムは、ファイルシステムの読出し専用コピー（ここではクローンという）を作成して、コピー・オン・ライト(Copy-On-Write)（ＣＯＷ）技術を用いて、アクティブなファイルシステムとデータを共有する。

エピソードシステムは、システムのクラッシュの後、ファイルシステムを回復させるのに、ロギング技術を用いている。ロギング技術は、ファイルシステムのメタデータが整合的であるかどうかを確認する。ビットマップテーブルは、ファイルシステム内の各ブロックが割り当てられているか否かについての情報を含む。また、ビットマップテーブルは、各ブロックがログ（記録）されているか否かをも示す。全ての最新のメタデータは、集合体の処理（transaction）ログを記憶しているログ「コンテナ」（container）に記憶される。ログは、ディスクブロックの循環バッファとして処理される。エピソードシステムの処理ロギングは、ファイルシステムの整合性を確認するために、元来データベースのために開発されたロギング技法を用いる。この技法は、注意深く順序づけられた書込み（write）と回復（recovery）プログラム（その中のデータベース技法によって補われる）を用いる。

ＩＢＭのＪＦＳ及びベリタス社（Veritas Corporation）のＶｘＦＳを含む他の先行技術システムは、回復処理を早めるために各形態の処理ロギングを用いているが、未だ回復処理を必要としている。

もう１つの従来技術方法は、「オーダードライト」（ordered write）技法と呼ばれ、それは、一連の関係する書込みを行っている間にシステムの故障が生じた時に損害を最小とするため、全てのディレクトリブロックに、注意深く決められた順序で書き込む。この従来技術では、発生した不整合性が無害のものであることを確認することを試みる。たとえば、少数の使用されていないブロックまたはｉノードは、割り当てられたものとしてマークされる。この技法の主な短所は、それがディスクのオーダに与える制限の為に、高性能を達成することが困難になることである。

他の従来技術システムは、「回復を伴うオーダードライト（ordered write with recovery）」技法と呼ばれる上記の第２の従来技術の改良方法である。この方法では、不整合性は、潜在的に有害となり得る。しかし、書込みのオーダは、回復プログラムで不整合が発見され処理されるように制限される。この方法の例としては、元のＵＮＩＸファイルシス

テムと、バークレイファストファイルシステム（ＦＦＳ）がある。この技法では、ディスクオーダリング（ordering）の性能上の減点を充分除く様にディスクオーダリングを減じている。もう１つの不利な点は、回復プロセスが、時間がかかる点である。それは、典型的には、ファイルシステムのサイズに比例している。そのため、たとえば５ＧＢのＦＦＳファイルシステムを回復するには、１時間以上を必要とする。

ファイルシステムのクローン

図１は、ファイル集合のクローンを作成するためのコピー・オン・ライト（ＣＯＷ）技術の使用を示すエピソードファイルシステムの従来技術の図である。アノード１１０は、セットされたＣＯＷビットを有する第１のポインタ１１０Ａからなる。ポインタ１１０Ａは直接データブロック１１４を参照する。アノード１１０は、クリアされたＣＯＷビットを有する第２のポインタ１１０Ｂからなる。アノードのポインタ１１０Ｂは、間接ブロック１１２を参照する。間接ブロック１１２は、直接データブロック１２４を参照するポインタ１１２Ａからなる。ポインタ１１２ＡのＣＯＷビットは、セットされている。間接ブロック１１２は、データブロック１２６を参照する第２のポインタ１１２Ｂからなる。ポインタ１１２ＢのＣＯＷビットはクリアされる。

クローンアノード１２０は、データブロック１１４を参照する第１のポインタ１２０Ａからなる。ポインタ１２０ＡのＣＯＷビットは、クリアされる。クローンアノード１２０の第２のポインタ１２０Ｂは間接ブロック１２２を参照する。ポインタ１２０ＢのＣＯＷビットはクリアされる。次に、間接ブロック１２２は、データブロック１２４を参照するポインタ１２２Ａからなる。ポインタ１２２ＡのＣＯＷビットはクリアされる。

図１に示すように、エピソードファイルシステムの各直接ポインタ１１０Ａ、１１２Ａ～１１２Ｂ、１２０Ａおよび１２２Ａ、および、間接ポインタ１１０Ｂと１２０Ｂは、ＣＯＷビットを含む。変更されていないブロックは、アクティブファイルシステムとクローンの両方に含まれ、ＣＯＷビットをセットする（１）。ポインタにより参照されるブロックが変更された時にＣＯＷビットはクリアされ（０）、そのため、アクティブなファイルシステムの１部であるが、クローンではない。

図１に示すように、コピー・オン・ライトブロックが変更されると、新たに、ブロックが割り当てられ更新される。次に、この新しいブロックへのポインタ内のＣＯＷフラグがセットされる。元のアノード１１０のポインタ１１０ＡのＣＯＷビットは、クリアされる。こうして、クローンアノード１２０が作られると、クローンアノード１２０のポインタ１２０Ａは、データブロック１１４も参照する。元のアノード１１０もクローンアノード１２０も共に、データブロック１１４を参照する。また、データブロック１２４も、元の間接ブロック１１２内のポインタ１１２Ａのクリアされたビットにより示される様に、変更される。このように、クローンアノードが作られるとき、間接ブロック１２２が作られる。間接ブロック１２２のポインタ１２２Ａは、データブロック１２４を参照し、ポインタ１２２ＡのＣＯＷビットがクリアされる。元のアノード１１０の間接ブロック１２２と、クローンアノード１２０の間接ブロック１２２は、共に、データブロック１２４を参照する。

図１は、単一ファイル用にクローンアノード１２０を作るためのアノードのコピー動作を示す。ただし、クローンアノードは、ファイルシステム中で変更したデータブロックを有する各ファイルに対して作成する必要がある。クローンの場合には、全てのｉノードをコピーしなければならない。ファイルシステム中ですべての変更されたファイルに対してクローンアノードを作ることにより、かなりの量のディスクスペースを消費することになる。さらに、エピソードシステムは、各ポインタが唯一のＣＯＷビットしか有していないので、多数のクローンをサポートできない。単一のＣＯＷは、１以上のクローンを区別することができない。１以上のクローンに対しては、セットできる第２のＣＯＷがない。

ファイル集合の「クローン」は、アクティブなファイル集合が読出し、また、書込める、アクティブなファイル集合の読出し専用コピーである。クローンは、ＣＯＷ技術を用いて実施され、ブロック毎に、アクティブなファイル集合とデータブロックを共有する。エピソードシステムは、ファイル集合に記憶されている各アノードをコピーしてクローン化を

実施する。初めにクローンを作るときは、アクティブなファイル集合の書込み可能アノードもクローンされたアノードと共に、同じデータブロックを指す。しかし、元のアノード中の直接ブロックおよび間接ブロックのディスクアドレスは、ＣＯＷとしてタグ（tag）される。そのため、書込み可能のファイル集合を更新してもクローンに影響を与えない。ＣＯＷブロックが変更されると、新しいブロックがファイルシステム内で割り当てられ、その変更に伴って更新される。この新しいブロックに対するポインタのＣＯＷフラグはクリアされる。

従来のエピソードシステムは、ファイルシステム内の全体のｉノードファイルと全ての間接ブロックを複製するクローンを作成する。エピソードシステムは、アクティブなファイルシステムとクローンの両方によって使われるブロックに対する全ポインタにコピー・オン・ライト（ＣＯＷ）ビットをセットできるように全てのｉノード及び間接ブロックを複製する。エピソードシステムでは、アクティブなファイルシステムに書き込まれた新しいデータが、クローンの１部で、そのために変更してはならない「古い」データを上書きしないように、これらのブロックを確認することが重要である。

この従来技術でクローンを作成するには、１ＧＢディスクで３２ＭＢまでを使用する。また、この従来技術では、ファイルシステムの８個のクローンを維持するのに、１ＧＢディスクで（４ＫＢのブロックに対して）２５６ＭＢのディスク空間を使用する。このように、先行技術では、データの損失を防止するのに多数のクローンを使用できない。そのかわりに、テープバックアップ装置のような、ディスクドライブ以外の補助記憶手段がファイルシステムのバックアップを行うのを容易にするために用いられる。クローンは、クローンが作成された瞬間に整合性のある状態でファイルシステムをバックアップするのに用いられる。ファイルシステムのクローンを作ることにより、クローンは、アクティブなファイルシステムをシャットダウンし、使用者がファイルシステムを用いることを妨げることなしに補助記憶手段に対してバックアップ可能となる。このようにして、クローンは、ファイルシステムが整合性を有するようにバックアップされる間、使用者がアクティブファイルシステムにアクセスを続けることを可能とする。次いで、バックアップが１度で完了すると、クローンは削除される。エピソードシステムは、各ポインタが１つのＣＯＷビットしか有していないので、多数のクローンをサポートできない。単一のＣＯＷビットは、１つ以上のクローンを区別できない。１より多いクローンに対してセットできる第２のＣＯＷビットはない。

ファイルシステムのクローンを作成するこの先行技術システムの１つの短所は、それがファイルシステム中の全てのｉノードと全ての間接ブロックの複製を必要とすることである。多くの小さなファイルを有するシステムでは、ｉノードだけでも、ファイルシステム中の全ディスク空間のかなりの割合を使用する。たとえば、４ＫＢのファイルで満たされた１ＧＢのファイルシステムは、３２ＭＢのｉノードを有している。このように、エピソードクローンを作成することは、かなりのディスク空間を使い、多量の（即ち、多くのメガバイトの）ディスクトラフィック（traffic）を生じる。この条件の結果として、ファイルシステムのクローンを作成するには、完成までにかなりの時間を必要とする。

この先行技術システムのもう１つの短所は、同じファイルシステムの多数のクローンを作成するのが難しいという点である。この結果、クローンは、ファイルシステムをテープでバックアップする等の短い期間の操作に対して、１時に一つ使用され、次に消去されるという傾向になっている。

発明の概要

本発明は、整合性のある状態にファイルシステムを維持し、かつ、ファイルシステムの読出し専用コピーを作成する方法を提供する。ファイルシステムを整合性のある状態に維持するために、ファイルシステムに対する変更は、きびしく制御されている。このファイルシステムは、１つの自己整合状態から別の自己整合状態へ進む。ルートｉノードによってルート（root）されるディスク上の自己整合性ブロックの集合は、整合点（ＣＰ）と呼ばれる。整合点を実施するために、ＷＡＦＬは、常に、ディスク上の割り当てられていないブロックに新しいデータを書くが、決して現存のデータを上書きすることはない。ファイ

ルシステム情報（fsinfo）ブロックがｉノードファイルのための新しいルートｉノードをその中に書込むことにより更新されるとき、新しい整合点が生じる。このように、ルートｉノードが更新されない限り、ディスク上に存在するファイルシステムの状態は変化しない。

また、本発明は、ファイルシステムの仮想読出し専用コピーであるスナップショット（snapshot）をも作成する。初めて作成される時は、スナップショットは、ディスク空間を使用しない。それは、同じファイルシステムに対して、多くの異なるスナップショットが作成できるように設計される。従来のファイルシステムでは、全体のｉノードファイルと全ての間接ブロックを複製することにより、クローンを作るが、これと異なり、本発明は、ｉノードファイルを記述するｉノードのみを複製する。こうして、スナップショットに必要な実際のディスク空間は複製ｉノードを格納するのに用いる１２８バイトにすぎない。スナップショットに必要な本発明の１２８バイトは、従来技術のクローンに用いられる数メガバイトよりかなり少ない。

本発明は、アクティブなファイルシステムに書き込まれた新しいデータが、スナップショット（１以上）の１部である「古い」データを上書きするのを防止する。古いデータは、それがスナップショットの１部である限り、上書きされないことが必要である。これは、多重ビットのフリーブロックのマップを用いることにより達成される。大部分の従来技術のファイルシステムは、ブロックが割り当てられているか否かを示すために、ブロック毎に単一ビットを有するフリーブロックマップを用いる。本発明は、３２ビットエントリを有するブロックマップを用いる。第１ビットは、ブロックが、アクティブなファイルシステムにより使用されるか否かを示し、残りの２０ビットは、２０個のスナップショットまで用いられるが、３１ビットの内のいくつかのビットは、他の目的に用いてもよい。

【図面の簡単な説明】

図１は、ファイルシステムの従来技術の「クローン」を示すブロック図である。
図２は、汚れバッファを有するｉノードのリストを示す図である。
図３は、ＷＡＦＬのオン・ディスクｉノードを示す図である。
図４Ａ～図４Ｄは、異なるレベルの間接度（indirection）を有するＷＡＦＬのオン・ディスクｉノードを示す図である。
図５は、整合点を生成する方法を示す流れ図である。
図６は、整合点を生成する図５のステップ５３０を示す流れ図である。
図７は、スナップショットを作成するための図５のステップ５３０を示す流れ図である。
図８は、本発明によるＷＡＦＬのメモリ内（incore）ｉノードを示す図である。
図９Ａ～図９Ｄは、本発明による、異なるレベルの間接度を有するＷＡＦＬのメモリ内のｉノードを示す図である。
図１０は、ファイル用のメモリ内ｉノード１０２０を示す図である。
図１１Ａ～図１１Ｄは、本発明によるブロックマップファイルを示す図である。
図１２は、本発明によるｉノードファイルを示す図である。
図１３Ａ～図１３Ｂは、本発明によるｉノードマップファイルを示す図である。
図１４は、本発明によるディレクトリを示す図である。
図１５は、ファイルシステム情報構造を示す図である。
図１６は、ＷＡＦＬファイルシステムを示す図である。
図１７Ａ～図１７Ｌは、整合点の生成を示す図である。
図１８Ａ～図１８Ｃは、スナップショットの生成を示す図である。
図１９は、ｉノードファイルへの変更を示す図である。
図２０は、ファイルシステムを整合状態に維持するために用いられるファイルシステム情報ブロックを示す図である。
図２１Ａ～図２１Ｆは、スナップショットの生成を示す詳細な図である。
図２２は、それぞれ共通ファイルを参照する３つのスナップショットを有するアクティブなＷＡＦＬファイルシステムを示す図である。
図２３Ａ～図２３Ｂは、ａタイムの更新を示す図である。

発明の詳細な説明

ファイルシステムにおける読出し専用コピー作成システムについて述べる。本発明に関する記述の全体を通して、より詳しく説明するために、ディスク（複数）の番号や性質、ディスクブロックサイズ等に関して、以下に詳細に説明する。しかし、明らかに、このような詳細な説明は無くても、当該技術に関する熟練者であれば、本発明を実施することは可能である。たとえば、本発明が不明瞭になるのを避けるために、公知の特徴については詳細には説明していない。

ライトエニウエアファイルシステムレイアウト

本発明は、ライトエニウエア(writeanywhere)（どこでも書込みできる）ファイルシステムレイアウト（ＷＡＦＬ）を用いる。このディスクフォーマットシステムは、ブロックベースつまり、断片的部分を含まない４ＫＢのブロック）であり、ファイルを表すのにｉノードを用い、単純に特別にフォーマットされたファイルからなるディレクトリを含む。ＷＡＦＬは、ファイルシステムのレイアウトを記述するメタデータを保存するためのファイル（複数）を用いる。ＷＡＦＬのメタデータファイルは、ｉノードファイル、ブロックマップ（blkmap）ファイルおよびｉノードマップ（inomap）ファイルを含む。ｉノードファイルは、フィルシステム用のｉノードテーブルを含む。ブロックマップファイルは、どのディスクブロックが割り当てられているかを示す。ｉノードマップは、どのｉノードが割り当てられているかを示す。オンディスク及びメモリ内（incore）のＷＡＦＬのｉノードについては、後に述べる。

オンディスクＷＡＦＬｉノード

ＷＡＦＬのｉノードは従来のｉノードとは異なる。オンディスク（ディスク上）ＷＡＦＬｉノードは、それぞれ、同レベルの間接度（indirection）を有する１６個のブロックを指す。ブロック番号は４ビット長である。一つのｉノード中の同レベルの間接度を有するブロック番号を使用することで、ファイルの再帰プロセスをより容易にしている。図３は、オンディスクｉノード３１０を表すブロック図である。オンディスクｉノード３１０は、標準ｉノード情報３１０Ａと同レベルの間接度を有する１６個のブロック番号エントリ３１０Ｂとからなる。ｉノード情報３１０Ａは、ファイルの所有者、許可、ファイルサイズ、アクセスタイム等に関する情報からなるが、これらは当業者にとって公知のことである。オンディスクｉノード３１０は、異なるレベルの間接度を有する複数個のブロック番号からなる従来のｉノードとは異なる。一つのｉノード３１０に含まれるすべてのブロック番号エントリ３１０Ｂの間接度を同レベルに保つことによって、ファイルシステムを単純にできる。

６４バイト以下の小さいファイルに関しては、１６個のブロック番号ではなく、ｉノード自体に直接データが保存される。図４Ａは、図３に示されたｉノード３１０に類似した、レベル０のｉノード４１０を表す。しかし、ｉノード４１０は１６個のブロック番号３１０Ｂの代わりに、６４バイトのデータ４１０Ｂからなる。したがって、非常に小さいファイルについては、ディスクブロックを割り当てる必要が無い。

６４ＫＢ以下の大きさのファイルに関しては、１６個のブロック番号のそれぞれが直接４ＫＢのデータブロックを参照する。図４Ｂは、１６ブロック番号３１０Ｂからなるレベル１のｉノード３１０を表す。ブロック番号エントリ０～１５は、対応する４ＫＢのデータブロック４２０Ａ～４２０Ｃを指す。

６４ＫＢ以上かつ６４ＭＢ以下の大きさのファイルに関しては、１６ブロック番号のそれぞれが単独間接ブロックを参照する。この場合、４ＫＢの単独間接ブロックは、４ＫＢのデータブロックを参照する１０２４個のブロック番号からなる。図４Ｃは、１６個の単独間接ブロック４３０Ａ～４３０Ｃを参照する１６個のブロック番号３１０Ｂからなるレベル２のｉノード３１０を表す。図４Ｃに示すように、ブロック番号エントリ０は、単独間接ブロック４３０Ａを指す。単独間接ブロック４３０Ａは、４ＫＢのデータブロック４４０Ａ～４４０Ｃを参照する１０２４個のブロック番号からなる。同様に、各単独間接ブロック４３０Ｂ～４３０Ｃは、１０２４個までのデータブロックをアドレスできる。

６４ＭＢ以上の大きさのファイルに関しては、ｉノード中の１６個のブロック番号が、２

重間接ブロックを参照する。それぞれ４ＫＢの２重間接ブロックは、対応する単独間接ブロックを指す１０２４個のブロック番号からなる。この場合、各単独間接ブロックは、４ＫＢデータブロックを指す１０２４個のブロック番号からなる。このようにして、６４ＧＢまでのアドレスが可能である。図４Ｄは、ブロック番号エントリ０、１および１５が、２重間接ブロック４７０Ａ，４７０Ｂおよび４７０Ｃをそれぞれ参照する１６個のブロック番号３１０Ｂからなるレベル３のｉノード３１０を表す。２重間接ブロック４７０Ａは、単独間接ブロック４８０Ａ～４８０Ｂを指す１０２４個のブロック番号エントリ０～１０２３からなる。各単独間接ブロック４８０Ａ～４８０Ｂは、この場合、１０２４個のデータブロックを参照する。図４Ｄに示すように、単独間接ブロック４８０Ａは１０２４個のデータブロック４９０Ａ～４９０Ｃを参照し、単独間接ブロック４８０Ｂは、１０２４個のデータブロック４９０Ｃ～４９０Ｆを参照する。

メモリ内のＷＡＦＬｉノード

図８は、メモリ内(in core)のＷＡＬＦｉノード８２０を表すブロック図である。メモリ内のｉノード８２０は、（図３に示す）オンディスクｉノード３１０の情報、ＷＡＦＬバッファデータ構造８２０Ａおよび１６個のバッファポインタ８２０Ｂからなる。メモリ内のＷＡＦＬｉノードは３００バイトの大きさを有する。ＷＡＦＬバッファは、ディスクに保存されている４ＫＢブロックに相当するメモリ内の４ＫＢである。メモリ内のｉノード８２０は、異なる間接度レベルを有するバッファを参照する従来のｉノードとは異なる。各のメモリ内のＷＡＦＬｉノード８２０は、同一の間接度レベルを有する１６個のバッファを指す。バッファポインタは、４バイト長である。一つのｉノード８２０に含まれる全バッファポインタ８２０Ｂを同一の間接度レベルに保つことにより、ファイルシステムを簡単にできる。メモリ内のｉノード８２０は、また、リンクリスト用のポインタ、整合点中(in-consistency)（ＩＮ＿ＣＰ）フラグおよび汚れフラグからなるメモリ内情報８２０Ｃを含む。汚れフラグは、ｉノード自体が変更されていること、または、汚れフラグが参照しているバッファが変更されたものであることを示している。ＩＮ＿ＣＰフラグはｉノードが整合点（後で説明する）の中にあることをマークするために用いられる。リンクリスト用ポインタについて、以下に述べる。

図１０は、ＷＡＦＬｉノード１０１０が参照するファイルを表す。このファイルは、間接ＷＡＦＬバッファ１０２０～１０２４および直接ＷＡＦＬバッファ１０３０～１０３４からなる。ＷＡＦＬのメモリ内のｉノード１０１０は、標準ｉノード情報１０１０Ａ（汚れバッファの計数値を含む）、ＷＡＦＬバッファデータ構造１０１０Ｂ、１６個のバッファポインタ１０１０Ｃおよび標準オンディスクｉノード１０１０Ｄからなる。メモリ内のＷＡＦＬｉノード１０１０は、およそ３００バイトの大きさを有する。オンディスクｉノードは１２８バイトの大きさである。ＷＡＦＬバッファデータ構造１０１０Ｂは、１６個のバッファポインタ１０１０Ｃを参照する第１ポインタと、オンディスクブロック番号１０１０Ｄを参照する第２ポインタとからなる。

各ｉノード１０１０は、参照する汚れバッファの計数値を持っている。ｉノード１０１０は、汚れｉノードのリストおよび／または汚れバッファを有するｉノードのリストに入れられる。一つのｉノードが参照する汚れバッファの全てがディスクへの書込みが予定されているか、または、実際に書込まれている場合には、ｉノード１０１０に対する汚れバッファの計数値が０にセットされる。そして、ｉノード１０１０は、そのフラグ（つまり、汚れバッファがないこと）に従って、ふたたびキューされる。このｉノード１０１０は、次のｉノードが処理される前に、クリアされる。さらに、ｉノードが整合点中にあることを示すｉノードのフラグがクリアされる。ｉノード１０１０自体が整合点においてディスクに書き込まれる。

ＷＡＦＬバッファ構造は、間接ＷＡＦＬバッファ１０２０によって表される。ＷＡＦＬバッファ１０２０は、ＷＡＦＬバッファデータ構造１０２０Ａと、１０２４個のＷＡＦＬバッファポインタからなる４ＫＢバッファ１０２０Ｂと、１０２４個のオンディスクブロック番号からなる４ＫＢバッファ１０２０Ｃからなる。ＷＡＦＬバッファデータ構造は５６バイトの大きさを有するとともに、２つのポインタからなる。ＷＡＦＬバッファデータ構

造１０２０Ａのポインタの一つは、４ＫＢバッファ１０２０Ｂを参照し、もう一つのポインタはバッファ１０２０Ｃを参照する。図１０において、ＷＡＦＬｉノード１０１０の１６個のバッファポインタ１０１０Ｃは、１６個の単独間接ＷＡＦＬバッファ１０２０～１０２４を指す。一方、ＷＡＦＬバッファ１０２０は、１０２４個の直接ＷＡＦＬバッファ構造１０３０～１０３４を参照する。ＷＡＦＬバッファ１０３０は、直接ＷＡＦＬバッファを代表している。

直接ＷＡＦＬバッファ１０３０は、ＷＡＦＬバッファデータ構造１０３０Ａと、対応するオンディスク４ＫＢデータブロックのキャッシュバージョンを含む４ＫＢ直接バッファ１０３０Ｂとからなる。直接ＷＡＦＬバッファ１０３０は、間接ＷＡＦＬバッファ１０２０のような４ＫＢバッファを有しない。ＷＡＦＬバッファデータ構造１０３０Ａの第２バッファポインタは、０にされ、第２の４ＫＢバッファを指さない。これにより、メモリの非効率な使用を防止して、メモリ空間が使われていないバッファに割り当てられないようにする。

図１０に示すように、ＷＡＦＬファイルシステムにおいて、ＷＡＦＬのメモリ内のｉノード構造１０１０は、ＷＡＦＬバッファ構造１０２０～１０２４および１０３０～１０３４のツリーを参照する。これは、間接ブロックおよび／または直接ブロックを指すブロック番号からなる標準ｉノードによって参照される、ディスク上のブロックツリーに類似している。このように、ＷＡＦＬｉノード１０１０は、１６個のボリュームブロック番号からなるオンディスクｉノード１０１０ＤおよびＷＡＦＬバッファ構造１０２０～１０２４及び１０３０～１０３４を指す１６個のバッファポインタ１０１０Ｃからなる。ＷＡＦＬバッファ１０３０～１０３４は、ボリュームブロック番号によって参照されたブロックのキャッシュ成分を含む。

ＷＡＦＬインコード（in-code）ｉノード１０１０は１６個のバッファポインタ１０１０Ｃを含む。次に、１６個のバッファポインタ１０１０Ｃは、ＷＡＦＬバッファ１０２０～１０２４および１０３０～１０３４のツリーのルートであるＷＡＦＬバッファ構造１０１０Ｂによって参照される。このように、各ＷＡＦＬｉノード１０１０は、ｉノード１０１０中の１６個のバッファポインタ１０１０Ｃを指すＷＡＦＬバッファ構造１０１０Ｂを含む。これにより、再帰的に実行されるバッファのツリーを処理するアルゴリズムが容易になる。仮にｉノード１０１０中の１６個のバッファポインタ１０１０ＣがＷＡＦＬバッファ構造１０１０Ｂによって表されないとしたら、バッファ１０２０～１０２４及び１０３０～１０３４のツリー全体を操作するための再帰的アルゴリズムの実施が困難になる。

図９Ａ～図９Ｄは、異なるレベルの間接度を有するｉノードを表す。図９Ａ～図９Ｄに、間接度を示すために、簡単化された間接及び直接のＷＡＦＬバッファが表される。しかし、図９のＷＡＦＬバッファは、対応する図１０の間接及び直接のバッファを表すと理解されるべきである。６４バイト以下の小さなファイルに関しては、データは、１６個のバッファポインタの代わりにｉノード中に直接に保存される。図９Ａは、図８に示されるｉノード８２０が１６個のバッファポインタ８２０Ｂの代わりに６４バイトのデータ９２０Ｂからなるという点を除いて、レベル０のｉノード８２０はこのｉノード８２０と同一であることを表す。それゆえ、追加のバッファが、非常に小さいファイルに対して割り当てられることがない。

６４ＫＢより小さいファイルに関しては、１６個のバッファポインタの各々が、直接４ＫＢの直接ＷＡＦＬバッファを参照する。図９Ｂは１６個のバッファポインタ８２０Ｂからなるレベル１のｉノード８２０を表す。バッファポインタＰＴＲ０～ＰＴＲ１５は、対応する４ＫＢの直接ＷＡＦＬバッファ９２２Ａ～９２２Ｃを指す。

６４ＫＢ以上かつ６４ＭＢ以下の大きさのファイルに関しては、１６個のバッファポインタの各々が、単独間接ＷＡＦＬバッファを参照する。この場合、各４ＫＢの単独間接ＷＡＦＬバッファは、４ＫＢ直接ＷＡＦＬバッファを参照する１０２４個のバッファポインタからなる。図９Ｃは、１６個の単独間接ＷＡＦＬバッファ９３０Ａ～９３０Ｃを参照する１６個のバッファポインタ８２０Ｂからなるレベル２のｉノード８２０を表す。図９Ｃに示すように、バッファポインタＰＴＲ０は、単独間接ＷＡＦＬバッファ９３０Ａを指す。

単独間接ＷＡＦＬバッファ９３０Ａは、４ＫＢ直接ＷＡＦＬバッファ９４０Ａ～９４０Ｃを参照する１０２４個のポインタからなる。同様に、単独間接ＷＡＦＬバッファ９３０Ｂ～９３０Ｃは、それぞれ、１０２４個までの直接ＷＡＦＬバッファをアドレスできる。

６４ＭＢ以上の大きなファイルに関しては、ｉノードの１６個のバッファポインタが２重間接ＷＡＦＬバッファを参照する。各４ＫＢ２重間接ＷＡＦＬバッファは、対応する単独間接ＷＡＦＬバッファを指す１０２４個のポインタからなる。この場合、各単独間接ＷＡＦＬバッファは、４ＫＢ直接ＷＡＦＬバッファを指す１０２４のポインタからなる。このように、６４ＧＢまでがアドレスできる。図９Ｄは、ポインタＰＴＲ０、ＰＴＲ１およびＰＴＲ１５が、２重間接ＷＡＦＬバッファ９７０Ａ、９７０Ｂおよび９７０Ｃをそれぞれ参照する１６個のポインタ８２０Ｂを有するレベル３のｉノード８２０を表す。２重間接ＷＡＦＬバッファ９７０Ａは、１０２４個の単独間接ＷＡＦＬバッファ９８０Ａ～９８０Ｂを指す１０２４個のポインタよりなる。次に、各単独間接ＷＡＦＬバッファ９８０Ａ～９８０Ｂは、１０２４個の直接ＷＡＦＬバッファを参照する。図９Ｄに示すように、単独間接ＷＡＦＬバッファ９８０Ａは、１０２４個の直接ＷＡＦＬバッファ９９０Ａ～９９０Ｃを参照する。そして、単独間接ＷＡＦＬバッファ９８０Ｂは、１０２４個の直接ＷＡＦＬバッファ９９０Ｄ～９９０Ｆを参照する。

ディレクトリ

ＷＡＦＬシステムにおけるディレクトリは、二つの区分に分かれる４ＫＢブロックに保存される。図１４は、本発明におけるディレクトリブロック１４１０を説明する。各ディレクトリブロック１４１０は、固定長ディレクトリエントリ構造１４１２～１４１４からなる第１区分１４１０Ａと、実際のディレクトリ名１４１６～１４１８を含む第２区分１４１０Ｂとからなる。各ディレクトリエントリは、ファイルＩＤ（id）および作成（generation）を含む。この情報は、当業界で周知であるので、図１４には示されない。ディレクトリブロックの第１区分１４１０Ａ中の各エントリ１４１２～１４１４は、第２区分１４１０Ｂ中に含まれているその名を指すポインタを有している。さらに、各エントリ１４１２～１４１４は、第２区分中に含まれているその名に依存するハッシュ値を含んでおり、ハッシュがヒット（ハッシュマッチ）した時に限りその名前が検査される。例えば、第１区分１４１０Ａのエントリ１４１２は、ハッシュ値１４１２Ａ及びポインタ１４１２Ｂからなる。ハッシュ値１４１２Ａは、第２区分１４１０Ｂの可変長エントリ１４１６に保存されているディレクトリ名「ＤＩＲＥＣＴＯＲＹ＿ＡＢＣ」に依存する値である。エントリ１４１０のポインタ１４１２Ｂは、第２区分１４１０Ｂの可変長エントリ１４１６を指す。第１区分１４１０Ａ中の固定長ディレクトリエントリ１４１２～１４１４を用いることによって、名前ルックアップ処理が迅速になる。ディレクトリブロック１４１０中の、次のエントリを見付け出すための操作は不要である。さらに、第１区分１４１０Ａ中のエントリ１４１２～１４１４を小さく保持することにより、ラインフィルデータキャッシュを有するファイルシステムのヒット速度が改善される。

メタデータ

ＷＡＦＬは、メタデータとして知られている、ファイル中にファイルシステムを記述する情報を保持している。メタデータは、ｉノードファイル、ｉノードマップファイルおよびブロックマップファイルからなる。ＷＡＦＬは、メタデータを、ディスクの任意の場所に（anywhere）書込まれる可能性のあるファイルに保存する。全てのＷＡＦＬメタデータは、ファイルの中に保持されているので、正にファイルシステム中の他のファイルと同様に、任意の場所に書込むことができる。

第１のメタデータファイルは、ファイルシステムにおける他の全てのファイルを記述するｉノードを含む「ｉノードファイル」である。図１２は、ｉノードファイル１２１０を表した図である。ｉノードファイル１２１０は、ディスクの定められた場所に「ｉノードテーブル」を書き込む従来のシステムとは異なり、ディスクの任意の場所に書込むことが出来る。ｉノードファイル１２１０は、ｉノードファイル１２１０自体を除いて、ファイルシステム中の各ファイルに対するｉノード１２１０Ａ～１２１０Ｆを含む。ｉノードファイル１２１０は、「ルートｉノード」として参照されるｉノードによって指される。ルー

トｉノードは、後述するファイルシステム情報（ｆｓｉｎｆｏ）ブロックとして参照されるディスク上の定められた場所に保持される。ｉノードファイル１２１０自体は、ディスク上の４ＫＢブロック（またはメモリ中の４ＫＢバッファ）中に保存される。図１２は、ｉノード１２１０Ａ～１２１０Ｃが４ＫＢバッファ１２２０中に保存されていることを表している。１２８バイトの大きさを有するオンディスクｉノードに関しては、４ＫＢバッファ（またはブロック）は、３２個のｉノードよりなる。メモリ内のｉノードファイル１２１０は、ＷＡＦＬバッファ１２２０からなる。一つのメモリ内のｉノード（つまり１２１０Ａ）がロードされた時には、メモリ内ｉノード１２１０Ａのオンディスクｉノード部分が、ｉノードファイル１２１０のバッファ１２２０のためにコピーされる。バッファデータ自体は、ディスクからロードされる。逆の順番で、ディスクにデータが書き込まれる。オンディスクｉノードのコピーであるメモリ内のｉノード１２１０Ａが、ｉノードファイル１２１０の対応バッファ１２２０にコピーされる。そして、ｉノードファイル１２１０が、書き込み割り当てされると共に、ｉノードファイル１２１０のバッファ１２２０中にデータが保存される。

他のメタデータファイルは、「ブロックマップ」（ｂｌｋｍａｐ）ファイルである。図１１Ａは、一つのブロックマップファイル１１１０を表す。ブロックマップファイル１１１０は、ディスクシステム中の各４ＫＢブロックに対する３２ビットエントリ１１１０Ａ～１１１０Ｃを含む。これは、また、フリーブロックマップファイルとして用いられる。ブロックマップファイル１１１０は、一つのディスクブロックが割り当てられたか否かを示す。図１１Ｂは、ブロックマップファイル１１１０（図１１Ａに示す）の一ブロックエントリ１１１０Ａの図である。図１１Ｂに示すように、エントリ１１１０Ａは、３２ビット（ビット０～ビット３１）からなる。エントリ１１１０Ａのビット０は、アクティブなファイルシステムビット（ＦＳビット）である。エントリ１１１０ＡのＦＳビットは、対応するブロックがアクティブなファイルシステムの一部であるか否かを示す。エントリ１１１０Ａのビット１～２０は、ブロックが対応するスナップショット１～２０の一部であるか否かを示すビットである。次の上位１０ビット（ビット２１～ビット３０）は予約されている。ビット３１は、エントリ１１１０Ａの整合点ビット（ＣＰビット）である。

ブロック用の３２ビットエントリ１１１０Ａ中の全てのビット（ビット０～ビット３１）がクリアになっている（０の値に設定される）時には、そのファイルシステムにおいて、一つのブロックがフリーブロックとして有効である。図１１Ｃは、ディスクブロックがフリーであることを示す図１１Ａのエントリ１１１０Ａを表す。このように、ビット０～３１の全てがゼロの値を有する時は、ブロックマップファイル１１１０のエントリ１１１０Ａによって参照されたブロックはフリーである。図１１Ｄは、アクティブなファイルシステムにおける割り当てられたブロックの全てを示す図１１Ａのエントリ１１１０Ａを表す。ＦＳビットとして参照されるビット０が１の値に設定されている時は、ブロックマップファイル１１１０のエントリ１１１０Ａは、アクティブなファイルシステムの一部であるブロックを示している。ビット１～ビット２０は、もし在る場合には、ブロックを参照する対応スナップショットを示すために用いる。スナップショットについては、後で詳細に述べる。仮にビット０が０の値に設定されても、必ずしもブロックが割り当て可能であること示すものではない。ブロックを割り当てるためには、全スナップショットビットがゼロでなければならない。しかし、整合点の一部として記録すべくメモリビット３１にロードされた時には、エントリ１１１０Ａのビット３１は、常にディスク上のビット０と同じ状態を有する。

その他のメタデータファイルは、フリーｉノードマップとして利用される「ｉノードマップ」（ｉｎｏｍａｐ）である。図１３Ａは、一つのｉノードマップファイル１３１０を表す。ｉノードマップファイル１３１０は、図１２に示されるｉノードファイル１２１０中の各ブロック用の８ビットエントリ１３１０Ａ～１３１０Ｃを含む。各エントリ１３１０Ａ～１３１０Ｃは、ｉノードファイル１２１０の対応ブロック中の割り当てられたｉノードの計数値である。図１３Ａは、エントリ１３１０Ａ～１３１０Ｃ中の、それぞれの値３２、５および０を示す。ｉノードファイル１２１０は、まだ、ブロック中のどのｉノード

がフリーであるかを調べなければならない。しかし、多数のランダムブロックをディスクからメモリにロードする必要はない。ｉノードファイル１２１０の各４ＫＢブロック１２２０は、３２個のｉノードを保持する。それゆえに、ｉノードファイル１２１０の各ブロック用の８ビットのｉノードマップエントリ１３１０Ａ～１３１０Ｃは、０から３２の範囲の値を有することができる。ｉノードファイル１２１０のブロック１２２０が、使用中のｉノードを有しない時は、そのｉノードマップファイル１３１０中のエントリ１３１０Ａ～１３１０Ｃはゼロである。ブロック１２２０のｉノードファイル１２１０中の全てのｉノードが使用中である時には、ｉノードマップファイル１３１０のエントリ１３１０Ａ～１３１０Ｃは、値３２を有する。

図１３Ｂは、ｉノードファイル１３４０の４ＫＢブロック１３４０Ａ～１３４０Ｃを参照するｉノードマップファイル１３５０を表す。例えば、ｉノードファイル１３４０は、３個の４ＫＢブロック１３４０Ａ～１３４０Ｃ中に、３７個のｉノードを保存する。ｉノードファイル１３４０のブロック１３４０Ａ～１３４０Ｃは、それぞれ３２、５および０の使用されたｉノードを含む。ブロックマップファイル１３５０のエントリ１３５０Ａ～１３５０Ｃは、それぞれ、ｉノードファイル１３４０のブロック１３４０Ａ～１３４０Ｃを参照する。このように、ｉノードマップファイルのエントリ１３５０Ａ～１３５０Ｃは、ｉノードファイル１３４０のブロック１３４０Ａ～１３４０Ｃ用として、それぞれ、３２、５および０の値を有している。一方、ｉノードマップファイルのエントリ１３５０Ａ～１３５０Ｃは、それぞれ、ｉノードファイル１３４０のブロック１３４０Ａ～１３４０Ｃの０、２７および３２のフリーのｉノードを示している。

図１３に示すように、計数値の代わりにｉノードマップファイル１３１０のエントリ１３１０Ａ～１３１０Ｃ用のビットマップを用いることは、不利である。何故ならば、ｉノードファイル１２１０（図１２に示す）のブロック１２２０用のエントリ１３１０Ａ～１３１０Ｃ当たり、１バイトでは無く４バイトが必要であるからである。ｉノードファイル１２１０のブロック１２２０中のフリーｉノードは、ｉノードマップファイル１３１０中に示す必要がない。なぜならば、ｉノード自体がそのような情報を含んでいるからである。

図１５は、ファイルシステム情報（fsinfo）構造１５１０を表す。ファイルシステムのルートｉノード１５１０Ｂは、ディスク上の定められた場所に保持されて、ファイルシステムのブート中に位置が分かるようにされている。ファイルシステム情報ブロックは、メタデータではないが、ＷＡＦＬシステムの一部である。ルートｉノード１５１０Ｂは、ｉノード１２１０を参照するｉノードである。それは、ファイルシステム中のブロックの番号、ファイルシステムの作成時間等を含む情報１５１０Ａを含むファイルシステム情報構造の一部である。この種々の情報１５１０Ａは、さらに、チェックサム１５１０Ｃ（後に述べる）からなる。別の実施例において、ルートｉノード１５１０Ｂ自体を除いて、この情報１５１０Ａは、メタデータファイル中に保持することができる。ファイルシステム情報構造１５１０の２つの同一コピーは、ディスク上の定められた場所に保持される。

図１６は、ディスク上で整合状態にあるＷＡＦＬファイルシステム１６７０を表し、このファイルシステムは、二つのファイルシステム情報ブロック１６１０および１６１２、ｉノードファイル１６２０、ブロックマップファイル１６３０、ｉノードマップファイル１６４０、ルートディレクトリ１６５０および典型的なファイル（またはディレクトリ）１６６０からなる。ｉノードファイル１６２０は、ファイルシステム１６７０中の他のファイル１６３０～１６６０を参照する複数のｉノード１６２０Ａ～１６２０Ｄからなる。ｉノードファイル１６２０のｉノード１６２０Ａは、ブロックマップファイル１６３０を参照する。ｉノード１６２０Ｂは、ｉノードマップファイル１６４０を参照する。ｉノード１６２０Ｃは、ルートディレクトリ１６５０を参照する。ｉノード１６２０Ｄは、典型的なファイル（またはディレクトリ）１６６０を参照する。このように、ｉノードファイルは、ファイルシステム情報ブロック１６１０および１６１２を除くファイルシステム１６７０中の全てのファイル１６３０～１６６０を指す。ファイルシステム情報ブロック１６１０および１６１２は、それぞれ、ｉノードファイル１６２０のｉノードのコピー１６１０Ｂおよび１６１２Ｂを含む。ファイルシステム情報ブロック１６１０及び１６１２のル

ートｉノード１６１０Ｂおよび１６１２Ｂは、全メタデータ１６３０～１６４０を含むファイルシステム１６７０中の残りのファイル１６３０～１６６０を記述するｉノードファイル１６２０を記述している。したがって、ルートｉノード１６１０Ｂおよび１６１２Ｂは、ブロックのツリーのルートとみなされる。ＷＡＦＬシステム１６７０は、このツリー構造をその更新方法（整合点）用に、そしてスナップショットを実行するために用いる。これらについては後述する。

汚れブロックを有するｉノードのリスト

ＷＡＦＬファイルシステムのメモリ内のｉノード（すなわち、図１０に示すＷＡＦＬｉノード１０１０）は、それぞれの状態に応じた異なるリンクリストに保持されている。ｉノードには、図２に示すように、汚れブロックを参照するｉノードが、汚れｉノードリストに保持されている。汚れていないバリッドデータを含むｉノードは、別のリストに保持される。そして、非バリッドデータを有するｉノードは、別に保持される。これらは、当業界において公知である。本発明は、汚れデータブロックを有するｉノードのリストを利用して、実行すべき書き込み割り当てを必要とする全てのｉノードを見つけることを容易にする。

図２は、本発明による汚れｉノードのリスト２１０を表す。汚れｉノードのリスト２１０は、ＷＡＦＬのメモリ内のｉノード２２０～１７５０からなる。図１７に示すように、ＷＡＦＬのメモリ内の各ｉノード２２０～２５０は、それぞれ、リンクリスト中の他のｉノードを指すポインタ２２０Ａ～２５０Ａからなる。たとえば、ＷＡＦＬｉノード２２０～２５０は、それぞれ、メモリ中の位置２０４８、２１５２、２８７８、３４４８および３７１２に保存される。このようにして、ｉノード２２０のポインタ２２０Ａはアドレス２１５２を含む。したがって、ＷＡＦＬｉノード２２２が指される。次に、ＷＡＦＬｉノード２２２は、アドレス２８７８を用いるＷＡＦＬｉノードを指す。ＷＡＦＬｉノード２３０は、ＷＡＦＬｉノード２４０を指す。ＷＡＦＬｉノード２４０は、ｉノード１７５０を指す。ＷＡＦＬｉノード２５０のポインタ２５０Ａはヌル値を含み、したがって、他のｉノードを指さない。このように、それは、汚れｉノードのリスト２１０中の最後のｉノードである。リスト２１０中の各ｉノードは、図１０に描かれたように、バッファツリーからなるファイルを表している。各ｉノード２２０～２５０によって参照されるブロックの少なくとも一つは、汚れバッファである。汚れバッファは、ＷＡＦＬシステムにおいて、新しいディスクの位置に書き込まねばならない変更されたデータを含んでいる。ＷＡＦＬは、常に汚れバッファをディスク上の新しい位置に書き込む。

整合点

今まで説明してきたＷＡＦＬディスク構造は、静的である。本発明において、ファイルシステム１６７０への変化は、ファイルシステム１６７０を整合状態に保つために、強く制御される。ファイルシステム１６７０は、１つの自己整合状態から他の自己整合状態へ進む。ルートｉノード１５１０Ｂによりルートされるディスク上の自己整合ブロックの集合（またはツリー）は、整合点（ＣＰ）という。整合点を提供するため、ＷＡＦＬは、常に、新しいデータをディスク上の割り当てられていないブロックに書く。ＷＡＦＬは、存在しているデータを上書きすることはない。こうして、ルートｉノード１５１０Ｂが更新されないかぎり、ディスクで表されるファイルシステム１６７０の日付は変わらない。しかし、ファイルシステム１６７０が有用であるためには、ファイルシステムは、常に、新しく書込まれたデータを参照せねばならず、したがって、新しい整合点が書込まれねばならない。

図１６を参照すると、新しい整合点は、まずすべてのファイルシステムブロック（ｉノードファイル１６２０、ブロックマップファイル１６３０、ｉノードマップファイル１６４０などのメタデータファイルにおけるブロックを含む）をディスク上の新しい位置にフラッシュ(flush)することにより、書込まれる。ファイルシステム１６７０のための新しいルートｉノード１６１０Ｂ、１６１２Ｂは、次に、ディスクに書込まれる。ファイルシステムを原子的に更新するこの方法を用いると、オンディスクファイルシステムは、決して整合的でない。このオンディスクファイルシステム１６７０は、ルートｉノード１６１０

Ｂ、１６１２Ｂが書込まれるまで、古い整合点を反映する。ルートｉノード１６１０Ｂ、１６１２Ｂが書込まれた後にすぐに、ファイルシステム１６７０は新しい整合点を反映する。ファイルシステム１６７０のデータ構造は、任意の順序に更新でき、ルートｉノード１６１０Ｂ、１６１２Ｂが更新される前にファイルシステム１６７０内の全ブロックがディスクに書込まれねばならないという１つの条件を除いて、ディスクへのデータの書込みについて順序の条件はない。

新しい整合点に変換するため、ルートｉノード１６１０Ｂ、１６１２Ｂは、信頼できるようにまた原子的に更新されねばならない。ルートｉノード１６１０Ｂ、１６１２Ｂを含むファイルシステム情報構造１６１０、１６１２の２つの同一のコピーを保持することにより、ＷＡＦＬは、これを実行する。ルートｉノード１６１０Ｂ、１６１２Ｂを更新する間、ファイルシステム情報構造１６１０の第１のコピーは、ディスクに書込まれ、次に、ファイルシステム情報構造１６１２の第２のコピーが書込まれる。ファイルシステム情報構造１６１０、１６１２におけるチェックサム１６１０Ｃ、１６１２Ｃは、それぞれ、ディスクに書込まれるとき、ｉノードのコピーをそれぞれ含むファイルシステム情報構造１６１０、１６１２のコピーの１つを破壊するシステムクラッシュの発生を検知するために使用される。通常は、この２つのファイルシステム情報構造１６１０、１６１２は同一である。

整合点を作成するアルゴリズム

図５は、整合点を作成する方法を説明する。ステップ５１０において、システムにおけるすべての「汚れ」ｉノード（変更されたデータを含む新しいブロックを指すｉノード）は、その内容が、またその内容のみがディスクに書込まれる整合点にあるとしてマークされる。これらの書込みが完全であるときにのみ、他のｉノードからの書込みがディスクに達することを許される。さらに、汚い書込みが発生している間、新しい変更は、整合点にあるｉノードに対して行うことができない。

整合点の１部であるすべての汚れｉノードに整合点フラグを立てることに加えて、グローバル整合点フラグが、使用者が要求する変化が強く制御された形で起こるように立てられる。グローバル整合点フラグが立てられると、使用者が要求する変化は、整合点にあるｉノードに影響することを許されない。さらに、整合点フラグが立てられたｉノードのみが、その汚れブロックに対してディスク空間を割り当てられる。その結果、ファイルシステムの状態は、整合点が始まるときに存在したように、ディスクに正確にフラッシュされる。

ステップ５２０において、正規の(regular)ファイルがディスクにフラッシュされる。この正規のファイルのフラッシュは、正規のファイルの中の汚れブロックのためのディスク空間を割り当てるステップと、対応するＷＡＦＬバッファをディスクに書込むステップからなる。ｉノードそれ自体は、次にｉノードファイルにフラッシュ（コピー）される。書込まれることが必要なすべてのｉノードが、汚れバッファを有するｉノードのリスト、または、汚いｉノードであるが汚れバッファを有しないｉノードのリストのいずれかに存在する。ステップ５２０が終わったとき、整合点には通常のｉノードはもはや存在せず、入ってくるすべてのＩ／Ｏ要求は、ディスクＩ／Ｏ操作のためになおロックされているバッファを使用しないなら、これらの要求はうまくいく。

ステップ５３０において、特別なファイルがディスクにフラッシュされる。この特別なファイルのフラッシュは、２つの特別なファイル、ｉノードファイルとブロックマップファイル、における汚れブロックに対してディスク空間を割り当てるステップ、整合点を更新してブロックマップファイルにおける各エントリに対しアクティブなファイルシステムビット（ＦＳビット）を調和させるステップ、および、ブロックをディスクに書込むステップからなる。ｉノードファイルとブロックマップファイルを割り当てる書込みは、それらを割り当てる書込みの過程がそれらのファイル自体を変化するので、複雑である。そうして、ステップ５３０において、変化が終わる前にディスクＩ／Ｏ操作において重要なブロックがロックされるのを防止するために、これらのファイルを変える間は、書込みは、ディスエーブルにされる。

また、ステップ５３０において、ファイルシステム情報ブロックを除いて、ファイルシステムが完全に自己整合性がありディスクに書込まれようとする唯一の時点であるので、スナップショットの作成と除去（後で説明する）がおこなわれる。同じスナップショットｉノードが１つのパスで使用できるように、１つのスナップショットは、新しいものが作成される前に、ファイルシステムから除去される。

図６は、ステップ５３０に含まれるステップを示す流れ図である。ステップ５３０は、ディスク空間をブロックマップファイルとｉノードファイルに割り当て、ブロックマップファイルにおける各エントリに対してアクティブなＦＳビットをＣＰビットにコピーする。

ステップ６１０において、ブロックマップファイルのためのｉノードは、ｉノードファイルにあらかじめフラッシュされる。このことは、ブロックマップファイルのｉノードを含むｉノードファイルにおけるブロックが汚くてステップ６２０がそのブロックにディスク空間を割り当てることを保証する。

ステップ６２０において、ディスク空間は、ｉノードとブロックマップファイルにおけるすべての汚れブロックに対して割り当てられる。この汚れブロックは、ブロックマップファイルの汚れｉノードを含むｉノードファイルにおけるブロックを含む。

ステップ６３０において、ブロックマップファイルのためのｉノードがふたたびフラッシュされるが、しかし、今回は、実際のｉノードが、ｉノードファイルにおける前記のまえもってフラッシュされたブロックに書込まれる。ステップ６１０は、ブロックマップファイルのｉノードを含むｉノードファイルのブロックをすでに汚くしている。こうして、ステップ６２０におけるように、もう１つの書込み割り当ては、スケジュールされる必要がない。

ステップ６４０において、ブロックマップファイルにおける各ブロックのためのエントリが更新される。各エントリは、ブロックマップファイルにおける汚れブロックにおける全エントリに対してアクティブなＦＳビットをＣＰビットにコピーする（すなわち、ビット０をビット３１にコピーする）ことにより、更新される。

ステップ６５０において、ブロックマップとｉノードファイルにおけるすべての汚れブロックは、ディスクに書込まれる。

ブロックマップファイルの汚れブロックにおけるエントリのみが、ステップ６４０において、アクティブなファイルシステムビット（ＦＳビット）を整合点ビット（ＣＰビット）にコピーさせる必要がある。整合点のすぐ後で、すべてのブロックマップエントリがアクティブなＦＳビットとＣＰビットの両方について同じ値を有する。時間が進むにつれ、ファイルシステムのブロックマップファイルエントリのいくつかのアクティブなＦＳビットがクリアされるか、セットされる。変化されたＦＳビットを含むブロックマップファイルのブロックは、したがって、汚いとマークされる。次の整合点の間、クリーンであるブロックは、ふたたびコピーされる必要がない。クリーンなブロックは、前の整合点で汚くなく、また、それらのブロック内においてそれ以後何も変わっていないので、コピーされない。こうして、すべてのブロックマップエントリにおいて同じ値を有するアクティブなＦＳビットとＣＰビットとともにファイルシステムが始めに生成されるかぎり、汚れブロックを有するエントリのみが各整合点において更新される必要がある。

図５を参照して、ステップ５４０において、ファイルシステム情報（fsinfo）ブロックが更新され、次に、ディスクにフラッシュされる。ファイルシステム情報ブロックは、ｉノードファイルについての新しいルートｉノードを書くことにより更新される。ファイルシステム情報ブロックは、２度書込まれる。はじめに１つの位置に書込まれ、次に、第２の位置に書込まれる。この２つの書込みは、システムクラッシュが書込みの間に起きたときに自己整合的なファイルシステムがディスク上に存在するように、行われる。したがって、もし第２のファイルシステム情報ブロックを書いている間にシステムがクラッシュしたならば、新しい整合点が存在し、または、もし第１のファイルシステム情報ブロックが故障したなら、（最近の整合点が始まる前のディスク上の）前の整合点が存在する。ファイルの故障の後でファイルシステムが再スタートされるとき、正しいチェックサム値を有するファイルシステム情報ブロックにおける整合点のための最高の作成（generation）計数

値が使用される。これは後で詳細に説明される。
ステップ５５０において、整合点が完成される。このことは、汚れｉノードが整合点の１部でなかったために遅延された当該整合点がふたたびキューされることを要求する。整合点の間に状態が変化されたｉノードは、整合点ウエイト（ＣＰ＿ＷＡＩＴ）キューにある。ＣＰ＿ＷＡＩＴキューは、ステップ５４０が終わる前で、かつ、整合点がスタートしたステップ５１０の後で変化したｉノードを保持する。整合点が終わると、ＣＰ＿ＷＡＩＴキューにおけるｉノードは、汚れバッファを有するｉノードの通常のリストと、汚れバッファを持たない汚れｉノードのリストにおいて、ふたたびキューされる。
整合性の単一の順序づけの条件
図２０Ａ～図２０Ｃに示されるように、本発明は、単一の順序づけの条件を有する。この単一の順序づけの条件は、すべての他のブロックがディスクに書込まれた後でのみファイルシステム情報ブロック１８１０がディスクに書込まれるということである。ファイルシステム情報ブロック１８１０の書込みは、原子的であり、そうでないと、全体のファイルシステム１８３０は、失われるであろう。こうして、ＷＡＦＬファイルシステムは、ファイルシステム情報ブロック１８１０が１度に、整合状態においてでなく書込まれることを要求する。図１５に示すように、各ファイルシステム情報ブロック１８１０（１５１０）は、チェックサム１５１０Ｃと作成計数値１５１０Ｄを含む。
図２０Ａは、ファイルシステム情報ブロック１８１０、１８７０の作成計数値１８１０Ｄ、１８７０Ｄの更新を説明する。整合点（またはスナップショット）が行われるたびに、ファイルシステム情報ブロックの作成計数値が更新される。図２０Ａは、ファイルシステムの整合点を示す同じ値Ｎをそれぞれ有する作成計数値１８１０Ｄ、１８７０Ｄを有する２つのファイルシステム情報ブロック１８１０、１８７０を示します。２つのファイルシステム情報ブロックは、前の整合点（ディスク上の古いファイルシステム）１８３０を参照する。ファイルシステムの新しいバージョンは、ディスク上に存在し、新しい整合点１８３１として参照される。作成計数値は、整合点ごとにインクリメントされる。
図２０Ｂにおいて、第１のファイルシステム情報ブロック１８１０の作成計数値１８１０Ｄは更新され、Ｎ＋１の値を与えられる。これは、次にディスクに書込まれる。図２０Ｂは、ファイルシステム情報ブロック１８１０の作成計数値１８１０Ｄの値Ｎ＋１を示すが、第２のファイルシステム情報ブロック１８７０の作成計数値１８７０Ｄは、値Ｎを有する。ファイルシステム情報ブロック１８１０は、新しい整合点１８３１を参照するが、これに対し、ファイルシステム情報ブロック１８７０は、古い整合点１８３０を参照する。
次に、図２０Ｃに示すように、ファイルシステム情報ブロック１８７０の作成計数値１８７０Ｄが更新され、ディスクに書込まれる。図２０Ｃにおいて、ファイルシステム情報ブロック１８７０の作成計数値１８７０Ｄは、値Ｎ＋１を有する。したがって、この２つのファイルシステム情報ブロック１８１０と１８７０は、同じ作成計数値Ｎ＋１を有する。システムクラッシュがファイルシステム情報ブロックの更新（複数）の間で起こるとき、ファイルシステム情報ブロック１８１０と１８７０の各コピーは、自己整合チェックサム（図示しない）を有するが、作成計数値１８１０Ｄと１８７０Ｄの一方がより大きい値を有する。システムクラッシュは、ファイルシステムが図２０Ｂに示す状態にあるときに起こる。たとえば、図２０Ｂに示すように本発明の好ましい実施例において、ファイルシステム情報ブロック１８１０の作成計数値１８１０Ｄは、第２のファイルシステム情報ブロック１８７０の前に更新される。したがって、作成計数値１８１０Ｄ（値１）は、ファイルシステム情報ブロック１８７０の作成計数値１８７０Ｄより大きい。第１のファイルシステム情報ブロック１８１０の作成計数値がより高いので、それは、システムクラッシュの後でファイルシステムを回復するために選択される。これは、第１のファイルシステム情報ブロック１８１０がその作成計数値１８１０Ｄにより示されるようにより新しいデータを含むためである。第１のファイルシステム情報ブロックが更新中にシステムクラッシュが起こったために壊れた場合は、ファイルシステム情報ブロックの他のコピー１８７０が、ファイルシステム１８３０を整合状態に回復するために使用される。
２つのファイル空間情報ブロック１８１０と１８７０を本発明において同時に更新するこ

とはできない。したがって、ファイル空間情報ブロック１８１０と１８７０の少なくとも１つのコピーがファイルシステムにおいて存在する。このことは、システムが常に整合状態に回復されることを可能にする。

ＷＡＦＬは、特殊な再生過程を必要としない。これは、ロギング、オーダードライト（順序だてられた書込み）、および、多くは、回復を伴うオーダドライトを用いる従来技術と異なる。これは、ＲＡＩＤにより保護されるデータ破壊またはソフトウエアのみがＷＡＦＬファイルシステムを破壊できるからである。システムが故障したときにデータを失うことを避けるために、ＷＡＦＬは、最新の整合点の後で行われたすべての操作の非破壊トランザクションのログを保持できる。このログは、ＷＡＦＬディスクフォーマットと完全に独立であり、操作がシステムクラッシュの間に失われるのを防ぐためにのみ必要である。しかし、ファイルシステムの整合性を維持するために必要ではない。

整合点の作成

先に説明したように、ＷＡＦＬファイルシステムへの変化は、ファイルシステムを整合状態に維持するために強く制御される。図１７Ａ～図１７Ｈは、ＷＡＦＬファイルシステムに対しての整合点の作成を示す。整合点の作成は、図５と図６を参照して説明される。

図１７Ａ～図１７Ｈにおいて、変更されなかったバッファは、その側にアスタリスクを有しない。したがって、バッファは、オンディスクブロックに対応するのと同じデータを含む。こうして、ブロックは、メモリにロードできるが、そのオンディスクバージョンに関して変化されていない。１つのアスタリスク（*）を側に持つバッファは、メモリにおける汚れバッファを指す（そのデータが変更される）。２重のアスタリスク（**）を側に持つバッファは、ディスク空間に割り当てられている汚れバッファを指す。最後に、３重のアスタリスク（***）を側に持つバッファは、ディスク上の新しいブロックに書込まれる汚れバッファである。バッファの状態を表すこの方法は、図２１Ａ～図２１Ｅにおいても用いられる。

図１７Ａは、ｉノード２３０６Ａと２３０６Ｂからなる汚れバッファを持つｉノードのリスト２３９０を示す。ｉノード２３０６Ａと２３０６Ｂは、バッファのツリー（複数）を参照する。ここで、各ツリーの少なくとも１つのバッファは変更されている。初めに、ｉノード２３０６Ａと２３０６Ｂの整合点フラグ２３９１と２３９２がクリアされる（０）。本システムについての汚れバッファを持つｉノードのリスト２３９０が示されるが、当業者に明らかであるが、ｉノードの他のリスト（複数）もメモリに存在してもよい。たとえば、汚れｉノードであるが、汚れバッファを持たないｉノードのリストがメモリに維持される。また、これらのｉノードは、整合点にあるとしてマークされねばならない。これらは、汚れｉノードが汚れブロックを参照しないけれども、ｉノードファイルの汚れコンテンツをディスクに書くために、ディスクに対しフラッシュされねばならない。これは、図５のステップ５２０でなされる。

図１７Ｂは、ファイルシステム情報ブロック２３０２、ｉノードファイル２３４６、ブロックマップファイル２３４４およびファイル２３４０、２３４２からなる前の整合点のＷＡＦＬファイルシステムを表す。ファイル２３４０は、それぞれデータ「Ａ」、「Ｂ」、「Ｃ」を含むブロック２３１０～２３１４からなる。ファイル２３４２は、それぞれデータ「Ｄ」、「Ｅ」、「Ｆ」を含むブロック２３１６～２３２０からなる。ブロックマップファイル２３４４はブロック２３２４からなる。ｉノードファイル２３４６は、２つの４ＫＢブロック２３０４と２３０６からなる。第２のブロック２３０６は、ファイル２３４０、ファイル２３４２およびブロックマップファイル２３４４をそれぞれ参照するｉノードファイル２３０６Ａ～２３０６Ｃからなる。これは、ｉノードにおけるファイル番号をリストすることによりブロック２３０６において示される。ファイルシステム情報ブロック２３０２は、ルートｉノードからなる。このルートｉノードは、ｉノードファイル２３４６のブロック２３０４と２３０６を参照する。こうして、図１７Ｂは、ルートｉノードを含むファイルシステム情報ブロック２３０２によりルートされるファイルシステムにおけるバッファのツリーを示す。

図１７Ｃは、メモリにおけるブロック２３１４と２３２２のための２つの変更されたバッ

ファを説明する。アクティブなファイルシステムは、データ「Ｃ」を含むブロック２３１４がファイル２３４０から除去されるように、変更される。また、ブロック２３２０に格納されるデータ「Ｆ」は、「Ｆプライム」に変更され、ディスクブロック２３２２のためのバッファに格納される。ディスクブロック２３１４と２３２２のためのバッファに含まれる変更されるデータは、この時点でメモリにおいてのみ存在する。図１７Ｃにおけるアクティブなファイルシステムにおける全ての他のブロックは、変更されず、したがって、そのそばにアスタリスクを持たない。しかし、これらのブロックのいくつか、または、全ては、メモリにおいて対応するきれいなバッファを持ってもよい。

図１７Ｄは、メモリにおけるブロックマップファイル２３４４のエントリ２３２４Ａ～２３２４Ｍを説明する。エントリ２３２４Ａ～２３２４Ｍは、ブロックマップファイル２３４４の４ＫＢブロック２３２４のためのバッファに含まれる。前に説明したように、ビット０とビット３１は、それぞれ、ＦＳビットとＣＰビットである。整合点ビット（ＣＰビット）は、１つの整合点が始まったが終わっていないときに対応するブロックが変更されないことを保証するために、整合点の間、セットされる。ビット１は、第１スナップショットビット（後で説明する）である。ブロックマップエントリ２３２４Ａと２３２４Ｂは、図１７Ｂに示すように、ｉノードファイル２３４６の４ＫＢブロック２３０４と２３０６がアクティブなファイルシステムにあり（ＦＳビット＝１）、整合点にある（ＣＰビット＝１）を示す。同様に、他のブロック２３１０～２３１２と２３１６～２３２０と２３２４は、アクティブなファイルシステムにあり、整合点にあることを示す。しかし、ブロック２３０８、２３２２および２３２６～２３２８は、（ビット０とビット３１にそれぞれ示すように）アクティブなファイルシステムにも整合点にもない。除去されたブロック２３１４のためのエントリは、ＦＳビットに値０を持ち、アクティブなファイルシステムから除かれたことを示す。

図５のステップ５１０において、システムにおけるすべての「汚れ」ブロックは、整合点にあるとしてマークされる。汚れｉノードは、汚れｉノードと、汚れバッファを参照するｉノードの両方を含む。図１７Ｉは、汚れバッファを持つｉノードのリストを示す。ここで、ｉノード２３０６Ａと２３０６Ｂの整合点フラグ２３９１と２３９２はセットされる（１）。ｉノード２３０６Ａは、アクティブなファイルシステムから除去されるべきファイル２３４０のデータ「Ｃ」を含むブロック２３１４を参照する。ｉノードファイル２３４２のブロック２３０６のｉノード２３０６Ａは、ファイル２３４２を参照する。データ「Ｆ」を含むブロック２３２０は、変更されており、データ「Ｆ」を含む新しいブロックが割り当てられねばならない。ステップ５１０において、汚れｉノード２３０６Ａと２３０６Ｂは、ブロック２３０８のためのバッファにコピーされる。ブロック２３０６のためのバッファは、続いてディスクに書込まれる（ステップ５３０）。これは、図１７Ｅに示される。変更されたデータは、メモリにおいてのみ存在し、バッファ２３０８は汚いとマークされる。ｉノード２３０６Ａと２３０６Ｂの整合点フラグ２３９１と２３９２は、図１７Ａに示すように、クリアされる（０）。これは、これらのｉノードを他のプロセスによる使用のために解放する。

ステップ５２０において、正規のファイルはディスクにフラッシュされる。こうして、ブロック２３２２はディスク空間に割り当てられる。ファイル２３４０のブロック２３１４は、削除され、したがって、整合点が終わるまでこのブロックに対してなにも起こらない。ブロック２３２２は、ステップ５２０において、ディスクに書込まれる。これらは、図１７Ｆにおいて示され、ここで、ブロック２３２２と２３１４のためのバッファはディスクに書込まれる（***によりマークされる）。ディスク空間の中間的割り当て（**）は示されない。ｉノードファイル２３４６のブロック２３０８のｉノード２３０８Ａと２３０８Ｂは、ｉノードファイルにフラッシュされる。ブロック２３０８のｉノード２３０８Ａは、ファイル２３４６のブロック２３１０と２３１２を参照する。ｉノード２３０８Ｂは、ファイル２３４２のためのブロック２３１６、２３１８、２３２２を参照する。図１７Ｆに示すように、ディスク空間は、ｉノード２３４６のブロック２３０８と、ファイル２３４２のための直接ファイル２３２２について割り当てられる。しかし、ファイルシステ

ムそれ自体は、更新されない。こうして、ファイルシステムは整合状態にとどまる。
ステップ５３０において、ブロックマップファイル２３４４は、ディスクにフラッシュされる。これは図１７Ｇに示され、ここで、ブロックマップファイル２３４４はアスタリスクにより汚いとして示される。
図６のステップ６１０において、ブロックマップファイルのためのｉノードは、図１７Ｈに示されるように、ｉノードファイルにまえもってフラッシュされる。ｉノード２３０８Ｃは、ｉノードファイル２３４６のブロック２３０６Ｂにフラッシュされている。しかし、ｉノード２３０８Ｃはなおブロック２３２４を参照する。ステップ６２０において、ディスク空間は、ブロックマップファイル２３４４とｉノードファイル２３４６のために割り当てられる。ブロック２３０８は、ｉノードファイル２３４６のために割り当てられ、ブロック２３２６は、ブロックマップファイル２３４４のために割り当てられる。上に説明したように、ｉノードファイル２３４６のブロック２３０８は、ブロックマップファイル２６４４のためのまえもってフラッシュされたブロック２３０８Ｃを含む。ステップ６３０において、ブロックマップファイル２３４４のためのｉノードは、ｉノード２３４６における上記のまえもってフラッシュされたブロック２３０８Ｃに書込まれる。こうして、メモリ内のｉノード２３０８Ｃは更新されて、ステップ６２０においてブロック２３２４を参照し、ブロック２３０８に書込まれるべきブロック２３０６を含むメモリ内のバッファにコピーされる。これは、図１７Ｈに示され、ここで、ｉノード２３０８Ｃはブロック２３２６を参照する。
ステップ６４０において、ブロックマップファイル２３４４における各ブロック２３０４～２３２６のためのエントリ２３２６Ａ～２３２６Ｌは、図１７Ｊにおいて更新される。図１７Ｂで整合点が始まってから変更していないブロックは、それらのエントリにおいて同じ値を有する。エントリは、ビット０（ＦＳビット）を整合点ビット（ビット３１）にコピーすることにより、更新される。ブロック２３０６は、アクティブなファイルシステムの１部ではない。したがって、ビット０は０に等しい（ビット０は、ステップ６２０でｉノードファイルのその部分に新しいデータを保持するようにブロック２３０８が割り当てらたときにオフにされる）。これは、図１７Ｊにおいてエントリ２３２６Ｂについて示される。同様に、ファイル２３４０のブロック２３１４のためのエントリ２３２６Ｆは０に等しいビット０とビット３１を有する。ファイル２３４２のブロック２３２０とブロックマップファイル２３４４のブロック２３２４は、同様に、エントリ２３６１と２３２６Ｋに示すように取り扱われる。ステップ６５０において、ｉノードファイル２３４６の汚れブロック２３０８とブロックマップファイル２３４４の汚れブロック２３２６は、ディスクに書込まれる。これは、図１７Ｋにおいて、ブロック２３０８と２３２６の側の３重アスタリスクにより示される。
図５を参照して、ステップ５４０において、ファイルシステム情報ブロック２３０２はディスクにフラッシュされ、これは２度行われる。こうして、図１７Ｌにおいて、ファイルスペース(fs)情報ブロック２３０２は、汚くされ、次に、（３重アスタリスクにより示すように）ディスクに書込まれる。図１７Ｌにおいて、単独のファイルスペース情報ブロック２３０２が示される。図に示すように、ファイルスペース情報ブロック２３０２は、いま、ｉノードファイル２３４６のブロック２３０４と２３０８を参照する。図１７Ｌにおいて、ブロック２３０６は、もはやアクティブなファイル空間におけるｉノードファイル２３４６の１部ではない。同様に、ｉノードファイル２３４６のｉノード２３０８Ａにより参照されるファイル２３４０は、ブロック２３１０と２３１２からなる。ブロック２３１４は、もはやこの整合点におけるファイル２３４０の１部ではない。ファイル２３４２は、この新しい整合点においてブロック２３１６、２３１８および２３２２からなるが、ブロック２３２０はファイル２３４２の１部ではない。さらに、ｉノードファイル２３４６のブロック２３０８は、ブロック２３２６からなる新しいブロックマップファイル２３４４を参照する。
図１７Ｌに示すように、１つの整合点において、アクティブなファイルシステムは、新しいファイル２３４６のｉノードをファイルスペース情報ブロック２３０２にコピーするこ

とにより、更新される。しかし、前の整合点のブロック２３１４、２３２０、２３２４および２３０６はディスク上に残っている。これらのブロックは、古い整合点１８３０と新しい整合点１８３１の両方が図２０においてステップ５４０の間に存在することを保証するために、ファイルシステムを更新するときに決して上書きされない。

スナップショット

ＷＡＦＬシステムは、スナップショットをサポートしている。スナップショットは、スナップショットが作成（generate）される瞬間の全ファイルシステムの読出し専用コピーである。新たに作成されるスナップショットは、アクティブなファイルシステムが指すものと同じディスクブロックを指す。したがって、それは短時間に作成され、追加のディスク空間を消費することはない。アクティブなファイルシステムにおけるデータブロックが変更されディスク上の新たな位置に書き込まれる時にのみ、スナップショットは追加の空間を使用し始める。

ＷＡＦＬは、１から２０まで番号付けされた２０個までの異なるスナップショットをサポートする。したがって、ＷＡＦＬは、同一ファイルシステムの多重の「クローン」の作成を可能にする。各スナップショットは、アクティブなファイルシステムがルートｉノードにより表わされるのと同様に、スナップショットｉノードにより表わされる。スナップショットは、ファイルシステムのルートデータ構造を複製することにより作成される。好ましい実施例において、ルートデータ構造はルートｉノードである。しかし、全ファイルシステムを表わすどのようなデータ構造も使用することができる。スナップショットｉノードは、ｉノードファイル中の固定位置に存在する。スナップショットの２０個の限界は、ブロックマップエントリの大きさにより決定されている。ＷＡＦＬは、新たなスナップショットNの作成に２つのステップを必要とする。すなわち、ルートｉノードをスナップショットN用のｉノードにコピーし、ビット０をブロックマップファイルにある各ブロックマップエントリのビットNにコピーする。ビット０は、ルートｉノードの下のツリーが指すブロックを示す。

その結果は、ルートｉノードと全く同じディスクブロックを示すスナップショットｉノードNによって確立された新たなファイルシステムツリーである。スナップショットにある各ブロックのブロックマップに対応するビットを設定すれば、アクティブなファイルがスナップショットブロックを使用しなくなっても、スナップショットブロックの解放が防止される。ＷＡＦＬは新たなデータを未使用のディスク位置に常に書き込むので、アクティブなファイルシステムが変わっても、スナップショットツリーは変わらない。新たに作成されたスナップショットツリーは、ルートｉノードと全く同じブロックを参照するので、新たなディスク空間は使用されない。時間が経過すると、スナップショットは、別途解放されたディスクブロックを参照する。したがって、時間が経過すると、スナップショットとアクティブなファイルシステムが共有するブロックは徐々に少なくなり、スナップショットが使用する空間は増大する。スナップショットは、それが許容できない数のディスクブロックを使用するようになると削除することができる。

アクティブなスナップショットのリストはスナップショットの名前とともに、スナップショットディレクトリと呼ばれるメタデータファイルに格納される。ディスクの状態は上に記したように更新される。他の全ての変更とともに、ある整合点より別の整合点に自動的に進むことにより、更新は行われる。変更されたブロックはディスク上の未使用位置に書き込まれ、その後に、更新されたファイルシステムを記載した新たなルートｉノードが書き込まれる。

スナップショットの概観

図１８Ａは、スナップショットが取り出される前のファイルシステム１８３０を示し、ＷＡＦＬファイルシステムの概観をより簡単にするため、間接性のレベルは除かれている。ファイルシステム１８３０は、図１６のファイルシステム１６９０を示している。ファイルシステム１８３０はブロック１８１２～１８２０からなる。ｉノードファイルのｉノードは、ファイルシステム情報ブロック１８１０に含まれる。ファイルシステム情報ブロック１８１０の一つのコピーしか図１８Ａには示されていないが、ファイルシステム情報ブ

ロックの第２のコピーもディスク上にあるものと解釈すべきである。ファイルシステム情報ブロック１８１０に含まれたｉノード１８１０Ａは、同一の間接性レベルの１６のブロックを指す１６のポインタを有している。図１８Ａのブロック１８１２～１８２０は、直接ブロック、間接ブロック等を含むファイルシステム１８３０内の全てのブロックを示している。５個のブロック１８１２～１８２０しか示されていないが、各ブロックは他のブロックを指してもよい。

図１８Ｂは、スナップショットの作成を示す。スナップショットは、全ファイルシステム１８３０用に、ファイルシステム情報ブロック１８１０に格納されているｉノードファイルのｉノード１８１０Ａをスナップショットｉノード１８２２に単にコピーすることにより作成される。ｉノードファイルのｉノード１８１０Ａをコピーすることにより、アクティブなファイルシステムと同じファイルシステムを示すｉノードの新たなファイルが作成される。ｉノードファイルそれ自体のｉノード１８１０Ａがコピーされるので、他のブロック１８１２～１８２０は２重にされる必要はない。コピーされたｉノードまたはスナップショットｉノード１８２２は、次に、ｉノードファイルにコピーされ、ｉノードファイルのブロックを汚す。一つまたはそれ以上の間接性レベルからなるｉノードファイルに対し、各間接ブロックが順に汚される。ブロックを汚すこのプロセスは、全ての間接性レベルを通して進む。ディスク上のｉノードファイルの各４キロバイトブロックは、３２のｉノードを含み、各ｉノードは１２８バイトの長さである。

図１８Ｂの新しいスナップショットｉノード１８２２は、スナップショット１８２２が取り出されると、ｉノードファイルのｉノード１８１０Ａにより参照される最高レベルの間接性のブロック１８１２～１８２０を指す。ｉノードファイルそれ自体は、ファイルシステム１８３０のスナップショットを含んでいるので、再帰的構造である。各スナップショット１８２２は、ｉノードファイルにコピーされるｉノードファイルのｉノード１８１０Ａのコピーである。

図１８Ｃは、スナップショット１８２２が取り出された後、アクティブなファイルシステム１８３０に変化が発生する時のアクティブなファイルシステム１８３０とスナップショット１８２２を示す。この図に示されるように、データ「Ｄ」を有するブロック１８１８は、スナップショット１８２２が取り出された後に変更されるので（図１８Ｂ）、データ「Ｄprime」を含む新たなブロック１８２４は、アクティブなファイルシステム１８３０用として割り当てられる。したがって、アクティブなファイルシステム１８３０はブロック１８１２～１８１６と１８２０～１８２４とを有するが、データ「Ｄ」を含むブロック１８１８を含まない。しかし、ＷＡＦＬシステムはディスク上のブロックに上書きしないので、データ「Ｄ」を含むブロック１８１８は上書きされない。ブロック１８１８は、ブロック１８１８用のブロックマップエントリに設定されているスナップショットビットにより上書きされないよう保護されている。したがって、スナップショット１８２２はなお、ブロック１８１２～１８１６と１８２０と同様、未変更のブロック１８１８を指している。本発明は、図１８Ａ～図１８Ｃに示されるように、ファイルシステムの「クローン」を作成しクローンがディスク上のｉノードファイルの全てのブロックのコピーである従来のシステムとは異なる。したがって、従来技術のｉノードファイルの全内容が２重にされるので、ディスクＩ／Ｏ操作にかなりの時間を必要とするばかりでなく、かなりの量（ＭＢ）のディスク空間を必要とする。

図１８Ｃにおいてアクティブなファイルシステム１８３０は変更されるとき、ブロック１８１２～１８２０を有するファイルシステムは上書きされないので、アクティブなファイルシステムはより多くのディスク空間を使用する。図１８Ｃにおいて、ブロック１８１８は直接ブロックとして示されている。しかし、実際のファイルシステムにおいては、同様に間接ブロックによりブロック１８１８を指してもよい。したがって、ブロック１８１８が変更され、ブロック１８２４として新たなディスク位置に格納されると、対応する直接ブロックおよび間接ブロックもまたコピーされ、アクティブなファイルシステム１８３０に割り当てられる。

図１９は、図１８Ｃのブロック１８２４に発生する変化を示す。図１８Ｃのブロック１８

２４は、図１９において破線１８２４の内側に示されている。図１９は、図１８Ｃのブロック１８２４に対するいくつかの間接性レベルを示している。図１８Ｃのディスクに書き込まれる新たなブロック１９１０は、図１９において１９１０の符号が付されている。ブロック１８２４は、２重間接性により参照される変更データを含むデータブロック１９１０を有しているので、二つの他のブロック１９１８と１９２６も変更される。単独間接ブロック１９１８のポインタ１９２４は新しいブロック１９１０を示すので、ブロック１９１８もディスクの新たな位置に書き込む必要がある。同様に、間接ブロック１９２６のポインタ１９２８はブロック１９１８を指すので、変更される。したがって、図１９に示されるように、データブロック１９１０を変更することにより、いくつかの間接ブロック１９１８と１９２６も変更を生じさせることができる。これはまた、ディスクの新たな位置へのブロック１９１８と１９２６の書込みを必要とする。

図１８Ｃのデータブロック１８２４の直接および間接のブロック１９１０、１９１８、１９２６は変えられ、新たな位置に書き込まれるので、ｉノードファイルのｉノードは新たなブロックに書き込まれる。ｉノードファイルの変更されたブロックは、データが上書きされないので、ディスク上の新たなブロックが割り当てられる。

図１９に示されるように、ブロック１９１０は間接ブロック１９２６、１９１８によってそれぞれ指し示される。したがって、ブロック１９１０が変更され、新たなディスク位置に格納されると、対応する直接および間接のブロックもまたコピーされ、アクティブなファイルシステムに割り当てられる。したがって、多くのデータ構造が更新される必要がある。直接ブロック１９１０および間接ブロック１９１８、１９２６を変えると、ブロックマップファイルが変更される。

スナップショット用キーデータ構造はブロックマップエントリであり、各エントリは一つのスナップショットに対し多重のビットを有する。これは、複数のスナップショットの作成を可能とする。スナップショットは、ファイルシステム（図１８の１８３０）であるブロックのツリーのピクチャである。新しいデータがスナップショットのブロック上に書き込まれないかぎり、スナップショットにより表わされるファイルシステムは変わらない。スナップショットは整合点と類似している。

本発明のファイルシステムは、ファイルシステム情報ブロック１８１０、１８７０が書き込まれた時点で完全に整合性がある。したがって、システムへの給電が妨げられても、再始動のときに、ファイルシステム１８３０は整合状態に達する。１ＧＢファイルシステムの従来の典型的な「クローン」では８～３２ＭＢのディスク空間が使用されているので、クローンは、本発明のように整合点またはスナップショットに導かない。

図２２を参照すると、ディスク上に既に二つのスナップショット２１１０Ａと２１１０Ｂが存在する。三つ目のスナップショットが作成される瞬間に、アクティブなファイルシステムを指すルートｉノードは、ｉノードファイル２１１０の第３のスナップショットのためにｉノードエントリ２１１０Ｃにコピーされる。同時に、通過する整合点において、スナップショット３が作成されたことをフラグが示す。全ファイルシステムは、ブロックマップファイルにおける各エントリごとにビット０がセットされるか（１）、あるいは、クリアされるか（０）をチェックすることにより処理される。各ブロックマップエントリの全てのビット０の値はスナップショット３のためのプレーンにコピーされる。終了すると、ファイルシステムの各アクティブブロック２１１０～２１１６と１２０７は、それが取り出される瞬間にスナップショットに存在する。

所定時間連続してディスク上に存在したブロックも、第３のスナップショット２１１０Ｃの前の対応するスナップショット２１１０Ａ～２１１０Ｂに存在する。あるブロックが十分長い時間ファイルシステムに存在したならば、それはすべてのスナップショットに存在する。ブロック１２０７はそのようなブロックである。図２２に示されるように、ブロック１２０７は、アクティブなｉノードファイルのｉノード２２１０Ｇにより示されるとともに、スナップショット１、２、３により間接的に示される。

スナップショットの順番は、ファイルシステムコピーの時間順を必ずしも示すわけではない。ファイルシステムの個々のスナップショットは任意の時に削除することができ、こう

することによりエントリをさらに使用することが可能となる。アクティブなファイルシステムを参照するブロックマップエントリのビット０がクリアされ（ブロックのアクティブなファイルシステムからの削除を示す）、いずれかのスナップショット参照ビットが設定されると、ブロックは再使用できない。これは、ブロックが、なお使用されているスナップショットの一部だからである。ブロックは、ブロックマップエントリの全てのビットが０に設定される時にのみ再使用できる。

スナップショットを作成するアルゴリズム

スナップショットの作成は、図５に示されるような規則的整合点の作成にほぼ完全に類似している。ステップ５１０において、全ての汚れｉノードは、整合点にあるものとしてマークが付けられる。ステップ５２０において、全ての正規のファイルはディスクに対しフラッシュされる。ステップ５３０において、特別なファイル（すなわち、ｉノードファイルとブロックマップファイル）は、ディスクに対しフラッシュされる。ステップ５４０において、ファイルシステム情報ブロックはディスクにフラッシュされる。ステップ５５０において、整合点にない全てのｉノードは処理される。図５については詳細に先に説明している。実際、スナップショットの作成は、整合点の作成の一部として行われる。スナップショットの作成と整合点の作成における基本的な違いは、ブロックマップファイルのすべてのエントリが、スナップショットビットにコピーされたアクティブなＦＳビットを有していることである。スナップショットビットは、スナップショットのブロックが上書きされることのないよう保護するために、対応するスナップショットを示している。スナップショットの作成および削除は、ステップ５３０で行われるが、これは、ファイルシステムが完全に自己整合性があり、ディスクにまさに移る唯一の点だからである。

異なるステップ（複数）が、新たなスナップショットが作成される時の整合点のために図６に示されているステップ５３０において行われる。ステップは、正規の整合点のステップと非常に類似している。図７は、スナップショットを作成するためにステップ５３０が有するステップを示す流れ図である。上述したように、ステップ５３０は、ディスク空間をブロックマップファイルとｉノードファイル用に割り当てるとともに、対応するスナップショットを示すスナップショットビットにアクティブＦＳビットをコピーし、スナップショットのブロックが上書きされないよう保護する。

ステップ７１０において、ブロックマップファイルのｉノードと作成されているスナップショットはディスクに前もってフラッシュされる。ブロックマップファイルのｉノードのｉノードファイルのブロックへのフラッシュ（整合点に対する図６のステップ６１０のように）に加えて、作成されているスナップショットのｉノードもまた、ｉノードファイルのブロックにフラッシュされる。その結果、スナップショットのｉノードを含むｉノードファイルのブロックは汚れることになる。

ステップ７２０において、ブロックマップファイルの各ブロックマップは汚れている。ステップ７６０（後で記載する）において、ブロックマップファイルのすべてのエントリは、汚れブロックのエントリの代わりに更新される。したがって、ブロックマップファイルのすべてのブロックはここで汚れているとのマークを付ける必要があり、ステップ７３０はディスク空間をそれに書き込み割り当てる。

ステップ７３０において、ディスク空間は、ｉノードとブロックマップファイルのすべての汚れブロックに割り当てられる。汚れブロックには、汚れブロックマップファイルのｉノードを含むｉノードファイルのブロックと新たなスナップショット用ｉノードを含むブロックとが含まれる。

ステップ７４０において、ファイルシステム用ルートｉノードの内容は、ｉノードファイルのスナップショットのｉノードにコピーされる。この時、新たな整合点の一部でありディスクに書き込まれる各ブロックには、それに割り当てられるディスク空間がある。したがって、ルートｉノードをスナップショットｉノードに２重化すると、アクティブな全ファイルシステムが効率的にコピーされる。スナップショットに存在する実際のブロックは、アクティブなファイルシステムの同じブロックである。

ステップ７５０において、ブロックマップファイルとスナップショットのｉノードは、ｉ

ノードファイルにコピーされる。
ステップ７６０において、ブロックマップファイルのエントリは更新される。アクティブなＦＳビットのエントリのためのＣＰビットへのコピーに加えて、アクティブなＦＳビットも、新たなスナップショットに対応するスナップショットビットにコピーされる。
ステップ７７０において、ブロックマップとｉノードファイルとの中のすべての汚れブロックはディスクに書き込まれる。
最後に、ある時点で、スナップショットそれ自体が、ステップ７６０においてファイルシステムから削除される。スナップショットは、アクティブなファイルシステムのｉノードファイルの中におけるそのスナップショットｉノードエントリをクリアし、ブロックマップファイルの各エントリにおけるスナップショット番号に対応する各ビットをクリアすることにより、ファイルシステムより削除される。また、設定値からクリアされる全てのブロックマップエントリの中のスナップショットの各ビットの計数が行われ、これにより、スナップショットを削除することにより解放されるブロックの計数値（解放されるディスク空間の大きさに相当する量）が得られる。このシステムは、最も古いスナップショットに基づいて削除すべきスナップショットを決定する。使用者もまた、特定のスナップショットの削除を手動で選択できる。
本発明は、スナップショットの総数を限定するとともに、エピソードのようにＣＯＷビットを有するポインタを使用する代わりに、スナップショットを追跡するための複数のビットを有するエントリを持つブロックマップファイルを維持する。未使用ブロックは、そのブロックマップファイルエントリにおけるビットがすべて０である。時間が経過すると、アクティブなファイルシステムビット０（ビット０）は、一般にある時点で１となる。ビット０を設定することにより、対応するブロックを、アクティブなファイルシステムにおいて割り当てられたものとして、同定する。上述したように、全てのスナップショットビットは、まず０に設定される。いずれかのスナップショットビットが設定される前にアクティブなファイルのビットがクリアされると、ディスク上に格納されたどのスナップショットにもブロックは存在しない。それ故、ブロックは再割り当てのためにすぐに使用でき、スナップショットから続いて回復できない。

スナップショットの作成

前述したように、スナップショットは整合点に非常に類似している。それ故、スナップショットの作成を、これと、図１７Ａ～図１７Ｌに示される整合点の作成との間の相違を参照して説明する。図２１Ａ～図２１Ｆは、スナップショット作成の相違を示している。
図１７Ａ～図１７Ｄは、スナップショット開始時のＷＡＦＬファイルシステムの状態を示している。すべての汚れｉノードは、ステップ５１０において整合点にあるものとしてマークされており、正規のファイルは、ステップ５２０においてディスクにフラッシュされる。したがって、スナップショットの最初の処理は、整合点のものと同一である。スナップショットのための処理は、ステップ５３０において整合点のものと相違している。図７によるスナップショットの処理を以下に説明する。
以下の説明は、ＷＡＦＬファイルシステムの第２のスナップショット用のものである。第１のスナップショットは、図１７Ｃのブロックマップエントリに記録されている。エントリ２３２４Ａ～２３２４Ｍに示されているように、ブロック２３０４～２３０６、２３１０～２３２０、２３２４は、第１スナップショットに含まれている。他の全てのスナップショットビット（ビット１～ビット２０）は、対応するスナップショットがディスク上にないことを示す０の値を持っているものとする。図２１Ａは、ステップ５１０とステップ５２０が終了した後のファイルシステムを示している。
ステップ７１０において、スナップショット２のｉノード２３０８Ｃと２３０８Ｄとブロックマップファイル２３４４はディスクに対しあらかじめフラッシュされる。その結果、スナップショット２のｉノードを包含しつつあるｉノードファイルのブロックが汚いことを保証する。図２１Ｂにおいて、ｉノード２３０８Ｃと２３０８Ｄは、スナップショット２に対し、かつ、ブロックマップファイル２３４４に対し、あらかじめフラッシュされる。

ステップ７２０において、全ブロックマップファイル２３４４は汚されている。その結果、ステップ７３０において、全体のブロックマップファイル２３４４がディスク空間に割り当てられる。ステップ７３０において、ディスク空間は、図２１Ｃに示されるように、汚れブロック２３０８と２３２６およびｉノードファイル２３４６とブロックマップファイル２３４４に対し割り当てられる。このことは、ブロック２３０８と２３２６の横に三重アスタリスク（***）で示されている。これは、図６のステップ６２０においてブロックマップファイル２３４４で変化したエントリを有するブロックにのみディスク空間が割り当てられる整合点の作成とは異なる。図２１Ｃのブロックマップファイル２３４４は単一ブロック２３２４を有している。しかし、ブロックマップファイル２３４４が１より多いブロックを有すると、ディスク空間はステップ７３０において全ブロックに割り当てられる。

ステップ７４０において、新しいファイルシステムのルートｉノードは、スナップショット２のｉノード２３０８Ｄにコピーされる。ステップ７５０において、ブロックマップファイル２３４４とスナップショット２のｉノード２３０８Ｃと２３０８Ｄは、図２１Ｄに示されるようにディスクにフラッシュされる。図は、スナップショット２のｉノード２３０８Ｄがブロック２３０６ではなく、ブロック２３０４と２３０８を参照することを示している。

ステップ７６０において、ブロックマップファイル２３４４のブロック２３２６におけるエントリ２３２６Ａ～２３２６Ｌは図２１Ｅに示されるように更新される。図２１Ｅは、スナップショット２のビット（ビット２）が、各エントリ２３２６Ａ～２３２６ＬのＦＳビットとＣＰビットと同様に更新されることを示している。したがって、ブロック２３０４、２３０８～２３１２、２３１６～２３１８、２３２２、２３２６はスナップショット２に含まれるのに対し、ブロック２３０６、２３１４、２３２０、２３２４は含まれない。ステップ７７０において、汚れブロック２３０８と２３２６はディスクに書き込まれる。

スナップショット２の次の処理は、図５に示される整合点の作成のものと同一である。ステップ５４０において、二つのファイルシステム情報ブロックはディスクにフラッシュされる。したがって、図２１Ｆは、このステップの後の整合状態にあるＷＡＦＬファイルシステムを示している。ステップ５４０が終了した後の、整合性のあるファイルシステムのファイル２３４０、２３４２、２３４４、２３４６は、図２１Ｆの破線の内側に示される。ステップ５５０において、整合点は、この整合点になかったｉノードを処理することにより終了する。

アクセスタイム上書き

ＵＮＩＸファイルシステムは、各ｉノードにおいて「アクセスタイム」（ａタイム(atime)）を維持する必要がある。ａタイムは、ファイルが読み出された最後の時間を示している。それは、ファイルがアクセスされるたびに更新される。その結果、ファイルが読出されると、ｉノードファイルにおいてｉノードを含むブロックは書き換えられ、ｉノードを更新する。これは、結果として、ファイルの読出しによりディスク空間を使い尽くす可能性があるので、スナップショットの作成には不利である。さらに、ファイルシステムのすべてのファイルの読出しにより、全ｉノードファイルの２重化を惹き起す。本発明は、この問題を解決する。

ａタイムにより、ｉノードを変更するとｉノードファイル用の新たなブロックがディスクに書き込まれるので、読出しはディスク空間を消費する可能性がある。さらに、ファイルシステムが一杯となるようなファイルシステムにとって異常な事態が発生すると、読出し動作は失敗に終わる可能性がある。

一般に、ディスク上のデータは、ディスクに格納されたデータを保護するように、ＷＡＦＬファイルシステムに上書きされることはない。この規則の唯一の例外は、図２３Ａ～図２３Ｂに示されるように、ｉノードに対するａタイム上書きである。「ａタイム上書き」が発生すると、ｉノードファイルのブロックで変更される唯一のデータは、それに含まれる一つあるいはそれ以上のｉノードのａタイムであり、ブロックは同じ位置で書き換えら

れる。これはＷＡＦＬシステムにおける唯一の例外であり、他の場合は、新しいデータは常に新しいディスク位置に書き込まれる。

図２３Ａは、古いＷＡＦＬｉノードファイルブロック２４２０のｉノード２４２２のａタイム２４２３と、ブロック２４２０を示すスナップショットｉノード２４３２のａタイム２４３３を示す。ブロック２４２０のｉノード２４２２は直接ブロック２４１０を参照する。ｉノード２４２２のａタイム２４２３は、「４／３０　９：１５ＰＭ」であるのに対し、スナップショットｉノード２４３２のａタイム２４３３は「５／１　１０：００ＡＭ」である。図２３Ａは、直接バッファ２４１０がアクセスされる前のファイルシステムを示している。

図２３Ｂは、直接ブロック２４１０がアクセスされた後の直接ブロック２４１０のｉノード２４２２を示す。図に示されるように、ｉノード２４２２のアクセスタイム２４２３は、それを示すスナップショット２４３２のアクセスタイム２４３３とともに上書きされる。したがって、直接ブロック２４１０のｉノード２４２２のアクセスタイム２４２３は「５／１　１１：２３ＡＭ」である。

新しいａタイムでｉノードファイルブロックを上書きすると、スナップショットにおいてわずかな非整合が発生する。スナップショットのファイルのａタイムは、実際には、スナップショットが作成された時間より後のはずである。ユーザがこの非整合を発見しないように、ＷＡＦＬは、スナップショットの全てのファイルのａタイムをファイルが最後にアクセスされた時間の代わりにスナップショットが実際に作成された時間に調節する。このスナップショットタイムは、スナップショットを全体として記載しているｉノードに格納される。したがって、スナップショットを介してアクセスすると、ｉノード２４２２のアクセスタイム２４２３は、常に「５／１　１０：００ＡＭ」として報告される。これは、「４／３０　９：１５ＰＭ」であると予想される更新前と、「５／１　１１：２３ＡＭ」であると予想される更新後の両方で発生する。アクティブなファイルシステムを介してアクセスすると、時間は更新前と更新後においてそれぞれ「４／３０　９：１５ＰＭ」と「５／１　１１：２３ＡＭ」として報告される。

このように、ファイルシステムを整合状態に維持し、ファイルシステムの読出し専用コピーを作成する方法が開示された。

【図１】

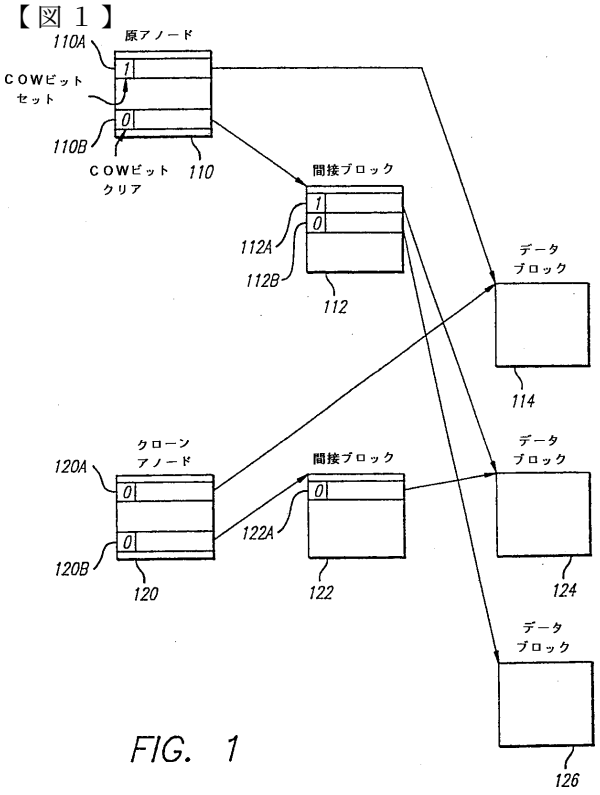


FIG. 1

【図２】

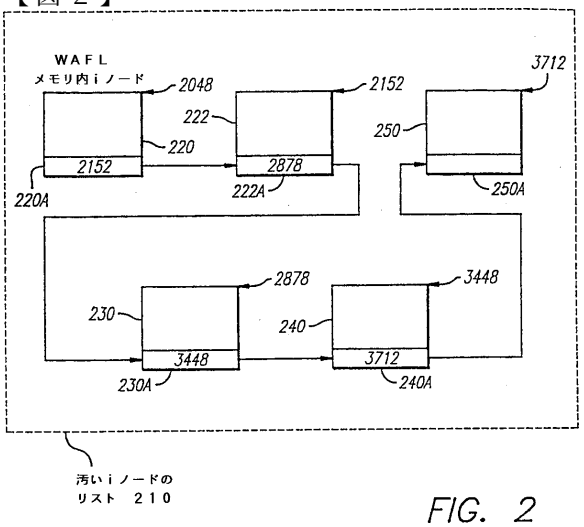
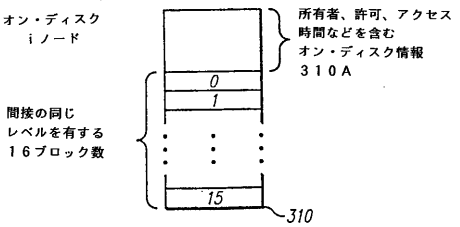


FIG. 2

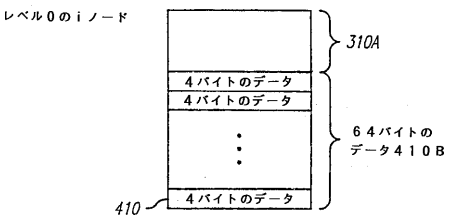
【図３】

FIG. 3



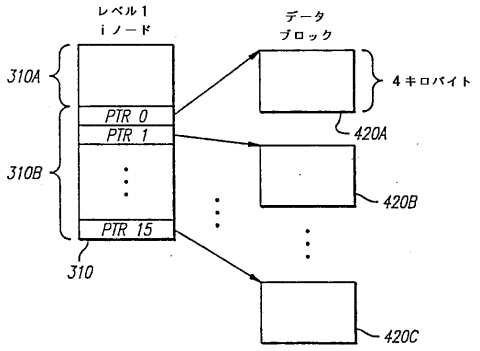
【図４Ａ】

FIG. 4A



【図４Ｂ】

FIG. 4B



【図４Ｃ】

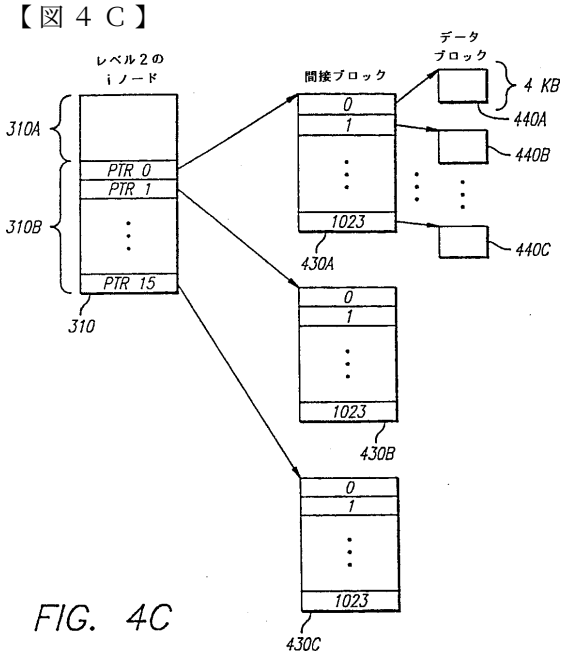


FIG. 4C

【図４Ｄ】

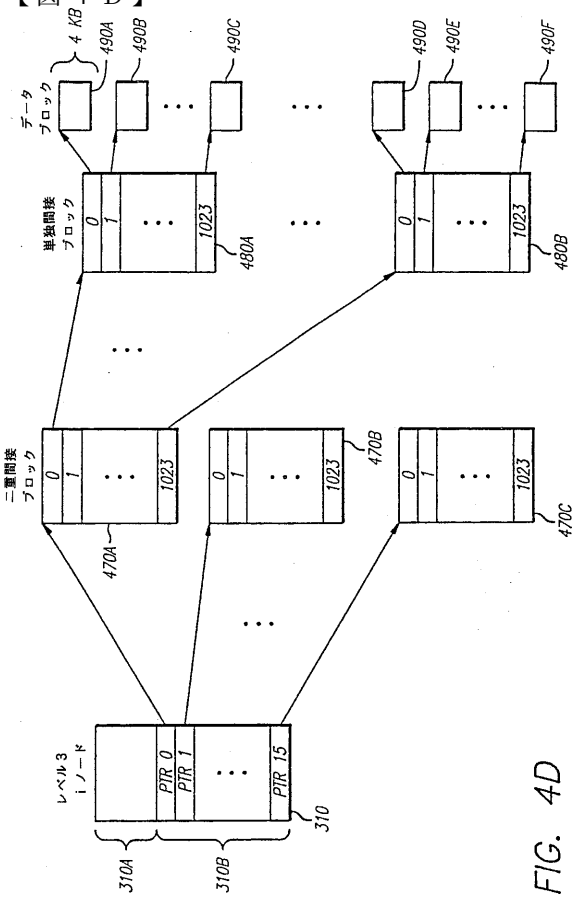


FIG. 4D

【図５】

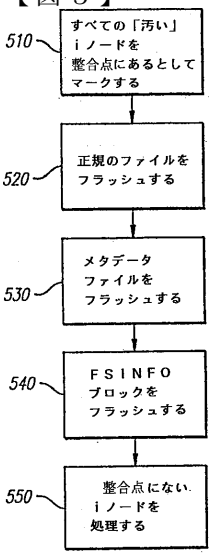


FIG. 5

【図６】

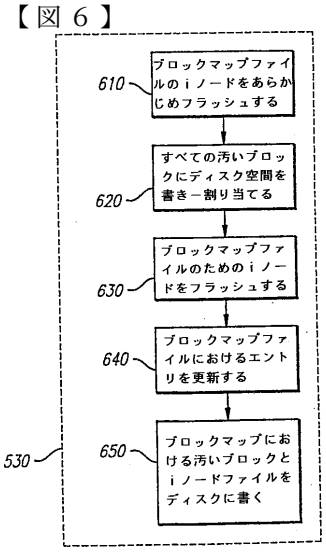


FIG. 6

【図７】

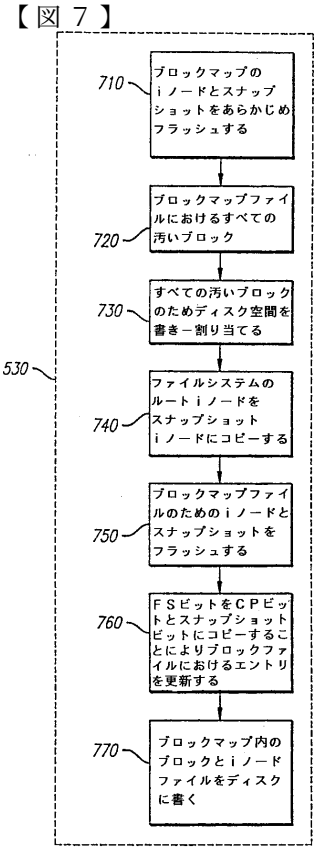
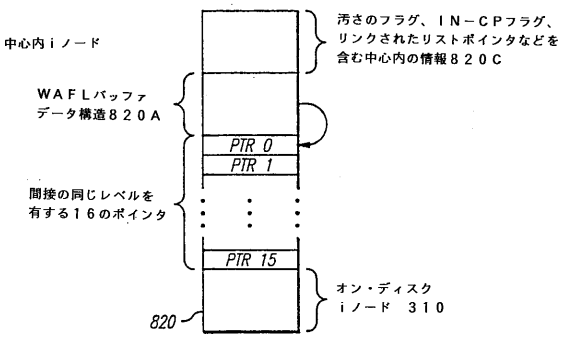
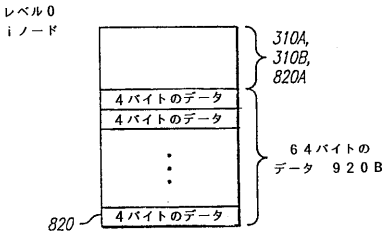


FIG. 7

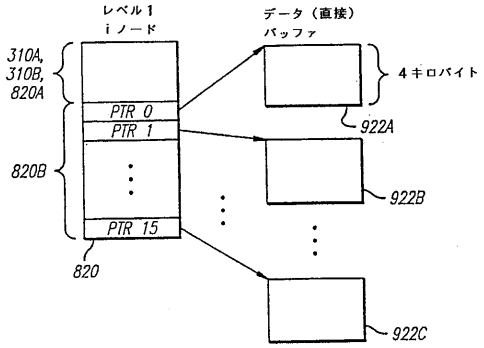
【図８】
FIG. 8



【図９Ａ】
FIG. 9A



【図９Ｂ】
FIG. 9B



【図９Ｃ】

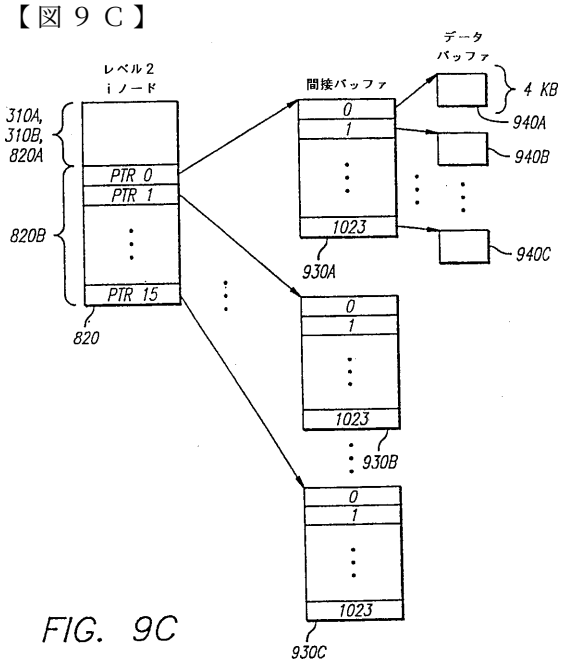


FIG. 9C

【図９Ｄ】

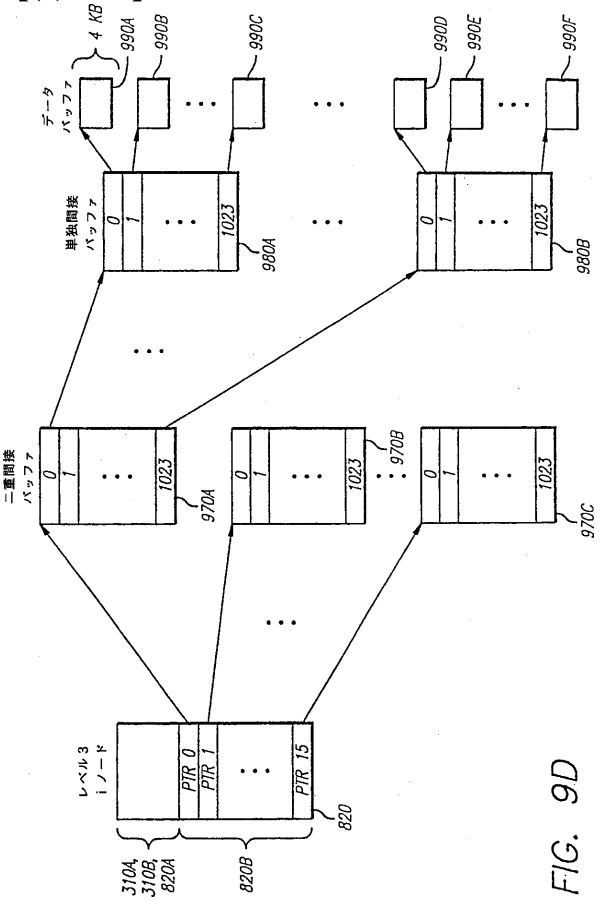


FIG. 9D

【図１０】

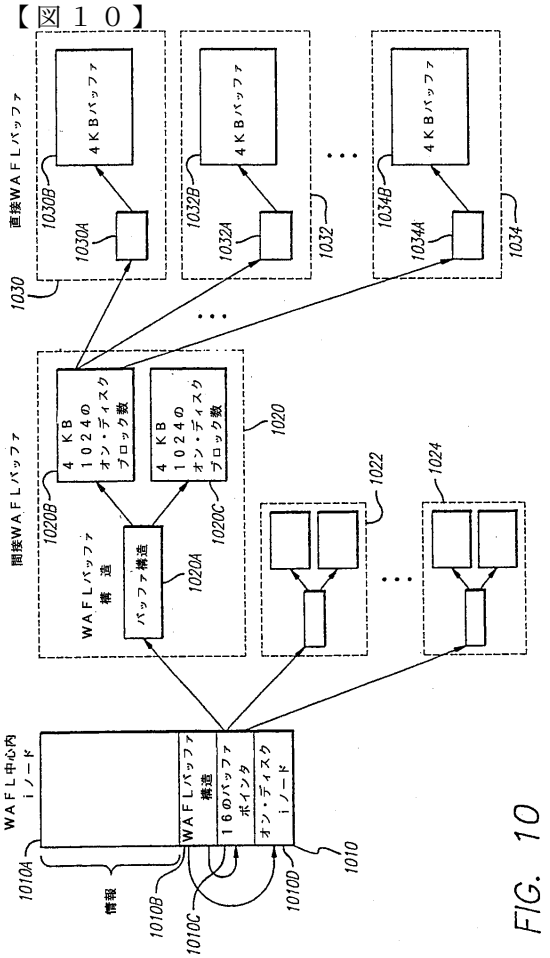
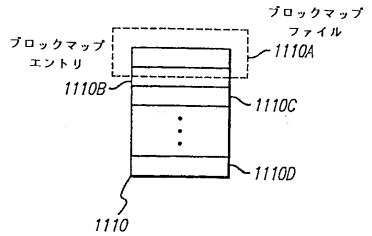


FIG. 10

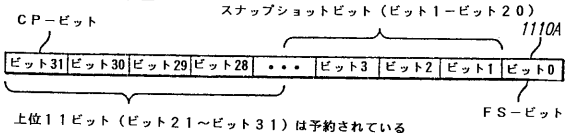
【図１１Ａ】

FIG. 11A



【図１１Ｂ】

FIG. 11B



【図１１Ｄ】

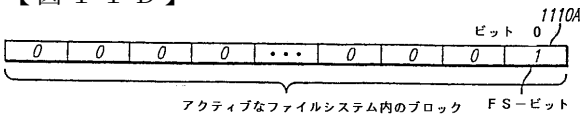


FIG. 11D

【図１１Ｃ】

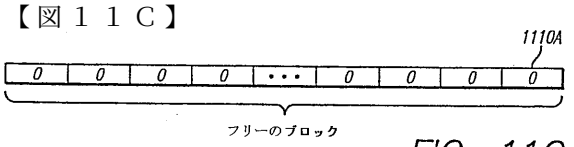


FIG. 11C

【図１２】

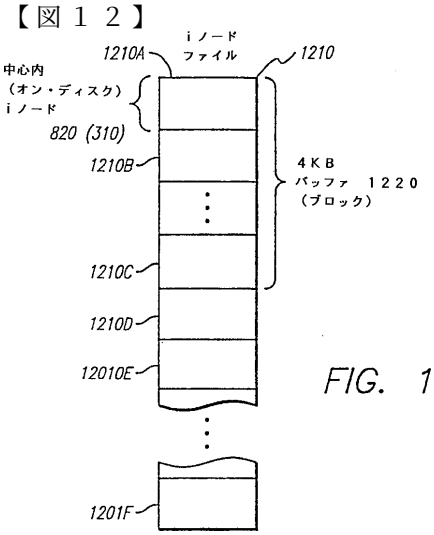
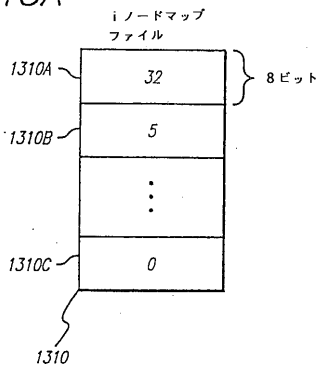


FIG. 12

【図１３Ａ】

FIG. 13A



【図１３Ｂ】

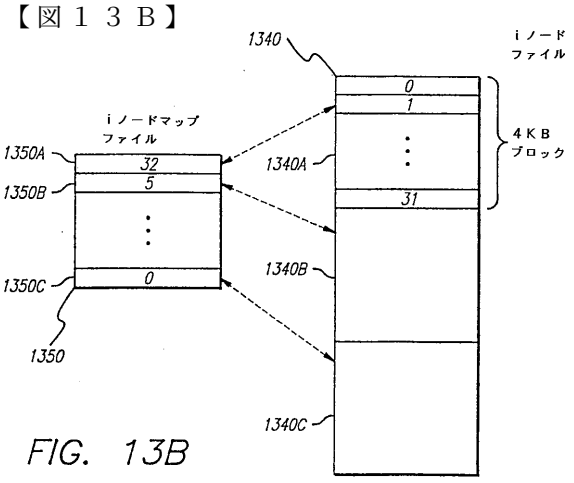


FIG. 13B

【図１４】

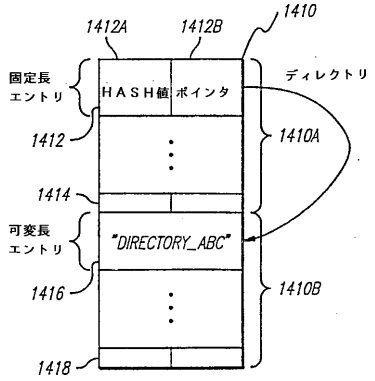


FIG. 14

【図１５】

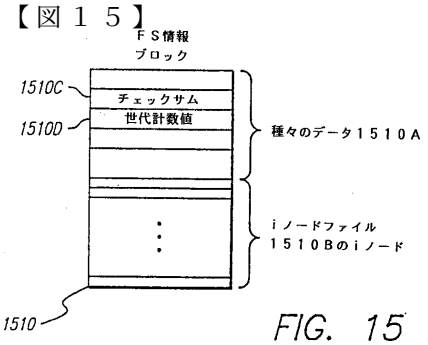


FIG. 15

【図１７Ａ】

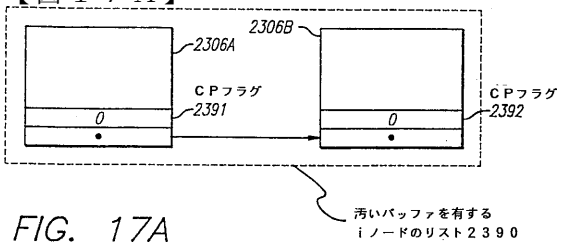


FIG. 17A

【図１７Ｉ】

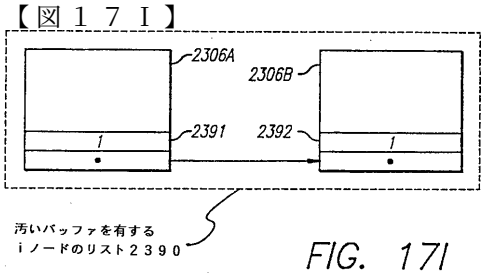


FIG. 17I

【図１６】

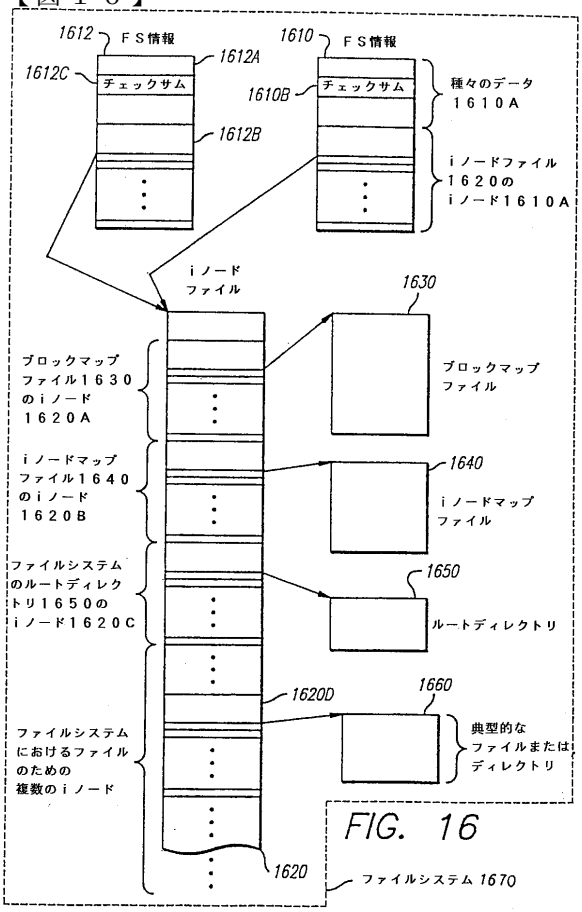


FIG. 16

【図１７Ｂ】

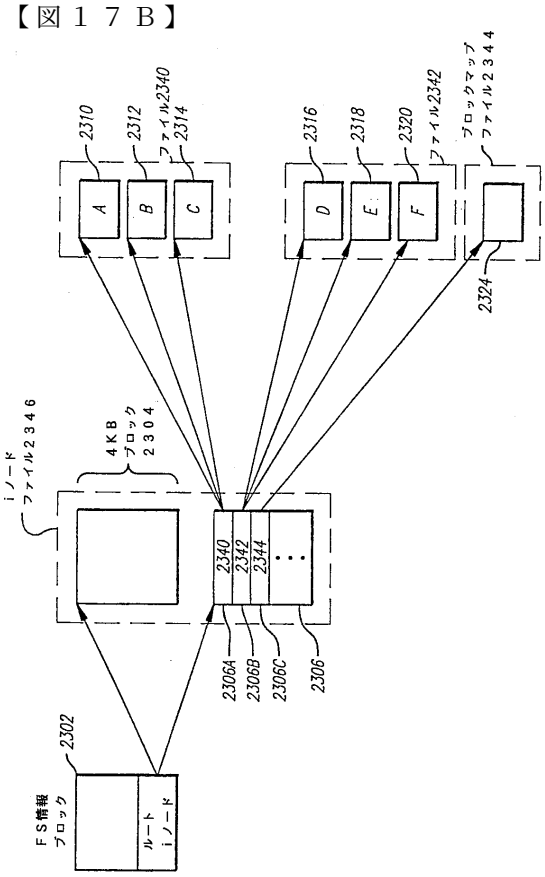


FIG. 17B

【図１７Ｃ】

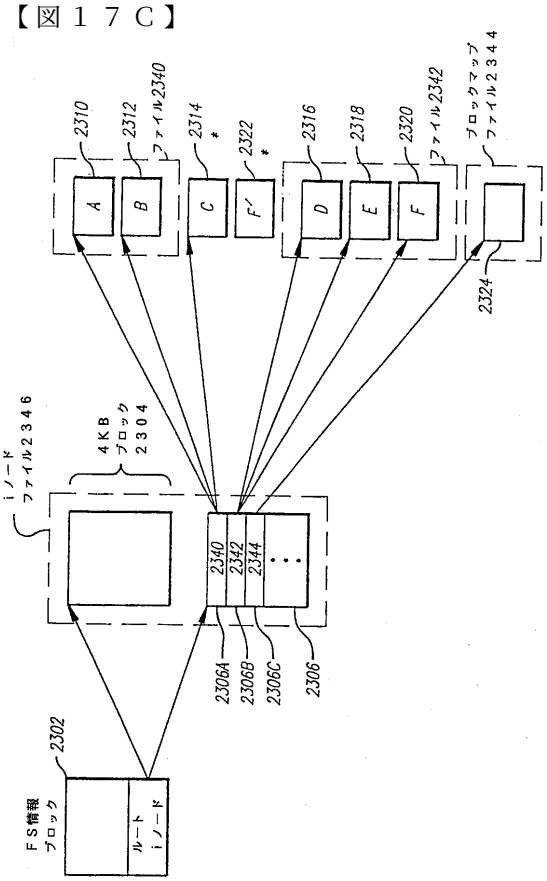


FIG. 17C

【図１７Ｄ】

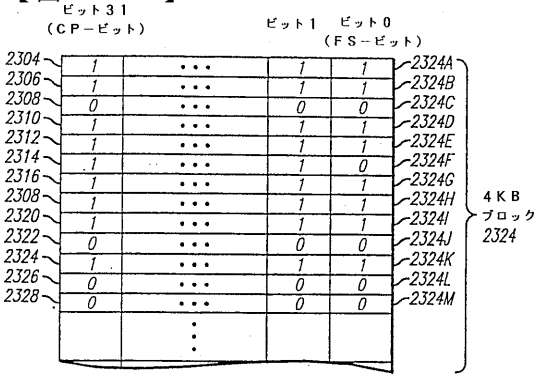


FIG. 17D

【図１７Ｅ】

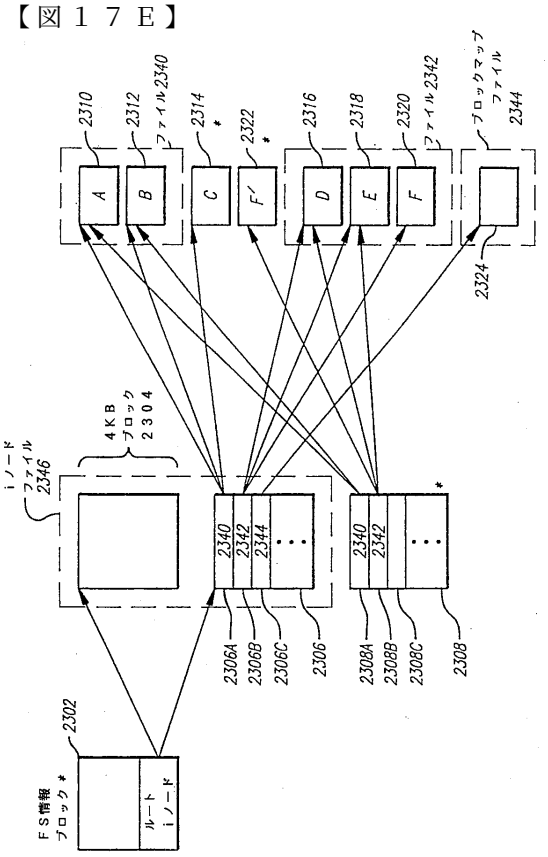


FIG. 17E

【図１７Ｊ】

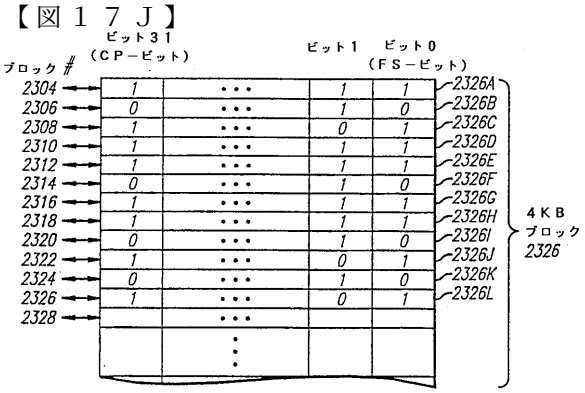


FIG. 17J

【図１７Ｆ】

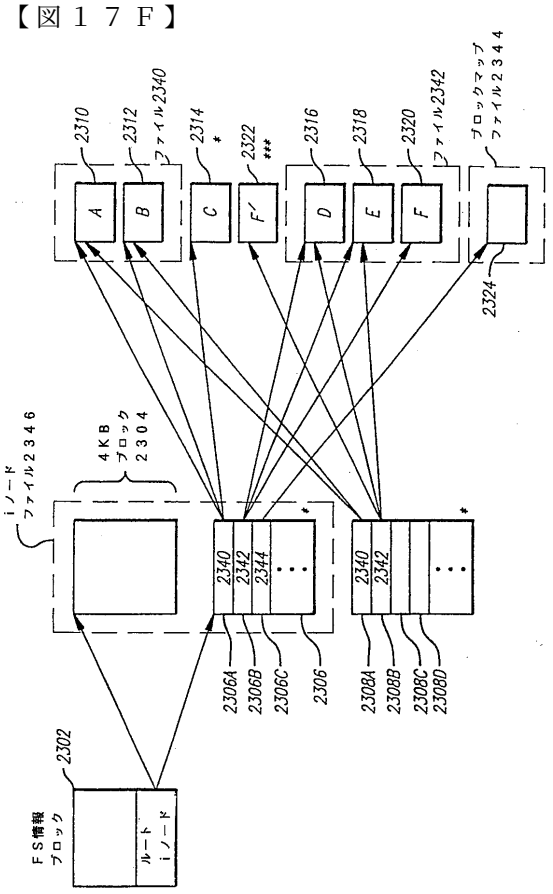


FIG. 17F

【図１７Ｇ】

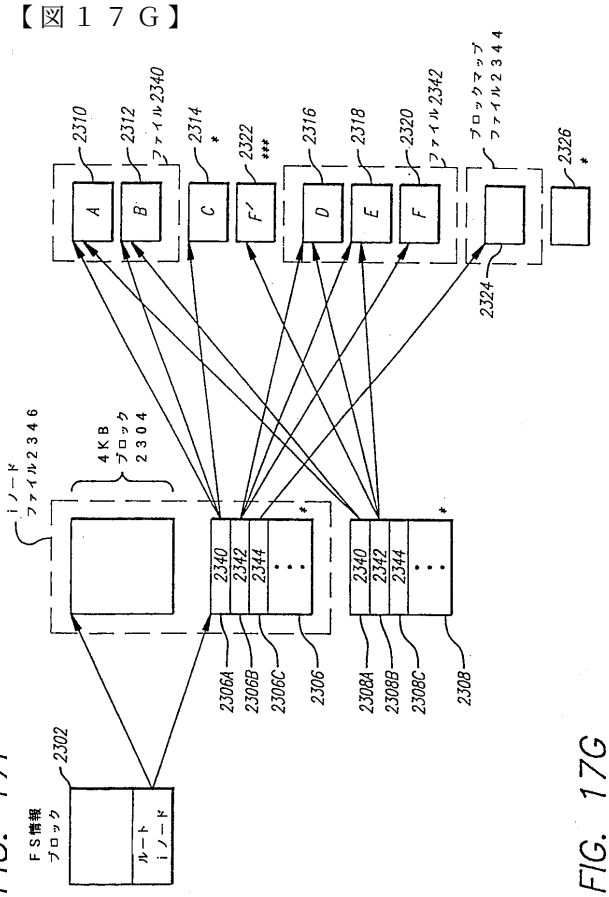


FIG. 17G

【図１７Ｈ】

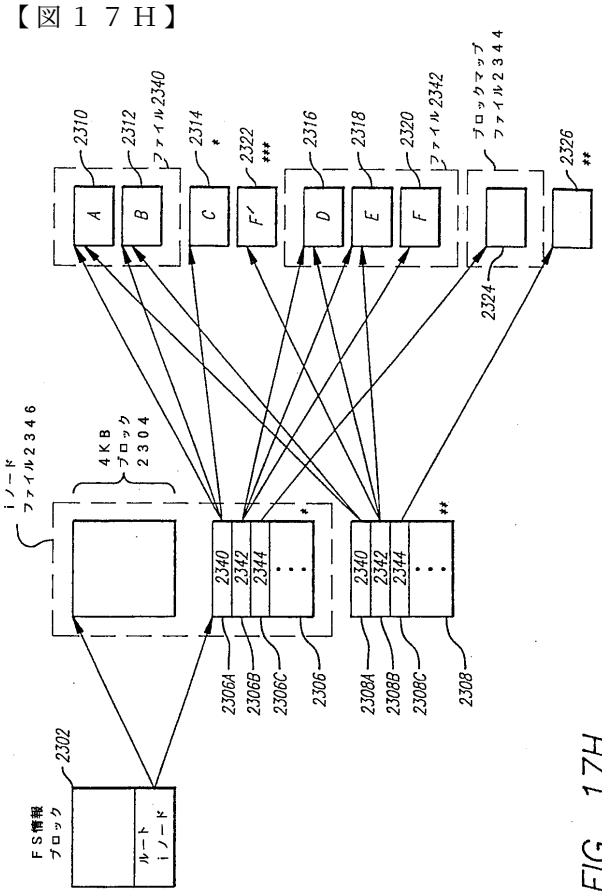


FIG. 17H

【図１７Ｋ】

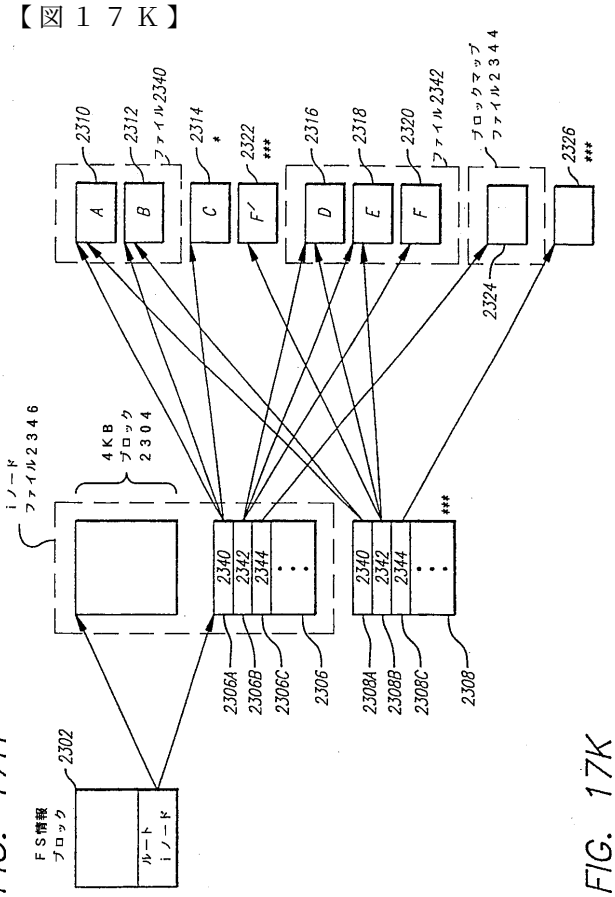


FIG. 17K

【図１７Ｌ】

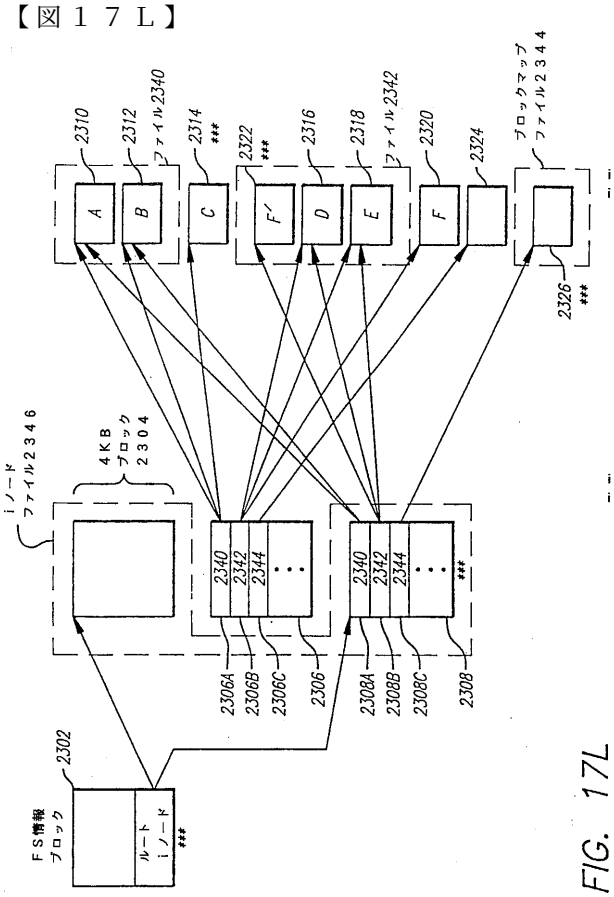
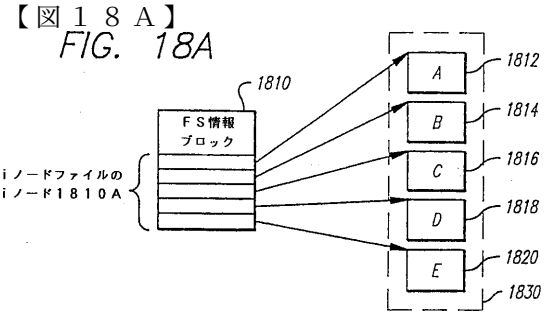


FIG. 17L

【図１８Ａ】

FIG. 18A



【図１８Ｂ】

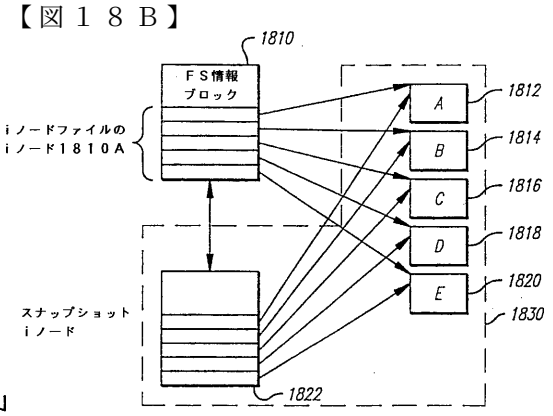
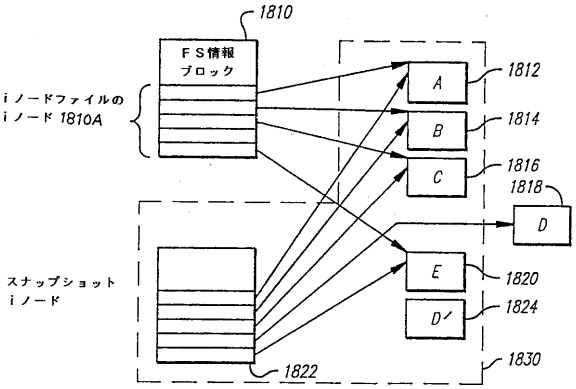


FIG. 18B

【図１８Ｃ】
FIG. 18C



【図１９】

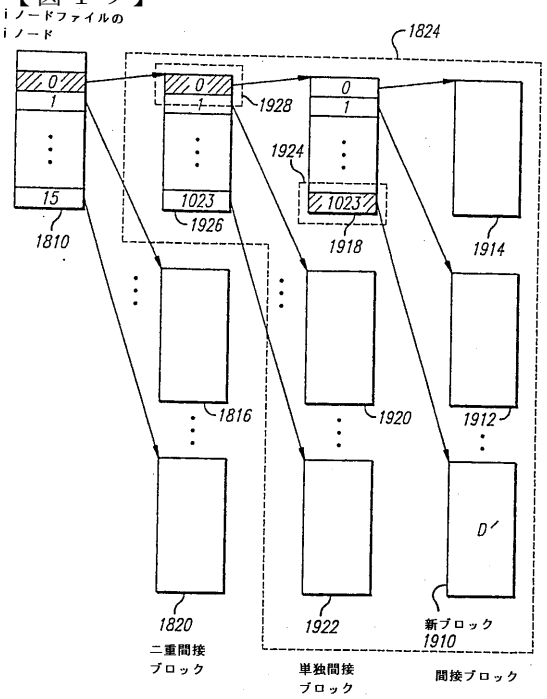
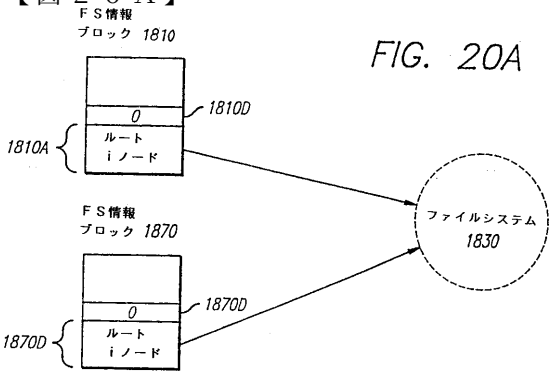
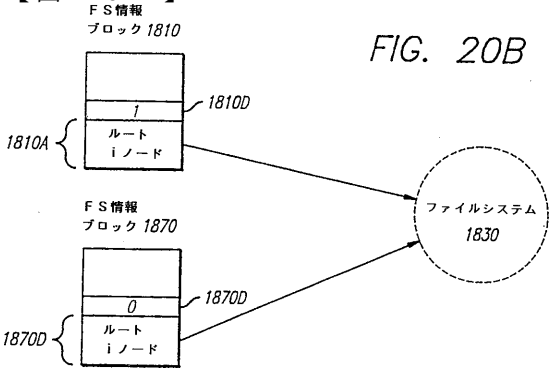


FIG. 19

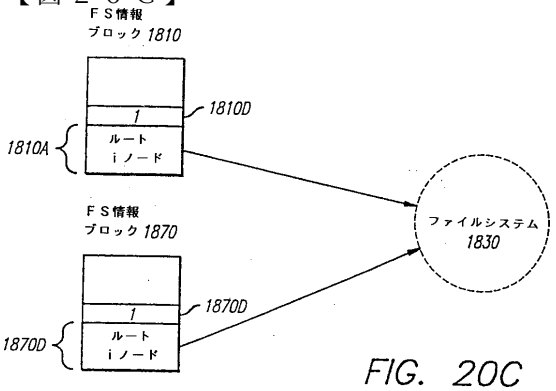
【図２０Ａ】
FIG. 20A



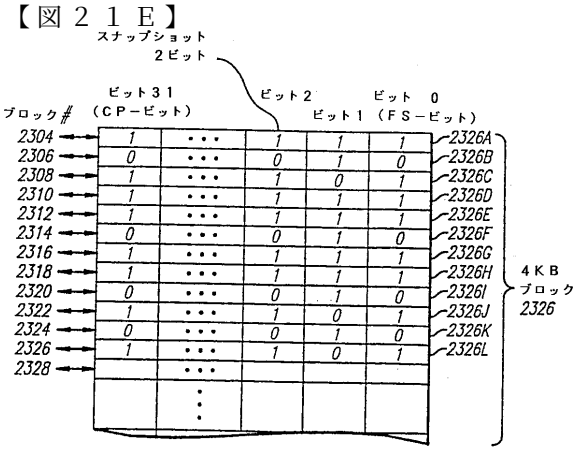
【図２０Ｂ】
FIG. 20B



【図２０Ｃ】
FIG. 20C



【図２１Ｅ】
FIG. 21E



【図２１Ａ】

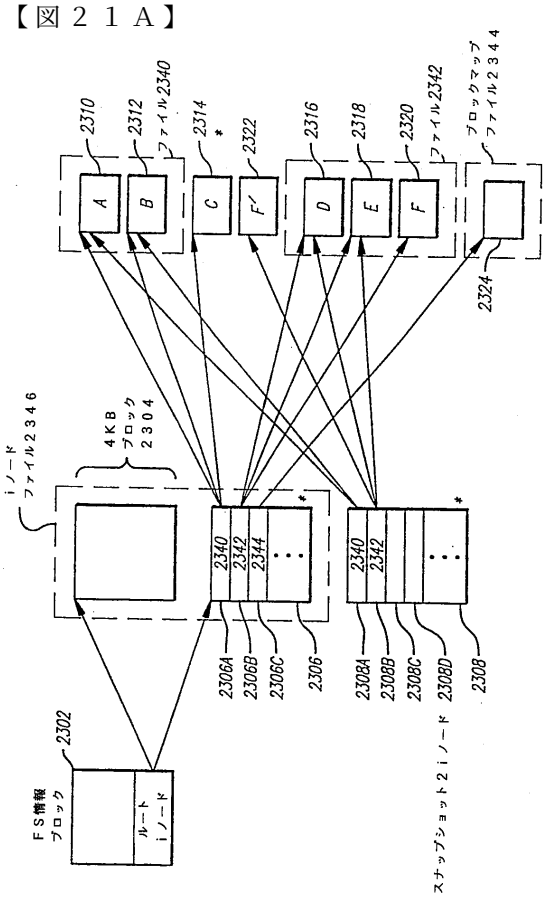


FIG. 21A

【図２１Ｂ】

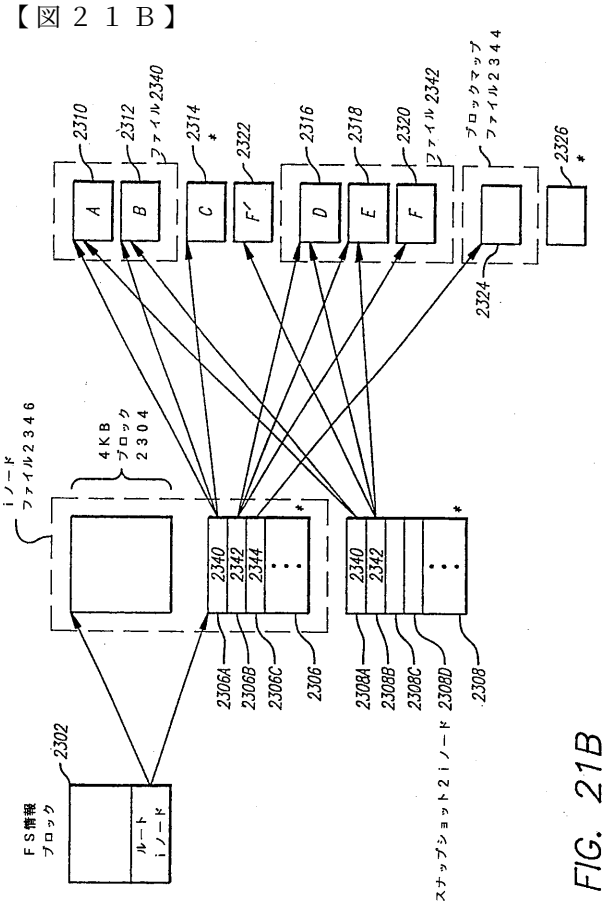


FIG. 21B

【図２１Ｃ】

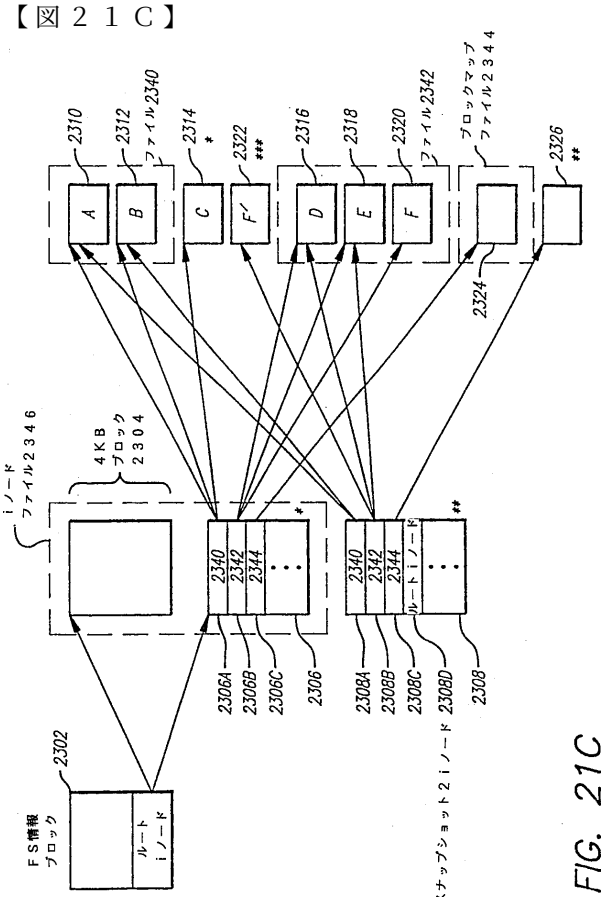


FIG. 21C

【図２１Ｄ】

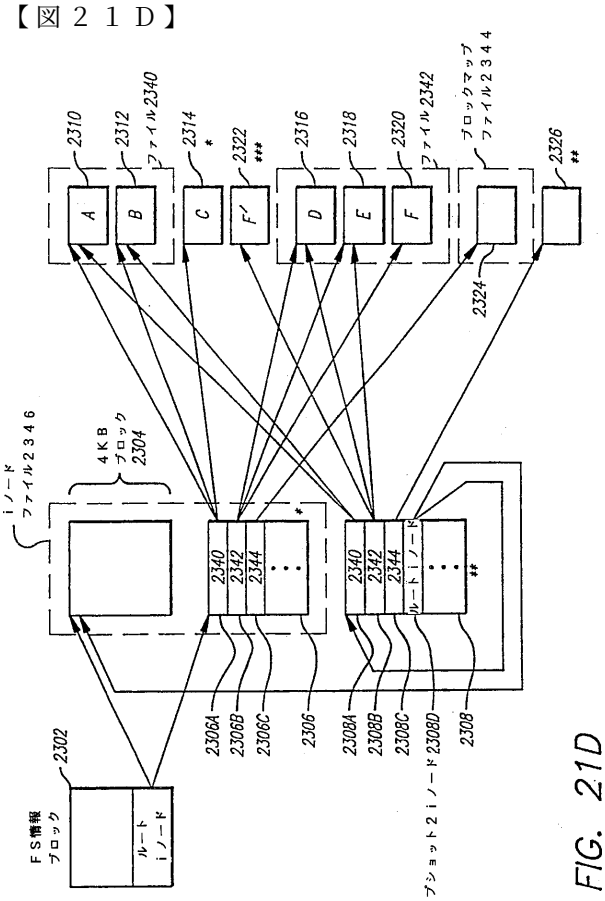


FIG. 21D

【図２１Ｆ】

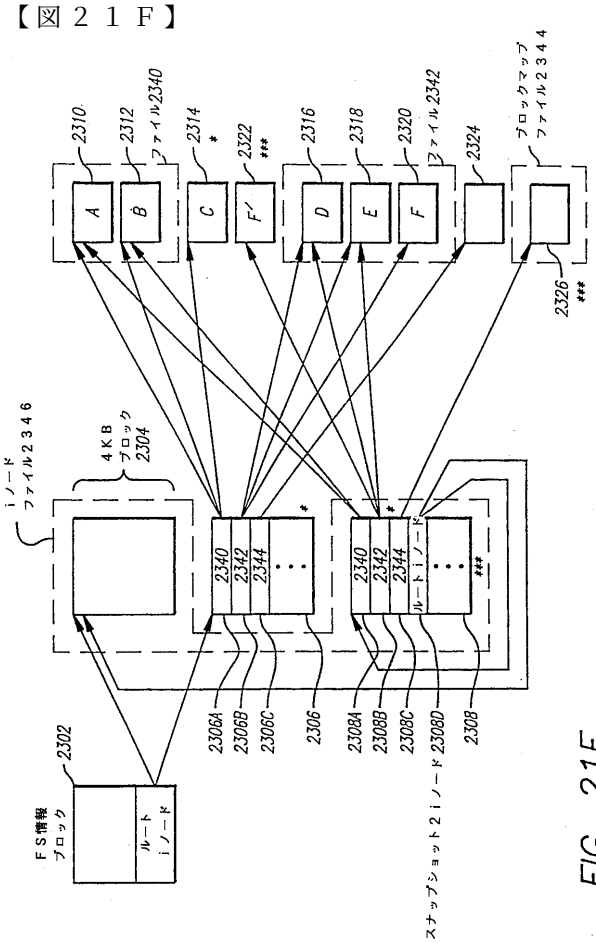


FIG. 21F

【図２２】

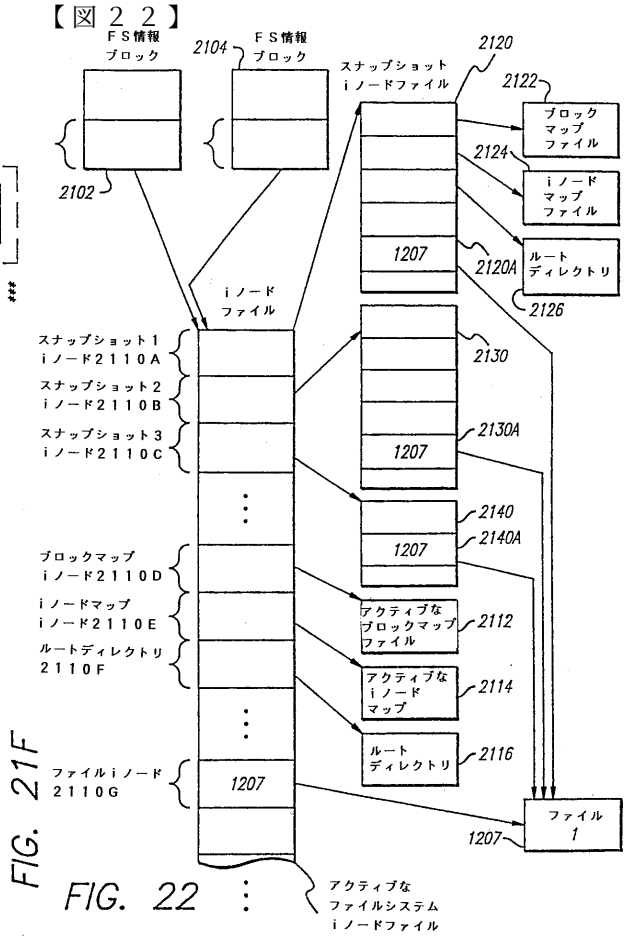


FIG. 22

【図２３Ａ】

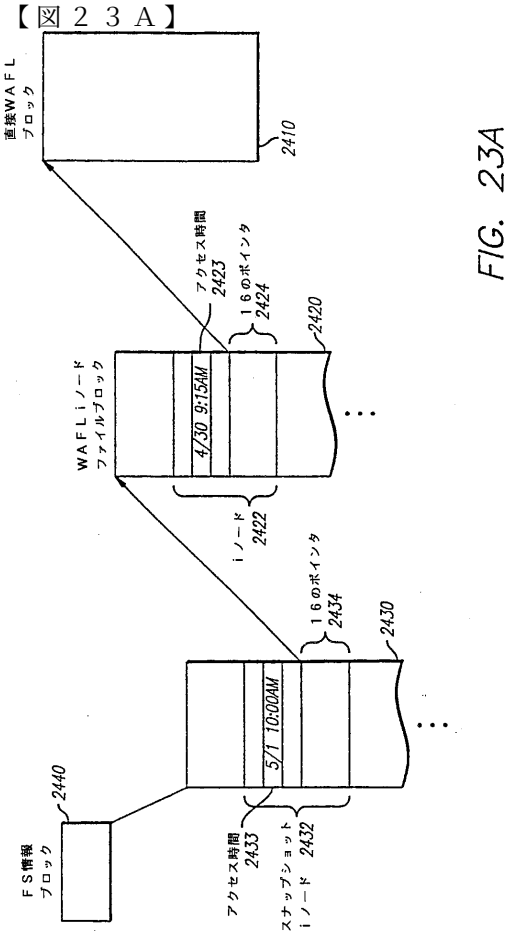


FIG. 23A

【図２３Ｂ】

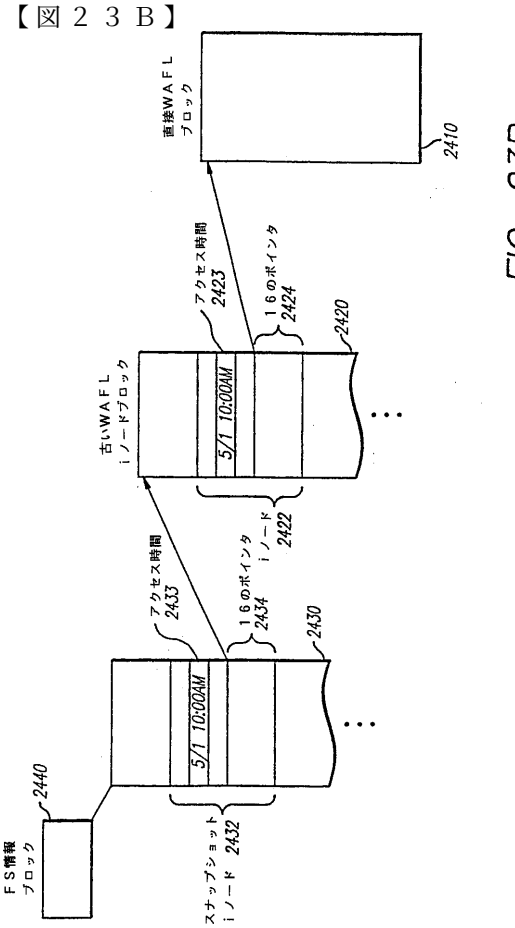


FIG. 23B

フロントページの続き

(72)発明者 ヒッツ、デイビッド
アメリカ合衆国カリフォルニア州９４０８６、サニーベイル、ウォルフェ・ロード・ナンバー２３、９２５番
(72)発明者 マルコム、マイケル
アメリカ合衆国カリフォルニア州９４０２２、ロス・アルトス、サウス・アバロン・ドライブ４８番
(72)発明者 ラウ、ジェイムズ
アメリカ合衆国カリフォルニア州９５０１４、カパチノ、アップランド・ウェイ１１５７０番
(72)発明者 ラキチス、バイロン
アメリカ合衆国カリフォルニア州９４０４３、マウンテン・ビュー、ノース・ウィッシマン・ナンバー１３０、１００番

審査官 相崎 裕恒

(56)参考文献 Chutani S. et al., The Episode File System , Proceedings of the USENIX Winter 1992 Technical Conference, 米国, USENIX, １９９２年, p.43-60

(58)調査した分野(Int.Cl., ＤＢ名)
G06F 12/00,17/30