

(19) 日本国特許庁(JP)

(12) 公開特許公報(A)

(11) 特許出願公開番号

特開2007-242019

(P2007-242019A)

(43) 公開日 平成19年9月20日(2007.9.20)

(51) Int. Cl.

G06F 3/06 (2006.01)

F I

G06F 3/06 301Z

テーマコード (参考)

5B065

審査請求 有 請求項の数 10 O L (全 44 頁)

(21) 出願番号 特願2007-55047 (P2007-55047)
 (22) 出願日 平成19年3月6日(2007.3.6)
 (31) 優先権主張番号 11/369653
 (32) 優先日 平成18年3月7日(2006.3.7)
 (33) 優先権主張国 米国 (US)

(71) 出願人 503003854
 ヒューレット・パカード デベロップメント カンパニー エル. ピー.
 アメリカ合衆国 テキサス州 77070
 ヒューストン 20555 ステイト
 ハイウェイ 249
 (74) 代理人 110000039
 特許業務法人アイ・ピー・エス
 (72) 発明者 ジェームズ・エム・ロイター
 アメリカ合衆国カリフォルニア州 パロアルト
 ハノーバー・ストリート 3000
 ヒューレット・パカード・カンパニー
 内

Fターム(参考) 5B065 BA06 CA30 CC03

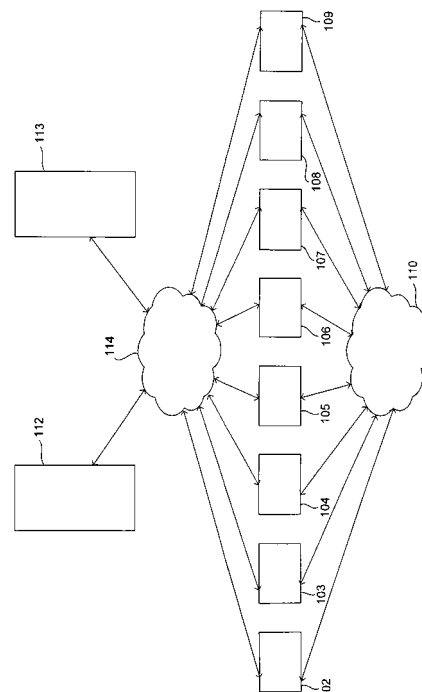
(54) 【発明の名称】 分散データの階層的管理のための方法及びシステム

(57) 【要約】

【課題】分散データの階層的管理のための方法及びシステムを提供する。

【解決手段】コンポーネントデータストレージシステムが、冗長性方式のマイグレーションの期間中、異なる冗長性方式の下で特定のデータセグメントの部分を記憶することを可能にするために、階層的データ構造体と、階層的データ構造体を並行にする階層的コーディネータルーチンと、コンポーネントデータストレージシステムの役割を含むブロックアドレス指定方式とを使用する。階層的制御ロジックは、最上位レベルコーディネータと、仮想ディスクイメージレベルコーディネータと、セグメント構成ノードレベルコーディネータと、構成グループレベルコーディネータと、構成レベルコーディネータとを備える。

【選択図】 図1



【特許請求の範囲】

【請求項 1】

ネットワーク接続されたコンポーネントデータストレージシステムで構成される分散データストレージシステム（102～109）の各コンポーネントデータストレージシステム内の階層的制御ロジックであって、前記ネットワーク接続されたコンポーネントデータストレージシステム上には、仮想ディスク（904～907）がセグメントの粒度で分散され、該仮想ディスクは、データブロックで構成されたデータセグメント（916、920）で構成されて、任意に仮想ディスクイメージ（908～910）として複製され、各データセグメントは、構成（1124）に従って分散され、

最上位レベルコーディネータ（2802）と、

仮想ディスクイメージレベルコーディネータ（2806）と、

セグメント構成ノードレベルコーディネータ（2810）と、

構成グループレベルコーディネータ（2814）と、

構成レベルコーディネータ（2820）と

を備える階層的制御ロジック。

10

【請求項 2】

前記種々のコーディネータ（2802、2806、2810、2814、2820）のそれぞれは、そのコーディネータのレベルに関連付けられるストレレジスタモデルベース一貫性方法を実行し、

ストレレジスタモデルベース一貫性方法は、

クォーラムベースのストレレジスタモデルベース一貫性方法と、

m-クォーラムベースのストレレジスタモデルベース一貫性方法と、

全体ベースのストレレジスタモデルベース一貫性方法と

を含み、

前記最上位レベルコーディネータ（2802）は、前記分散データストレージシステムのデータ状態を記述する階層的データストレージモデルの仮想ディスクレベルに関連付けられる

請求項 1 に記載の階層的制御ロジック。

20

【請求項 3】

前記階層的データストレージモデルの前記仮想ディスクレベルは、

仮想ディスクイメージを表すエントリーを含む仮想ディスクテーブル

を含み、

前記最上位レベルコーディネータは、仮想ディスクへのアクセスを管理し、

前記仮想ディスクイメージレベルコーディネータ（2806）は、前記分散データストレージシステムのデータ状態を記述する階層的データストレージモデルの仮想ディスクイメージレベルに関連付けられ、

各仮想ディスクイメージテーブルは、一般に地理的に同じ場所に配置されたコンポーネントデータストレージシステムのサブセットに記憶された仮想ディスクの 1 つの複製を表し、

各仮想ディスクイメージテーブルは、

仮想ディスクセグメントを表すエントリー

を含み、

前記仮想ディスクイメージレベルコーディネータは、仮想ディスクイメージへのアクセスを管理する

請求項 2 に記載の階層的制御ロジック。

30

【請求項 4】

前記セグメント構成ノードレベルコーディネータ（2810）は、前記分散データストレージシステムのデータ状態を記述する階層的データストレージモデルのセグメント構成ノードレベルに関連付けられ、

前記セグメント構成ノードレベルコーディネータは、全体ベースのストレレジスタ

40

50

モデルベース一貫性を使用し、

前記階層的データストレージモデルの前記セグメント構成ノードレベルは、
複数のセグメント構成ノード
を含み、

各セグメント構成ノードは、1つ又は複数の冗長性方式に従って複数のコンポーネント
データストレージシステム上に分散される1つ又は複数の仮想ディスクセグメントを表し

、
前記セグメント構成ノードレベルコーディネータは、第1の冗長性方式から第2の冗長
性方式への仮想ディスクセグメントのマイグレーションを管理する

請求項2に記載の階層的制御ロジック。

10

【請求項5】

前記構成グループレベルコーディネータ(2814)は、前記分散データストレージシ
ステムのデータ状態を記述する階層的データストレージモデルの構成グループレベルに関
連付けられ、ミラーリング冗長性方式を使用する構成グループにはクォーラムベースの技
法を使用し、イレージャ符号化冗長性方式を使用する構成グループにはm-クォーラムベ
ースの技法を使用し、

前記階層的データストレージモデルの前記構成グループレベルは、
複数の構成グループデータ構造体要素
を含み、

各構成グループデータ構造体要素は、冗長性方式に従った分散構成で複数のコンポーネ
ントデータストレージシステム上に分散される1つ又は複数の仮想ディスクセグメントを
表し、

20

前記構成グループレベルコーディネータは、仮想ディスクセグメントの再構成を管理す
る

請求項2に記載の階層的制御ロジック。

【請求項6】

前記構成レベルコーディネータ(2820)は、前記分散データストレージシステムの
データ状態を記述する階層的データストレージモデルの構成レベルに関連付けられ、ミラ
ーリング冗長性方式を使用する構成にはクォーラムベースの技法を使用し、イレージャ符
号化冗長性方式を使用する構成にはm-クォーラムベースの技法を使用し、

30

前記階層的データストレージモデルの前記構成レベルは、
複数の構成データ構造体要素
を含み、

各構成データ構造体要素は、冗長性方式に従った分散構成で複数のコンポーネントデー
タストレージシステム上に分散された1つ又は複数の仮想ディスクセグメントを表し、

前記構成レベルコーディネータは、コンポーネントデータストレージシステムの健全性
ステータスを決定し、コンポーネントデータストレージシステムの故障のハンドリングを
容易にする

請求項2に記載の階層的制御ロジック。

【請求項7】

40

ネットワーク接続されたコンポーネントデータストレージシステムで構成される分散デ
ータストレージシステム(102~109)内のデータを管理するための方法であって、

コンポーネントデータストレージシステムにわたってセグメントの粒度でデータを分散
させることであって、該データは、仮想ディスク(904~907)内に階層的に編成され、
該仮想ディスクは、データセグメント(916、920)から構成されて、任意に仮想
ディスクイメージ(908~910)として複製され、前記データセグメントはデータ
ブロックで構成される、分散させることと、

前記分散されたデータのデータ状態を表す階層的データ構造体を保持することであって
、該階層的データ構造体は、仮想ディスクレベル、仮想ディスクイメージレベル、セグメ
ント構成ノードレベル、構成グループレベル、及び構成レベルを含む、保持することと、

50

階層的に順序付けられたコーディネータールーチン（２８０２、２８０６、２８１０、２８１４、２８２０）を実行することであって、各コーディネータールーチンは、階層的データ構造体レベルに関連付けられ、各階層的コーディネータールーチンは、その階層的コーディネータールーチンが関連付けられる前記階層的データ構造体レベルによって表される階層レベルにおけるデータアクセス及びデータ一貫性を管理する、実行することを含む方法。

【請求項８】

前記階層的コーディネータールーチンは、
最上位レベルコーディネータ（２８０２）と、
仮想ディスクイメージレベルコーディネータ（２８０６）と、
セグメント構成ノードレベルコーディネータ（２８１０）と、
構成グループレベルコーディネータ（２８１４）と、
構成レベルコーディネータ（２８２０）と
を含む
請求項７に記載の方法。

10

【請求項９】

前記異なるコーディネータのそれぞれは、該コーディネータのレベルに関連付けられたストレージレジスタモデルベース一貫性方法を実行し、
ストレージレジスタモデルベース一貫性方法は、
クォーラムベースのストレージレジスタモデルベース一貫性方法と、
m-クォーラムベースのストレージレジスタモデルベース一貫性方法と、
全体ベースのストレージレジスタモデルベース一貫性方法と
を含み、

20

前記最上位レベルコーディネータ（２８０２）は、前記分散データストレージシステムのデータ状態を記述する階層的データストレージモデルの仮想ディスクレベルに関連付けられ、

前記階層的データストレージモデルの前記仮想ディスクレベルは、
仮想ディスクイメージを表すエントリーを含む仮想ディスクテーブル
を含み、

前記最上位レベルコーディネータは、仮想ディスクへのアクセスを管理し、

30

前記仮想ディスクイメージレベルコーディネータ（２８０６）は、前記分散データストレージシステムのデータ状態を記述する階層的データストレージモデルの仮想ディスクイメージレベルに関連付けられ、

前記階層的データストレージモデルの前記仮想ディスクイメージレベルは、
複数の仮想ディスクイメージテーブル
を含み、

各仮想ディスクイメージテーブルは、一般に地理的に同じ場所に配置されたコンポーネントデータストレージシステムのサブセットに記憶された仮想ディスクの１つの複製を表し、

各仮想ディスクイメージテーブルは、
仮想ディスクセグメントを表すエンティティ
を含み、

40

前記仮想ディスクイメージレベルコーディネータは、仮想ディスクイメージへのアクセスを管理し、

前記セグメント構成ノードレベルコーディネータ（２８１０）は、前記分散データストレージシステムのデータ状態を記述する階層的データストレージモデルのセグメント構成ノードレベルに関連付けられ、全体ベースのストレージレジスタモデルベース一貫性を使用し、

前記階層的データストレージモデルの前記セグメント構成ノードレベルは、
複数のセグメント構成ノード

50

を含み、

各セグメント構成ノードは、1つ又は複数の冗長性方式に従って複数のコンポーネントデータストレージシステム上に分散された1つ又は複数の仮想ディスクセグメントを表し、

前記セグメント構成ノードレベルコーディネータは、第1の冗長性方式から第2の冗長性方式への仮想ディスクセグメントのマイグレーションを管理する

請求項8に記載の方法。

【請求項10】

前記構成グループレベルコーディネータ(2814)は、前記分散データストレージシステムのデータ状態を記述する階層的データストレージモデルの構成グループレベルに関連付けられ、ミラーリング冗長性方式を使用する構成グループにはクォーラムベースの技法を使用し、イレージャ符号化冗長性方式を使用する構成グループにはm-クォーラムベースの技法を使用し、

前記階層的データストレージモデルの前記構成グループレベルは、複数の構成グループデータ構造体要素

を含み、

各構成グループデータ構造体要素は、冗長性方式に従った分散構成で複数のコンポーネントデータストレージシステム上に分散された仮想ディスクセグメントを表し、

前記構成グループレベルコーディネータは、仮想ディスクセグメントの再構成を管理し

前記構成レベルコーディネータ(2820)は、前記分散データストレージシステムのデータ状態を記述する階層的データストレージモデルの構成レベルに関連付けられ、ミラーリング冗長性方式を使用する構成にはクォーラムベースの技法を使用し、イレージャ符号化冗長性方式を使用する構成にはm-クォーラムベースの技法を使用し、

前記階層的データストレージモデルの前記構成レベルは、複数の構成データ構造体要素

を含み、

各構成データ構造体要素は、冗長性方式に従った分散構成で複数のコンポーネントデータストレージシステム上に分散された仮想ディスクセグメントを表し、

前記構成レベルコーディネータは、コンポーネントデータストレージシステムの健全性ステータスを決定し、コンポーネントデータストレージシステムの故障のハンドリングを容易にする

請求項9に記載の方法。

【発明の詳細な説明】

【技術分野】

【0001】

本発明は、分散データの階層的管理のための方法及びシステムに関する。

【背景技術】

【0002】

コンピュータネットワークシステム及び相互接続システムが、能力、信頼性、及びスループットにおいて着実に進歩するとともに、ネットワークシステムおよび相互接続システムに基づく分散コンピューティングシステムが、それに対応して、サイズ及び能力において増大したことから、分散コンピューティング問題の理論的理解の発展において多大な進歩がなされ、その結果として、コンピューティングタスクを分散システム内に分散させるための強力且つ有益なツール及び手法の開発及び広範囲に及ぶ普及が可能になった。分散システムの開発の初期においては、大きな計算タスクの処理を分散させるために、それぞれがマストレイジデバイスを含む多数の周辺デバイスを有する大きなメインフレームコンピュータ及びミニコンピュータが、直接又はネットワークを通じて相互接続されていた。ネットワークシステムが、よりローバスト、より有能且つより経済的になるにつれて、1つ又は複数のネットワークを通じてリモートホストコンピュータと相互接

10

20

30

40

50

続された独立したディスクアレイ等の独立したマストレージデバイスが、メインフレームからパーソナルコンピュータに至るまでの多数のコンピュータシステムによって共有された大量のデータを記憶するために開発された。近年、以下で詳細に説明するように、開発努力は、1つ又は複数のネットワークによって相互接続された多数のマストレージデバイスにわたってマストレージシステムを分散させる方向に向けられ始めてきている。

【0003】

マストレージデバイスは、単一のコンピュータシステムに別個に取り付けられて単一のコンピュータシステムにより制御される周辺デバイスから、リモートホストコンピュータによって共有される独立したデバイスへ発展し、最終的には、相互にネットワーク接続された多数の別個のマストレージユニットで構成された分散システムへ発展したことから、データを共有することに関連した問題、及び、一貫性があり且つローバスタな状態で共有データを保持することに関連した問題が劇的に増加している。分散システムの設計者、開発者、製造業者、ベンダ、及び最終的にはユーザは、すでに開発された分散コンピューティング方法及びルーチンを拡張する必要性を認識し続け、より大きく、より複雑で且つより高度に分散されたシステムにおいて所望のレベルのデータローバスタ性及び一貫性を提供する新しい方法及びルーチンの必要性を認識し続けている。

10

【発明の開示】

【発明が解決しようとする課題】

【0004】

本発明は、分散データの階層的管理のための方法及びシステムを提供することを目的とする。

20

【課題を解決するための手段】

【0005】

本発明の種々の方法及びシステムの実施の形態は、ネットワーク接続されたコンポーネントデータストレージシステムで構成される分散データストレージシステムの各コンポーネントデータストレージシステム内の階層的制御ロジックであって、ネットワーク接続されたコンポーネントデータストレージシステム上には、仮想ディスクがセグメントの粒度で分散され、この仮想ディスクは、データブロックで構成されたデータセグメントで構成されて、任意に仮想ディスクイメージとして複製される、階層的制御ロジックを対象とする。各データセグメントは、構成に従って分散される。階層的制御ロジックは、本発明の一実施の形態では、最上位レベルコーディネータと、仮想ディスクイメージレベルコーディネータと、セグメント構成ノードレベルコーディネータと、構成グループレベルコーディネータと、構成レベルコーディネータとを備える。

30

【発明を実施するための最良の形態】

【0006】

本発明のさまざまな方法及びシステムの実施の形態は、コンポーネントデータストレージシステムが、冗長性方式のマイグレーションの期間中、異なる冗長性方式の下で特定のデータセグメントの部分を記憶することを可能にするために、階層的データ構造体と、階層的データ構造体を並行にする階層的コーディネータルーチンと、コンポーネントデータストレージシステムの役割を含むブロックアドレス指定方式とを使用する。以下では、現在開発中の分散マストレージデバイスを背景として本発明の実施の形態を説明する。以下の小節では、分散マストレージシステムのコンポーネント及び機能、並びに、分散マストレージシステムの処理コンポーネントによって使用されるさまざまな方法を使用して、本発明のさまざまな実施の形態が説明される。

40

【0007】

〔FAB概論〕

ブリックの連合アレイ（「FAB (federated array of bricks)」）アーキテクチャは、マストレージの新しい、高度に分散した手法を表す。図1は、本発明の一実施の形態によるFABマストレージシステムの高レベルの図を示している。FABマストレージシステムを以下では「FABシステム」と呼ぶ。FABシステムは、複数の小さな別

50

個のコンポーネントデータストレージシステム、すなわちマスタストレージデバイス102～109を備える。これらのマスタストレージデバイス102～109は、第1の通信媒体110を通じて互いに相互通信を行い、第2の通信媒体114を通じて、複数のリモートホストコンピュータ112～113から要求を受信することができ、複数のリモートホストコンピュータ112～113へ返答を送信することができる。各別個のコンポーネントデータストレージシステム102～109は、「ブリック (brick)」と呼ばれる場合がある。ブリックは、インターフェースを含むことができ、このインターフェースを通じて、リモートホストコンピュータから要求を受信することができ、受信された要求に対する応答をリモートホストコンピュータへ返信することができる。FABシステムのいずれのブリックも、ホストコンピュータからの要求を受信することができ、この要求に応答することができる。FABシステムの1つのブリックは、任意の特定の要求に関して調整役を引き受け、その特定の要求に対する応答に関与するすべてのブリックのオペレーションを調整し、FABシステムのどのブリックも、所与の要求に関する調整役を引き受けることができる。したがって、FABシステムは、大部分がソフトウェアで実施される一種の対称分散コンピューティングシステムである。いくつかの代替的な実施の形態では、ブリックの相互接続、及び、FABシステムのリモートホストコンピュータへの相互接続の双方に単一のネットワークを使用することができる。他の代替的な実施の形態では、3つ以上のネットワークを使用することもできる。

10

【0008】

図2は、本発明の一実施の形態による一例示のFABブリックの高レベルの図を示している。図2に示すFABブリックは、ディスクI/Oプロセッサ214にインターフェースしている12個のSATAディスクドライブ202～213を含む。ディスクI/Oプロセッサ214は、1つ又は複数の高速バス216を通じて中央ブリッジデバイス218に相互接続されている。次に、中央ブリッジ218は、1つ又は複数の汎用プロセッサ220、ホストI/Oプロセッサ222、相互ブリックI/Oプロセッサ224、及び1つ又は複数のメモリ226～228に相互接続されている。ホストI/Oプロセッサ222は、第2の通信媒体(図1の114)への通信インターフェースを提供する。この通信インターフェースを通じて、ブリックは、リモートホストコンピュータと通信する。相互ブリックI/Oプロセッサ224は、第1の通信媒体(図1の110)への通信インターフェースを提供する。この通信インターフェースを通じて、ブリックは、FABの他のブリックと通信する。1つ又は複数の汎用プロセッサ220は、多くのタスク及び担当の中で、リモートホストコンピュータ及びリモートブリックからの要求の処理、1つ又は複数のメモリ226～228及びストレージデバイス202～213に記憶されている状態情報の管理、並びにブリック内のデータストレージ及びデータ一貫性の管理を行うための制御プログラムを実行する。1つ又は複数のメモリは、データのキャッシュとしての機能に加えて、FABシステム内に記憶されたデータへのアクセスを制御するプロセス、及び、FABシステム内のデータを一貫性のある状態に維持する制御プロセスによって使用される、タイムスタンプ及びデータ構造体を含むさまざまなエンティティのストレージロケーションとしての機能を果たす。これらのメモリには、通常、揮発性メモリ及び不揮発性メモリの双方が含まれる。以下の解説では、1つ又は複数の汎用プロセッサ、1つ又は複数のメモリ、及び他のコンポーネントは、フレーズ「1つ又は複数」を繰り返すことを回避するために単数形で参照される場合がある。なお、上記他のコンポーネントの1つ又は複数のものは、最初に、含まれるものとして言及したものである。

20

30

40

【0009】

本発明の一定の実施の形態では、FABのすべてのブリックは、基本的に同一であり、同じ制御プログラムを実行し、それらのメモリ226及びマスタストレージデバイス202～213内に基本的に同じデータ構造体及び制御情報を保持し、I/Oプロセッサを通じてホストコンピュータ、FAB内の他のブリック、及び内部ディスクドライブへ標準インターフェースを提供する。本発明のこれらの実施の形態では、FAB内のブリックは、制御プログラムのバージョン、内部ディスクドライブの特定のモデル及び能力、さまざまな

50

ハードウェアコンポーネントのバージョン、並びに他のこのような変形に関して互いにわずかに異なってもよい。インターフェース及び制御プログラムは、このような変形をFAB内で許容することを可能にするために、下位互換性及び上位互換性の双方で設計される。

【0010】

また、各ブリックは、図2に図示しない他の多数のコンポーネントも含むことができる。これら他の多数のコンポーネントには、1つ又は複数の電源、冷却システム、制御パネル又は他の外部制御インターフェース、標準的なランダムアクセスメモリ、及び他のこのようなコンポーネントが含まれる。ブリックは、比較的簡単なデバイスであり、一般に、コモディティI/Oプロセッサ及びディスクドライブを含むコモディティコンポーネントから構築される。12個の100-GB SATAディスクドライブを使用するブリックは、1.2テラバイトの記憶容量を提供する。その記憶容量のわずかし内部の使用に必要とされない。FABは、数百個又は数千個のブリックを備えることができ、大きなFABシステムは、5,000個～10,000個のブリックを収容することが現在想定されており、ペタバイト(「PB」)の記憶容量を提供する。このように、FABマストレージシステムは、記憶容量及び費用効率において、現在のディスクアレイ及びネットワーク接続型ストレージデバイスを超える膨大な増加を提供する。

10

【0011】

[冗長性]

FABシステム等の大きなマストレージシステムは、膨大な記憶容量を提供するだけでなく、冗長ストレージの提供及び管理も行い、ブリックの故障、ディスクドライブの故障、ディスクドライブ上の特定のシリンダ、トラック、セクタ、若しくはブロックの故障、電子コンポーネントの故障、又はそれ以外の故障のために、記憶データの一部分が喪失した場合に、ホストコンピュータによる介入もユーザによる手動の介入もなく、大規模マストレージシステムによって記憶され管理されている冗長データから喪失データをシームレスに且つ自動的に回復できるようになっている。データベースシステム及びエンタープライズクリティカルデータ(enterprise-critical data)を含む重要なデータストレージの用途の場合、地理的に分散した複数のデータインスタンスを記憶して保持するために、2つ以上の大規模マストレージシステムが使用されることが多く、壊滅的な事象であっても回復不可能なデータ喪失につながらないように、より高いレベルの冗長性が提供されている。

20

30

【0012】

本発明の一定の実施の形態では、FABシステムは、少なくとも2つの異なるクラスの下位レベルの冗長性を自動的にサポートする。第1のクラスの冗長性は、1つのブリックの故障が回復不可能なデータ喪失につながらないように、ブリックレベルのミラーリングを含む。換言すれば、2つ以上のブリックにデータオブジェクトの複数の別個のコピーを記憶することを含む。図3～図4は、データミラーリングの概念を示している。図3は、本発明の一実施の形態によるデータオブジェクト302及び3つのブリック304～306のコンテンツの論理表現を示している。データオブジェクト302は、データユニット308等、図3の「1」～「15」の番号を付けられた15個の連続したデータユニットを備える。データオブジェクトは、ボリューム、ファイル、データベース、又は別のタイプのデータオブジェクトとすることができ、データユニットは、ブロック、ページ、又は他のこのような連続アドレス指定されるストレージロケーションのグループとすることができ、図4は、本発明の一実施の形態による3つのブリック304～306におけるデータオブジェクト302のトリプルミラーリング冗長ストレージを示している。3つのブリックのそれぞれは、データオブジェクト302内の15個のデータユニットのすべてのコピーを含む。ミラーリングの多くの説明図では、データユニットのレイアウトは、データオブジェクトのすべてのミラーコピーにおいて同一であることが示されている。しかしながら、実際には、ブリックは、その内部ディスクドライブのどの場所にもデータユニットを記憶することを決めることができる。図4では、データオブジェクト302内のデー

40

50

タユニットのコピーは、3つの異なるブリック内で異なる順序及び位置に示されている。3つのブリック304～306のそれぞれは、データオブジェクトの完全なコピーを記憶するので、たとえ3つのブリックの2つが故障しても、データオブジェクトは回復可能である。単一のブリックの故障確率は、一般に比較的わずかであり、3ブリックミラーの3つすべてのブリックの故障の組み合わせ確率は、一般に極めて小さい。一般に、FABシステムは、数百万個、数十億個、数兆個、又はそれよりも多くの異なるデータオブジェクトを記憶することができ、異なる各データオブジェクトは、FABシステム内で異なる個数のブリック上に個別にミラーリングすることができる。たとえば、或るデータオブジェクトは、ブリック1、7、8、及び10上にミラーリングすることができる一方、別のデータオブジェクトは、ブリック4、8、13、17、及び20上にミラーリングすることができる。

10

【0013】

第2の冗長クラスは、「イレージャ符号化(eraser coding:抹消符号化)」冗長性と呼ばれる。イレージャ符号化冗長性は、ミラー冗長性よりも幾分複雑である。イレージャ符号化冗長性は、雑音のあるチャネルを通じて転送される通信メッセージ及び他のデジタルデータの誤り制御符号化に使用されるリード・ソロモン符号化技法を使用することが多い。これらの誤り制御符号化技法は、2元線形符号の具体的な例である。

【0014】

図5は、イレージャ符号化冗長性を示す高レベルの図を示している。図5では、 $n=4$ 個のデータユニットを備えるデータオブジェクト502が、 n よりも多くの個数のブリック504～509にわたって分散されている。最初の n 個のブリック504～507は、それぞれ、 n 個のデータユニットの1つを記憶する。最後の $m=2$ 個のブリック508～509は、データオブジェクトから計算されたチェックサムデータ又はパリティデータを記憶する。図5に示すイレージャ符号化冗長性方式は、 $m+n$ イレージャ符号化冗長性方式の一例である。 $n=4$ であり、 $m=2$ であるので、図5に示す具体的な $m+n$ イレージャ符号化冗長性方式は、「4+2」冗長性方式と呼ばれる。8+2方式、3+3方式、及び他の方式を含めて、他の多くのイレージャ符号化冗長性方式が可能である。一般に、 m は n 以下である。故障したブリックが m 個又は $m+n$ 個よりも少ない限り、それらの故障したブリックがデータを含んでいようとパリティ値を含んでいようと、データオブジェクト全体を復元することができる。たとえば、図5に示すイレージャ符号化方式では、ブリック505及び508等のどのブリック対の故障にもかかわらず、データオブジェクト502を完全に回復させることができる。

20

30

【0015】

図6は、図3及び図4で使用されたものと同じ説明図の規則を使用する一例示の3+1イレージャ符号化冗長性方式を示している。図6では、15個のデータユニットのデータオブジェクト302が、4つのブリック604～607にわたって分散されている。これらのデータユニットは、4つのディスクにわたってストライプされ、データオブジェクトの各3つのデータユニットは、ブリック604～606にわたって連続的に分散され、ストライプのチェックサムデータユニット又はパリティデータユニットは、ブリック607に配置されている。最初のストライプは、3つのデータユニット608から成り、図6において矢印610～612によって示されている。図6では、チェックサムデータユニットは、すべて単一のブリック607に配置されているが、ストライプは、ブリックに関して別のやり方で整列することができ、各ブリックが、チェックサムデータユニット又はパリティデータユニットの或る部分を含むことができる。

40

【0016】

イレージャ符号化冗長性は、一般に、データユニットの各バイト、各ワード、又は各ロングワードのチェックサムビット又はパリティビットを数学的に計算することによって実行される。したがって、 m ビットのパリティビットは、 n ビットのデータビットから計算される。ここで、 $n=8$ 、16、若しくは32、又はそれよりも大きな2のべき乗である。たとえば、8+2イレージャ符号化冗長性方式では、2ビットのパリティチェックビッ

50

トが、データの各バイトについて生成される。したがって、8 + 2 イレージャ符号化冗長性方式では、データの8つのデータユニットが、チェックサムビット又はパリティビットの2つのデータユニットを生成する。これら2つのデータユニットのすべては、10 データユニットのストライプに含めることができる。以下の解説では、用語「ワード」は、符号化が行われるデータユニットの粒度を指し、ビットからロングワード又はそれよりも長いデータユニットに変化することができる。データストレージの用途では、データユニットの粒度は、通常、512 バイト以上とすることができる。

【0017】

i 番目のチェックサムワード c_i は、関数 $F_i(d_1, d_2, \dots, d_n)$ により、n 個のすべてのデータワードの関数として計算することができる。この関数 F_i は、以下のよう 10
に、データワード d_j のそれぞれに係数 $f_{i,j}$ を乗算したものの線形結合である。

【0018】

【数1】

$$c_i = F_i(d_1, d_2, \dots, d_n) = \sum_{j=1}^n d_j f_{i,j}$$

【0019】

20

行列表記では、この式は、

【0020】

【数2】

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,n} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,n} \\ \vdots & \vdots & & \vdots \\ f_{m,1} & f_{m,2} & \cdots & f_{m,n} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}$$

30

【0021】

又は、

【0022】

【数3】

40

$$C = FD$$

【0023】

となる。リード・ソロモン技法では、関数 F は、要素 $f_{i,j}$ が j^{i-1} に等しい $m \times n$ バンデルモンド行列となるように選ばれる。すなわち、

【0024】

【数 4】

$$F = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 2 & \dots & n \\ \vdots & \vdots & & \vdots \\ 1 & 2^{m-1} & \dots & n^{m-1} \end{bmatrix}$$

10

【0025】

である。特定のワード d_j が、新しい値 d_j' を有するように変更された場合、新しい i 番目のチェックサムワード c_i' は、

$$c_i' = c_i + f_{i,j} (d_j' - d_j)$$

又は

【0026】

【数 5】

20

$$c' = C + FD' - FD = C + F(D' - D)$$

【0027】

として計算することができる。したがって、新しいチェックサムワードは、前のチェックサムワード及び行列 F の単一の列から容易に計算される。

【0028】

30

ストライプから喪失されたワードは、行列の反転によって回復される。行列 A 及び列ベクトル E は、以下のように構築される。

【0029】

【数 6】

$$A = \begin{bmatrix} I \\ F \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 3 & \dots & n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 2^{m-1} & 3^{m-1} & \dots & n^{m-1} \end{bmatrix}$$

40

50

【 0 0 3 0 】

【 数 7 】

$$\mathbf{E} = \left[\frac{\mathbf{D}}{\mathbf{C}} \right] = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ c_1 \\ c_2 \\ \vdots \\ c^m \end{bmatrix}$$

10

【 0 0 3 1 】

以下であることが容易に分かる。

20

【 0 0 3 2 】

【 数 8 】

$$\mathbf{AD} = \mathbf{E}$$

or:

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & & 1 \\ 1 & 1 & 1 & \dots & n \\ 1 & 2 & 3 & \dots & n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 2^{m-1} & 3^{m-1} & \dots & n^{m-1} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}$$

30

40

【 0 0 3 3 】

変更された行列 A' 及び E' を作成するために、行列 A の任意の m 行及びベクトル E の対応する行を削除することができる。ここで、A' は正方行列である。次に、元のデータワードを表すベクトル D は、以下のように行列の反転によって回復させることができる。

50

【0034】

【数9】

$$A'D = E'$$

$$D = A'^{-1}E'$$

【0035】

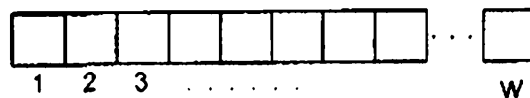
したがって、 m 個以下のデータワード又はチェックサムワードが消去又は喪失された場合、それら m 個以下の喪失されたデータワード又はチェックサムワードを含む m 個のデータワード又はチェックサムワードをベクトル E から削除することができ、上記に示したように、行列 A から対応する行が削除され、元のデータワード又はチェックサムワードは、行列の反転によって回復させることができる。

【0036】

行列の反転は、実数については、加算、減算、乗算、及び除算のよく知られている実数算術演算を使用して容易に実行されるが、デジタル誤り制御符号化に使用される離散値の行列要素及び列要素は、離散値が、対応する離散算術演算の下で閉じた算術体 (arithmetic field) を形成する場合にのみ、行列乗算に適している。一般に、チェックサムビットは、長さ w のワード

【0037】

【数10】



【0038】

について計算される。 w ビットワードは、 2^w 個の異なる値のいずれかを有することができる。ガロア体として知られている数学体 (mathematical field) は、 2^w 個の要素を有するように構築することができる。ガロア体の要素の算術演算は、好都合なことに、

【0039】

【数11】

$$a \pm b = a \oplus b$$

$$a * b = \text{anti log} [\log(a) + \log(b)]$$

$$a \div b = \text{anti log} [\log(a) - \log(b)]$$

【0040】

である。ここで、ガロア体の要素の対数及び真数の表は、 w 次原始多項式を含む伝播方法を使用して計算することができる。

【0041】

ミラー冗長性方式は、概念的にはより簡単であり、さまざまな再構成オペレーションに明らかに向いている。たとえば、3ブリックトリプルミラー冗長性方式の1つのブリックが故障した場合、ダブルミラーリング冗長性方式の下で、残りの2つのブリックを2ブリックミラー対として再構成することができる。或いは、新しいブリックを選択して、故障したブリックと取り替えることができ、残存しているブリックの1つからこの新しいブリックヘデータをコピーして、3ブリックトリプルミラー冗長性方式を復元することもでき

10

20

30

40

50

る。対照的に、イレージャ符号化冗長性方式の再構成は、それほど簡単ではない。たとえば、ストライプ内の各チェックサムワードは、そのストライプのすべてのデータワードに依存している。4+2イレージャ符号化冗長性方式を8+2イレージャ符号化冗長性方式に変換することが望まれる場合、チェックサムビットのすべてが再計算される場合があり、データは、4+2方式の6個のブリックの関連があるコンテンツを新しいロケーションにコピーするのではなく、新しい8+2方式に使用される10個のブリック上に再分散される場合がある。その上、同じイレージャ符号化方式のストライプサイズの変更であっても、チェックサムデータユニットのすべてを再計算し、それらデータを新しいブリックロケーションに再分散させることが伴う場合がある。ミラーリング冗長性方式の場合には、元のブリックから新しいブリックヘデータをコピーすることで、複数のブリックの1つのブリックの消去又は1つのブリックの追加を行うが、イレージャ符号化方式に対する変更は、ほとんどの場合、そのような消去又は追加ではなく、古い構成から取り出されたデータに基づいて新しい構成を完全に構築することを伴う。ミラーリングは、一般に、イレージャ符号化よりも空間効率がよくないが、処理サイクルの時間及び費用の効率はよい。

【0042】

[FABストレージユニット]

上述したように、FABシステムは、膨大な量のデータストレージ空間を提供することができる。全ストレージ空間は、階層的なデータユニットに論理的に区画することができる。最も低くない各階層レベルのデータユニットは、次に最も低い(next-lowest)階層レベルのデータユニットで論理的に編成される。論理データユニットは、1つ又は複数のブリック内の物理ストレージ空間にマッピングすることができる。

【0043】

図7は、本発明の一実施の形態を表す、現在のFABの実施態様で使用される階層的なデータユニットを示している。最上位レベルのデータユニットは、「仮想ディスク」と呼ばれ、FABシステム内の全利用可能ストレージ空間は、1つ又は複数の仮想ディスクに区画されているとみなすことができる。図7では、全ストレージ空間702は、第1の仮想ディスク704を含む5つの仮想ディスクに区画されて示されている。仮想ディスクは、「セグメント」と呼ばれる次に最も低い階層的なデータユニットのサイズ以上の任意のサイズを有するように構成することができる。図7では、第3の仮想ディスク706は、複数のセグメント708に論理的に区画されていることが示されている。これらのセグメントは、連続して順序付けることができ、合わせることによって、仮想ディスクに対応する線形論理ストレージ空間を構成する。図7に示すように、セグメント4(図7の710)等の各セグメントは、特定の冗長性方式に従って複数のブリック712上に分散させることができる。セグメントは、ブリックにわたるデータ分散の粒度を表している。たとえば、図7では、セグメント4(図7の710)は、8+2イレージャ符号化冗長性方式に従ってブリック1~9及び13上に分散させることができる。したがって、8+2イレージャ符号化冗長性方式の下で、パリティデータがセグメントデータとは別に記憶される場合、ブリック3は、セグメントデータの8分の1を記憶することができ、ブリック2は、セグメントのパリティデータの2分の1を記憶することができる。ブリック7(図7の714)等の各ブリックは、そのブリックの内部ディスク716のいずれか又はキャッシュメモリ上に、セグメント又はセグメントの一部を分散させることを決めることができる。セグメント又はセグメントの一部は、内部ディスク又はキャッシュメモリに記憶されると、図7に示すページ718等の複数のページを備えていると論理的にみなされる。各ページは、図7に示すブロック720等の連続した一連のブロックを備える。ブロック(たとえば、図7の720)は、タイムスタンプが関連付けられているデータユニットレベルである。タイムスタンプは、後述するストレージレジスタデータ一貫性レジーム(storage register data-consistency regime)に従って管理される。開発中の或るFABシステムでは、セグメントは、連続した256メガバイトを備え、ページは、8メガバイトを備え、ブロックは512バイトを備える。

【0044】

図 8 A～図 8 D は、本発明の一実施の形態を表す、F A B システムのブリック及び内部ディスクへの論理データユニットの仮想的なマッピングを示している。図 8 A～図 8 D は、すべて、図 8 A に関して次に解説する同じ説明図の規則を使用している。F A B システムは、16 個のブリック 8 0 2～8 1 7 として表されている。各ブリックは、ブリック 8 0 2 内の内部ディスクドライブ 8 2 0～8 2 3 等、4 つの内部ディスクドライブを含むものとして示されている。図 8 A～図 8 D において、図示されている論理データユニットは、図の左側に示されている。図 8 A で図示される論理データユニットは、全利用可能ストレージ空間 8 2 6 である。内部ディスクドライブの正方形表現内の陰影は、その図で図示される論理データユニットがマッピングされる内部ディスクドライブの領域を示している。たとえば、図 8 A では、全ストレージ空間 8 2 6 は、すべてのブリックのすべての内部ディスクにおいて利用可能な全空間にわたってマッピングされることが示されている。一定の少量の内部ストレージ空間を、各ブリックの制御ロジックによる制御目的及び管理目的で予約できることに留意すべきであるが、その内部空間は、図 8 A には示されていない。また、データは、ディスクに書き込まれる前に、ランダムアクセスメモリのキャッシュに存在する場合があるが、このストレージ空間は、説明図を簡単にするために、図 8 A～図 8 D においては、各ブリックについて 4 つの内部ディスクのみを備えているとみなされる。

【0045】

図 8 B は、F A B システム 8 0 0 のストレージ空間への仮想ディスク論理データユニット 8 2 8 の一例示のマッピングを示している。図 8 B は、F A B システム 8 0 0 のブリック内の多くの内部ディスクの一部に、又は、すべての内部ディスクの一部にも仮想ディスクをマッピングできることを示している。図 8 C は、F A B システム 8 0 0 の内部ストレージ空間への仮想ディスクイメージ論理データユニット 8 3 0 の一例示のマッピングを示している。仮想ディスクイメージ論理データユニットは、F A B システム内のかかなりの個数のブリックの内部ストレージ空間の大部分にマッピングすることができる。仮想ディスクイメージ論理データユニットは、仮想ディスクのコピー又はイメージを表している。高レベルの冗長性を提供するために、仮想ディスクは、2 つ以上の仮想ディスクイメージとして複製することができる。各仮想ディスクイメージは、F A B システム内のブリックの別々のパーティションに置かれる。仮想ディスクの複製によって、たとえば、F A B システム内のブリックの地理的に異なった別々のパーティション上に仮想ディスクを複製することが可能になり、1 つの地理的ロケーションにおける大規模な惨事が仮想ディスクデータの回復不可能な損失にならないようにされる。

【0046】

図 8 D は、F A B システム 8 0 0 のブリック内の内部ストレージ空間へのセグメント 8 3 2 の一例示のマッピングを示している。図 8 D に見ることができるよう、セグメントは、F A B システム内のブリックの比較的小さなサブセットの内部ディスクの多くの小さな部分にマッピングすることができる。上述したように、セグメントは、本発明の多くの実施の形態では、イレージャ符号化方式及びミラーリング方式を含む下位レベル冗長性方式によるデータの分散のための論理データユニットレベルである。したがって、データ冗長性が所望されていない場合には、セグメントを単一のブリックの単一のディスクドライブにマッピングすることができる。一方、ほとんどの目的では、セグメントは、2 つのブリックに少なくともミラーリングされる。上述したように、ブリックは、セグメントのページ又はセグメントの一部を、さまざまな考慮すべき事項に従ってその内部ディスク間に分散させる。これらさまざまな考慮すべき事項には、利用可能な空間が含まれ、また、内部ディスクドライブのさまざまな特性を利用するための最適な分散が含まれる。これら内部ディスクドライブのさまざまな特性には、ヘッドの移動遅延、回転遅延、アクセス頻度、及び他の考慮すべき事項が含まれる。

【0047】

図 9 は、本発明の一実施の形態を表す、F A B システム内で使用される論理データユニットを示している。全利用可能データストレージ空間 9 0 2 は、仮想ディスク 9 0 4～9

07に区画することができる。仮想ディスクは、次に、所望される場合に、複数の仮想ディスクイメージに複製される。たとえば、仮想ディスク904は、仮想ディスクイメージ908～910に複製される。仮想ディスクが複製されない場合、仮想ディスクは、単一の仮想ディスクイメージを備えているとみなすことができる。たとえば、仮想ディスク905は、単一の仮想ディスクイメージ912に対応する。各仮想ディスクイメージは、順序付けられた一連のセグメントを備える。たとえば、仮想ディスクイメージ908は、セグメントの順序付きリスト914を備える。各セグメントは、冗長性方式に従って1つ又は複数のブリックにわたって分散される。たとえば、図9では、セグメント916は、8+2イレージャ符号化冗長性方式に従って10個のブリック918にわたって分散される。別の例では、セグメント920は、トリプルミラーリング冗長性方式に従って3つのブリック922にわたって分散されたものとして図9に示されている。 10

【0048】

〔FABデータ状態記述データ構造体〕

上述したように、FABシステム内の各ブリックは、基本的に同じ制御プログラムを実行することができ、各ブリックは、リモートホストコンピュータからの要求を受信することができ、その要求に応答することができる。したがって、人体の各細胞が有機体全体の全DNAの符号化された構造（DNA-encoded architecture）を含むのと同様に、各ブリックは、個々のブリックによって適切に管理されるブリック特有の状態情報（一般的には、この状態情報は含まない）に至るまで、FABシステムの全データ状態を表すデータ構造体を、内部の揮発性ランダムアクセスメモリ、不揮発性メモリ、及び／又は内部ディスク空間に含む。全データ状態は、ブリックの動作状態、すなわち健全性と、セグメントが記憶される冗長性方式とに関する情報と共に、図9に示す階層的なデータユニットのサイズ及びロケーションを含む。一般に、内部のページ及びブリック内に記憶されたデータのブロックアドレスを含めて、ブリック特有のデータ状態情報は、FABシステムの全データ状態の一部とはみなされない。 20

【0049】

図10Aは、各ブリックによって保持されるデータ構造体であって、FABシステムの全データ状態を記述し、且つ、本発明の一実施の形態を表すデータ構造体を示している。このデータ構造体は、一般に、前の小節で説明した階層的な論理データユニットをミラーリングするために階層的である。最上位レベルでは、データ構造体は、仮想ディスクテーブル1002を含むことができる。仮想ディスクテーブル1002の各エントリーは、仮想ディスクを記述する。各仮想ディスクテーブルエントリー（「VDTE」）は、1つ又は複数の仮想ディスクイメージ（「VDI」）テーブルを参照することができる。たとえば、VDTE1004は、図10Aでは、VDIテーブル1006を参照する。VDIテーブルは、仮想ディスクイメージの各セグメントについて、セグメント構成ノード（「SCN」）への参照子を含むことができる。このデータ構造体に充てられるメモリ及びストレージ空間を節約するために、複数のVDIテーブルエントリーは、単一のSCNを参照することができる。図10Aでは、VDIテーブルエントリー1008はSCN1010を参照する。各SCNは、1つ又は複数の構成グループ（「cgrp」）を表すことができる。たとえば、図10Aでは、SCN1010はcgrp1012を参照する。各cgrpは、1つ又は複数の構成（「cfg」）を参照することができる。たとえば、図10Aでは、cgrp1014はcfg1016を参照する。最後に、各cfgは、単一のレイアウトのデータ構造体要素に関連付けることができる。たとえば、図10Aでは、cfg1016は、レイアウトデータ構造体要素1018に関連付けられている。レイアウトデータ構造体要素は、関連付けられているcfg内に収容しても、cfgとは別個としてもよく、関連付けられているcfg内にブリックの表示を含んでもよい。VDIテーブルは、かなり大きな場合があり、効率的なストレージ方式を使用して、VDIテーブル又はVDIテーブルの一部をメモリ及び不揮発性記憶媒体に効率的に記憶することができる。たとえば、UNIX（登録商標）のようなiノード構造体がある。このiノード構造体は、セグメントへの参照子を直接含んだルートノードと、間接参照子又は2重間接参照子を 30 40 50

有する追加ノードとを有する。2重間接参照子は、iノード参照子を含んだノードを通じて、セグメント参照子を含んだ追加ノードに向かう参照子である。他の効率的なストレージ方式も可能である。

【0050】

V D I テーブルと、F A B システムの全データ状態を記述する、各ブリックによって保持されるデータ構造体の他のすべてのデータ構造体要素との双方について、多種多様な物理表現及びストレージ技法を使用することができる。一例として、可変長のデータ構造体要素を、可能な最大個数のデータエントリ又は予想される最大個数のデータエントリを収容するのに十分なサイズの固定長のデータ構造体要素として割り当てることができる。或いは、可変長のデータ構造体要素は、リンクリスト、ツリー、又は、他のこのような動的なデータ構造体要素として表すこともできる。他のこのような動的なデータ構造体要素とは、新しいデータを収容するか、又はもはや必要とされないデータを削除するために、必要に応じてリアルタイムでサイズの変更が可能なものである。図10A及び図11A～図11Hに示すツリーのような表現において、分離され且つ区別されるものとして表されているノードは、実際の実施態様では、共にテーブルに記憶することができる。ただし、ノード又はテーブルに記憶されるものとして示されているデータ構造体要素は、代替的に、リンクリスト、ツリー、又はより複雑な他のデータ構造体の実施態様に記憶することもできる。

10

【0051】

上述したように、V D I は、仮想ディスクの複製を表すのに使用することができる。したがって、V D T E から V D I への階層的なファンアウトは、仮想ディスクの複製を表しているとみなすことができる。S C N は、或る冗長性方式から別の冗長性方式へのセグメントのマイグレーションを可能にするのに使用することができる。4+2 イレージャ符号化冗長性方式に従って分散されたセグメントを8+2 イレージャ符号化冗長性方式へ転写することが望ましいか又は必要な場合がある。セグメントのマイグレーションには、場合によっては新しいブリックグループにわたって分散される新しい冗長性方式の空間を作成すること、新しい構成を既存の構成と同期させること、および新しい構成が既存の構成と同期されると既存の構成を削除することが伴う。したがって、マイグレーションが行われている期間中、S C N は、或る冗長性方式の下での既存の構成と異なる冗長性方式の下での新しい構成とを含む過渡状態を表す2つの異なる c g r p を同時に参照することができる。マイグレーション中にセグメントに対して実行されるデータ変更オペレーション及びデータ状態変更オペレーションは、完全な同期が達成されるまで、過渡状態の双方の構成に対して実行され、古い構成は削除することができる。同期には、新しい構成のすべてのブロックについて、後述するクォーラムを確立すること、必要に応じて古い構成から新しい構成へデータをコピーすること、およびマイグレーション中にセグメントに向けられたオペレーションを実行するのに必要とされるすべてのデータ更新を実行することが伴う。場合によっては、新しい構成の構築中の故障によって、構成が回復不可能な損傷を受けたまま残されるので、過渡状態は、新しい構成が完全に構築されるまで維持される。後述する場合を含めて、それ以外の場合には、古い構成のすべての既存のクォーラムは、新しい構成で引き続き有効であるので、最小限の同期しか必要ない。

20

30

40

【0052】

既存の冗長性方式に従ってセグメントを分散させる一組のブリックは、新しい冗長性方式に従ってセグメントを分散させる一組のブリックと重なり合う場合がある。したがって、セグメントが現在マイグレーション中である場合に、F A B システム内のブロックアドレスは、その特定の冗長性方式を記述する追加フィールド又は追加オブジェクト、すなわちブロックのロールを含むことができる。したがって、ブロックアドレスは、2つの異なる冗長性方式の下で単一のブリックに記憶された同じセグメントの2つのブロックを区別する。図10Bは、本発明の一実施の形態によるブリックロールを組み込んだブリックセグメントアドレスを示している。図10Bに示すブロックアドレスは、次のフィールド、すなわち、(1) そのブロックアドレスによって参照されるブロックを含んだブリックの

50

識別情報を含むブリックフィールド1020、(2)そのブロックアドレスによって参照されるブロックを含んだセグメントの識別情報を含むセグメントフィールド1022、(3)セグメントフィールドで特定されるセグメント内のブロックの識別情報を含むブロックフィールド1024、(4)セグメントが記憶される冗長性方式の表示を含むフィールド1026、(5)セグメントがイレージャ符号化冗長性方式の下で記憶される場合に、イレージャ符号化冗長性方式内のブリックフィールドによって特定されるブリックのブリック位置の表示を含むフィールド1028、及び(6)セグメントがイレージャ符号化冗長性方式の下で記憶される場合に、イレージャ符号化冗長性方式のストライプサイズの表示を含むフィールド1030、を含む。ブロックアドレスは、必要に応じて、所与のFABの実施態様においてブロックの位置を十分に記述するための追加フィールドを含むことができる。一般に、フィールド1026、1028、及び1030は、合わさってブリックロールを構成する。このブリックロールは、参照されたブロックを記憶しているブリックが果たす役割を定義するものである。冗長性方式、ブリック位置、及びストライプサイズのさまざまな数値符号化のいずれを使用しても、ブリックロール符号化に充てられるビット数を最小にするすることができる。たとえば、このFABの実施態様が、さまざまなイレージャ符号化冗長性方式について少数の異なるストライプサイズしか使用しない場合、ストライプサイズは、或る一覧表のさまざまな値によって表すことができる。すなわち、換言すれば、少数の異なるストライプサイズの数値表現を含むのに十分な比較的小さなビットフィールドによって表すことができる。

【0053】

cgrpは、再構成を受けている時、複数のcfgデータ構造体要素を参照することができる。再構成は、セグメントを分散させるブリックの変更を伴う場合があるが、ミラーリング冗長性方式からイレージャ符号化冗長性方式への変更、4+3等の或るイレージャ符号化冗長性方式から8+2等の別のイレージャ符号化冗長性方式への変更、又は、複数のブリックのコンテンツの再構成若しくは変更を伴う他のこのような変更を伴う場合はない。たとえば、再構成は、ブリック1、2、及び3に記憶されているトリプルミラーをブリック2及び3に記憶されるダブルミラーに再構成することを伴う場合がある。

【0054】

cfgデータ構造体要素は、一般に、特定の冗長性方式の下で特定のセグメントを共に記憶する一組の1つ又は複数のブリックを記述する。cfgデータ構造体要素は、一般に、cfgデータ構造体要素によって表される構成内のブリックの健全性、すなわち動作状態についての情報を含む。

【0055】

図10Aのレイアウト1018等のレイアウトデータ構造体要素は、特定の冗長性方式の下で特定のセグメントを分散させるすべてのブリックの識別子を含む。レイアウトデータ構造体要素は、表されるセグメントが記憶される特定の冗長性方式を記述する1つ又は複数のフィールドを含むことができ、追加フィールドも含むことができる。図10Aに示すデータ構造体の他のすべての要素は、必要に応じて、データ構造体によって表されるデータ分散方式に従ったデータの記憶及び維持を容易にするための追加フィールド及び記述的サブ要素を含むことができる。図10Aの下部には、次に続くレベルのデータ構造体要素間のマッピング関係の表示が設けられている。1つ又は複数のVDIテーブル内の複数の異なるセグメントエントリーは、単一のSCNノードを参照でき、同じ冗長性方式に従った、同一の一組のブリックにわたる異なるセグメントの分散を表していることに留意すべきである。

【0056】

各ブリックによって保持されるデータ構造体であって、FABシステムの全データ状態を記述し、且つ、本発明の一実施の形態を表すデータ構造体は、絶えず変化する動的表現であり、且つ、ブロックが記憶、アクセス、及び削除される、ブリックが追加及び削除される、ブリック及び相互接続が故障する、冗長性方式並びにFABシステムの他のパラメータ及び特性が管理インターフェースを通じて変更される、並びに他のイベントが発生す

るときに、さまざまな制御ルーチンを引き起こして状態変化をさらに起こす動的表現である。ロック方式がデータ構造体の一部に向けられたオペレーションを制御して直列化するための大きなオーバーヘッドを回避するために、c g r pレベルからレイアウトレベルに至るまでのすべてのデータ構造体要素は、不変であるとみなすことができる。それらのコンテンツ又は相互接続を変更する必要がある場合、c g r pレベルからレイアウトレベルに至るまでのデータ構造体要素がロックされて、変更され、そしてロック解除されるのではなく、新しいコンテンツ及び／又は新しい相互接続を有する新しいデータ構造体要素が追加され、前のバージョンへの参照子が最終的には消去される。この方法で取って代わられたデータ構造体要素は、古いデータ構造体要素によって表されるデータ及び新しいデータ構造体要素によって表されるデータが、新しいクォーラムを確立してあらゆる必要な更新を実行することにより同期された後、最終的には孤立状態になり、そして、孤立状態のデータ構造体要素は、その後、ガベージコレクトされる。この手法は、c g r pレベルからレイアウトレベルに至るまでのデータ構造体要素を不変であると呼ぶことによって要約することができる。

10

20

30

40

50

【0057】

各ブリックによって保持されるデータ構造体であって、F A Bシステムの全データ状態を記述し、且つ、本発明の一実施の形態を表すデータ構造体の別の態様は、各ブリックが、データ構造体のメモリ内バージョン又は部分的なメモリ内バージョンと、不揮発性データ記憶媒体に記憶された永続バージョンとの双方を保持できることである。メモリ内バージョン又は部分的なメモリ内バージョンは、最も頻繁にアクセスされるレベル及びデータ構造体要素並びに最も近時にアクセスされたレベル及びデータ構造体要素に高速アクセスするために保持される。データ構造体のメモリ内バージョンのデータ要素は、データ構造体の永続バージョンには含まれない追加フィールドを含むことができる。この追加フィールドは、一般に、図10A、図11A～図11H、及びその後の図には示されていない。たとえば、メモリ内バージョンは、ポインタ等の逆マッピング要素を含むことができる。この逆マッピング要素によって、図に示すポインタの下向き方向によって示されるトップダウンのトラバースに加えて、ボトムアップ方向、横方向、及びより複雑な方向でのデータ構造体の効率的なトラバースが可能になる。データ構造体のメモリ内バージョンのデータ構造体要素の中のいくつかのものは、参照カウントフィールドも含むことができる。この参照カウントフィールドによって、ガベージコレクションが容易になり、且つ、そのデータ構造体を含んだブリックの状態を変更する、制御ルーチンの実行するオペレーションの調整が容易になる。

【0058】

図11A～図11Hは、本発明の一実施の形態を表す、F A Bシステム内における図10Aに示すデータ記述データ構造体に反映されるさまざまな異なるタイプの構成の変更を示している。図11A～図11Dは、ブリックの健全性ステータスの変更を伴う簡単な構成の変更を示している。この場合、トリプルミラーリング冗長性方式に従ってブリック1、2、及び3上に分散されたセグメント（図11Aの1102）は、（1）ブリック3の修復により、トリプルミラーリング方式に従ってブリック1、2、及び3（図11Bの1104）上、（2）ブリック3の故障及びブリック4内のスペアのストレージ空間によるブリック3の取り替えにより、トリプルミラーリング方式に従ってブリック1、2、及び4（図11Cの1106）上、又は（3）ブリック3の故障により、ダブルミラーリング方式に従ってブリック1及び2（図11Dの1108）上、のいずれかに分散されるように再構成される。ブリック3の故障が最初に検出されると、新しいc g r p 1112がデータ構造体に追加される。この新しいc g r p 1112は、ブリック3がデッドであることを示すブリック3のブリック健全性表示1114を有する新しいc f g 1110に加えて、初期c f gのコピー1011も含む。この追加によって、分散されたセグメントの初期c g r p、初期c f g、及び初期レイアウト表現が取り替えられる（図11Aの1102）。ブリック3の健全性ステータスとして記憶される「デッドブリック」表示は、図10Aに示す全データ構造体の重要な特徴である。「デッドブリック」ステータスによって

、その後故障したブリックの以前の関与の記録をデータ構造体に保存することが可能になり、それによって、その後の同期、及び、故障したブロックの前の関与に気付くことが必要となり得る他のオペレーションが可能になる。ブリック3の故障により現在のクォーラムを有しないブロックの新しいクォーラムを確立することを含めて、初期構成と新しい構成との間のあらゆる同期が完了し、分散されたセグメント1116の新しい表現がデータ構造体に追加されると、データ構造体要素1110～1112を備える分散されたセグメントの過渡的な2-cfg表現を消去してガベージコレクトすることができ、ブリック3が故障したことを示す単一のcfgデータ構造体を有する分散されたセグメント1116の最終的な表現が残される。図11A～図11D及びその後の図では、たとえば、図11Aに示すcgrpが1つ又は複数のSCNノードによって参照されるとの理解を前提にして、データ構造体の関連がある部分のみが示されている。 10

【0059】

図11B～図11Dは、ブリック3が故障した場合の3つの異なる結果を説明している。各結果は、図11Aの下部に示す、分散されたセグメント1116の表現から開始している。3つのすべての結果には、データ構造体の中間状態として示す過渡的な2-cfg状態が伴う。この過渡的な2-cfg状態は、2つの新しいcfgデータ構造体要素を参照するさらに別の新しいcgrpで構成される。2つの新しいcfgデータ構造体要素の一方は、図11Aの下部に示す分散されたセグメント1116の表現からのcfgのコピーを含み、他方は、新しいブリック健全性情報を含む。図11Bでは、ブリック3が修復され、過渡的な2-cfg1118は、ブリック3の故障状態の記述と、ブリック3の修復状態の記述との双方を含む。図11Cでは、ブリック3は、ブリック4のスペアのストレージ空間によって取り替えられており、過渡的な2-cfg状態1120は、ブリック3の故障状態の記述と、ブリック3がブリック4によって取り替えられた新しい構成の記述との双方を含む。図11Dでは、ブリック3は完全に故障し、セグメントは、3つのブリックではなく2つのブリック上への分散に再構成され、過渡的な2-cfg状態1122は、ブリック3の故障状態の記述と、データがブリック1及び2上に分散されているダブルミラーリング構成の記述との双方を含む。 20

【0060】

図11E及び図11Fは、4+2イレージャ符号化冗長性方式に従ってセグメントが分散されるブリックが喪失され、その喪失されたブリックの代わりに新しいブリックが使用されることを示している。最初に、セグメントは、ブリック1、4、6、9、10、及び11上に分散されている（図11Eの1124）。ブリック4の故障が検出されると、2つの新しいcfgデータ構造体要素を参照する新しいcgrpを含んだ過渡的な2-cfg状態1126が、ブリック4が故障したことを示す新しいcfg1128を得る。分散されたセグメント1124の初期表現は、その後、ガベージコレクトすることができる。故障したブリック4を有する新しい構成の同期が古い構成に対して実行され、ブリック4が故障したことを示す単一のcfgデータ構造体要素を参照する新しいcgrpを有する、分散されたセグメント1132の記述が追加されると、過渡的な2-cfg表現1126をガベージコレクトすることができる。次に、新しい構成が追加されて、過渡的な2-cfg状態1133が作成される。この新しい構成は、ブリック5におけるスペアのストレージ空間が、ブリック4により前に提供されたストレージ空間に取って代わったものである。その後、前の表現1132はガベージコレクトされる。ブリック5がブリック4に取って代わった新しい構成の同期が完了し、分散されたセグメントの最終的な新しい表現1136が追加されると、過渡的な2-cfg表現1134をガベージコレクトすることができる。 40

【0061】

図11Fのcfg1135等の新しい構成が、図11Fのcfg1134等の古い構成と同期されている時間の間、図11Fのcfg1134及び1135等の2-cfg過渡状態の2つの代替的な構成は、図11A～図11Fに示す過渡的な2-cfg表現に同時に保持される。たとえば、ブリック5のコンテンツが、前の小節で解説した行列反転方法 50

に従って再構築されている間、セグメントに発行される新しいWR I T E（書き込み）オペレーションは双方の構成に発行され、それらのWR I T Eオペレーションが、各構成におけるブリックのクォーラムで成功して完了することが確実にされる。クォーラム及び他の一貫性メカニズムについては後述する。最後に、新しい構成1 1 3 5が完全に再構築され、新しい構成のデータ状態が、古い構成のデータ状態1 1 1 4に対して完全に同期されると、古い構成は、表現1 1 3 3全体を、最終的な構成のみを含む新しい表現1 1 3 6と取り替えることによって削除することができる。過渡的な2-c f g表現は、その後ガベージコレクトされる。c g r pレベル及びそれより低レベルの既存のデータ構造体要素を変更しないが、その代わり、2-c f g過渡状態を通じて新しいデータ構造体要素を追加することによって、適切な同期を完成させることができ、データ構造体へのアクセスを制御するために、ロッキングも他の直列化技法も使用する必要はない。WR I T Eオペレーションは、1つ又は複数のブリック内のデータ状態を変更する、データに対するオペレーションの例示である。したがって、この解説では、WR I T Eオペレーションは、オペレーション又はタスクの実行中に、F A Bシステムのデータ状態が変更されることにより、データ一貫性問題が発生するそれらオペレーション又はタスクのクラスを表すのに使用される。一方、他のオペレーション及びタスクもデータ状態を変更する場合があります、このような他のオペレーション及びタスクがF A Bの実施態様で実行される場合に、上述した技法によって、構成間の適切な遷移が可能になる。さらに他の場合において、初期構成の下でのブロックのすべてのクォーラムが、基本的に変更されずに維持され、且つ、新しい構成において有効であるとき、2-c f g過渡表現が必要とされない場合があります、長い期間保持する必要がある場合がある。たとえば、ダブルミラーリングされたセグメントが、2つのブリックの一方の故障によって非冗長構成に再構成される場合、すべてのクォーラムは有効のままである。その理由は、ダブルミラーリングされた構成のブリックの過半数が各ブロックの値について一致していることを必要とされていたからである。これは、したがって、すべてのブリックが前の構成で一致していたことを意味する。2つのブリックの一方の喪失に起因して、あいまいさも損傷したクォーラムも生じない。

【0 0 6 2】

図1 1 Gは、F A Bシステムのブリック上にセグメントを分散させる冗長性方式の変更を伴う、さらに複雑な構成の変更を示している。図1 1 Gに示すケースでは、当初、4+2イレージャ符号化冗長性に従ってブリック1、4、6、9、10、及び11（図1 1 Gの1 1 4 0）上に分散されたセグメントが、ブリック4、13、及び18（図1 1 Gの1 1 4 2）上のトリプルミラーリング冗長性方式にマイグレーションされる。冗長性方式の変更は、新しい構成1 1 2 8が前の構成1 1 4 0と同期されている間、S C Nノード1 1 4 6から参照される2つの異なるc g r pデータ構造体要素1 1 4 4～1 1 4 5を保持することを伴う。新しい構成が古い構成と同期されている間、S C Nレベルの制御ロジックは、2つの異なる構成に対するWR I T Eオペレーションの方向を調整する。その理由は、WR I T Eオペレーションの一貫性のある実行を確実にするための技法が、2つの異なる冗長性方式の間で異なるからである。S C Nノードをロックすることができ、また、S C Nノードへのアクセスを別の方法で操作可能に制御することもできるので、S C Nノードの状態をマイグレーション中に変更することができる。しかしながら、S C Nノードは、複数のV D Iテーブルエントリによって参照される場合があるので、新しいS C Nノード1 1 4 6が、一般に、マイグレーションオペレーション用に割り当てられる。

【0 0 6 3】

最後に、図1 1 Hは、F A Bシステム内の仮想ディスクの一例示の複製を示している。仮想ディスクは、単一のV D Iテーブル1 1 5 0を参照するV D T Eエントリ1 1 4 8によって表される。仮想ディスクの複製は、元のV D Iテーブル1 1 5 0と共に、V D T E 1 1 3 2から同時に参照される新しいV D Iテーブル1 1 5 2を作成することを伴う。制御ロジックの階層内における仮想ディスクレベルの制御ロジックは、新しいV D Iの前のV D Iとの同期を調整し、同期プロセス中に仮想ディスクに向けられるWR I T Eオペレーションを受け止め続ける。

【0064】

図10Aに示すデータ記述データ構造体内における階層レベルは、FABシステムの各ブリックによって実行される制御ロジック内における制御ロジックレベルを反映している。この制御ロジックレベルは、データ状態記述データ構造体の対応するレベルにおけるデータ構造体要素と、そのレベルよりも下のデータ構造体要素とを操作する。ホストコンピュータから受信された要求は、最初に、最上位処理レベルで受信され、最上位処理レベルにより、実行用の1つ又は複数のオペレーションとして、適切な仮想ディスクに向けられる。仮想ディスクレベルの制御ロジックは、次に、仮想ディスクの1つ又は複数の複製を表す1つ又は複数のVDIにこのオペレーションを向ける。VDIレベルの制御ロジックは、オペレーションが向けられる1つ又は複数のVDIのセグメントを決定し、適切なセグメントにオペレーションを向ける。SCNレベルの制御ロジックは、オペレーションを適切な構成グループに向け、構成グループレベルの制御ロジックは、オペレーションを適切な構成に向け。構成レベルの制御ロジックは、要求をその構成のブリックに向け、ブリック内の内部ブリックレベル制御ロジックは、要求を内部ディスクドライブ内の特定のページ及びブロックにマッピングして、ローカルな物理アクセスオペレーションを調整する。

10

【0065】

[ストレージレジスタモデル]

FABシステムは、クォーラムベースの分散READ（読み出し）オペレーション及び分散WRITE（書き込み）オペレーションについてストレージレジスタモデルを使用することができる。ストレージレジスタは、データの分散単位である。現在のFABシステムでは、ブロックがストレージレジスタとして扱われる。

20

【0066】

図12～図18は、分散ストレージレジスタの基本オペレーションを示している。図12に示すように、分散ストレージレジスタ1202は、好ましくは、或る特定の電子デバイスのハードウェアで実施される物理レジスタではなく、抽象的なレジスタ、すなわち仮想レジスタである。プロセッサ又はコンピュータシステム1204～1208で実行される各プロセスは、分散ストレージレジスタに関係する少数のルーチンと共に、ダイナミックメモリに記憶された少数の値を使用して、分散ストレージレジスタ1202を共同で実施する。これら少数の値は、オプションとして、不揮発性メモリにバックアップされる。最低でも、一組の記憶された値及びルーチンが、分散ストレージレジスタにアクセスする各処理エンティティに関連付けられる。いくつかの実施態様では、物理プロセッサ又はマルチプロセッサシステムで実行される各プロセスは、それ自身の記憶された値及びルーチンを管理することができ、他の実施態様では、特定のプロセッサ又はマルチプロセッサシステムで実行されるプロセスは、記憶された値及びルーチンを共有することができるが、共有がローカルに調整されて、プロセッサで実行される複数のプロセスによる同時アクセス問題が防止されることが条件である。

30

【0067】

図12では、各コンピュータシステムは、分散ストレージレジスタのローカル値1210～1214を保持する。一般に、異なるコンピュータシステムによって記憶されたローカル値は通常、同一であり、分散ストレージレジスタ1202の値と等しい。しかしながら、時に、ローカル値は、図12に示す例のように、すべて同一であるとは限らない場合がある。この場合、コンピュータシステムの過半数が、現在ローカルに記憶されている単一の値を保持している場合、分散ストレージレジスタの値が過半数保持値(majority-held value)となる。

40

【0068】

分散ストレージレジスタは、当該分散ストレージレジスタを共同で実施する複数の相互通信プロセスに2つの基本高レベル関数を提供する。図13に示すように、プロセスは、READ要求1302を分散ストレージレジスタ1202に向けることができる。図14において分散ストレージレジスタ1202内に値「B」により示すように、分散ストレ

50

ジレジスタが有効な値を現在保持している場合、この現在の有効な値が要求側プロセスへ返される（1402）。一方、図15に示すように、分散ストレージレジスタ1202が現在、有効な値を含んでいない場合、値NIL1502が要求側プロセスへ返される。値NILは、分散ストレージレジスタ内に記憶されている有効な値となることができない値である。

【0069】

また、プロセスは、分散ストレージレジスタに値を書き込むこともできる。図16では、プロセスは、WRITEメッセージ1602を分散ストレージレジスタ1202へ向ける。このWRITEメッセージ1602は、分散ストレージレジスタ1202に書き込まれる新しい値「X」を含む。図17に示すように、分散ストレージレジスタに現在どのような値が記憶されていようとも、分散ストレージレジスタへ送信された値の上書きが成功した場合、ブール値「TRUE」が、WRITE要求を分散ストレージレジスタへ向けたプロセスへ返される（1702）。そうでない場合、図18に示すように、WRITE要求は失敗し、ブール値「FALSE」が、WRITE要求を分散ストレージレジスタへ向けたプロセスへ返され（1802）、分散ストレージレジスタに記憶された値は、WRITE要求によって変更されない。一定の実施態様では、分散ストレージレジスタは、2値「OK」及び「NOK」を返す。OKは、WRITE要求の実行が成功したことを示し、NOKは、分散ストレージレジスタのコンテンツが不明確であることを示し、また換言すれば、WRITEは、成功している場合もあるし、成功していない場合もある。

【0070】

図19は、複数の他のプロセス及び／又は処理エンティティ $P_{j \neq i}$ と共に分散ストレージレジスタを実施するプロセス又は処理エンティティ P_i によって使用されるコンポーネントを示している。プロセッサ又は処理エンティティは、3つの低レベルプリミティブ、すなわち、タイマメカニズム1902、一意のID1904、及び時計1906を使用する。プロセッサ又は処理エンティティ P_i は、ローカルタイマメカニズム1902を使用する。このローカルタイマメカニズム1902によって、 P_i は、タイマを指定された期間に設定して、その後、そのタイマが満了するのを待つことが可能になる。 P_i は、或るオペレーションを続けるために、タイマの満了時に通知を受ける。プロセスは、タイマを設定して、タイマの満了をチェック又はポーリングしながら、実行を続けることもできるし、タイマを設定して、実行を一時停止し、タイマが満了した時に再び目覚めることもできる。いずれの場合も、タイマによって、プロセスは、オペレーションを論理的に一時停止し、その後、指定された期間後にオペレーションを再開することも可能であり、またタイマが満了するまで、指定された期間の間、或るオペレーションを実行することも可能になる。また、プロセス又は処理エンティティ P_i は、確実に記憶され且つ確実に取り出し可能なローカルプロセスID（「PID」）1904も有する。各プロセッサ又は各処理エンティティは、分散ストレージレジスタを共に実施する他のすべてのプロセス及び／又は処理エンティティに対して一意のローカルPIDを有する。最後に、プロセッサ及び／又は処理エンティティ P_i は、或る絶対時刻にほぼ調整されたリアルタイム時計1906を有する。分散ストレージレジスタを共に共同で実施するすべてのプロセス及び／又は処理エンティティのリアルタイム時計は、正確に同期されている必要はないが、絶対時刻の或る共有概念を適度に反映しているべきである。パーソナルコンピュータを含むほとんどのコンピュータは、バッテリーで電力供給を受けるシステム時計を含む。このシステム時計は、現在の世界時の値を反映している。分散ストレージレジスタの実施態様を含めて、ほとんどの目的において、これらのシステム時計は、正確に同期されている必要はないが、現在の世界時をほぼ反映していることだけが必要とされる。

【0071】

各プロセッサ又は各処理エンティティ P_i は、揮発性メモリ1908を含み、いくつかの実施の形態では、不揮発性メモリ1910も含む。揮発性メモリ1908は、実行用の命令と、分散ストレージレジスタプロトコルに使用される複数の変数のローカル値とを記憶するのに使用される。不揮発性メモリ1910は、いくつかの実施の形態では、分散ス

10

20

30

40

50

トレージレジスタプロトコルに使用される変数を永続的に記憶するのに使用される。変数値の永続ストレージによって、クラッシュ又は通信中断後に、プロセスが分散ストレージレジスタの共同実施への関与を再開することが比較的簡単になる。一方、永続ストレージは、クラッシュしたプロセッサ又は一時的に分離されたプロセッサが、分散ストレージレジスタの共同実施への関与を再開することには必要とされない。その代わり、非永続ストレージの実施の形態において、ダイナミックメモリに記憶された変数値が喪失される場合にすべて同時に喪失されるという条件で、且つ、喪失された変数が適切に再初期化されるという条件で、且つ、プロセッサのクォラムが常に機能的で且つ相互接続された状態にあるという条件で、分散ストレージレジスタプロトコルは、正常に動作し、分散ストレージレジスタを使用するプロセス及び処理エンティティの進行は維持される。各プロセス P_i は、3つの変数、すなわち、(1) 分散ストレージレジスタの現在のローカル値を保有する `val1934`、(2) 分散ストレージレジスタの現在のローカル値に関連付けられたタイムスタンプ値を示す `val-ts1936`、及び(3) `WRITE` オペレーションに関連付けられた最も近時のタイムスタンプを示す `ord-ts1938` を記憶する。変数 `val` は、特に非永続ストレージの実施の形態では、値 `NIL` に初期化される。この値 `NIL` は、プロセス又は処理エンティティによって分散ストレージレジスタに書き込まれるどの値とも異なる。すなわち、したがって値 `NIL` は、分散ストレージレジスタの他のすべての値と区別可能である。同様に、変数 `val-ts` 及び `ord-ts` の値は、値「`initialTS`」に初期化される。この値「`initialTS`」は、タイムスタンプ値を生成するのに使用されるルーチン「`newTS`」によって返されるどのタイムスタンプ値よりも小さな値である。`val`、`val-ts`、及び `ord-ts` が、共にこれらの値に再初期化されるという条件で、共同で実施される分散ストレージレジスタは、通信中断並びにプロセス及び処理エンティティのクラッシュに耐性を有する。ただし、プロセス及び処理エンティティの少なくとも過半数が回復して正常なオペレーションを再開することが条件である。

【0072】

各プロセッサ又は各処理エンティティ P_i は、他のプロセス及び処理エンティティ $P_{j \neq i}$ からメッセージを受信し(1912)、他のプロセス及び処理エンティティ $P_{j \neq i}$ へメッセージ送信する(1914)ために、メッセージベースのネットワークを介して他のプロセス及び処理エンティティ $P_{j \neq i}$ に相互接続することができる。各プロセッサ又は各処理エンティティ P_i は、ルーチン「`newTS`」(新しいTS)1916を含む。このルーチン「`newTS`」1916は、呼び出された時にタイムスタンプ `TS_i` を返し、タイムスタンプ `TS_i` は、或る初期値「`initialTS`」(初期TS)よりも大きい。ルーチン「`newTS`」は、呼び出されるたびに、前に返したどのタイムスタンプよりも大きなタイムスタンプ `TS_i` を返す。また、プロセッサ又は処理エンティティ P_i によって呼び出された `newTS` が返すどのタイムスタンプ値 `TS_i` も、他のどのプロセッサ又は処理エンティティ P_j によって呼び出された `newTS` が返すどのタイムスタンプ値 `TS_j` とも異なるべきである。`newTS` を実施するための1つの実際的な方法は、`newTS` が、システム時計1906によって報告された現在の値とローカル `PID1904` を連結したものを含むタイムスタンプ `TS` を返すことである。分散ストレージレジスタを実施する各プロセッサ又は各処理エンティティ P_i は、4つの異なるハンドラルーチン、すなわち、(1) `READ` ハンドラ1918、(2) `ORDER` (命令) ハンドラ1920、(3) `WRITE` ハンドラ1922、及び(4) `ORDER & READ` (命令&読み出し) ハンドラ1924を含む。ハンドラルーチンは、クリティカルセクション、すなわち、ロックによってシングルスレッド化されたコードセクションを使用して、さまざまなローカルデータ値の試験及び設定における競合状態を防止することが必要な場合があることに留意することは重要である。また、各プロセッサ又は各処理エンティティ P_i は、4つの操作ルーチン、すなわち、(1) `READ1926`、(2) `WRITE1928`、(3) `RECOVER` (回復) 1930、及び(4) `MAJORITY` (過半数) 1932も有する。4つのハンドラルーチン及び4つの操作ルーチンの双方については、以下で

詳述する。

【0073】

分散ストレージレジスタの正常なオペレーション、並びに、分散ストレージレジスタを使用するプロセス及び処理エンティティの活性、すなわち進行は、複数の前提に依存する。各プロセス又は各処理エンティティ P_i は、悪意をもって振る舞わないことが前提とされる。換言すれば、各プロセッサ又は各処理エンティティ P_i は、分散ストレージレジスタプロトコルを忠実に順守する。もう1つの前提は、分散ストレージレジスタを共同で実施するプロセス及び／又は処理エンティティ P_i の過半数が、決してクラッシュしないか、又は最終的にはクラッシュを停止して確実に実行することである。上述したように、分散ストレージレジスタの実施態様は、メッセージの喪失、通信中断、並びにプロセス及び処理エンティティのクラッシュに対して耐性を有する。プロセス又は処理エンティティのクォーラムを破壊するほど十分ではない個数のプロセス又は処理エンティティがクラッシュ又は分離された場合、分散ストレージレジスタは、正常状態及び活性状態を維持する。十分な個数のプロセス又は処理エンティティが、クラッシュ又は分離されて、プロセス又は処理エンティティのクォーラムが破壊された場合、システムは、正常状態を維持するが、活性状態を維持しない。上述したように、プロセス及び／又は処理エンティティのすべてが、メッセージベースのネットワークによって十分に相互接続されている。このメッセージベースのネットワークは、非同期とすることができ、メッセージ送信回数 (message-transmission times) に限界がない。しかしながら、このネットワークのフェアロス (fair-loss) プロパティが前提とされる。このフェアロスプロパティは、基本的に、 P_i が P_j からメッセージ m を受信した場合に、 P_j がメッセージ m を送信したことを保証し、また、基本的に、 P_i がメッセージ m を P_j へ繰り返し送信した場合において、 P_j が正常なプロセス又は処理エンティティであるときは、 P_j が最終的にメッセージ m を受信することを保証するものである。この場合も、上述したように、すべてのプロセス又は処理エンティティのシステム時計がすべて、或る共有された時刻標準を適度に反映しているが、正確に同期されている必要はないことが前提とされる。

【0074】

これらの前提は、分散ストレージレジスタプロトコルの正常性を証明し、且つ、進行を保証するのに有益である。しかしながら、或る実際的な実施態様では、これらの前提の1つ又は複数は違反される場合があり、適度に機能的な分散ストレージレジスタが得られる場合がある。加えて、ハードウェアプラットフォーム及び処理エンティティの特定の不備を克服するために、ハンドラルーチン及び操作ルーチン内に安全装置を追加して構築することができる。

【0075】

分散ストレージレジスタのオペレーションは、クォーラムの概念に基づいている。図20は、クォーラムによる分散ストレージレジスタの現在の値の求値を示している。図20は、図12～図18で使用されたものと類似の説明図の規則を使用する。図20では、プロセス又は処理エンティティ2002～2006のそれぞれは、プロセス又は処理エンティティ2002によって保持されたローカル変数2007等のローカル変数 $val-ts$ を保持する。このローカル変数 $val-ts$ は、分散ストレージレジスタのローカルタイムスタンプ値を保有する。図16と同様に、共同で分散ストレージレジスタを実施するさまざまなプロセス及び／又は処理エンティティによって保持されるローカル値の過半数が、分散ストレージレジスタに関連付けられたタイムスタンプ値 $val-ts$ について現在一致している場合、分散ストレージレジスタ2008の現在の値は、プロセス又は処理エンティティの過半数によって保有される変数 val の値であるとみなされる。プロセス及び処理エンティティの過半数が、タイムスタンプ値 $val-ts$ について一致することができない場合、又は過半数が保有する単一の値が存在しない場合、分散ストレージレジスタのコンテンツは定義されない。一方、その場合に、分散ストレージレジスタを回復させるために、少数派が保有する値を選択して、プロセス及び／又は処理エンティティの過半数により一致させることができる。

【0076】

図21は、図19に概略的に示すルーチンハンドラ及び操作ルーチンの擬似コードの実施態様を示している。これらの擬似コードの実施態様は、詳細なエラーハンドリング、並びに、低レベル通信プリミティブ、ローカルロック、及びコンピュータプログラミングの技術分野の当業者によってよく理解され簡単に実施される他の細部の具体的な詳細を省略していることに留意すべきである。ルーチン「majority」2102は、ライン2において、プロセス又は処理エンティティ P_i から P_i 自身及び他のすべてのプロセス又は処理エンティティ $P_{j \neq i}$ へメッセージを送信する。他のすべてのプロセス又は処理エンティティ $P_{j \neq i}$ は、 P_i と共に共同で分散ストレージレジスタを実施するものである。このメッセージは、十分な個数の返答が受信されるまで、周期的に再送信される。多くの実施態様では、このステップに有限時間及び実行制限を設けるために、タイマが設定される。次に、ライン3～4において、ルーチン「majority」は、メッセージに対する返答が受信されるのを待ち、その後、ライン5において受信された返答を返す。上述した、プロセスの過半数が正常であるという前提によって、タイマが使用されていまいと、ルーチン「majority」は最終的には復帰することが基本的に保証される。実際的な実施態様では、タイマによって、ハンドリングエラーの発生がタイミングよく発生することが促進される。プロセス P_i が受信した返答を前に送信されたメッセージと相関させることができるように、各メッセージは、一般にタイムスタンプまたは他の一意の番号によって一意に特定されることに留意されたい。

10

【0077】

ルーチン「read」2104は、分散ストレージレジスタから値を読み出す。ライン2において、ルーチン「read」は、ルーチン「majority」を呼び出して、READメッセージを P_i 自身及び他のプロセス又は処理エンティティ $P_{j \neq i}$ のそれぞれへ送信する。このREADメッセージは、プロセス P_i によって保有されるローカルな現在の分散ストレージレジスタ値に関連付けられたタイムスタンプ値 $vals$ だけでなく、当該メッセージがREADメッセージであることの表示も含む。ルーチン「majority」が一組の返答を返し、ライン3で判断されるように、すべてがブール値「TRUE」を含む場合、ルーチン「read」は、ローカルな現在の分散ストレージレジスタ値 val を返す。そうでない場合、ライン4において、ルーチン「read」は、ルーチン「recover」を呼び出す。

20

30

【0078】

ルーチン「recover」2106は、クォーラム技法によって分散ストレージレジスタの現在の値を求めようとする。最初に、ライン2において、ルーチン「newTS」を呼び出すことによって、新しいタイムスタンプ ts が得られる。次に、ライン3において、ルーチン「majority」が呼び出されて、プロセス及び／又は処理エンティティのすべてへORDER&READメッセージが送信される。ライン4において、ルーチン「majority」によって返された返答のいずれかのステータスが「FALSE」である場合、「recover」は値NILを返す。そうでない場合、ライン5において、分散ストレージレジスタのローカルな現在の値 val は、ルーチン「majority」によって返された一組の返答における最も高い値のタイムスタンプに関連付けられた値に設定される。次に、ライン6において、ルーチン「majority」が再び呼び出されて、WRITEメッセージが送信される。このWRITEメッセージは、ライン2で得られた新しいタイムスタンプ ts と、分散ストレージレジスタの新しいローカルな現在の値 val とを含む。すべての返答のステータスがブール値「TRUE」を有する場合、WRITEオペレーションは成功し、プロセス及び／又は処理エンティティの過半数は、現在、ライン5においてローカルコピーに記憶されたその新しい値 val と一致する。そうでない場合、ルーチン「recover」は、値NILを返す。

40

【0079】

ルーチン「write」2108は、分散ストレージレジスタに新しい値を書き込む。新しいタイムスタンプ ts がライン2において得られる。ルーチン「majority」

50

が、ライン3において呼び出され、新しいタイムスタンプを含むORDERメッセージをプロセス及び／又は処理エンティティのすべてへ送信する。ルーチン「majority」によって返された返答メッセージで返されるステータス値のいずれかが「FALSE」である場合、ライン4において、値「NOK」が、ルーチン「write」によって返される。そうでない場合、ライン5において、ルーチン「majority」を介してWRITEメッセージを送信することにより、値valが、他のプロセス及び／又は処理エンティティに書き込まれる。ルーチン「majority」によって返された返答のすべてのステータス値が「TRUE」であるとライン6で判断された場合、ルーチン「write」は値「OK」を返す。そうでない場合、ライン7において、ルーチン「write」は値「NOK」を返す。ルーチン「recover」2106及びルーチン「write」の双方の場合において、分散ストレージレジスタ値valのローカルコピー及びタイムスタンプ値val-tsのローカルコピーは、共に、後述するローカルハンドラルーチンによって更新されることに留意されたい。

【0080】

次に、ハンドラルーチンを解説する。手始めに、ハンドラルーチンは、受信された値をローカル変数値と比較し、次に、比較の結果に従ってローカル変数値を設定することに留意すべきである。これらのタイプのオペレーションは、厳密に直列化され、複数の値を記憶するデータ構造体について、各プロセス及び／又は各処理エンティティ内での競合状態から保護されることが必要な場合がある。ローカルな直列化は、不可分なテスト・アンド・セット命令に基づくクリティカルセクション又はローカルロックを使用して容易に行われる。READハンドラルーチン2110は、READメッセージを受信し、このREADメッセージに対してstatus（ステータス）値で返答する。このstatus値は、受信プロセス又は受信エンティティにおけるタイムスタンプval-tsのローカルコピーが、READメッセージで受信されたタイムスタンプと等しいか否かと、READメッセージで受信されたタイムスタンプtsが、ローカル変数ord-tsの現在の値以上であるか否かとを示している。WRITEハンドラルーチン2112は、ライン2において、WRITEメッセージを受信し、ローカル変数statusの値を求める。このローカル変数statusは、受信プロセス又は受信エンティティにおけるタイムスタンプval-tsのローカルコピーが、WRITEメッセージで受信されたタイムスタンプよりも大きいと、WRITEメッセージで受信されたタイムスタンプtsが、ローカル変数ord-tsの現在の値以上であるか否かとを示している。statusローカル変数の値が「TRUE」であるとライン3で判断された場合、WRITEハンドラルーチンは、ライン4～5において、ダイナミックメモリ及び永続メモリの双方にローカルに記憶された値val及びタイムスタンプval-tsを、WRITEメッセージで受信された値及びタイムスタンプで更新する。最後に、ライン6において、ローカル変数statusに保有された値は、WRITEハンドラルーチン2112によってハンドリングされたWRITEメッセージを送信したプロセス又は処理エンティティに返される。

【0081】

ORDER&READハンドラ2114は、ライン2において、ローカル変数statusの値を計算し、その値を、受信されたORDER&READメッセージの送信元のプロセス又は処理エンティティへ返す。statusの計算値は、ORDER&READメッセージで受信されたタイムスタンプが、ローカル変数val-tsに記憶された値及びローカル変数ord-tsに記憶された値の双方よりも大きいと示すブール値である。statusの計算値が「TRUE」である場合、受信されたタイムスタンプtsは、ダイナミックメモリ及び永続メモリの双方の変数ord-tsに記憶される。

【0082】

同様に、ORDERハンドラ2116は、ライン2において、ローカル変数statusの値を計算し、そのステータスを、受信されたORDERメッセージの送信元のプロセス又は処理エンティティへ返す。このステータスは、受信されたタイムスタンプが、ローカル変数val-ts及びord-tsに保有された値よりも大きいと示すブール値を反映してい

る。statusの計算値が「TRUE」である場合、受信されたタイムスタンプtsは、ダイナミックメモリ及び永続メモリの双方の変数ord-tsに記憶される。

【0083】

上述した分散ストレージレジスタ方法及びプロトコルを使用すると、分散データストレージシステムに絶えず一貫して保持される共有状態情報を、一組の分散ストレージレジスタに記憶することができる。記憶は、1つのレジスタにつき1ユニットの共有状態情報で行われる。レジスタのサイズは、共有状態情報のユニットの異なる自然のサイズに適合するように変化することができる。状態情報ユニットの粒度は、パフォーマンスモニタリングによるか、又は特定の分散システム内の状態情報のユニットの予想交換率の解析によって決定することができる。ユニットが大きいほど、プロトコル変数及び分散ストレージレジスタ用に保持された他のデータの被るオーバーヘッドは小さくなるが、ユニットの異なる部分が異なる時刻にアクセスされる場合、通信オーバーヘッドが増加する場合がある。また、上記擬似コード及び説明図は、単一の分散ストレージレジスタの実施態様を対象にしているが、これらの擬似コードルーチンは一般化できることにも留意すべきである。この一般化は、特定の分散ストレージレジスタを特定する、状態情報のユニットのパラメータであって、オペレーションが向けられたパラメータを追加すること、及びこの特定するパラメータによってインデックスされるval-ts、val、ord-ts等の変数のアレイを保持することによって行われる。

【0084】

〔一般化されたストレージレジスタモデル〕

ストレージレジスタモデルは一般に、ミラーリング冗長性方式に従って分散されたセグメントにわたって一貫性を維持するために、FABシステムによってブロックレベルで適用される。換言すれば、セグメントの各ブロックは、複数のブリックにわたって分散されたストレージレジスタであるとみなすことができ、クォーラム及びメッセージパッシングを伴う上記分散技法は、ミラーコピーにわたってデータ一貫性を維持するのに使用される。一方、ストレージレジスタ方式は、イレージャ符号化冗長性方式をハンドリングするように拡張することができる。第1に、上記節で説明し、ミラーリング冗長性方式に使用されるように、ブロックが分散されたブリックの過半数から成るクォーラムではなく、イレージャ符号化冗長性方式は、 $m + \lceil (n - m) / 2 \rceil$ 個のブリックのクォーラムを使用し、任意の2つのクォーラムの交わりが、少なくともm個のブリックを含むようにする。このタイプのクォーラムは、「m-クォーラム」と呼ばれる。第2に、新しく受信された値を、WRITEオペレーションの第2のフェーズで、内部ストレージのブロックに書き込むのではなく、ブリックはその代わりに、その新しい値をその値に関連付けられたタイムスタンプと共にログに記録することができる。ログは、その後、ログに記録されたエントリーのm-クォーラムが受信されてログに記録された時に、非同期に処理されて、ログに記録されたWRITEをコミットすることができる。ログ記録が使用される理由は、ブリックのm-クォーラムが特定のWRITEオペレーションを受信せず、正常の実行していなかった場合に、ミラーリング冗長性方式と異なって、ブリックのクラッシュによりデータを回復させることができないからである。図22は、図17に提供した擬似コードに類似した修正擬似コードを示している。この修正擬似コードは、本発明の一実施の形態を表す、FABシステム内におけるイレージャ符号化冗長性方式に従ってブリックにわたるセグメントの分散をハンドリングする、ストレージレジスタモデルへの拡張を含む。たとえば、m個のブリックが、最も近時に書き込まれた値をログに記憶できなかった場合、その最も近時に書き込まれた値は、ログ内の少なくともm個のコピーに存在する前の値又は少なくともm個のブリック内に記憶された前の値にロールバックされる。

【0085】

図23は、本発明の一実施の形態を表す、FABシステム内におけるストレージレジスタモデルに基づいたデータ一貫性技法によるタイムスタンプに対する大きな依存関係を示している。図23では、ブロック2302は、トリプルミラーリング冗長性方式に従って3つのブリック2304～2306にわたって分散され、且つ、3+2イレージャ符号化

方式に従って5つのブリック2308～2312にわたって分散されて示されている。トリプルミラーリング冗長性方式では、ブロック2314等のブロックの各コピーは、前の小節で解説したように、2つのタイムスタンプ2316及び2317に関連付けられる。イレージャ符号化冗長性方式では、最初のブロック2318等の各ブロックは、少なくとも2つのタイムスタンプに関連付けられる。ブロック2320及び2321と、ブロックのストライプにおける他のブロックとから計算されたチェックサムビットは、2つのタイムスタンプに関連付けられるが、ブロック2324等のブロックは、これに加えて、ログエントリー2326等の（以下に図示され、且つ、ブロックによって重ねられた）ログエントリーにも関連付けることができる。ログエントリーのそれぞれも、タイムスタンプ2328等のタイムスタンプに関連付けられる。明らかに、ストレージレジスタモデルに基づくデータ一貫性技法は、非常の多くのタイムスタンプの記憶及び保持を伴う可能性があり、タイムスタンプに充てられた全ストレージ空間は、FABシステム内の全体の利用可能なストレージ空間のかなりの割合となるおそれがある。その上、ストレージレジスタに向けられた上述したREADオペレーション及びWRITEオペレーションの期間中にブリック間でタイムスタンプを渡すことに起因して、メッセージトラフィックのオーバーヘッドが発生するおそれがある。

【0086】

タイムスタンプに関係する膨大なオーバーヘッドが潜在することから、FABシステムは、複数の技法を使用して、タイムスタンプに関係する記憶オーバーヘッド及びメッセージングオーバーヘッドを改善することができる。第1に、単一のWRITEオペレーションで書き込まれた多数の連続したブロックに単一のタイムスタンプに関連付けることができるように、ブリックは、タイムスタンプを不揮発性ランダムアクセスメモリに階層的に記憶することができる。図24は、本発明の一実施の形態を表す階層的なタイムスタンプ管理を示している。図24では、タイムスタンプは、「区間木」として知られている大きな一種の非循環グラフのリーフノードに関連付けられている。非循環グラフの小さな部分のみが図24に示されている。グラフの表示部分では、2つのリーフノード2402及び2404が、ブロック1000～1050及び1051～2000にそれぞれ関連付けられたタイムスタンプを表している。その後のWRITEオペレーションにおいて、WRITEがブロック1051～1099に向けられている場合、元の非循環グラフのリーフノード2404は、2つの下位レベルブロック2406及び2408に分割されて、修正非循環グラフになる。個別のタイムスタンプをそれら新しいリーフノードブロックのそれぞれに関連付けることができる。逆に、ブロック1051～2000が、その後、単一のWRITEオペレーションで書き込まれた場合、2つのブロック2406及び2408は、その後、合体されて、この非循環グラフは元の非循環グラフ2400に戻される。単一のWRITEオペレーションで書き込まれるブロックグループにタイムスタンプを関連付けることによって、ブリックにより保持されるタイムスタンプの個数を大幅に減少させることができる。

【0087】

ブリックによって保持されるタイムスタンプの個数を減少させる別の方法は、タイムスタンプを積極的にガベージコレクトすることである。前の小節で解説したように、タイムスタンプをブロックに関連付けて、ストレージレジスタモデルのクォーラムベースの一貫性方法を容易にすることができる。しかしながら、ブロックが分散されるすべてのブリックの更新が成功した場合、それらのブロックは、完全に一貫性のある状態にあり、且つ、十分冗長に記憶された状態にあるので、それらのブロックに関連付けられたタイムスタンプはもはや必要とされない。したがって、FABシステムは、ストレージレジスタモデルをさらに拡張して、WRITEオペレーションの全完了の後に、タイムスタンプの積極的なガベージコレクションを含めることができる。FABシステムが、タイムスタンプに関係するオーバーヘッドを減少させるのに使用するさらに別の方法には、他のメッセージ内におけるタイムスタンプに関係するメッセージをピギーバックすること、及び、後述する階層的降格を含めて、結合された処理タスクにおいて、関係するタイムスタンプを同時に

処理することが含まれ得る。

【0088】

クォーラムベースのストレージレジスタモデルは、前の小節で上述した再構成及びマイグレーションをハンドリングするようにさらに拡張することができる。クォーラムベースのストレージレジスタモデルでは、レイアウト及び冗長性方式が変更される。その小節で解説したように、再構成オペレーションの期間中、前の構成の削除及びガベージコレクションの前に、新しい構成が前に存在する構成と同期されている間、2つ以上の異なる構成を同時に保持することができる。WRITEオペレーションは、同期プロセスの期間中、双方の構成に向けられる。したがって、c f gグループ又はSCNレベルの制御ロジックが、受信されたWRITEオペレーションの完了が成功したとみなす前に、構成の上位レベルのクォーラムは、WRITEオペレーションの完了に成功している必要がある。図25及び図26は、さらに拡張されたストレージレジスタモデルの擬似コードを提供する。このさらに拡張されたストレージレジスタモデルは、本発明の一実施の形態を表す、FABシステム内における分散されたセグメントの再構成に起因して存在し得る複数のアクティブな構成に対するクォーラムベースの書き込みの概念を含む。

10

【0089】

あいにく、マイグレーションは、ストレージレジスタモデルにさらなる拡張を必要とする場合があるさらに別のレベルの再構成である。前述した再構成のシナリオのように、マイグレーションは、新しい構成を古い構成と同期させている間、SCNレベルの制御ロジックがWRITEオペレーションを向ける複数のアクティブな構成を伴う。しかしながら、再構成レベルと異なり、マイグレーションレベルは、アクティブな構成に向けられたWRITEがアクティブな構成のクォーラムではなくすべての構成において成功して完了することを必要とする。その理由は、冗長性方式がアクティブな構成によって異なり、或る冗長性方式で失敗したWRITEは、異なる冗長性方式を使用する異なるアクティブな構成から回復可能でない場合があるからである。したがって、マイグレーションレベルでは、アクティブな構成のクォーラムは、アクティブな構成のすべてから成る。したがって、ストレージレジスタモデルをマイグレーションレベルに拡張することによって、より一般的なストレージレジスタのようなモデルが得られる。図27は、本発明の一実施の形態を表す、FABシステム内でストレージレジスタモデルをマイグレーションレベルへ拡張するための高レベルの擬似コードを示している。さらに異なる考慮すべき事項が、複製レベルで適用される場合がある。複製レベルでは、WRITEは、仮想ディスクの複数の複製に向けられる。しかしながら、VDIレベルの考慮すべき事項が一般的なストレージレジスタモデルに組み込まれる場合、図27に関して上述した最も一般的なストレージレジスタモデルの拡張は、VDIレベル及び仮想ディスクレベルの適用について十分一般的である。

20

30

【0090】

上述したストレージレジスタモデルの拡張及び考慮すべき事項の結果、FABシステム内の階層的な制御ロジック及び階層的なデータストレージの最終的な高レベルの記述が得られる。図28は、本発明の一実施の形態を表す、FABシステム内の制御処理及びデータストレージの双方の全階層構造を示している。「最上位レベルコーディネータ」2802と呼ばれる最上位レベルコーディネータロジックは、階層的データストレージモデルの仮想ディスクレベル2804に関連付けることができる。「VDIレベルコーディネータ」2806と呼ばれるVDIレベル制御ロジックは、データストレージモデルのVDIレベル2808に関連付けることができる。「SCNコーディネータ」2810と呼ばれるSCNレベル制御ロジックは、データストレージモデルのSCNレベル2812に関連付けることができる。「構成グループコーディネータ」2814と呼ばれる構成グループレベル制御ロジックは、データストレージモデルの構成グループレベル2816に関連付けることができる。最後に、「構成コーディネータ」2818と呼ばれる構成レベル制御ロジックは、データストレージモデルの構成レベル2820に関連付けることができる。図28及び図28で使用する説明図の規則を使用するその後の図では、c f gデータ構造

40

50

体要素及びレイアウトデータ構造体要素が、互いに結合されて、1つのデータストレージモデルノードにされることに留意されたい。コーディネータの階層編成におけるコーディネータのそれぞれは、コーディネータの階層レベルに適した拡張されたストレージレジスタモデル一貫性方法を実行する。たとえば、c f g グループコーディネータは、ミラーリング冗長性方式についてはクォラムベースの技法を使用し、イレージャ符号化冗長性方式についてはm-クォラムベース技法を使用する。これとは対照的に、S C N コーディネータは、W R I T E オペレーションが成功しているとみなされるように、参照されるすべての構成グループによるW R I T E オペレーションの完了を必要とする拡張されたストレージレジスタモデルを使用する。

【0091】

本発明を特定の実施の形態の観点から説明してきたが、本発明がこれらの実施の形態に限定されることは意図されていない。本発明の精神の範囲内での変更は、当業者に明らかである。たとえば、上述した階層的制御ロジックは、仮想ディスクを表す構成データ構造体をミラーリングする。この仮想ディスクは、データセグメントから構成されて、任意に仮想ディスクイメージとして複製され、データセグメントはデータブロックで構成される。構成データ構造体は、データストレージシステムのデータストレージコンポーネント上にセグメントの粒度で分散される。階層的制御構造の代替的な実施の形態は、異なる階層的データ分散編成を階層的にミラーリングすることができる。たとえば、データオブジェクトは、異なる量のデータを表す異なる階層レベルに階層的に編成されることができる。本発明の階層的制御ロジックの代替的な実施の形態は、これらの異なる階層レベルに対応する制御ロジックレベル、すなわちコーディネータを含むことができる。本発明の階層的制御ロジックは、クォラムベースのデータ一貫性メカニズムを使用するが、代替的な実施の形態では、代替的なデータ一貫性メカニズムを使用することができる。

【0092】

上記説明は、説明のためのものであり、具体的な専門用語を使用して本発明の徹底した理解を提供している。しかしながら、本発明を実施するのに、具体的な細部は必要とされないことが当業者には明らかであろう。本発明の具体的な実施の形態の上記説明は、図示及び説明の目的で提示されるものである。それら説明は、網羅的であることを目的とするものでもなければ、開示した正確な形に本発明を限定することを目的とするものでもない。上記教示に鑑み、多くの変更及び変形が可能であることは明らかである。実施の形態は、本発明の原理及びその実用的な用途を最も良く説明し、それによって、当業者が、本発明及びさまざまな変更を有するさまざまな実施の形態を、検討した特定の使用に適するように最も良く利用することを可能にするために図示して説明されたものである。本発明の範囲は、添付の特許請求の範囲及びその均等物によって画定されることが意図されている。

【図面の簡単な説明】

【0093】

【図1】本発明の一実施の形態によるF A B マスストレージシステムの高レベルの図である。

【図2】本発明の一実施の形態による例示的なF A B ブリックの高レベルの図である。

【図3】データミラーリングの概念を示す図である。

【図4】データミラーリングの概念を示す図である。

【図5】イレージャ符号化冗長性を示す高レベルの図である。

【図6】図3及び図4で使用されたのと同じ説明図の規則を使用する3+1イレージャ符号化冗長性方式を示す図である。

【図7】本発明の一実施の形態を表す、現在のF A B の実施態様で使用される階層的なデータユニットを示す図である。

【図8A】本発明の一実施の形態を表す、F A B システムの物理ディスクへの論理データユニットの仮想的なマッピングを示す図である。

【図8B】本発明の一実施の形態を表す、F A B システムの物理ディスクへの論理データ

10

20

30

40

50

ユニットの仮想的なマッピングを示す図である。

【図 8 C】本発明の一実施の形態を表す、F A B システムの物理ディスクへの論理データユニットの仮想的なマッピングを示す図である。

【図 8 D】本発明の一実施の形態を表す、F A B システムの物理ディスクへの論理データユニットの仮想的なマッピングを示す図である。

【図 9】本発明の一実施の形態を表す、F A B システム内で使用される論理データユニットを、異なる説明図の規則で示す図である。

【図 10 A】各ブリックによって保持されたデータ構造体であって、F A B システムの全データ状態を記述するとともに本発明の一実施の形態を表すデータ構造体を示す図である。

【図 10 B】本発明の一実施の形態によるブリックロールを組み込むブリックセグメントアドレスを示す図である。

【図 11 A】本発明の一実施の形態を表す、F A B システム内における図 10 A に示すデータ記述データ構造体に反映されるさまざまな異なるタイプの構成の変更を示す図である。

【図 11 B】本発明の一実施の形態を表す、F A B システム内における図 10 A に示すデータ記述データ構造体に反映されるさまざまな異なるタイプの構成の変更を示す図である。

【図 11 C】本発明の一実施の形態を表す、F A B システム内における図 10 A に示すデータ記述データ構造体に反映されるさまざまな異なるタイプの構成の変更を示す図である。

【図 11 D】本発明の一実施の形態を表す、F A B システム内における図 10 A に示すデータ記述データ構造体に反映されるさまざまな異なるタイプの構成の変更を示す図である。

【図 11 E】本発明の一実施の形態を表す、F A B システム内における図 10 A に示すデータ記述データ構造体に反映されるさまざまな異なるタイプの構成の変更を示す図である。

【図 11 F】本発明の一実施の形態を表す、F A B システム内における図 10 A に示すデータ記述データ構造体に反映されるさまざまな異なるタイプの構成の変更を示す図である。

【図 11 G】本発明の一実施の形態を表す、F A B システム内における図 10 A に示すデータ記述データ構造体に反映されるさまざまな異なるタイプの構成の変更を示す図である。

【図 11 H】本発明の一実施の形態を表す、F A B システム内における図 10 A に示すデータ記述データ構造体に反映されるさまざまな異なるタイプの構成の変更を示す図である。

【図 12】分散ストレージレジスタの基本オペレーションを示す図である。

【図 13】分散ストレージレジスタの基本オペレーションを示す図である。

【図 14】分散ストレージレジスタの基本オペレーションを示す図である。

【図 15】分散ストレージレジスタの基本オペレーションを示す図である。

【図 16】分散ストレージレジスタの基本オペレーションを示す図である。

【図 17】分散ストレージレジスタの基本オペレーションを示す図である。

【図 18】分散ストレージレジスタの基本オペレーションを示す図である。

【図 19】多数の他のプロセス及び／又は処理エンティティ $P_j \neq i$ と共に分散ストレージレジスタを実施するプロセス又は処理エンティティ P_i によって使用されるコンポーネントを示す図である。

【図 20】クォラムによる分散ストレージレジスタの現在の値の求値を示す図である。

【図 21】図 19 に図的に示すルーチンハンドラ及び操作ルーチンの擬似コードの実施態様を示す図である。

【図 22】図 17 に提供した擬似コードに類似した修正擬似コードあって、本発明の一実

10

20

30

40

50

施の形態を表すF A Bシステム内におけるイレージャ符号化冗長性方式に従ってブリックにわたるセグメントの分散をハンドリングするストレージレジスタモデルへの拡張を含む、修正擬似コードを示す図である。

【図 2 3】本発明の一実施の形態を表す、F A Bシステム内におけるストレージレジスタモデルに基づいたデータ一貫性技法によるタイムスタンプに対する大きな依存関係を示す図である。

【図 2 4】本発明の一実施の形態を表す階層的なタイムスタンプ管理を示す図である。

【図 2 5】本発明の一実施の形態を表す、F A Bシステム内における分散されたセグメントの再構成に起因して存在し得る複数のアクティブな構成に対するクォーラムベースの書き込みの概念を含むさらに拡張されたストレージレジスタモデルの擬似コードを提供する図である。

10

【図 2 6】本発明の一実施の形態を表す、F A Bシステム内における分散されたセグメントの再構成に起因して存在し得る複数のアクティブな構成に対するクォーラムベースの書き込みの概念を含むさらに拡張されたストレージレジスタモデルの擬似コードを提供する図である。

【図 2 7】本発明の一実施の形態を表す、F A Bシステム内でストレージレジスタモデルをマイグレーションレベルへ拡張するための高レベルの擬似コードを示す図である。

【図 2 8】本発明の一実施の形態を表す、F A Bシステム内の制御処理及びデータストレージの双方の全階層構造を示す図である。

【符号の説明】

20

【0 0 9 4】

1 0 2 ~ 1 0 9 . . . マスストレージデバイス

1 1 0 , 1 1 4 . . . 通信媒体

1 1 2 , 1 1 3 . . . リモートホストコンピュータ

2 0 2 ~ 2 1 3 . . . S A T A ディスクドライブ

2 1 4 . . . ディスク I O P

2 1 6 . . . 高速バス

2 1 8 . . . 中央ブリッジ

2 2 0 . . . プロセッサ

2 2 2 . . . ホスト I O P

30

2 2 4 . . . ブリック間 I O P

2 2 6 , 2 2 7 , 2 2 8 . . . メモリ

7 0 2 . . . 全ストレージアドレス空間

7 0 4 , 7 0 6 . . . 仮想ディスク

7 0 8 , 7 1 0 . . . セグメント

7 1 2 , 7 1 4 . . . ブリック

7 1 6 . . . ディスク

7 1 8 . . . ページ

7 2 0 . . . ブロック

8 2 6 . . . 全ストレージ

40

8 2 8 . . . 仮想ディスク

8 3 0 . . . 仮想ディスクイメージ

8 3 2 . . . セグメント

9 0 2 . . . 全ストレージ

9 0 4 , 9 0 5 , 9 0 6 , 9 0 7 . . . 仮想ディスク

9 0 8 , 9 0 9 , 9 1 0 , 9 1 2 . . . 仮想ディスクイメージ

9 1 6 , 9 2 0 . . . セグメント

9 1 4 . . . 順序付きリスト

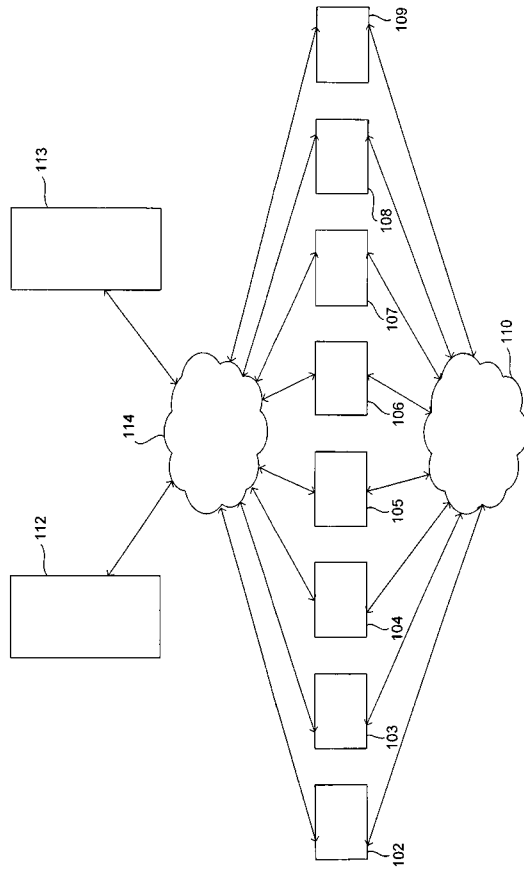
9 1 8 , 9 2 2 . . . ブリック

1 0 0 2 . . . 仮想ディスクテーブル

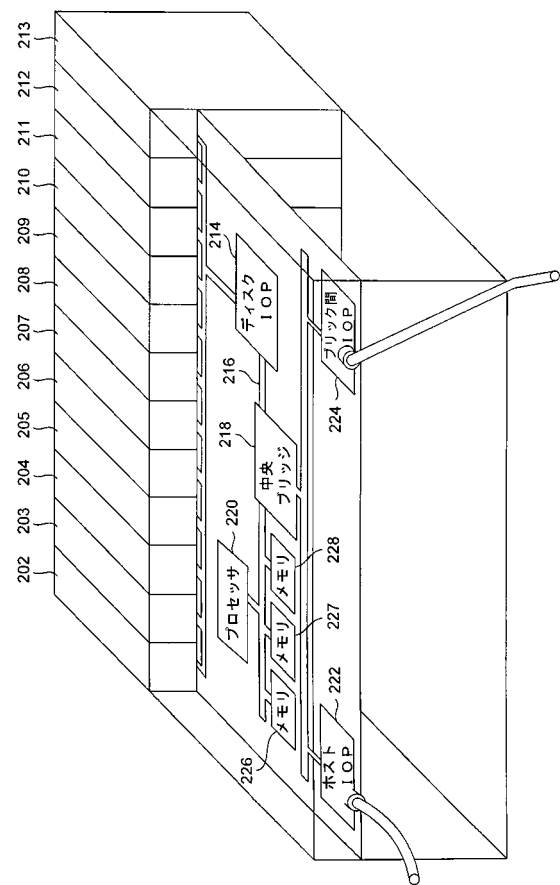
50

1 0 0 4 . . . V D T E エントリー	
1 0 0 6 , 1 0 0 8 . . . V D I テーブル	
1 0 1 0 . . . S C N	
1 0 1 2 , 1 0 1 4 . . . c g r p	
1 0 1 6 . . . c f g	
1 0 1 8 . . . レイアウト	
1 0 2 0 . . . ブリック	
1 0 2 2 . . . セグメント	
1 0 2 4 . . . ブロック	
1 0 2 6 . . . 冗長性方式	10
1 0 2 8 . . . ブリック位置	
1 0 3 0 . . . ストライプサイズ	
1 1 4 1 , 1 1 4 5 . . . c g r p	
1 1 4 6 . . . S C N	
1 1 4 8 . . . V D T E エントリー	
1 1 5 0 , 1 1 5 2 . . . V D I テーブル	
1 9 0 2 . . . タイマメカニズム	
1 9 0 4 . . . P I D	
1 9 0 6 . . . リアルタイム時計	
1 9 0 8 . . . 揮発性メモリ	20
1 9 1 0 . . . 不揮発性メモリ	
1 9 1 8 . . . R E A D ハンドラ	
1 9 2 0 . . . O R D E R ハンドラ	
1 9 2 2 . . . W R I T E ハンドラ	
1 9 2 4 . . . O R D E R デュアルハンドラ	
2 3 0 2 , 2 3 1 4 , 2 3 1 8 , 2 3 2 0 , 2 3 2 1 , 2 3 2 4 . . . ブロック	
2 3 0 4 ~ 2 3 1 2 . . . ブリック	
2 3 1 6 , 2 3 1 7 , 2 3 2 8 . . . タイムスタンプ	
2 3 2 6 . . . ログエントリー	
2 4 0 0 . . . 非循環グラフ	30
2 4 0 2 , 2 4 0 4 . . . リーフノード	
2 4 0 6 , 2 4 0 8 . . . ブロック	
2 8 0 2 . . . 最上位レベルコーディネータ	
2 8 0 4 . . . 仮想ディスクレベル	
2 8 0 6 . . . V D I コーディネータ	
2 8 0 8 . . . V D I レベル	
2 8 1 0 . . . S C N コーディネータ	
2 8 1 2 . . . S C N レベル	
2 8 1 4 . . . 構成グループコーディネータ	
2 8 1 6 . . . 構成グループレベル	40
2 8 1 8 . . . 構成コーディネータ	
2 8 2 0 . . . 構成レベル	

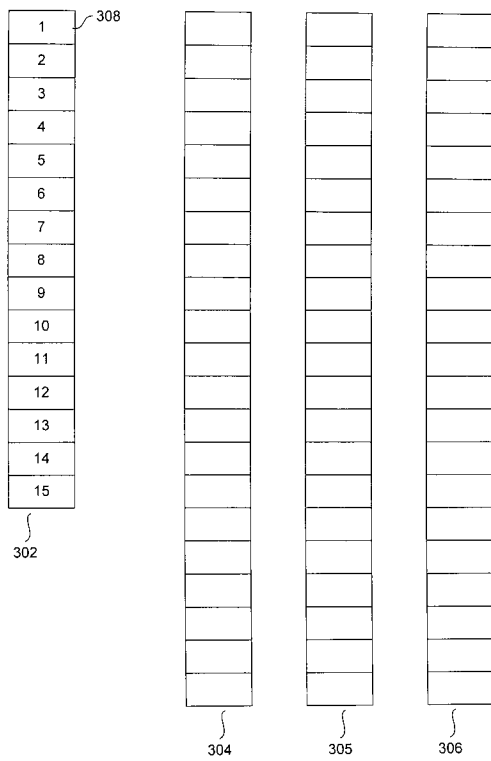
【図 1】



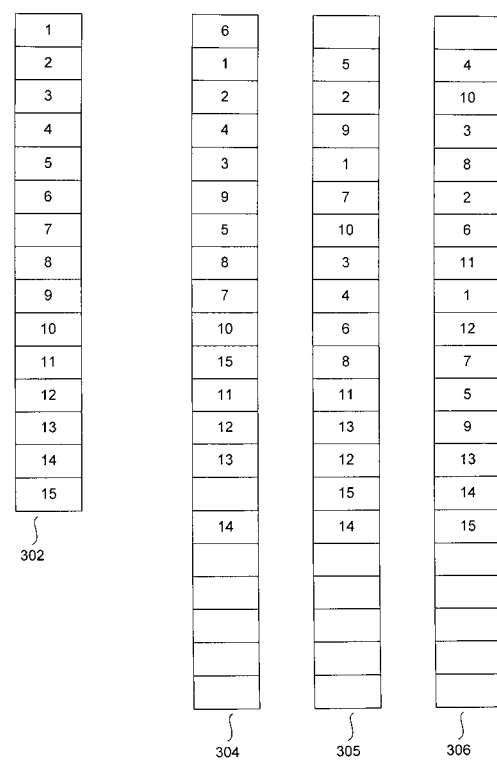
【図 2】



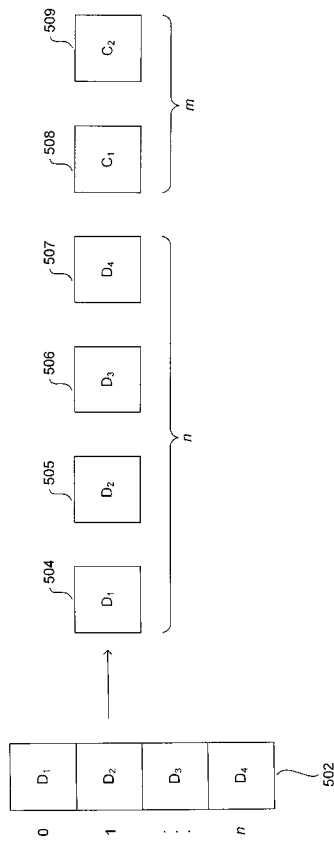
【図 3】



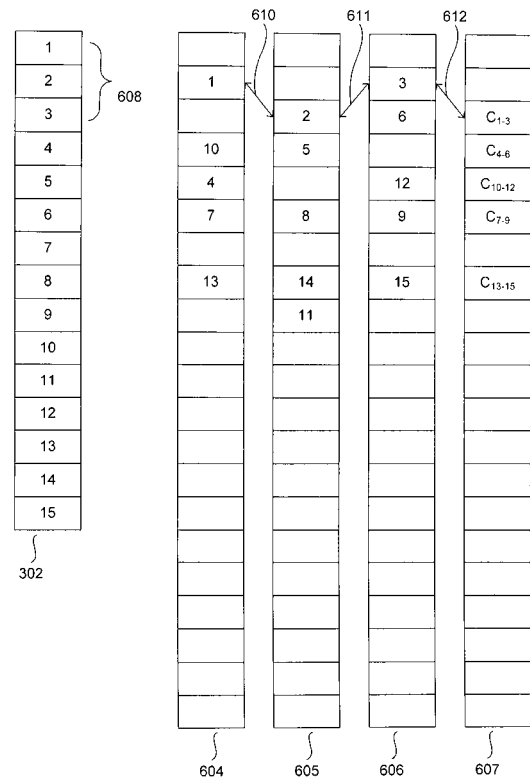
【図 4】



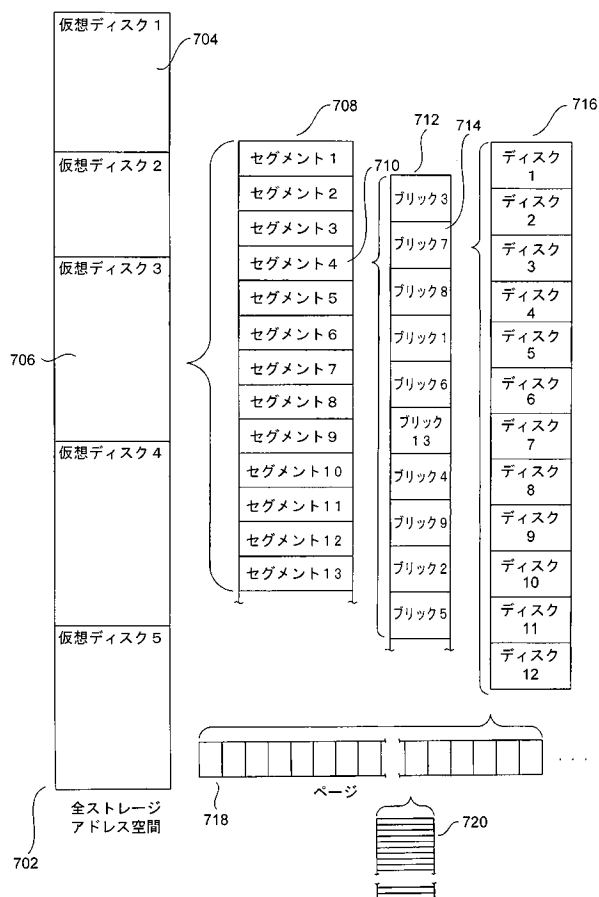
【図 5】



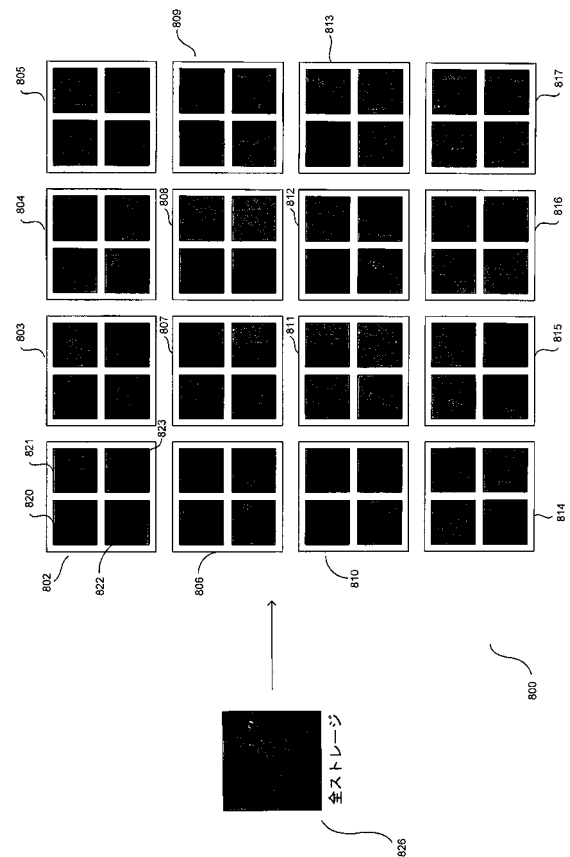
【図 6】



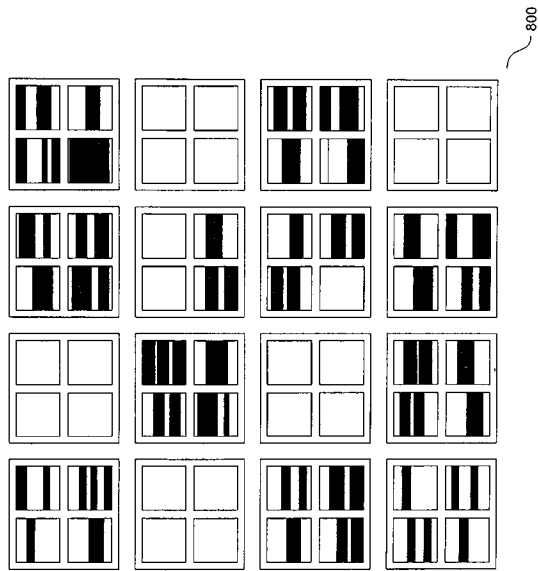
【図 7】



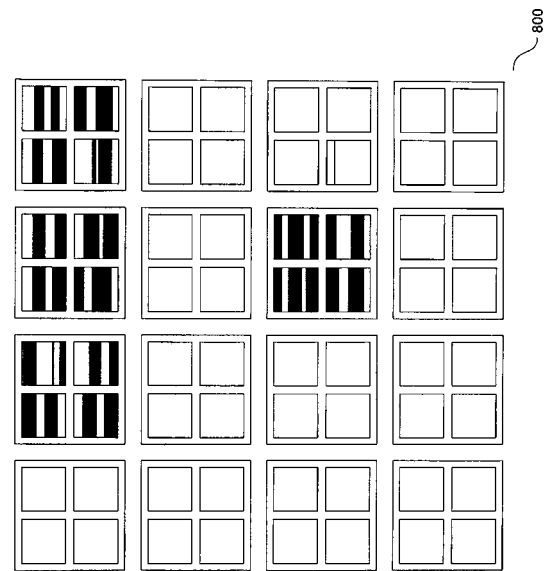
【図 8 A】



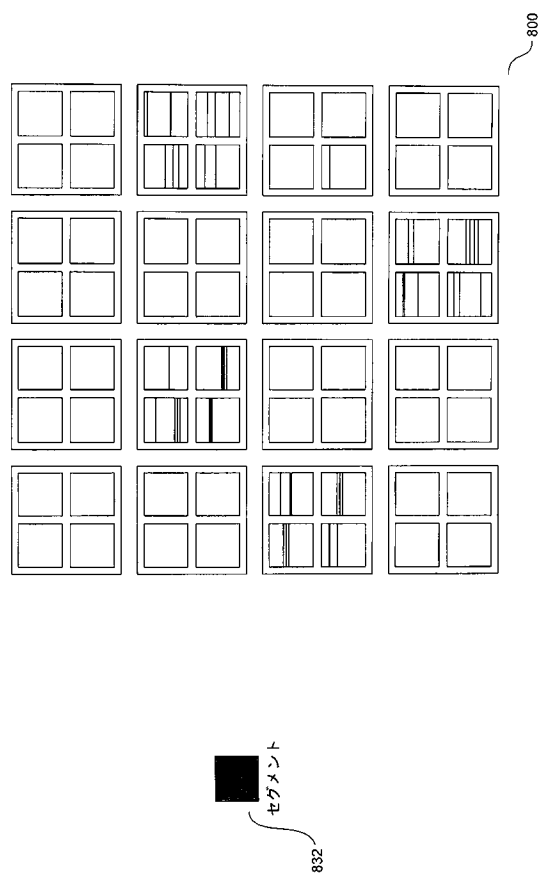
【図 8 B】



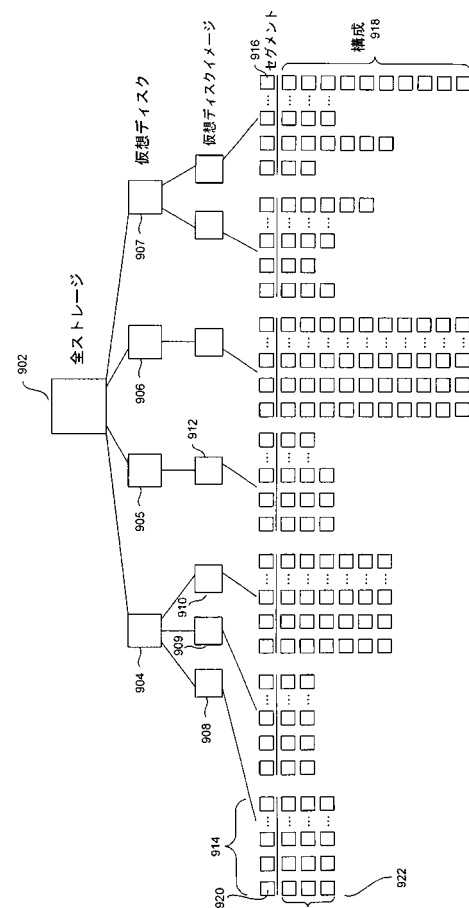
【図 8 C】



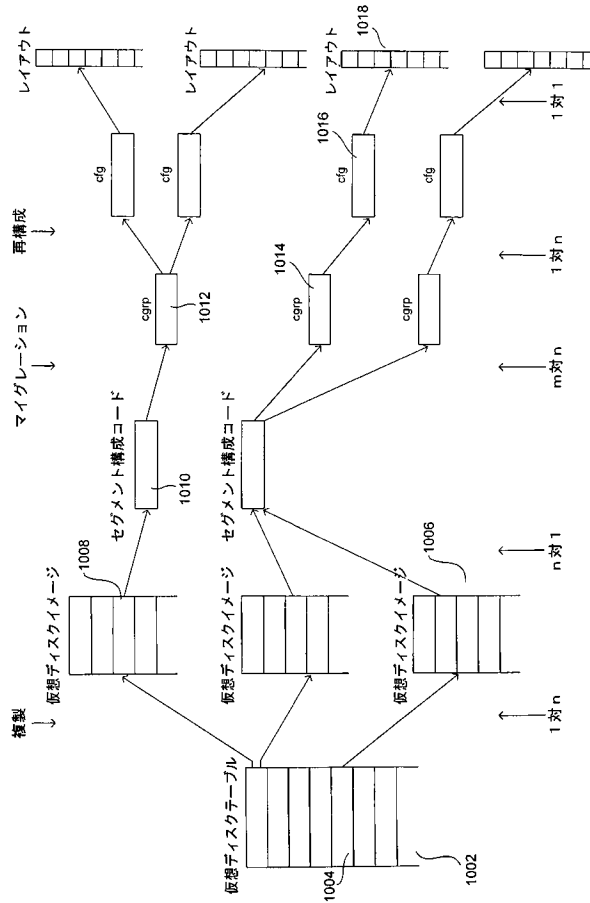
【図 8 D】



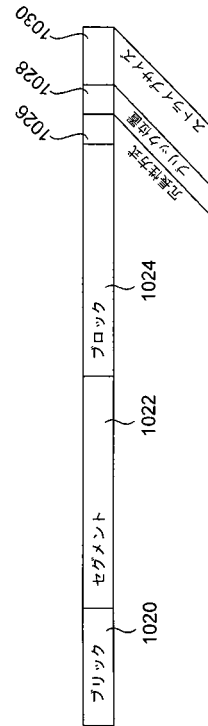
【図 9】



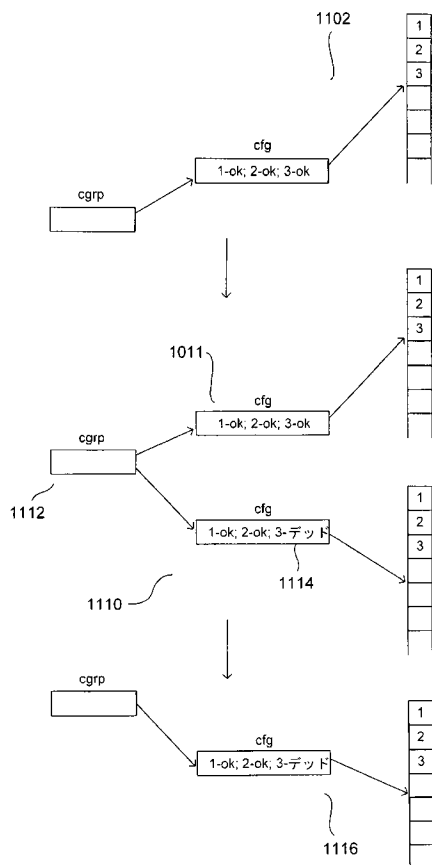
【図 10 A】



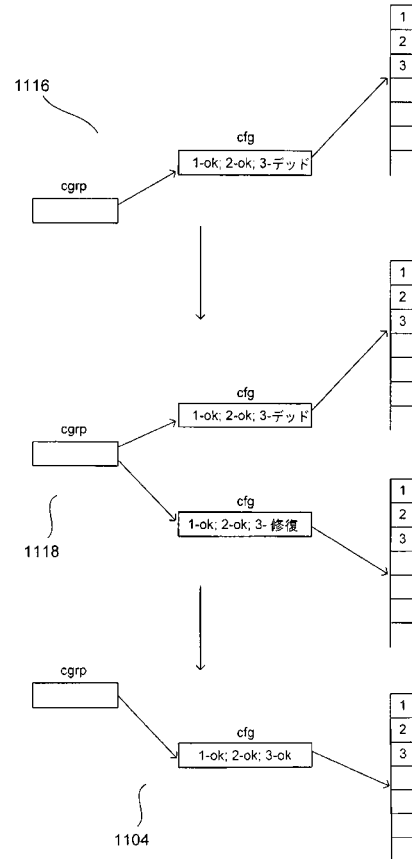
【図 10 B】



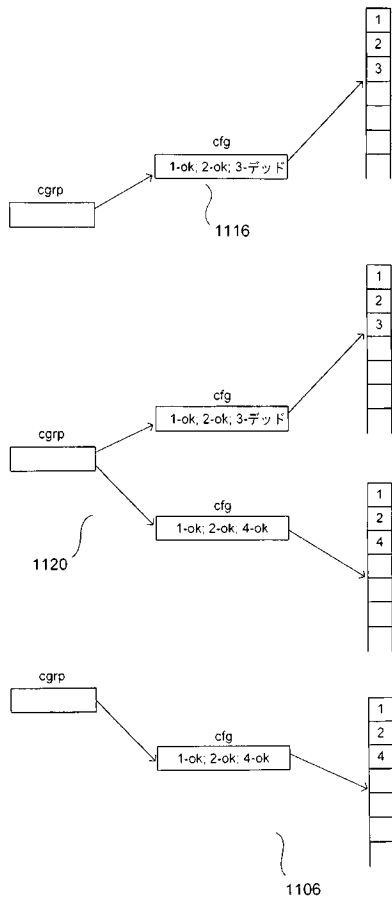
【図 11 A】



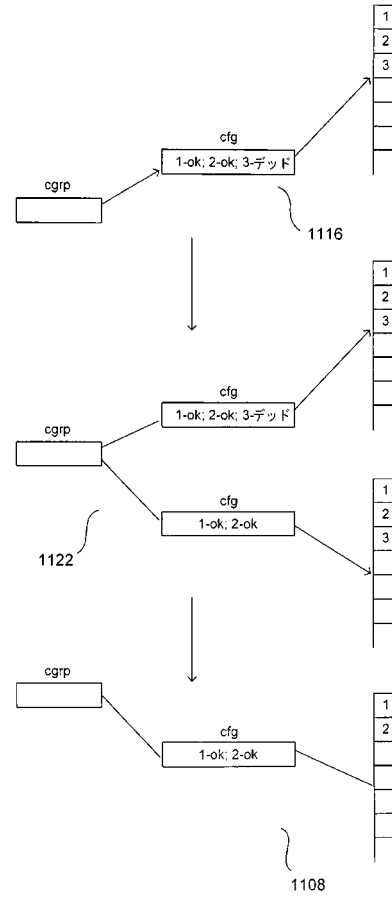
【図 11 B】



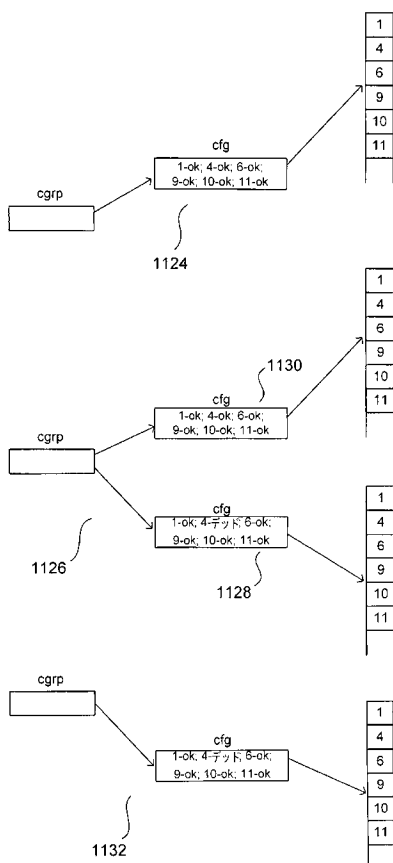
【図 1 1 C】



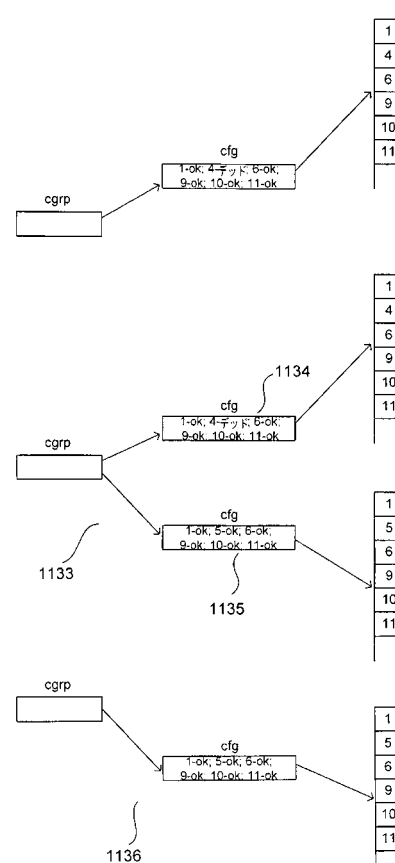
【図 1 1 D】



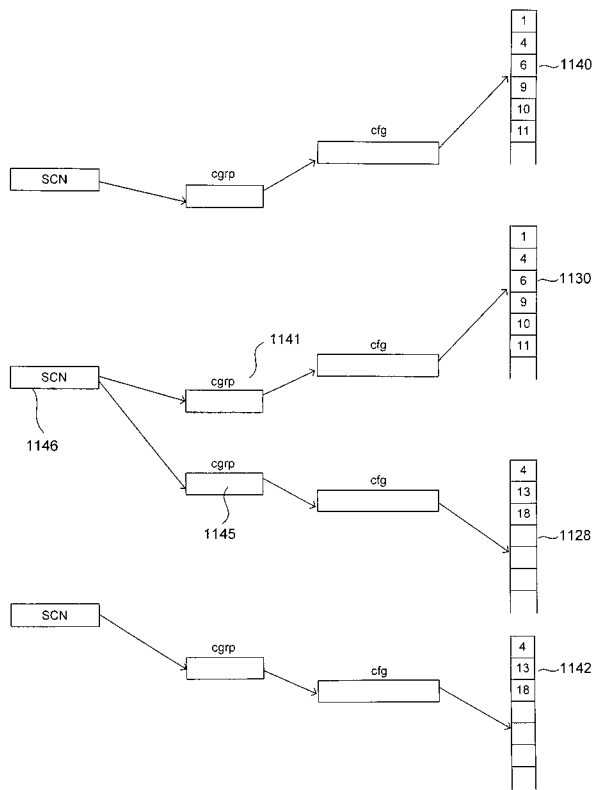
【図 1 1 E】



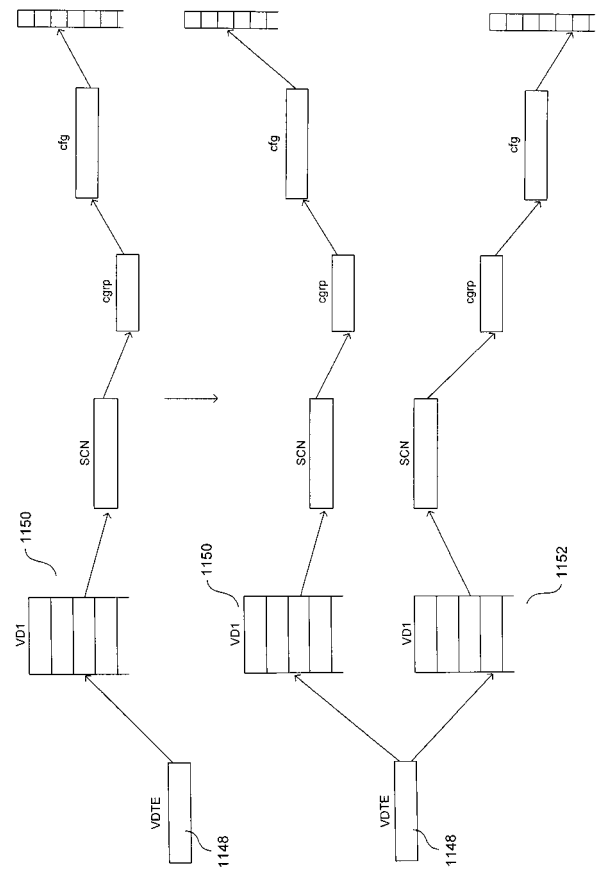
【図 1 1 F】



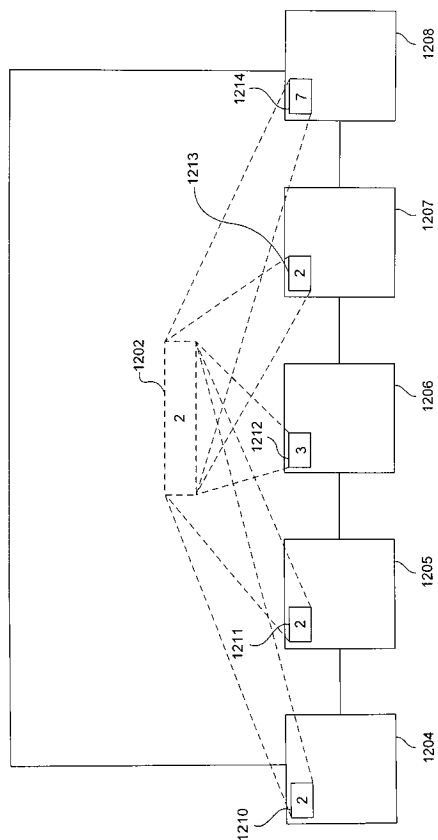
【図 1 1 G】



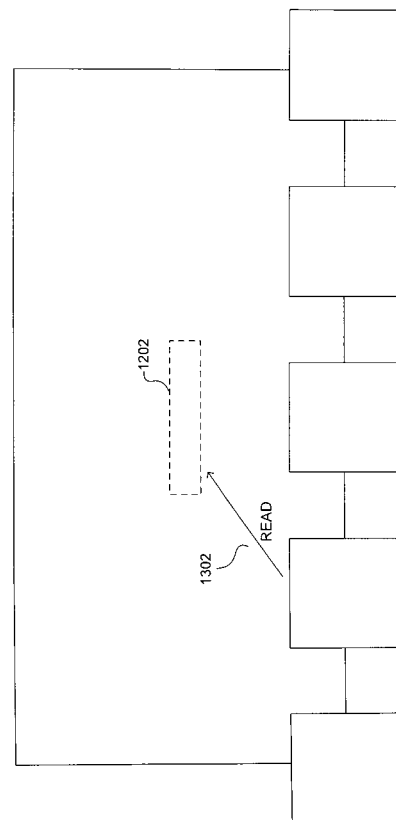
【図 1 1 H】



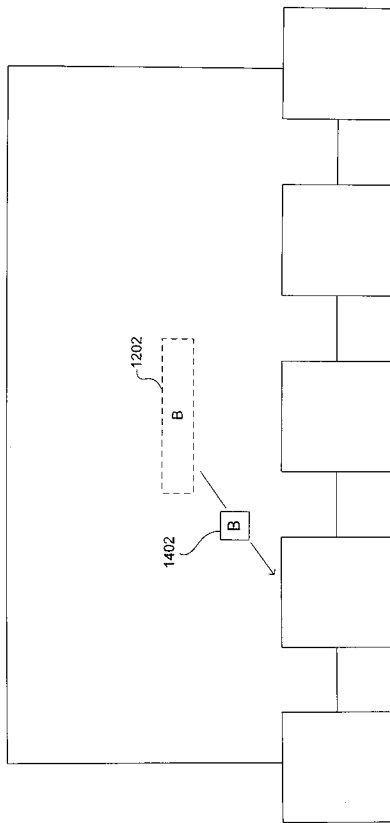
【図 1 2】



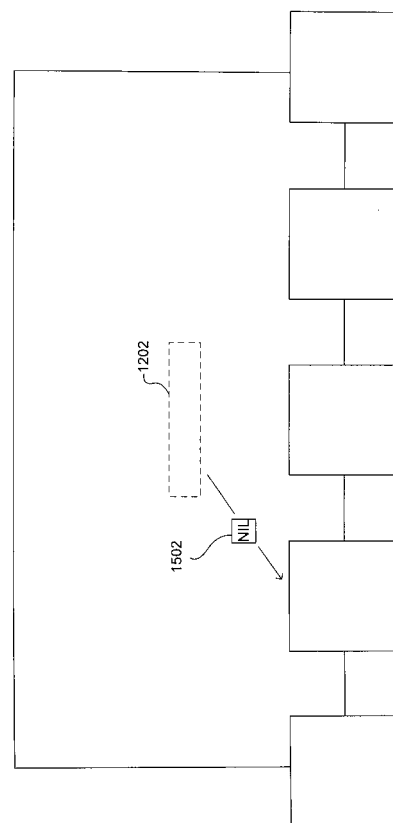
【図 1 3】



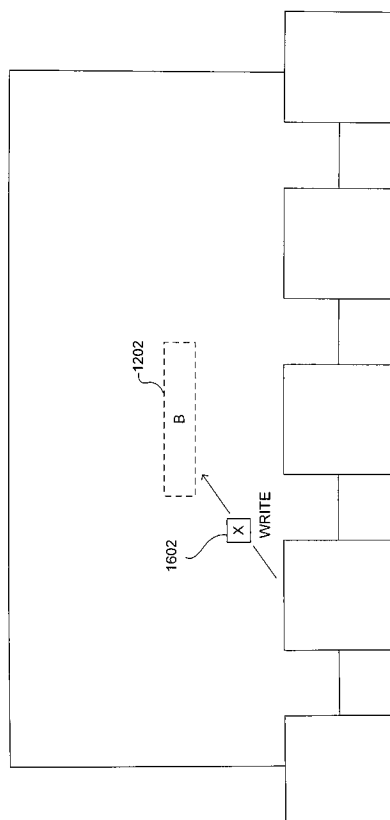
【図 1 4】



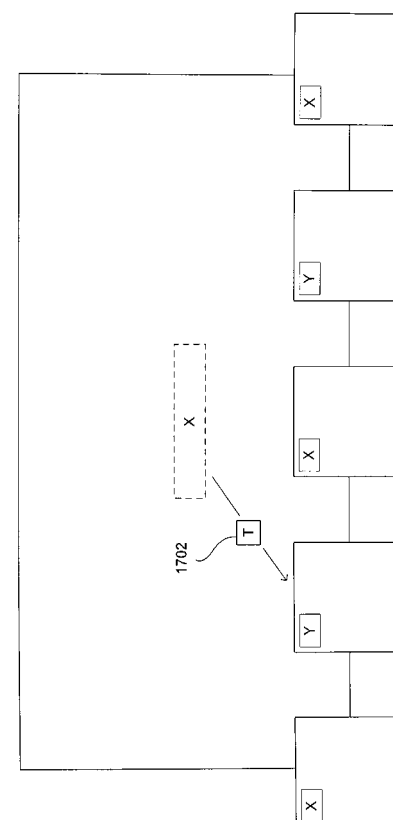
【図 1 5】



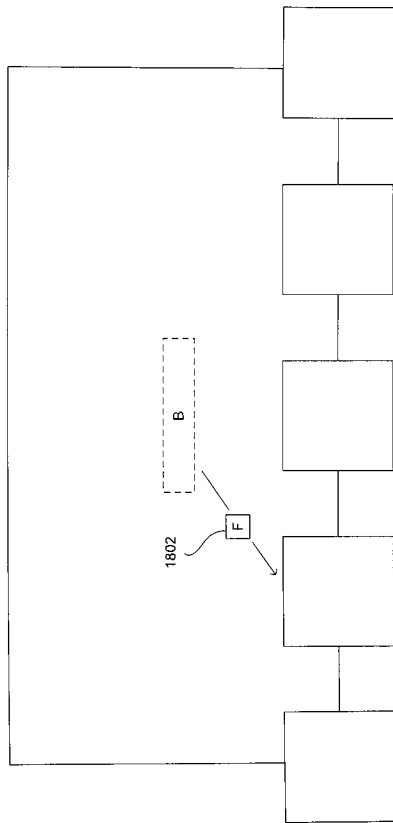
【図 1 6】



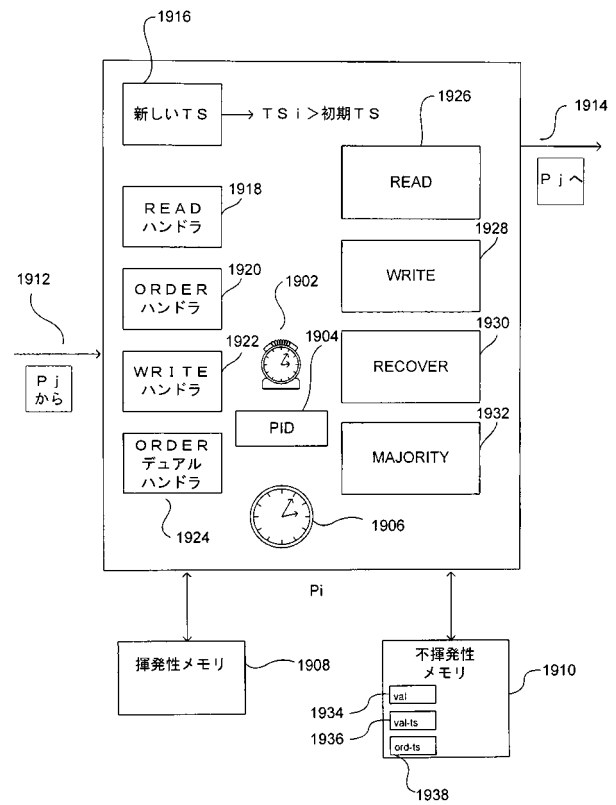
【図 1 7】



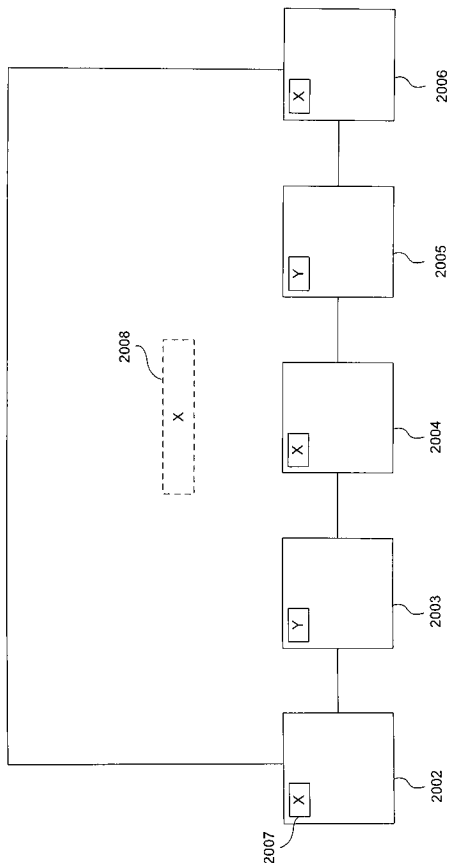
【図 18】



【図 19】



【図 20】



【図 21】

```

1 : procedure read()
2 :   返答 ← majority([Read, val-ts])
3 :   if すべての返答のステータスがTRUEである then return val
4 :   else return recover()

1 : procedure write()
2 :   ts ← newTS()
3 :   返答 ← majority([Order, ts])
4 :   if 返答のいずれかのステータスがFALSEである then return NOK
5 :   返答 ← majority([Write, val, ts])
6 :   if すべての返答のステータスがTRUEである then return OK
7 :   else return NOK

1 : procedure recover()
2 :   ts ← newTS()
3 :   返答 ← majority([Order&Read, ts])
4 :   if 返答のいずれかのステータスがFALSEである then return NIL
5 :   val ← 返答から最も高いval-tsを有する値
6 :   返答 ← majority([Write, val, ts])
7 :   if すべての返答のステータスがTRUEである then return val
8 :   else return NIL

1 : procedure majority(msg)
2 :   Send msg to all, 周期的に再送する
3 :   [n+1]個のプロセスから (rep) の受信を待つ
4 :   repがmsgと一致するように
5 :   return 一組の受信された返答

read Handler()
1 : when コーディネータから [Read, ts] を受信する
2 :   ステータス ← (ts = val-ts and ts ≥ ord-ts)
3 :   reply[Read-R, ステータス] to コーディネータ

write Handler()
1 : when コーディネータから [Write, new-val, ts] を受信する
2 :   ステータス ← (ts > val-ts and ts ≥ ord-ts)
3 :   if ステータス then
4 :     val ← new-val; store(val)
5 :     val-ts ← ts; store(val-ts)
6 :     reply[Write-R, ステータス] to コーディネータ

order&Read Handler()
1 : when コーディネータから [Order&Read, ts] を受信する
2 :   ステータス ← (ts > max(val-ts, ord-ts))
3 :   if ステータス then ord-ts ← ts; store(ord-ts)
4 :   reply[Order&Read-R, val-ts, val, ステータス]

order handler()
1 : when コーディネータから [Order, ts] を受信する
2 :   ステータス ← (ts > max(val-ts, ord-ts))
3 :   if ステータス then ord-ts ← ts; store(ord-ts)
4 :   reply[Order-R, ステータス]

```

【図 2 2】

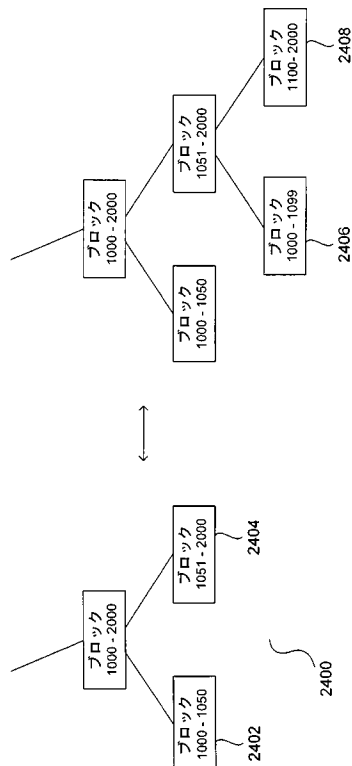
```

// 1ノコーディネータコード「idx」はスクリプト内でのブロック番号である。
proc write(val, idx)
  ts ← NewTimestamp()
  send [Order, [idx], ts] to セグメントグループのブリック
  if m-クォーラムが「yes」を返答 and idx番目のブリックがoldvalで返答
    デルタ←Delta(oldval, val, idx)
    send [Write-EO, val, ts] to idx番目のブリック
    send [Write-EO, NULL, ts] to 他のデータブリック
    send [Write-EO, デルタ, ts] to パリティブリック
    if m-クォーラムが「yes」を返答
      send [コミット, ts] toセグメントグループのブリック
    return OK
  return ABORTED
proc read(idx)
  send [Read] toセグメントグループのブリック
  if m-クォーラム及びidxが「yes」を返答 and すべてのタイムスタンプが等しい
    return idx番目のブリックによって返されたval
  ts ← NewTimestamp() // 低速回復パスが開始する
  send [Order&ReadLog, ts] toセグメントグループのブリック
  ts' ← 少なくともm個の返答に現れる最大のタイムスタンプを選んだもの
  ストリップ=ts'の元のストリップを再構築したもの
  send [Write, strip[i], ts] to i番目のブリック for セグメントグループの各i
  if m-クォーラムが「yes」を返す
    send [commit, ts] toセグメントグループのブリック
    return strip[idx]
  return ABORTED

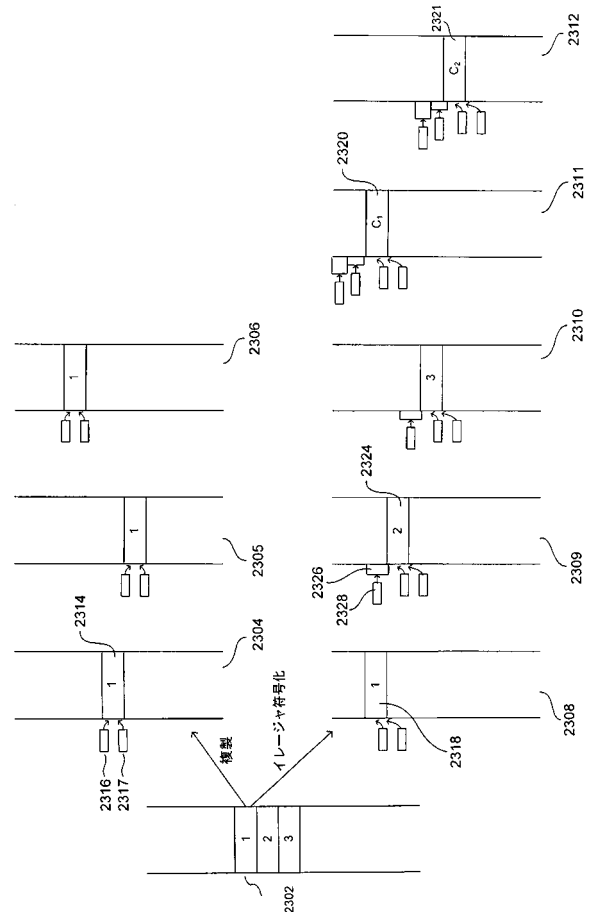
// ストレージハンドラコード
when [Write-EO, newval, ts] を受信する
  status← (ts > valTs and ts ≥ ordTs)
  if status
    if このブリックがパリティ用である, [xor(newval, val), ts]をログに追加する
    elseif newval ≠ NULL, [newval, ts] をログに追加する
    else [val, ts] をログに追加する
  reply status
when [Order&ReadLog, ts] を受信する
  status← (ts > max(valTs, ordTs))
  reply [status, すべてのログエントリ]
when [Commit, ts] を受信する
  しばらくの間、新しくないタイムスタンプを有する要求の拒否を待つ
  If tsのログエントリがある
    val ← 関連付けられるログ値
    タイムスタンプts又はそれよりも小さなタイムスタンプを有するログエントリを削除する

```

【図 2 4】



【図 2 3】



【図 2 5】

レジスタメソッド

```

1: val-ts=ord-ts=initialTS
2: val←val
3: procedure read()
4:   返答←Q-form([Read, val-ts])
5:   if すべての返答のステータスが真である then return val
6:   else return recover()
7: procedure write(val)
8:   ts←newTS()
9:   返答←Q-form([Order, ts])
10:  if いずれかの返答のステータスが偽である then return NOK
11:  返答←Q-form([Write, val, ts])
12:  if すべての返答のステータスが真である then return OK
13:  else NOKをreturn
14: procedure recover()
15:   ts←newTS()
16:   返答←Q-form([Order&Read, ts])
17:  if いずれかの返答のステータスが偽である then return NIL
18:  val←返答から最も高いval-tsを有する値
19:  返答←Q-form([Write, val, ts])
20:  if すべての返答のステータスが真である then return val
21:  else return NIL

```

レジスタメッセージハンドラ

```

1: when Q-receive([Read, ts], コーディネータ)
2:   ステータス←(ts=val-ts and ts≥ord-ts)
3:   Q-reply([ステータス], コーディネータ)
4: when Q-receive([Order, ts], コーディネータ)
5:   ステータス←(ts > max(val-ts, ord-ts))
6:   if ステータス then ord-ts←ts: store(ord-ts)
7:   Q-reply([ステータス], コーディネータ)
8: when Q-receive([Write, newval, ts], コーディネータ)
9:   ステータス←(ts > val-ts and ts≥ord-ts)
10:  if ステータス then
11:    val←newval: store(val)
12:    val-ts: store(ord-ts)
13:  Q-reply([ステータス], コーディネータ)
14: when Q-receive([Order&Read, ts], コーディネータ)
15:   ステータス←(ts > max(val-ts, ord-ts))
16:   if ステータス then ord-ts←ts: store(ord-ts)
17:   Q-reply([val-ts, val, ステータス], コーディネータ)

```

【図 26】

クォーラムベースの通信

```

1: activeConfs←tempConfs←{cinit}
2: msgSet←tempMsgs←φ
3: goConf←cu
4: procedure Q-form(req)
5:   msg←[Reply, rep, activeConfs, goConf]
6:   msgSet←msgSet∪{msg}
7:   await repsの要素がreqに一致し、且つ、activeConfsの
     すべての構成のクォーラムを構成するような
     repSetのサブセットreps
8:   msgSet←msgSet\{msg}
9:   replySet←replySet∪reps
10:  return reps
11: procedure Q-reply(rep, target)
12:  [Reply, rep, activeConfs, goConf]を
   ターゲットへ送信する、周期的に再送する
13: when activeConfs.tempConfs≠φ or
   msgSet.tempMsgs≠φ
14:  msgSetのメッセージをactiveConfsのすべての
   プロセスへ送信する、周期的に再送する
15:  tempConfs←activeConfs
16:  tempMsgs←msgSet
17: when [Request, req, ac, gc]をpから受信する
18:  update-confs(ac, gc)
19:  trigger Q-receive(req, p)
20: when [Reply, rep, ac, gc]を受信する
21:  replySet←replySet∪{reps}
22:  update-confs(ac, gc)
23: when conf-deliver(c)
24:  if c > gcConf then
25:    activeConfs←activeConfs∪{c}
26:    store(gcConf, activeConfs)
27: procedure update-confs(ac, gc)
28:  if gc > gcConf then
29:    gcConf←gc
30:    activeConfs←activeConfs∪{c}
31:  c≧gcConfを有するactiveConfsのすべてのcを削除する
32:  store(gcConf, activeConfs)

```

簡単なガベージコレクション方式

```

1: when [activeConfs] > 1
2:  cand←min(aactiveConfs)
3:  stat←recover()
4:  if stat≠NIL and cand=min(activeConfs) then
5:    activeConfs←activeConfs\{cand}
6:    goConf←cand
7:    store(gcConf, activeConfs)

```

楽観的状态同期

```

1: when [activeConfs] > 1
2:  cand←min(aactiveConfs)
3:  replicas←Q-form([Read, val-ts])
4:  if すべての返答のステータスが真である then
5:    stat←true
6:  else
7:    stat←(recover()≠NIL)
8:  if stat and (cand=min(activeConfs)) then
9:    activeConfs←activeConfs\{cand}
10:   goConf←cand
11:   store(gcConf, activeConfs)

```

【図 27】

Sc_read()

```

返答←totality([Read, val-ts, activeConfs, gsConf])
If すべての返答のステータスがTRUEである then return val
Else return sc_recover()

```

Sc_write()

```

Ts ←newts()
返答←totality([Order, ts, activeConfs, gsConf])
If いずれかの返答のステータスがFALSEである then return NOK
返答←totality([Write, ts, v, activeConfs, gsConf])
If いずれかの返答のステータスがFALSEである then return NOK
return OK

```

Sc_recover()

```

Ts ←newts()
返答←totality([Order&Read, ts, activeConfs, gsConf])
If いずれかの返答のステータスがFALSEである then return NIL
Val←返答からの最も高いval-tsを有する値
返答←totality([Write, val, ts, activeConfs, gsConf])
If いずれかの返答のステータスがFALSEである then return NIL
return val

```

Totality(req)

```

For req, activeConfsのあらゆるvdi
  1つの返答←vdi.req-func(req)
Update_Confs(req, activeConfs, req, gcConf)
複数の返答←複数の返答∪1つの返答
return 複数の返答

```

【図 28】

