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

特許第5274772号
(P5274772)

(45)発行日　平成25年8月28日(2013.8.28)　(24)登録日　平成25年5月24日(2013.5.24)

(51)Int.Cl.　F I
G06F 12/00 (2006.01)　G06F 12/00 501A
G06F 3/06 (2006.01)　G06F 12/00 517
G06F 3/06 301Z

請求項の数 15　(全 32 頁)

(21)出願番号　特願2006-534180 (P2006-534180)
(86)(22)出願日　平成16年9月30日(2004.9.30)
(65)公表番号　特表2007-507811 (P2007-507811A)
(43)公表日　平成19年3月29日(2007.3.29)
(86)国際出願番号　PCT/US2004/032490
(87)国際公開番号　WO2005/033938
(87)国際公開日　平成17年4月14日(2005.4.14)
審査請求日　平成19年9月19日(2007.9.19)
審判番号　不服2012-744 (P2012-744/J1)
審判請求日　平成24年1月13日(2012.1.13)
(31)優先権主張番号　10/675,188
(32)優先日　平成15年9月30日(2003.9.30)
(33)優先権主張国　米国(US)

(73)特許権者　505014122
シマンテック・オペレーティング・コーポレーション
アメリカ合衆国・９４０４３・カリフォルニア州・マウンテンビュー・エリス　ストリート・３５０
(74)代理人　100147485
弁理士　杉村　憲司
(74)代理人　100158148
弁理士　荒木　淳
(74)代理人　100166213
弁理士　永久保　宅哉

最終頁に続く

(54)【発明の名称】データ・ストレージ内の時相データを維持するためのシステムおよび方法

(57)【特許請求の範囲】
【請求項１】
現在データと履歴データとを含む時相データに対する入出力要求を生成するように構成したアプリケーションと、
前記時相データを物理ストレージに格納するように構成され、前記物理ストレージにマップされた論理デバイスである時相ボリュームと、
該時相データに対して、前記入出力要求が示すオペレーションを時相ボリュームレベルで実行するための時相ボリューム・マネージャと
を備え、
前記時相ボリューム・マネージャは、さらに、
前記入出力要求によって指定されたタイムスタンプを前記時相データに割り当てるための複数のチェックポインティング・オペレーションを行い、
生成されるべきスライス・イン・タイム・イメージの要求を指定し、且つ２つのタイムスタンプを指定するスライス・イン・タイム入出力要求を前記アプリケーションから受けとり、ここで、該スライス・イン・タイム・イメージは、前記時相データの一部であって、該２つのタイムスタンプの間の時間期間によって定められる部分に対応するものであり、
前記スライス・イン・タイム入出力要求に応答して、前記スライス・イン・タイム・イメージを生成する
ように構成されているシステム。

【請求項２】
前記入出力要求の一つはタイムスタンプを指定する書込み要求であり、前記時相ボリューム・マネージャは、該書込み要求に応答して、その書込み要求によって指定された時相データを、前記時相ボリュームに書き込むとともにその書き込んだ時相データに前記タイムスタンプを付すようにさらに構成されていることを特徴とする請求項１に記載のシステム。
【請求項３】
前記入出力要求の一つはタイムスタンプを指定する読取り要求であり、前記時相ボリューム・マネージャは、該読取り要求に応答して、その読取り要求のタイムスタンプによって指定された時相データを、前記アプリケーションに返却するようにさらに構成されていることを特徴とする請求項１に記載のシステム。
【請求項４】
前記入出力要求の一つが、時相ボリュームのポイント・イン・タイム・イメージを作成するよう指定するとともにそのポイント・イン・タイム・イメージの作成時点を示すタイムスタンプを指定し、前記時相ボリューム・マネージャは、その作成時点のポイント・イン・タイム・イメージを生成するようにさらに構成されていることを特徴とする請求項１に記載のシステム。
【請求項５】
前記入出力要求が、前記入出力要求によって指定されたタイムスタンプに対応する時点で前記時相ボリュームの履歴データを切り捨てるよう指定し、前記時相ボリューム・マネージャは、前記入出力要求によって指定された時点で前記時相ボリュームの履歴データを切り捨てるようにさらに構成されていることを特徴とする請求項１に記載のシステム。
【請求項６】
前記入出力要求が、その入出力要求によって指定された時点まで前記時相ボリュームの特定の部分の履歴データをたどらせるよう指定し、前記時相ボリューム・マネージャは、前記時相ボリュームの当該特定の部分に対して、当該入出力要求が示すオペレーションを実施するようにさらに構成されていることを特徴とする請求項１に記載のシステム。
【請求項７】
前記時相ボリュームは複数の領域に分割されており、
前記入出力要求が、該入出力要求が示すオペレーションが実施される１つまたは複数の時相ボリュームの領域を指示することを特徴とする請求項１に記載のシステム。
【請求項８】
時相ボリューム・マネージャが、ハードウェア処理装置によって、
物理ストレージに格納された時相ボリュームに記憶された時相データにタイムスタンプを割り当てるための複数のチェックポインティング・オペレーションを行うステップであって、前記時相データは現在データと履歴データとを含み、前記時相ボリュームは前記物理ストレージにマップされた論理デバイスであるステップと、
前記チェックポインティング・オペレーションを行うステップ後に、スライス・イン・タイム入出力要求をアプリケーションから受けとるステップであって、前記スライス・イン・タイム入出力要求は、生成されるべきスライス・イン・タイム・イメージの要求を指定し、且つ２つのタイムスタンプを指定し、前記スライス・イン・タイム・イメージは、前記時相データの一部であって、該２つのタイムスタンプの間の時間期間によって定められる部分に対応するものであるステップと、
前記スライス・イン・タイム入出力要求を受けとるステップ後に、前記スライス・イン・タイム入出力要求に応答して、前記スライス・イン・タイム・イメージを生成するステップと
を含み、
前記時相ボリューム・マネージャは、前記ハードウェア処理装置によって、前記時相データに対するオペレーションを時相ボリュームレベルで実行し、
前記オペレーションは、前記時相ボリューム・マネージャが前記アプリケーションから

受け取った入出力要求によって指定され、
前記入出力要求は、前記チェックポインティング・オペレーションのための前記タイムスタンプを指定すること
を特徴とする方法。
【請求項９】
前記入出力要求の一つがタイムスタンプを指定する書込み要求であり、前記時相ボリューム・マネージャが、前記書込み要求に応答して、その書込み要求によって指定された時相データを前記時相ボリュームに書き込むとともにその書き込んだ時相データに前記タイムスタンプを付すことを特徴とする請求項８に記載の方法。
【請求項１０】
前記入出力要求の一つがタイムスタンプを指定する読取り要求であり、前記時相ボリューム・マネージャが、前記読取り要求に応答して、その読取り要求のタイムスタンプによって指定された時相データを前記アプリケーションに返却することを特徴とする請求項８に記載の方法。
【請求項１１】
前記入出力要求の一つは、ポイント・イン・タイム・イメージの作成時点を示すタイムスタンプを指定するとともにそのタイムスタンプに対応する時相ボリュームの部分のポイント・イン・タイム・イメージを作成するよう指定し、前記時相ボリューム・マネージャが該指定されたタイムスタンプに対応する時相ボリュームのポイント・イン・タイム・イメージを生成することを特徴とする請求項８に記載の方法。
【請求項１２】
前記入出力要求の一つは、該入出力要求によって指定されたタイムスタンプに対応する時点で前記時相ボリュームの履歴データを切り捨てるよう指定し、前記時相ボリューム・マネージャは、前記入出力要求によって指定された前記時点で前記時相ボリュームの履歴データを切り捨てることを特徴とする請求項８に記載の方法。
【請求項１３】
前記入出力要求の一つは、該入出力要求によって指定された時点まで前記時相ボリュームの特定の部分の履歴データをたどらせるよう指定し、前記時相ボリューム・マネージャは、前記時相ボリュームの当該特定の部分に対して、当該入出力要求が示すオペレーションを実施することを特徴とする請求項８に記載の方法。
【請求項１４】
前記時相ボリュームは複数の領域に分割されており、
前記入出力要求の一つが、該入出力要求が示すオペレーションが実施される１つまたは複数の時相ボリュームの領域を指示する
ことを特徴とする請求項８に記載の方法。
【請求項１５】
ハードウェア処理装置に、前記請求項８乃至１４のいずれかの方法を実行させるためのプログラム命令を具備したコンピュータ可読記憶媒体。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、コンピュータ・システムの分野に関し、より詳細には、データ・ストレージ・システムに関する。
【背景技術】
【０００２】
典型的な従来技術のデータ・ボリュームでは、データに変更があった場合には、その場で対応するブロックが変更される。これは、今日のリレーショナル・データベースのようなアプリケーションやファイル・システムに関する限りは適切である。しかし、従来のデータ・ボリューム管理技術は、時相情報すなわち径時変化情報を維持し処理することには不向きである。データ・マイニング、データ・ウェアハウジング、メディア・ライブラリ

、医療レコードなどのアプリケーションは、時相情報すなわち径時変化情報の維持／または処理が必要なこともある。従来技術では、これらの分野のアプリケーションが時相データを管理できる単一のインフラストラクチャが存在しない。ストレージのインフラストラクチャが欠けている故に、従来技術のアプリケーションは通常、時相パラダイムをサポートするのに非効率となりがちなブルート・フォース型の方法を使用する。したがって、様々なアプリケーションをまたいで時相データを一般的に管理する、論理デバイス・レベルのインフラストラクチャを提供することが望ましい。

【発明の開示】

【課題を解決するための手段】

【０００３】

データ・ストレージ内の時相データを論理デバイス・レベルで維持するためのシステムと方法に関する諸実施態様を説明する。これらの諸実施態様は、コンテンツを保存するために、論理デバイス（ボリューム）レベルでデータにタイムスタンプを付けるための一般メカニズムを備えている。これらの諸実施態様は、時相ボリュームを管理しそれにアクセスするためのメカニズムを備えている。いくつかの実施態様をボリューム・マネージャと統合することができる。時相ボリュームを管理しそれにアクセスするためのメカニズムを時相ボリューム・マネージャと呼ぶ。時相ボリューム・マネージャの諸実施態様は、アプリケーションおよび／またはアプリケーション・エージェントが時相ボリューム・マネージャと通信して、１つまたは複数の時相ボリューム上の時相情報を管理し追跡するのを可能にするインターフェースを設けている。これらの諸実施態様は、時相データベース、バージョニング・ファイル・システム／レポジトリ、データ・アーカイブ、ストリーミング・メディアなどデータの履歴に働きかけるアプリケーションが、時相データを管理するためのインフラストラクチャを提供する。

【０００４】

時相ボリュームは、現在データに加えて非現在データも維持するボリュームである。時相ボリュームは、それに格納されているデータの履歴を維持し、過去の任意の時点におけるデータのコピーを検索するための方法をアプリケーションに提供することができる。時相ボリュームでは、データのブロックが変更されると常に、まず既存のブロックが保存され、次いで新しいデータが上書きされる。古いバージョンのブロックは、そのブロックがアプリケーションによってデータから削除された場合にも、維持される。これには、過去におけるデータの１つまたは複数の状態のコピーを維持する効果がある。時相ボリュームを、たとえば、ホスト・ベース環境、ネットワーク・ベース環境（スイッチまたはアプライアンス）、アレイ・ストレージ環境で使用することができる。さらに、時相ボリュームを、バンド内、バンド外仮想化において使用することができることに留意されたい。

【０００５】

一実施態様では、アプリケーションは、時相ボリュームを（デバイスとして）直接使用するのを望まない場合は、アプリケーション・エージェント（以下で説明する）を使用して、それらがボリュームの時間特性を利用するのに役立てることができる。これにより、時相データの管理の負担がアプリケーションから取り除かれ、したがって、アプリケーションは、データをどのように格納するかについてではなく、データをどのように消費するかについて集中することができる。

【０００６】

これらの諸実施態様では、それだけに限らないが、入出力制御型チェックポインティング、アプリケーション制御型チェックポインティング、周期型チェックポインティングを含めたいくつかの方法で、時相ボリューム内のデータの履歴を維持することができる。入出力制御型チェックポインティングでは、時相ボリュームを使用するアプリケーションは、時相ボリュームへの書込みおよび／または時相ボリュームの読取りを行うときに、タイムスタンプを生成する。アプリケーションは、その必要がある場合またはそれが望まれる場合、書込みの度にその都度タイムスタンプを生成する代わりに、ＵＮＩＸ（登録商標）環境におけるＩＯＣＴＬ（Ｉ／Ｏ　ｃｏｎｔｒｏｌ　ｃｏｍｍａｎｄ）など、ある領域（

または全体のボリューム）を対象とするタイムスタンプを指定する、入出力要求またはコマンドを発行することができる。この方法を、アプリケーション制御型チェックポインティングと呼ぶこともある。別の代替態様は、時相ボリューム・マネージャが周期的に、たとえば１０秒毎または１０分毎に自動チェックポインティングを行うためのものである。これを、周期型チェックポインティングと呼ぶこともある。

【０００７】

時相ボリュームは次元としての時間を有する。一実施態様では、時相ボリューム・マネージャは、ユーザが時相ボリューム内の時間次元（履歴）をトラバースするのを可能にするインターフェイスを設ける。一実施態様は、それ自体の独立した履歴を有する時相ボリュームの時相イメージを生成するためのメカニズムを備えていることができる。一実施態様は、２つのタイムスタンプ間での時相ボリュームのスライス・イン・タイム・イメージを生成するためのメカニズムを備えていることができる。一実施態様は、時相ボリュームのポイント・イン・タイム時相イメージを生成するためのメカニズムを備えることができる。ポイント・イン・タイム・イメージは、単一時間におけるボリュームのイメージであり、時相ボリュームの１次元イメージと見ることができる。スライス・イン・タイム・イメージも、ポイント・イン・タイム・イメージも、時相イメージのベースとして使用することができる。

【０００８】

以下の詳細な説明は、後で簡潔に説明する添付の図面を参照して行う。

【０００９】

本発明は、例示のため、いくつかの実施形態や例示的な図について説明してあるが、本明細書に記載の諸実施形態または図に限定されるものではないことを、当業者には理解されるであろう。添付の図面やその詳細な説明は、本発明を開示の特定の形態に限定するものではなく、本発明は、それとは対照的に、添付の特許請求の範囲で定義される本発明の趣旨と範囲内にあるすべての変更形態、均等物、代替形態を含むものであることを理解されたい。本明細書で使用する表題は、単なる構成上の目的にすぎず、本明細書の記載または添付の特許請求の範囲を限定するために使用されるものではない。本願の全体を通して使用する「することができる」という用語は、義務的な意味（すなわち、必須という意味）にではなく、任意的な意味（すなわち、そうする可能性があるという意味）で使用する。同様に、「含む」、「含めて」、「備える」という用語は、「それだけに限らない」ということを意味する。

【発明を実施するための最良の形態】

【００１０】

データ・ストレージ内の時相データを論理デバイス・レベルで維持するためのシステムと方法の諸実施形態を説明する。これらの諸実施形態は、コンテンツを保存するために、論理デバイス（ボリューム）レベルでデータにタイムスタンプを付けるための一般メカニズムを提供する。これらの諸実施形態では、本明細書において時相ボリュームと呼ぶこともある時相データ・ボリュームは、管理、入出力オペレーション、レプリケーションのためのインターフェースを設けていることができ、また、それだけに限らないが、バックアップ、復元、階層型ストレージ管理（Hierarchical Storage Management：ＨＳＭ）を含めたオペレーションをサポートすることができる。これらの諸実施形態は、時相ボリュームを管理しそれにアクセスするためのメカニズムを提供する。いくつかの実施形態をベリタス社のＶＥＲＩＴＡＳ　Ｖｏｌｕｍｅ　Ｍａｎａｇｅｒなどのボリューム・マネージャと統合することができる。本明細書では、時相ボリュームを管理しそれにアクセスするためのメカニズムを時相ボリューム・マネージャと呼ぶこともある。時相ボリューム・マネージャの諸実施形態は、アプリケーションが時相ボリューム・マネージャと通信して、１つまたは複数の時相ボリューム上の時相情報を管理し追跡するのを可能にするインターフェースを設けている。

【００１１】

本明細書内で使用されるいくつかの用語を定義する用語解説が、本明細書の最後に設け

られていることに留意されたい。

【００１２】

これらの諸実施形態は、それだけに限らないが、時相データベース（それらが使用している時相モデルは問わない）、バージョニング・ファイル・システム／レポジトリ、データ・アーカイブ、ストリーミング・メディアを含めたデータの履歴に働きかけるアプリケーションが、時相データを管理するためのインフラストラクチャを提供する。これらの諸実施形態は、時相データの管理が必要になる将来のアプリケーション用のビルディング・ブロックとしても働くことができる。

【００１３】

時相ボリュームは、現在データに加えて非現在データも維持するボリュームである。時相ボリュームは、そこに格納されているデータの履歴を維持することができ、したがって、アプリケーションが過去の任意の時点におけるデータのコピーを検索するための方法を提供することができる。通常のボリュームでは、データに変更がある場合には、その場で対応するデータ・ブロックが変更される。時相ボリュームでは、データのブロックが変更される場合には、まず既存のブロックが保存され、次いで新しいデータが上書きされる。古いバージョンのブロックは、そのブロックがアプリケーションによってデータから削除された場合にも維持される。これには、過去におけるデータの１つまたは複数の状態のコピーを維持する効果がある。このプロセスは、ボリューム上のデータの途切れないバージョニングを行い、それに変更があった場合にはそのボリュームのスナップショットを作成するプロセスとも考えることができる。別の一実施形態では、新しいデータを離れた位置に書き込むことができ、時相ボリューム内のメタデータ（データ領域を指すポインタなど）を操作することができる。

【００１４】

時相ボリュームを、それだけに限らないが、ホスト・ベース環境、ネットワーク・ベース環境（スイッチまたはアプライアンス）、アレイ・ストレージ環境を含めたストレージ環境で使用することができる。また、時相ボリュームを、バンド内やバンド外仮想化において使用することもできることに留意されたい。

【００１５】

図１は、一実施形態による、時相ボリュームにおける時相オペレーションを管理する時相ボリューム・マネージャの図である。時相ボリューム・マネージャ１００は、アプリケーション１０６および／またはオペレーティング／ファイル・システム１０４の代わりに、時相ボリューム１０２に対する時相データ・オペレーションを管理することができる。アプリケーション１０６は、時相データベース、バージョニング・ファイル・システム／レポジトリ、データ・アーカイブ、ストリーミング・メディアなど、時相データを管理するためにデータの履歴を維持することに関心のある、どんなタイプのアプリケーションでもよい。時相ボリューム・マネージャ１００は、様々な読取り、書込み、または時相ボリューム１０２に対するその他のオペレーションを実施するための１つまたは複数のＡＰＩを、アプリケーション１０６および／またはオペレーティング／ファイル・システム１０４に提供することができる。

【００１６】

時相ボリュームは、時間次元に関心のある任意のアプリケーションが使用できる、一般的な時相データのインフラストラクチャと考えることができる。一実施形態では、アプリケーションは、時相ボリュームを（デバイスとして）直接使用するのを望まない場合は、アプリケーション・エージェント（以下で説明する）を使用して、それらがボリュームの時間特性を利用するのに役立てることができる。アプリケーション・エージェントは、アプリケーションがデータをどのように格納するかについてではなく、データをどのように消費するかについて集中することができるように、時相データの管理の負担をアプリケーションから取り除く。たとえば、データベースはテーブルのコピーをどのように格納するかを気にせずに、効率的な問合せ処理に集中することができ、ファイル・システムはファイルをどのようにバージョン化するのかを気にせずに、ファイル・オペレーションに集中

することができ、レポジトリはデータ履歴をアーカイブすることにではなく、索引付けをより迅速に行うことに集中することができる。

【００１７】

図２は、一実施形態による、アプリケーション・エージェントを使用して、時相ボリューム・マネージャとインターフェイスするアプリケーションの図である。時相ボリューム・マネージャ２００は、アプリケーション２０６の代わりに、時相ボリューム２０２に対する時相データ・オペレーションを管理することができる。アプリケーション２０６は、時相データベース、バージョニング・ファイル・システム／レポジトリ、データ・アーカイブ、ストリーミング・メディアなど、時相データを管理するためにデータの履歴を維持することに関心のある、どんなタイプのアプリケーションでもよい。時相ボリューム・マネージャ２００は、様々な読取り、書込み、または時相ボリューム２０２に対するその他のオペレーションを実施するための、１つまたは複数のＡＰＩを提供することができる。アプリケーション・エージェント２０４は、アプリケーション２０６に代わって時相マネージャ２００と相互作用するブローカであり、アプリケーション２０６の（一部または全部の）セマンティクスをサポートしている。アプリケーション・エージェント２０４は、アプリケーション２０６が時相ボリューム２０２と容易に相互作用できるようにすることが好ましい。さらに、アプリケーション・エージェント２０４は、アプリケーション２０６がデータを生データのままではなく、必要とするフォーマットで抽出することを可能にする。

【００１８】

データの履歴をストレージ（論理デバイス）レベルで格納することによっても、管理が容易になる。これにより、アプリケーション・レベルでストレージを時間的に管理することの労力が軽減される。１つの例として、データベースは、Ｂツリーなどそれ自体のメタデータにおけるブランチのコピーを作成する必要がなく、ファイル・システムは、それ自体のｉノードのコピーを維持する必要がなく、レポジトリは、それ自体の索引を複製する必要がない。このことは、システム管理者にとっては、異なる時相データベース、異なるバージョニング・ファイル・システム、異なるアーカイブ・レポジトリを管理するのではなく、１つの時相データソースだけを管理することを意味する。

【００１９】

時相ボリュームは、時相情報を維持しそれにアクセスするためのインフラストラクチャを提供する。時相ボリュームは、ファイル・システムやデータベースを含む、すべてのレベルのアプリケーションが使用できる。さらに、時相ボリュームは、ファイル・システムやバックアップ製品（たとえば、ＶＥＲＩＴＡＳ　Ｆｉｌｅ　Ｓｙｓｔｅｍ、およびＶＥＲＩＴＡＳ　Ｎｅｔ　Ｂａｃｋｕｐ製品）と統合することによって、データのアーカイブ、バージョニング、レプリケーション、バックアップ、ＨＳＭのためのビルディング・ブロックとしても使用することができる。時相ボリュームは、後のある時点でのスナップショット、増分バックアップ、レプリケーション、破損したボリュームまたは削除されたファイルの復元用にそれを使用できるように、時相コンテンツを保存しておく。

【００２０】

時相ボリュームは、ＨＳＭなどのプロセスを使用して、データのオフライン・ストレージへの移行／オフライン・ストレージからの呼出しを自動化することにより、仮想的に無制限のストレージ容量をアプリケーションに提供することができる。増分バックアップを含めた自動バックアップを行うために、時相ボリュームを、ＶＥＲＩＴＡＳ　ＮｅｔＢａｃｋｕｐなどのバックアップ・ユーティリティと統合することもできる。時相ウィンドウおよび／または周期型チェックポインティングを使用して、時相ボリュームを対象とする周期的なレプリケーションを行うことができる。

【００２１】

時相ボリュームは、それだけに限らないが、次のうちの１つまたは複数を含む分野のアプリケーションで使用することができる。

・　ビジネス・インテリジェンス／データ・マイニング：これは、顧客データのうちか

ら傾向と習性を発見することに関するものである。通常、かかる結果を発見するには、時相データが必要となる。時相データを使用すると、たとえば、小売業者は、自社の顧客の使用傾向を発見するためにデータ・マイニングを行うことができ、銀行は、クレジット履歴、詐欺などを発見するために顧客データを分析することができる。

・　データ・ウェアハウス：データ・ウェアハウスは、特定の企業に関するすべての情報を格納するために一般に使用される企業規模のデータベースである。データ・ウェアハウスは、データ・マイニング・ツールによって収集される有益な情報のリポジトリであり、知識発見に使用される。これらは、様々な企業のバックエンドとしても働くことができる。時間次元を格納し、それをデータの問合せの際に利用することができるのは、このデータ・ウェアハウス内であり、あらゆるデータ・ウェアハウスは、それ自体を時相データベースにする時間次元を有しており、したがって、本明細書に記載の時相ボリュームに適している。

・　マルチメディアとイメージング：ストリーミング・オーディオ／ビデオ、画像、電子書籍、さらにＸ線やＭＲＩのような医療データは、ある種のバージョニングまたはストリーミング（経時変化コンポーネント）を関連付けることができるデータの例である。メディア・ライブラリは、メディア・ファイルを、時相フォーマットで格納し、接続速度などの要素に基づいてストリーミングし、衛星撮影や環境撮影などの画像データベースは、同じ位置の画像のバージョンを維持し、医療または臨床データベースは、Ｘ線、身体スキャンなどに伴う患者の履歴を維持する。これらは、経時変化メディアを何らかの形で使用でき、したがって、本明細書に記載の時相ボリュームに適しているアプリケーションのほんの一例にすぎない。

・　固定コンテンツ：ＷＯＲＭ（Ｗｒｉｔｅ－Ｏｎｃｅ－Ｒｅａｄ－Ｍａｎｙ：追記型）ストレージとも呼ばれる。本明細書に記載の時相ボリュームを使用すると、ＷＯＲＭストレージ上のデータを過去の時点までトレースすることが可能になる。

・　科学技術計算と研究室：時相ボリュームは、遺伝子データベース、データ解析、パターン発見、予測、集中計算、簡易臨床検査、データ監視、信号処理、数学、バイオインフォーマティックスなど、経時変化データを有するこれらのすべてを対象とするストレージの問題を解決することができる。

・　データ・アーカイブと監査：時相ボリュームは、それが格納するデータの履歴を保存しているので、たとえばその履歴をオフラインで取得し、テープまたはその他のメディアに格納することによって、これをアーカイビングに使用することができる。アーカイブは、後に分析または監査に使用することができる。アーカイブの例としては、それだけに限らないが、旧従業員データベース、旧売上げデータ、センサス・データ、ログ、およびサーバ履歴がある。別の例としては、様々な法律および規制により必要とされる規制上のアーカイビングがある。

・　データ・バージョニング：文書のバージョニングを、時相ボリュームに実装することができる。

【００２２】

一実施形態では、時相ボリューム・マネージャ・レベルで、索引付けシステムまたは構造（たとえば、キャッシュ・オブジェクト、アレイなど）を使用して、時相ボリュームのコンテンツを保護することができる。キャッシュ・オブジェクトは、制限されたストレージ・スペースを使用しながらも、そのユーザにストレージが無限にあるかのような錯覚を投影するオブジェクトとして定義することができる。キャッシュ・オブジェクトにストレージを割り付けることにより、キャッシュ・オブジェクトを、スペース最適化スナップショットを作成するのに使用することができる。キャッシュ・オブジェクトは、そのベースとなるキャッシュ・ボリュームと呼ばれるボリュームから、それ自体のストレージを導出する。ポリシーに基づく無限容量の取決めを守る必要がある場合には、キャッシュ・ボリュームを増大させることもできる。

【００２３】

一実施形態では、ボリュームを１つまたは複数の領域に分割することができる。領域は

、ディスクの１つの物理ブロックから、数キロバイト、数メガバイト、数ギガバイトなどまでのどの大きさの領域でもよい。ボリュームは、複数の領域に分割することができ、各領域は、それぞれに関連するタイムスタンプを有する。一実施形態では、アプリケーション（ファイル・システム、データベースなど）は、その領域にどのようなタイムスタンプを関連付けるべきかを指定することができる。一実施形態では、データが時相ボリュームに書き込まれるとき、アプリケーションがタイムスタンプを指定する。従来技術のシステムでは、書込みに関するタイムスタンプをアプリケーションが指定することが許可されていないことに留意されたい。

【００２４】

これらの諸実施形態では、それだけに限らないが、入出力制御型チェックポインティング、アプリケーション制御型チェックポインティング、周期型チェックポインティングを含めたいくつかの方法で、時相ボリューム内のデータの履歴を維持することができる。

【００２５】

入出力制御型チェックポインティングでは、時相ボリュームを使用するアプリケーションは、時相ボリュームへの書込みおよび／または時相ボリュームの読取りを行ったときに、タイムスタンプを生成する。入出力制御型チェックポインティングでは、アプリケーションは、入出力要求がある度にその都度、タイムスタンプを時相ボリューム・マネージャに供給する、あるいは時相データと何らかの関係がある入出力要求のときにだけ、タイムスタンプを時相ボリューム・マネージャに供給することもできる。本明細書では、入出力要求という用語は、任意の入出力要求、コマンド、入出力制御要求、ＩＯＣＴＬ、あるいはアプリケーションまたはその他のエンティティがボリュームと相互作用するために、たとえばボリュームにデータを読み書きするために使用するその他のメカニズムを指すために使用する。一実施形態では、アプリケーションは、何らかの入出力要求（たとえば、読取りまたは書込み）があれば、供給したタイムスタンプを用いて書込みを行う領域にタイムスタンプを付けるよう指定することができる。一実施形態では、時相ボリューム・マネージャは、入出力制御型チェックポインティングにおいて、タイムスタンプを指定する入出力要求（ＵＮＩＸのＩＯＣＴＬなどの入出力制御コマンドまたは要求）をアプリケーションが発行するのを可能にする入出力要求インターフェースを設けていることができる。一実施形態では、時相ボリューム・マネージャとの時相読取りと時相書込みインターフェイスを使用して、入出力制御型チェックポインティングを実現することができる。たとえば、時相データベースは、入出力制御型チェックポインティングを使用して、書込みの度にその都度タイムスタンプを指定することができる。

【００２６】

図３は、一実施形態による、入出力制御型チェックポインティングを使用して、時相ボリュームを論理デバイス・レベルで管理するための方法の流れ図である。３００で示されるように、まず、あるアプリケーションを対象とする時相データを格納するために、時相ボリュームを生成する。３０２で示されるように、時相ボリューム・マネージャは、アプリケーション（またはエージェントが使用されている場合はアプリケーションのエージェント）から入出力要求を受信する。入出力要求が時相要求である場合は、次いで、入出力要求は、時相ボリューム上の時相データを対象とする１つまたは複数のタイムスタンプを指定する。時相ボリューム・マネージャは、時相ボリュームにアクセスし、時相ボリュームに対する時相オペレーションを要求する際に使用するためのＡＰＩを、アプリケーション（またはアプリケーション・エージェント）に提示する。入出力要求が時相読取り要求である場合は、３０６で示されるように、時相ボリューム・マネージャは、その入出力要求の指定する１つまたは複数のタイムスタンプで指示された時相データを、アプリケーションに返却する。入出力要求が時相書込み要求である場合は、３０８で示されるように、時相ボリューム・マネージャは、その入出力要求の指定するタイムスタンプに従って、時相ボリュームにおける１つ（または複数）の領域のチェックポイントを生成し、３１０で示されるように、その入出力要求の指定するデータを時相ボリュームに書き込む。

【００２７】

アプリケーションは、その必要がある場合またはそれが望まれる場合、書込みの度にその都度タイムスタンプを生成する代わりに、ある領域または全体のボリュームを対象とするタイムスタンプを指定する入出力要求を発行する。この方法を、アプリケーション制御型チェックポインティングと呼ぶこともある。アプリケーション制御型チェックポインティングでは、書込みの度にその都度タイムスタンプを供給するのではなく、時相ボリューム内の新しいチェックポイント（ポイント・イン・タイム・コピー）を指定する入出力要求を発行する。アプリケーション制御型チェックポインティングでは、アプリケーションは、チェックポイント／バージョン（時相ボリュームのポイント・イン・タイム・コピー）をいつ作成するかを時相ボリュームに告げる。一実施形態では、これを、入出力制御（ＩＯＣＴＬ）オペレーションなどの入出力要求を使用して行うことができる。アプリケーション制御型の入出力要求がある度にその都度、１つまたは複数の領域にタイムスタンプが付けられる。一実施形態では、アプリケーション制御型チェックポインティングを、２つ以上の時相ボリュームにまたがって自動的に行うことができる。

【００２８】

図４は、一実施形態による、アプリケーション制御型チェックポインティングを使用して、時相ボリュームを論理デバイス・レベルで管理するための方法の流れ図である。４００で示されるように、まず、あるアプリケーションを対象とする時相データを格納するために、時相ボリュームを生成する。次いで、アプリケーションは、タイムスタンプを指定できず、したがって時相ボリュームのチェックポイントが生成されない１つまたは複数の「通常の」読取りおよび／または書込みを、時相ボリュームに対して行う。４０２で示されるように、時相ボリューム・マネージャは、時相ボリュームを対象とした新しいチェックポイントを生成するよう指定する入出力要求を、アプリケーションから受信する。次いで、４０４で示されるように、時相ボリューム・マネージャは、その入出力要求の指定するタイムスタンプに従って、時相ボリュームのチェックポイントを生成する。

【００２９】

別の方法としては、周期的に、たとえば１０秒毎または１０分毎に自動チェックポインティングを行う方法がある。これを、周期型チェックポインティングと呼ぶこともある。周期型チェックポインティングでは、時相ボリューム・マネージャは、時相ボリューム・マネージャによる論理デバイス（ボリューム）レベルでの周期型チェックポインティングの構成を可能にするインターフェースを設けている。周期型チェックポインティングでは、時相ボリュームのインフラストラクチャ（たとえば、時相ボリューム・マネージャ）が周期的に、データの新しいチェックポイントを作成する。これにより、好ましくは、データへのそれぞれの変更を格納することによって浪費されていたかもしれないストレージ・スペースが節約される。従来技術では通常、あらゆる書込みが保存されることに留意されたい。

【００３０】

図５は、一実施形態による、周期型チェックポインティングを使用して、時相ボリュームを論理デバイス・レベルで管理するための方法の流れ図である。５００で示されるように、まず、あるアプリケーションを対象とする時相データを格納するために、時相ボリュームを生成する。５０２で示されるように、次いで、時相ボリューム・マネージャは、タイムスタンプを指定できず、したがって時相ボリュームのチェックポイントが生成されない入出力要求をアプリケーションから受信する。一実施形態では、入出力要求は、チェックポイントを取得すべき時間間隔（または期間）を指定する。５０４で示されるように、時相ボリューム・マネージャは周期的に（たとえば、ｎ分、ｎ時間、またはｎ日毎に）、時相ボリュームのチェックポイントを生成する。一実施形態では、時相ボリューム・マネージャは、タイムスタンプを指定し、周期型チェックポインティングを使用する場合はチェックポイントを生成するよう指定する入出力要求を、アプリケーションから受信できることに留意されたい。

【００３１】

アプリケーションが時相書込みインターフェイスを使用している場合は、そのアプリケ

ーションは、書込みデータと共にタイムスタンプを指定することができ、ボリューム・マネージャは時相書込みを実行する。一実施形態では、タイムスタンプは、特定の書込みに適用され、ボリューム上に格納されているその他のデータは、それに時相情報を格納することも、格納しないこともできる。言い換えれば、一実施形態では、ボリューム上には時相データも存在すれば、時相データ以外のデータも存在し得る。データが時相データでない場合は、タイムスタンプが付されない。一実施形態では、アプリケーションがデータにタイムスタンプを付けるための入出力要求を発行した場合、あるいは周期型チェックポインティングが実施された場合にだけ、データにタイムスタンプが付される。これを、１つまたは複数の領域あるいは全体のボリュームに対して適用する。

【００３２】

時相ボリュームは、ある次元としての時間を有する。一実施形態では、時相ボリューム・マネージャは、ユーザが時相ボリューム内の時間次元（履歴）をトラバースすることを可能にするインターフェースを設けている。ユーザは、たとえば、時相ボリュームの履歴における以前の各時点での時相ボリュームのデータに対して、１つまたは複数のオペレーションを実施するためのインターフェイスを使用して、時相ボリュームの時間次元をトラバースする。かかるオペレーションの例として、それだけに限らないが、時相ボリューム上の特定のバージョンまたは値のデータの有無を時間次元において探索すること、時相ボリュームをその時相ボリュームの履歴における特定の時点の状態に復元すること、時相ボリュームのスライス・イン・タイム・イメージまたはポイント・イン・タイム・イメージを生成することなどのうちの、１つまたは複数のオペレーションを挙げることができる。一実施形態は、それ自体の独立した履歴を有する時相ボリュームの時相イメージを生成するためのメカニズムを用意する。

【００３３】

一実施形態は、時相ボリュームのスライス・イン・タイム・イメージを生成するためのメカニズムを用意する。一実施形態では、時相ボリューム・マネージャは、ユーザ／アプリケーションがある領域の、たとえばタイムスタンプＴ１とＴ２の間のコンテンツ（スライス・イン・タイム・イメージ）を要求するのを可能にするインターフェースを設けている。スライス・イン・タイム・イメージはそれ自体、時相ボリュームであるが、そのエクステントは当初、タイムスタンプＴ１とＴ２の間に制限される。スライス・イン・タイム・イメージは、２つのタイムスタンプの間の時相ボリュームのスライスである。時相ボリュームのスライス・イン・タイム・イメージは、その時相ボリューム上にあるデータの履歴のスライスを提示する。スライス・イン・タイム・イメージを、それ自体の履歴も保存する別個の時相ボリュームとして閲覧することができる。一実施形態では、スライス・イン・タイム・イメージに対して読取りと書込みを実行する。スライス・イン・タイム・イメージに対しては、その他のオペレーション、たとえばデータ・マイニングを実施することもできる。たとえば、データをある期間にわたって収集することができ、その期間内のある月（たとえば、８月）を対象とするスライス・イン・タイム・イメージを取得することができ、そのスライス・イン・タイム・イメージに対して、データ・マイニングまたはその他のオペレーションを実施することができる。

【００３４】

図６は、一実施形態による、時相ボリュームのスライス・イン・タイム・イメージを生成するための方法の流れ図である。６００で示されるように、まず、あるアプリケーションを対象とする時相データを格納するために、時相ボリュームを生成する。６０２で示されるように、次いで、時相ボリューム・マネージャは、アプリケーションから、少なくともそのうちのいくつかは時相ボリュームのチェックポイントを生成できる時相および／または非時相入出力要求を受信する。周期型チェックポインティングが使用されている場合では、次いで、時相ボリューム・マネージャが周期的に、時相ボリューム内のチェックポイントを生成していたことに留意されたい。６０４で示されるように、時相ボリューム・マネージャは、時相ボリュームを対象とした、時相ボリュームのスライス・イン・タイム・イメージを作成するよう要求する入出力要求をアプリケーションから受信する。６０６

で示されるように、時相ボリューム・マネージャは、その入出力要求に応答して、それが指定する２つのタイムスタンプに従い、時相ボリュームのスライス・イン・タイム・イメージを生成する。

【００３５】

一実施形態は、時相ボリュームのポイント・イン・タイムの時相イメージを生成するためのメカニズムを備えている。ポイント・イン・タイム・イメージは、単一時間におけるボリュームのイメージであり、時相ボリュームの１次元イメージと見ることができる。ポイント・イン・タイム・イメージを時相イメージのベースとして使用する。その場合、ポイント・イン・タイム・イメージは、そのイメージのベースとなる時点に初期化された時相ボリュームと見ることができる。入出力（書込み）がある度にその都度タイムスタンプが供給される場合は、ボリュームのポイント・イン・タイム・イメージの数は有限であるはずである。時相ボリュームは、２つ以上のポイント・イン・タイム・イメージを含むデータ・ボリュームと考えられる。２つ以上のポイント・イン・タイム・イメージを組み合わせることもできる。時相ボリュームのスライス・イン・タイム・イメージは、２つ以上のポイント・イン・タイム・イメージを含む。

【００３６】

図７は、一実施形態による、時相ボリュームのポイント・イン・タイム・イメージを生成するための方法の流れ図である。７００で示されるように、まず、あるアプリケーションを対象とする時相データを格納するために、時相ボリュームを生成する。７０２で示されるように、次いで、時相ボリューム・マネージャは、アプリケーションから、少なくともそのうちのいくつかは時相ボリュームのチェックポイントを生成できる時相および／または非時相入出力要求を受信する。周期型チェックポインティングが使用されている場合には、次いで、時相ボリューム・マネージャが周期的に、時相ボリューム内のチェックポイントを生成することに留意されたい。７０４で示されるように、時相ボリューム・マネージャは、時相ボリュームを対象とした、時相ボリュームのポイント・イン・タイム・イメージを作成するよう要求する入出力要求をアプリケーションから受信する。７０６で示されるように、時相ボリューム・マネージャは、その入出力要求に応答して、それが指定するタイムスタンプに従い、時相ボリュームのポイント・イン・タイム・イメージを生成する。

【００３７】

一実施形態は、時相ボリューム上にある２つのバージョンのデータ間における増分変更を判定するためのメカニズムを備えている。判定された増分変更を、増分バックアップやレプリケーションなどのアプリケーションで使用する。一実施形態では、時相ボリューム・マネージャは、任意の２つの時点の間に時相ボリュームにどのような変化があったのかを判定するためのインターフェースを設けている。たとえば、タイムスタンプ付けを、ある期間にわたって実施する。ユーザは、２つの時間の間でデータのバックアップまたはアーカイブをとることを望むかもしれない。増分イメージを作成する。増分イメージは、差異を示すイメージである。増分イメージを使用して、時間Ｔ１からＴ２までの（たとえば、ある時点から以前のバックアップ時までの）バックアップまたはレプリケーションを生成する。時相ボリュームのコンテンツを使用して、それらの差異を識別する。このメカニズムは、好ましくは、増分バックアップまたはレプリケーション用アプリケーションまたはユーティリティが容易に実装できるようにし、また、アプリケーションまたはユーティリティが、変更部分を算定し、スナップショットを管理することを強いられるのではなく、時相ボリューム・マネージャがスナップショットを管理する。バックアップまたはレプリケーション用アプリケーションは単に、時相ボリューム・マネージャに、時間Ｔ１とＴ２の間における時相ボリュームのコンテンツ（あるいは、コンテンツの差異または差分（ｄｅｌｔａ））を要求するだけである。

【００３８】

一実施形態では、時相ボリュームの内部で維持されているデータの履歴を、たとえばＩＯＣＴＬなどの入出力要求を使用して切り捨てる。スペース不足の故に切捨てが必要なこ

とも、履歴がもはや必要とされなくなった故に切捨てが望まれることもある。一実施形態では、このインフラストラクチャは、時相ボリュームのユーザがどれほどの量の履歴を維持するのかを決定する時相ウィンドウを設定できるようにする。時相ウィンドウの範囲を超える変更は自動的に切り捨てる。一実施形態では、時相ボリューム・マネージャは、スペースを解放するために時相ボリュームを切り捨てるためのインターフェースを設けている。たとえば、このインターフェイスは、時相ボリュームが３ヶ月間保持すべき情報をアプリケーションによって指定できるようにする。３ヶ月よりも古い情報を、時相ボリューム・マネージャによって削除する。３ヶ月よりも古いデータを、周期的に、たとえば毎日または毎週削除する。

【００３９】

一実施形態は、それ自体のポイント・イン・タイム・イメージの１つ、および／または２つの特定のタイムスタンプ間のスライス・イン・タイム・イメージから、ボリュームを迅速に復元するためのメカニズムを備えている。たとえば、あるアプリケーションがそのデータに１０分毎にタイムスタンプを付けており、そのボリュームが何らかの形で破損している場合は、時相データを使用してそのボリュームの（たとえば、最近の破損していないタイムスタンプ付きの時相データへの）復元を実施する。

【００４０】

時相ボリュームと時相ボリューム・マネージャ

以下では、ボリューム・マネージャとキャッシュ・オブジェクトを使用して、データ・ストレージ内の時相データを論理デバイス・レベルで維持するためのシステムと方法に関する例示的な諸実施形態を説明する。これらの諸実施形態は、例示的なものであり、時相ボリュームをイネーブルするためのその他のメカニズムを使用することもできることに留意されたい。これらの諸実施形態は、時相ボリュームを対象とする入出力要求をアプリケーションから受信する手段と、入出力要求に応答して時相ボリュームに対する論理デバイス・レベルの時相オペレーションを実施する手段とを備えている。

【００４１】

これらの諸実施形態は、たとえば、ＶＥＲＩＴＡＳ　Ｖｏｌｕｍｅ　Ｍａｎａｇｅｒ（ＶｘＶＭ）などのボリューム・マネージャを使用して、時相ボリュームを作成し使用するためのメカニズムを備えている。一実施形態は、データの履歴のアーカイブをとるために、キャッシュ・オブジェクトを使用する。一実施形態は、時相ボリュームのアプリケーションまたはユーザが、通常のボリューム・オペレーションに加えて時間ベースのオペレーションも実施できるようにする、１つまたは複数のインターフェースを設けている。

【００４２】

索引付けシステムまたは構造（たとえば、キャッシュ・オブジェクト、アレイ、Ｂ＋ツリーなど）を使用して、ボリューム・マネージャ内で、たとえばＶｘＶＭ内で時相ボリュームを実装する。一実施形態では、キャッシュ・オブジェクトを使用して、時相ボリュームを実装する。図８は、一実施形態による例示的なキャッシュ・オブジェクトを示している。図８に示されるように、キャッシュ・オブジェクトは、制限されたストレージ・スペースを使用しながらも、そのユーザに「無限の」ストレージを投影するオブジェクトである。キャッシュ・オブジェクトは永続的なスペース最適化ストアと考えることもできる。一実施形態では、キャッシュ・オブジェクトは、ユーザがスペース最適化スナップショットを作成できるようにする。スペース最適化スナップショットは、キャッシュ・オブジェクトを介して作成されることにより、好ましくは、その元のボリュームほど多くのストレージを使用しない。実際のデータは、それが元のボリューム上で変更されたときにだけ、スペース最適化スナップショット上に書き込まれる（これをコピー・オン・ライトと呼ぶ）。スペース最適化スナップショットが必要とするどんなストレージ・スペースも、そのベースとなるキャッシュ・オブジェクトから提供される。次いで、キャッシュ・オブジェクトは、それ自体のストレージをキャッシュ・ボリュームから導出する。キャッシュ・オブジェクトは、ユーザが定義したいくつかのポリシーに従い、必要に応じてキャッシュ・ボリュームを増大させることによって、無限のストレージという錯覚を維持する。

【００４３】
一実施形態では、キャッシュ・オブジェクトのスペース最適化を可能にする基本的な技術は、入出力ブロックの再ベクトル化(re-vectoring)である。キャッシュ・オブジェクトは、この再ベクトル化を、永続的な変換マップを使用して実現する。スペース最適化スナップショットに対してコピー・オン・ライトで書き込まれたデータを格納することになる同じ物理ストア（キャッシュ・ボリューム）にこれらの変換マップを格納する。さらに、キャッシュ・オブジェクトを用いると、その上に複数のストレージ・ユニットを切り出すことが可能となり、それによって、キャッシュ・オブジェクト上で複数のスペース最適化スナップショットを作成することが可能になる。

【００４４】
キャッシュ・オブジェクトは、複数のストレージ・ユニットをその上で作り出すことも可能にする。これは、キャッシュ・オブジェクトが異なるボリュームを対象とする複数の仮想アドレス可能範囲を提供することを意味する。したがって、各変換は、ストレージ・ユニットの識別子と、キャッシュ・ボリューム上のデータの物理オフセットにマッピングされるボリューム・オフセットとを備える探索キーから成る。探索可能なアドレス空間を、探索キーのサイズに従ってかなり広くすることができ、（無限ではないにせよ）膨大なストレージ・スペースが与えられている場合には、さらに広くすることもできる。これは、実際のデータ・ブロックへの再ベクトル化または変換に使用される構造が、探索可能性が高く、非常に効率的なものになることを意味する。Ｂ＋ツリーは、この用件を満たす１つの構造であり、例示のために本明細書の諸実施形態において使用されているが、その他の実施形態では、他の構造も使用することに留意されたい。

【００４５】
一実施形態では、キャッシュ・オブジェクト用のＢ＋ツリーを、スーパー・ブロック、復元ログ・エリア、フリー・ブロック・リストなど、その他のメタデータと共にキャッシュ・ボリュームに格納する。メタデータ以降のエリアは、データ領域を格納するのに使用される。キャッシュ・オブジェクト内のＢ＋ツリーの各ノードが、（たとえば、ボリューム・マネージャによって決定される）１つのページ・サイズである。このツリーのリーフ・ノードが、キャッシュ・ボリューム上の実際の物理データ・ブロックを指す。

【００４６】
一実施形態では、キャッシュ・オブジェクト上のあらゆる入出力が、再ベクトル化される。再ベクトル化は、Ｂ＋ツリーを使用して行うことができ、入出力ではまず、キャッシュ・ボリューム上のデータの物理オフセットを割り出すために、Ｂ＋ツリー・ウォークを実施する。「有効」領域へのすべての読取りと書込みは、再ベクトル化する必要があることもある。領域は、アプリケーションによって１回または複数回そこへの書込みが行われた場合に「有効」と見なされ、そうでない場合は「無効」と見なされる。ボリュームの無効領域への書込みの場合は、キャッシュ・オブジェクトは、Ｂ＋ツリー内の新しいエントリを新しいオフセットに割り付けることができ、これを、割付け書込み(allocating write)と呼ぶこともある。

【００４７】
以後の書込みで、それ以前のコンテンツを上書きする。一実施形態では、書込み以前のコンテンツが保護されるように、書込みはすべて、独立した割付け書込みとして扱われる。

【００４８】
図９は、一実施形態による時相ボリュームＴＶＯＬの構成を示している。ＴＶＯＬは、通常のデータ・プレックス(data plex)Ｐ１、Ｐ２などを有する。プレックスは、以下の用語解説で定義されている。時相データを保持するＰｔと呼ばれる特別なプレックスも存在する。Ｐｔは、キャッシュ・オブジェクトを介して作成できる１つのストレージ・ユニットＳＤｔを有している。次いで、このキャッシュ・オブジェクトは、それ自体のストレージをキャッシュ・ボリュームから検索する。一実施形態では、キャッシュ・オブジェクトは、先に説明したＢ＋ツリーを使用して、書込みを再ベクトル化する。別の一実施形態

では、時相ボリュームに、通常のデータ・プレックスを所在させず、Ｐｔだけを所在させる。

【００４９】

通常の書込みを使用してボリュームに変更が加えられた場合には、その変更を、Ｐｔを含めたすべてのプレックスに反映させる。Ｐ１やＰ２のような通常のプレックスは、単に新しいデータを用いて既存のデータを上書きすることしかできないが、Ｐｔはこれに対し、まだタイムスタンプの付けられていない最新のデータを上書きする。最新のデータにタイムスタンプが既に付けられている場合は、どんなタイムスタンプも割り当てずに、それを新しい位置に書き込む。

【００５０】

一実施形態では、時相オペレーションを対象とする書込みは、Ｐｔを含めたすべてのプレックスに及ぶことになる。時相オペレーションでは、タイムスタンプを指定することが必要となる。一実施形態では、時相書込みを、図１０に示したように処理する。

【００５１】

図１０は、一実施形態による、時相ボリューム上のデータ・ブロックの変更を処理する様子を示し、さらに、所与の時点でのキャッシュ・オブジェクト下の構造を示している。キャッシュ・オブジェクトのベースとなるキャッシュ・ボリュームは、ディスク上の実際のデータ・ブロックを指すポインタをそれ自体のリーフが保持するツリー型の構造を含む。キャッシュ・オブジェクト内にまだ所在していない領域Ｂへの書込み（割付け書込み）では、タイムスタンプ$Ｔ_0$をエントリ$Ｂ_0$に割り付ける。Ｂのコンテンツに変更があった場合は、$Ｂ_0$を上書きするのではなく、キャッシュ・ボリューム（$Ｂ_1$）上の新しい領域にタイムスタンプ$Ｔ_1$を割り付ける。キャッシュ・オブジェクトは、$Ｂ_1$をリストに追加する。$Ｂ_1$は、そのブロックの最近のコピーを含んでおり、$Ｂ_0$は、それ以前のコピーである。この連鎖は、ブロックＢにさらなる変更が加えられる限り、引き続き行う。ｎ個の変更が加えられた場合は、タイムスタンプ$Ｔ_n$～$Ｔ_0$を用いて、ブロックを$Ｂ_n$．．．$Ｂ_2$、$Ｂ_1$、$Ｂ_0$の順に連鎖する。この連鎖が、非常に長い連鎖となることもある。一実施形態では、かかる場合には、より高速な索引付けのために、連鎖自体をツリーの形に転換する。一実施形態では、後で説明するように、タイムスタンプを、そのコンテンツを識別するためのコピーのリスト中にある各ノードを用いて維持する。

【００５２】

一実施形態では、ブロックの変更に従って、より古いバージョンをキャッシュ・オブジェクトの下に連鎖する。実際、キャッシュ・オブジェクトは、そのボリュームに属するすべてのデータ・ブロックの履歴を保持している。一実施形態では、最新のコピーは、通常のプレックスＰ１、Ｐ２内などで発見するが、Ｐｔ（図９参照）の下には、より古いバージョンしか格納することができない。このように、現在データと履歴データの間には、明確な区画が存在する。このため、新しい時相オペレーションを、すべてのプレックスまたはそのサブセットに対して行うのではなく、直接Ｐｔに対して行うことも可能になる。

【００５３】

一実施形態では、時相ボリュームは、バージョニングを実現するためにタイムスタンプを使用する。時相ボリュームに対する通常の書込みは時相を実現しない。一実施形態では、ユーザまたはアプリケーションは、変更データの履歴を保持するために、時相書込みインターフェイス（以下でさらに説明する）を使用する。通常の書込みが使用された場合は、新しいデータが単に、古いデータを上書きするだけである。時相書込みに関しては、新しいデータは、古いデータに連鎖され、所与のタイムスタンプが付けられる。一実施形態では、タイムスタンプをアプリケーションから供給する。

【００５４】

一実施形態では、各タイムスタンプは、相互に比較可能な状態にすることができ、時間的順序に配列できるように構成する。これは、時相読取りオペレーションを行うために、またデータの履歴に関する情報を得るために望まれ、あるいは必要とされることもある。タイムスタンプに使用できるデータ型としては、それだけに限らないが、整数データ型を

挙げることができる。これらの諸実施形態では、タイムスタンプは、実際のシステム時間、バージョン番号、増分カウンタ、または他の任意の適当なフォーマットでよい。一実施形態では、タイムスタンプがボリューム・マネージャによって解釈可能であることを必要としなくてよい。ボリューム・マネージャがタイムスタンプを解釈する必要がないことにより、ユーザがデータについて、将来のある時間とし得る有効時間を有することが可能になる。しかし、ボリューム・マネージャがタイムスタンプを解釈する必要がないということは、時相ボリュームがタイムスタンプの示す時間の一貫性を保証できないことも意味する。したがって、一実施形態では、一貫性のあるタイムスタンプが維持されるかどうかはアプリケーションまたはユーザ次第である。一実施形態では、ユーザは、後で説明するアプリケーション・エージェントのフレームワークを使用して、タイムスタンプの維持の負担を軽減する。

【００５５】

一実施形態では、時相ボリュームは、（キャッシュ・オブジェクトによって提供される「無限の」ストレージのおかげで）「無限の」履歴を維持するが、ユーザは、履歴の一部または１ウィンドウを保持することしか望まないこともある。たとえば、現在とそれ以前のバージョンのデータにしか関心がなく、そのため、１つの変更／バージョンの時相ウィンドウしか必要としないアプリケーションがあるかもしれない。あるいは、ある期間だけに、たとえば最近のｎ分、ｎ時間、ｎ日などの期間だけに関する履歴を維持するのを望み、時相ウィンドウが、ｎ分、ｎ時間、ｎ日などとなるアプリケーションもあるかもしれない。

【００５６】

一実施形態では、時相ウィンドウを、時相ボリュームの作成中に設定することができ、それが望まれまたは必要とされる場合は、後にそれを変更することもできる。時相ウィンドウの範囲を超える時相データへのどんな変更も、格納されず、したがって失われる。したがって、たとえば時相ウィンドウが５分の場合には、５分よりも古いどんな変更も、使用不可能となる。一実施形態では、時相ウィンドウはデフォルトで、「無限」とされる。時相ウィンドウを増加させると、履歴の量が増加するが、時相ウィンドウを減少させると、時相ボリュームが履歴を切り捨てることが必要になることもある。

【００５７】

時相ウィンドウの用途は、上記で示した例から明らかである。１つの変更の時相ウィンドウに関しては、現在バージョンと最終バージョンのデータだけが維持される。これは、たとえば、２つのバージョン間の差異を使用する増分バックアップまたはレプリケーションに有益なこともある。バージョン間の差異だけが使用されるので、好ましくは、バンド幅の使用を最適化する。第２の例では、時相ウィンドウは５分である。これは、たとえば周期的な増分バックアップまたは周期的なレプリケーションに有益なこともある。ユーザは、ｎ分毎、ｎ時間毎、ｎ日毎など増分的にデータのバックアップをとることを望む場合は、ｎよりも若干広い時相ウィンドウを使用することができ、バージョン間の差異を増分バックアップに使用する。

【００５８】

ボリュームの時相ウィンドウを、１対のタイムスタンプ＜T_a，T_b＞として表す。ただし、T_aは、最も古いタイムスタンプであり、T_bは、最新のタイムスタンプである。時相ボリュームが作成された場合でも、そのボリュームに対する最初の時相書込みがあるまで、それ自体の時相ウィンドウを定義せずにおくこともできる。タイムスタンプT_xを伴う最初の書込みがあると、時相ウィンドウは、＜T_x，T_x＞に初期化される。以後の（たとえばタイムスタンプT_iを伴う）時相書込みが、時相ウィンドウを＜T_x，T_i＞に変更する。履歴がT_xからT_yに切り捨てられた場合（以下で詳細に説明する）は、時相ウィンドウは、＜T_y，T_i＞に変わる。

【００５９】

一実施形態は、時相ボリュームに対する時相入出力オペレーションを可能にする１つまたは複数のインターフェースを設けている。一実施形態では、通常のボリュームで許され

ているオペレーションを、時相ボリューム内で、それと同様にまたは同じ形で処理する。一実施形態では、時相ボリュームに対する通常の（非時相）読取りを、通常のボリューム内でそうするのと同様にまたは同じ形で進める。（時相プレックスＰｔ以外には）通常のプレックスが存在しない場合は、いかなるタイムスタンプも付いていないデータを、返却する。タイムスタンプの付いていないデータが何ら存在しない場合は、最新のタイムスタンプを伴うデータを返却する。一実施形態では、時相ボリュームに対する通常の書込みを、データレス・スペース最適化スナップショット（本明細書の後段で説明する）を作成することを除けば、通常のボリューム上でそうするのと同様にまたは同じ形で進める。一実施形態では、通常の読取りを、通常のボリュームにおいてそうするのと全く同様に時相ボリューム上で進めることができ、過去のデータのコピーを検索するための１つまたは複数の入出力要求を定義する。

【００６０】

一実施形態は、ユーザが過去における特定の時点のエクステントのコンテンツを読み取ることを可能にするインターフェースを設けている。一実施形態では、エクステントを、ボリューム上のオフセットとそのレングス(length)によって指定する。一実施形態では、タイムスタンプを使用して、そのコンテンツが必要とされる時間を指定することができ、そのタイムスタンプの解釈をユーザに委ねる。一実施形態では、タイムスタンプのタイプは、それに対して整数比較を実施できるように、ボリューム・マネージャ（たとえばＶｘＶＭ）によって指定されたタイプであることが好ましい。タイムスタンプの例としては、それだけに限らないが、エポックからの秒数として表されるトランザクション時間を表現するデータベース用のタイムスタンプ、バージョン番号を表現するバージョニング・システム用のタイムスタンプ、システム・クロックによって表される時間とするファイル・システム用のタイムスタンプ、整数カウンタである他の何らかのアプリケーション用のタイムスタンプを挙げることができる。

【００６１】

一実施形態では、タイムスタンプが０の場合、現在のデータのコピーを返却する。これは、通常の読取りと等価なものである。タイムスタンプが負である、たとえば－ｎである場合は、そのデータのｎ番目に新しいコピーが返却される。これは、たとえば実際の時間には関心がなく、変更またはバージョンにしか関心のないアプリケーションに有益なこともある。

【００６２】

以下は、諸実施形態が提供できる、時相ボリューム上の様々な時相オペレーションを論理デバイス・レベルで行うための例示的なＡＰＩである。これらのＡＰＩは例示的なものであり、限定的なものではないことに留意されたい。

【００６３】

一実施形態は、エクステントのコンテンツを非解釈バイト・バッファとして返却するのに使用するＡＰＩを提供する。ボリュームは、それ自体のデバイス番号（ｄｅｖｉｃｅ）を使用して指定する。以下は、ＵＮＩＸ環境でのＡＰＩの例示的なフォーマットである。

```
void*  vol_temporal_read(voldevno_t     device,
                         timestamp_t    timestamp,
                         voff_t         offset,
                         size_t         length);
```

【００６４】

一実施形態では、ｖｏｌ＿ｔｅｍｐｏｒａｌ＿ｒｅａｄにより、ＩＯＣＴＬ（たとえばＶＯＬ＿ＴＥＭＰＯＲＡＬ＿ＲＥＡＤ）をデバイス上で使用する。ＶＯＬ＿ＴＥＭＰＯＲＡＬ＿ＲＥＡＤは、［ｏｆｆｓｅｔ，ｌｅｎｇｔｈ］の対で指定されたデータを、ユーザが用意したバッファに読み込むための例示的なＩＯＣＴＬである。読み取るべきコピーは、タイムスタンプによって指定する。ＶＯＬ＿ＴＥＭＰＯＲＡＬ＿ＲＥＡＤは、指定されたタイムスタンプ（または最近のタイムスタンプ）にマッチする１つ（または複数）の領域を返却する。以下は、返却される構造体の例示的なフォーマットである。

```
struct vol_temporal_io {
        int             flags;          ／＊フラグ＊／
        timestamp_t     timestamp;      ／＊タイムスタンプ＊／
        caddr_t         buf;            ／＊バッファ＊／
        size_t          buf_len;        ／＊バッファ長＊／
        voff_t          offset;         ／＊オフセット＊／
}
```

【００６５】

一実施形態は、期間を読み取るのに使用されるＡＰＩを提供する。このインターフェイスは、ユーザが２つの時点におけるエクステントのすべてのバージョンを読み取ることができる。このエクステントは、ｏｆｆｓｅｔとｌｅｎｇｔｈを使用して指定する。このインターフェイスでは、２つのタイムスタンプを、すなわち、期間の開始に対応する第１のタイムスタンプと、その期間の終了に対応する第２のタイムスタンプとを提供する。一実施形態では、どちらのタイムスタンプも包括的である。一実施形態では、２つのタイムスタンプ（たとえばｐｅｒｉｏｄ＿ｓｔａｒｔとｐｅｒｉｏｄ＿ｅｎｄ）が等しい場合は、期間の開始におけるコピーが、返却される。２つのタイムスタンプがどちらも０である場合は、現在のコピーが返却される。タイムスタンプがどちらも負である、たとえば、それぞれ－ｍと－ｎ（ｍ＞ｎ）である場合は、ｍ番目に新しいコピーとｎ番目に新しいコピーの間のすべてのコピーが返却されることになる（一実施形態ではどちらも包括的）。以下は、ＡＰＩの例示的なフォーマットである。

```
void* vol_temporal_period_read (voldevno_t      device,
                                timestamp_t     period_start,
                                timestamp_t     period_end,
                                voff_t          offset,
                                size_t          length);
```

【００６６】

一実施形態では、ｖｏｌ＿ｔｅｍｐｏｒａｌ＿ｐｅｒｉｏｄ＿ｒｅａｄにより、ＩＯＣＴＬ（たとえばＶＯＬ＿ＴＥＭＰＯＲＡＬ＿ＰＥＲＩＯＤ＿ＲＥＡＤ）をデバイス上で使用する。このＩＯＣＴＬは、２つの時点（ある期間）におけるエクステント［ｏｆｆｓｅｔ，ｌｅｎ］のすべてのコピーを返却する。この期間は、開始と終了のタイムスタンプを使用して指定される。実際のデータを返却する前に、このＩＯＣＴＬを使用して、バッファのサイズを見出す。このために、空のバッファがサブミットされることもあり、ＩＯＣＴＬは、それが発見し得る領域の数（ｒｅｇ＿ｎｕｍ）だけを返却する。ユーザは、領域の数を見出すと、メモリをバッファに割り付けることができ、２回目のＩＯＣＴＬの呼出しを行う。ユーザは、構造体（たとえばｐｅｒｉｏｄ＿ｒｅｇ＿ｔ）がバッファ内での領域のコピーのレイアウトを記述するのに十分なメモリを割り付けることが好ましい。この２回目では、ＩＯＣＴＬは、２つのポインタを返却する。たとえば、第１のポインタは、その領域のコンテンツのコピーを次々とそこに詰め込む、ユーザ・バッファ自体（ｂｕｆ）である。その領域からはブロックを削除できるので、それぞれのコピーは、異なるサイズになることもある。第２のポインタ（ｒｅｇ＿ｐｔｒ）は、ユーザ・バッファ内での領域とそのコピーのレイアウトを記述する構造体を指す。この構造体は、各行がエクステントの個々のコピーを記述し、各列が１つのコピー内の領域を記述するマトリックスと考える。以下は、返却される構造体ｖｏｌ＿ｔｅｍｐｏｒａｌ＿ｒｅｇｉｏｎ＿ｐｅｒｉｏｄ＿ｉｏの例示的なフォーマットである。

```
typedef struct period_reg {
  voff_t            vtbr_offset;          ／＊領域のオフセット＊／
  timestamp_t       vtbr_timestamp;       ／＊タイムスタンプのコピー＊／
  struct period_reg *vtbr_next_region;    ／＊次の領域＊／
  struct period_reg *vtbr_prev_region;    ／＊前の領域＊／
  struct period_reg *vtbr_next_copy;      ／＊次のコピー＊／
```

```
  struct  period_reg  *vtbr_prev_copy;      ／＊前のコピー＊／
} *period_reg_t;

struct  vol_temporal_region_period_io  {
  int               flags;                ／＊フラグ＊／
  timestamp_t       start_timestamp;      ／＊期間の開始＊／
  timestamp_t       end_timestamp;        ／＊期間の終了＊／
  caddr_t           buf;                  ／＊バッファ＊／
  period_reg_t      reg_ptr;              ／＊データのレイアウト＊／
  size_t            buf_len;              ／＊バッファのレングス＊／
  caddr_t           reg_num;              ／＊レジスタの数＊／
  voff_t            offset;               ／＊エクステントのオフセット＊／
  size_t            len;                  ／＊エクステントのレングス＊／
}
```

【００６７】

一実施形態では、時相書込みは、通常の書込みと全く同様であるが、ユーザが定義したタイムスタンプを含む。時相書込みの例示的なインターフェイスは、次の通りである。

```
size_t  vol_temporal_write(voldevno_t    device,
                           timestamp_t   timestamp,
                           void          *buffer,
                           voff_t        offset,
                           size_t        length);
```

【００６８】

このボリュームは、それ自体のデバイス番号（ｄｅｖｉｃｅ）を使用して指定することができ、書き込むべきデータは、バッファに供給する。書き込むべき時間は、エクステントと共にタイムスタンプの形で供給され、（何らかのエクステントが存在すれば）そのエクステントの現在のタイムスタンプ以上であることが好ましい。エクステントは通常通り、［ｏｆｆｓｅｔ，ｌｅｎｇｔｈ］の対を使用して指定する。この関数に使用されるカーネルＩＯＣＴＬ（たとえばＶＯＬ＿ＴＥＭＰＯＲＡＬ＿ＷＲＩＴＥ）は、所与のタイムスタンプを使用して時相ボリューム上の領域への書込みを行う。

```
struct  vol_temporal_region_io  {
        int               flags;        ／＊フラグ＊／
        timestamp_t       timestamp;    ／＊タイムスタンプ＊／
        caddr_t           buf;          ／＊バッファ＊／
        size_t            buf_len;      ／＊バッファのレングス＊／
        voff_t            offset;       ／＊オフセット＊／
}
```

【００６９】

以下は、タイムスタンプの指定した時間より前に時相ボリューム（デバイス）の履歴を切り捨てるのに使用できる例示的なインターフェイスである。

```
int vol_temporal_truncate_history (voldevno_t device, timestamp_t times
tamp);
```

【００７０】

この返却値がステータス・インジケータである。この関数では、ＩＯＣＴＬ（たとえばＶＯＬ＿ＴＥＭＰＯＲＡＬ＿ＴＲＵＮＣＡＴＥ）を使用し、そのＩＯＣＴＬによって時相ボリュームの履歴を切り捨てる。ユーザは、タイムスタンプを使用して、履歴をそれ以前に削除すべき時間を指定する。

```
struct  vol_temporal_truncate{
        int               flags;        ／＊フラグ＊／
        timestamp_t       timestamp;    ／＊タイムスタンプ＊／
```

```
}
```

【００７１】

以下は、時相ウィンドウを変更するための例示的なインターフェイスである。

```
int  vol_temporal_change_window(voldevno_t     device,
                                timestamp_t    new_window,
                                twindow_t      window_type);
```

【００７２】

新しい時相ウィンドウ（ｎｅｗ＿ｗｉｎｄｏｗ）は、タイムスタンプのタイプを使用して指定する。タイムスタンプの解釈は、ウィンドウのタイプがどのようなタイプであるかに依存する。このタイプは、たとえば、時間に基づくものでも、変更の数に基づくものでも、（タイムスタンプが解釈されないので）現在のタイムスタンプと最も古いタイムスタンプとの間の差異に基づくものでもよい。この関数は、ＩＯＣＴＬ（たとえばＶＯＬ＿ＴＥＭＰＯＲＡＬ＿ＣＨＡＮＧＥ＿ＷＩＮＤＯＷ）を使用して、時相ボリュームの時相ウィンドウを変更する。新しいウィンドウが古いウィンドウよりも狭い場合は、このＩＯＣＴＬは、必要な履歴切捨て・アクティビティを実施する。以下は、このＩＯＣＴＬの例示的な構造体フォーマットである。

```
struct  vol_temporal_change_window{
        int             flags;          ／＊フラグ＊／
        timestamp_t     window;         ／＊新しいウィンドウ＊／
        twindow_t       type;           ／＊ウィンドウのタイプ＊／
}
```

【００７３】

時相ボリューム（デバイス）内の変更またはバージョニングの維持を中断するために、以下の例示的なインターフェイスを使用する。

```
int  vol_temporal_pause_history(voldevno_t  device);
int  vol_temporal_resume_history(voldevno_t  device);
```

【００７４】

これらの２つの関数は、対応するオペレーションのステータスを返却する。これらの２つの関数は、あるカーネルＩＯＣＴＬ（たとえばＶＯＬ＿ＴＥＭＰＯＲＡＬ＿ＰＡＵＳＥ、ＶＯＬ＿ＴＥＭＰＯＲＡＬ＿ＲＥＳＵＭＥ）を使用し、それらのＩＯＣＴＬによりそれぞれ、履歴の保持が中断し再開する。このオペレーションは、たとえば、アーカイビングのために全体の履歴をオフラインで取得するのに有益なこともある。中断期間中にあたるタイムスタンプを伴ってボリュームに加えられたどの変更も、ボリュームが中断されている間は失われる可能性がある。あるいは、それらの入出力は、受け付けられない。

【００７５】

時相ボリューム内のデータの履歴に関する情報を取得するために、以下の例示的なインターフェイスを使用する。

```
int  vol_temporal_info(  voldevno_t     device,
                         voff_t         offset,
                         unsigned  int  *copy_num,
                         timestamp_t    *min_timestamp,
                         timestamp_t    *max_timestamp,
                         timestamp_t    **change_timestamps,
                         voff_t         **changed_regions);
```

【００７６】

時相ボリュームは、それ自体のデバイス番号（ｄｅｖｉｃｅ）を使用して指定する。特定の領域についての履歴が望まれている場合は、その履歴を、それ自体のオフセットを使用して指定する。このオフセットが負である場合は、全体のボリュームについての情報を返却する。返却される最初の情報は、その領域のコピーの数（ｃｏｐｙ＿ｎｕｍ）である。領域上の最小タイムスタンプと最大タイムスタンプはそれぞれ、ｍｉｎ＿ｔｉｍｅｓｔ

ａｍｐとｍａｘ＿ｔｉｍｅｓｔａｍｐである。ｃｈａｎｇｅ＿ｔｉｍｅｓｔａｍｐｓは、領域上のすべてのタイムスタンプのリストである。全体のボリュームについての情報が望まれている場合は、すべての変更された領域（ｃｈａｎｇｅｄ＿ｒｅｇｉｏｎｓ）のリストを返却する。上記のインターフェイスは、ＩＯＣＴＬ（たとえばＶＯＬ＿ＴＥＭＰＯＲＡＬ＿ＩＮＦＯ）を使用して、コピーの数、タイムスタンプ、変更されたブロックのリストなど、履歴に関する情報を取得する。以下は、ＶＯＬ＿ＴＥＭＰＯＲＡＬ＿ＩＮＦＯの例示的な構造体である。

```
struct  vol_temporal_info{
    voff_t     offset;            ／＊領域のオフセット＊／
    caddr_t    copy_num;          ／＊コピーの数＊／
    caddr_t    start_timestamp;   ／＊最初の変更＊／
    caddr_t    end_timestamp;     ／＊最後の変更＊／
    caddr_t    change_timestamps; ／＊タイムスタンプのリスト＊／
    caddr_t    changed_regions;   ／＊変更された領域のオフセットのリスト＊／
  }
```

【００７７】

時相ボリュームが過去におけるボリュームのすべてのコピーを保持しているので、一実施形態は、特定の時点のボリュームの完全なイメージを取得するためのメカニズムを備えている。このイメージを、その特定の瞬間におけるボリュームのコンテンツを表す別個のボリュームである。このイメージは、データで一杯の状態（ｄａｔａ－ｆｕｌｌ）であることがあり、その場合は、すべてのブロックが新しいストレージ上にコピーされ、あるいは、データレスであることもあり、その場合は、実際のデータのコピーは存在せず、元の時相ボリュームを指す何らかの追跡構造体(tracking structure)だけが存在する。

【００７８】

図１１は、一実施形態による非時相スナップショットを示している。データのコピー・オン・ライトは、スナップショットの元のボリューム上にあるデータが上書きされたときに起こるが、その元のボリュームが時相ボリュームである場合には、これを回避する。図１１に示されるように、ＴＶＯＬは、通常の入出力が発生する時相ボリュームである。この時相ボリュームに時相アプリケーションが存在しない場合は、履歴は、プレックスＰｔの下で維持されず、その領域には、タイムスタンプが付けられない。あるスナップショットがT_0において取得されたときは、Ｐｔの下のすべての領域が、タイムスタンプT_0を用いてマークされ、したがって、これらの領域に対する以後のどんな書込みも回避される。ＴＶＯＬへの新しい書込みは、たとえばT_1まで達した場合、その領域にT_0のタイムスタンプが付いているのを知り、上書きを行わない。その代わりに、この新しい書込みには、（T_0にある）古いデータが連鎖されることになる。

【００７９】

ＴＶＯＬが通常のボリュームであれば、T_1の書込みが発生した場合は、コピー・オン・ライト・ポリシーの故に、その領域の古いコンテンツ（T_0のコンテンツ）をＳＴＶＯＬ（スナップショットの時相ボリューム、または時相スナップショット）にプッシュしているはずである。これは、ＳＴＶＯＬがT_0のＴＶＯＬのイメージを表す故に必要とされることがある。しかし、ＴＶＯＬは、時相ボリュームであるので、コピー・オン・ライトは必要とされない。ＴＶＯＬは、T_1の書込みが発生した場合でも、その領域にはT_0のスタンプが付いているので、それらのバージョニングを自動的に行うことができ、したがって、T_0の領域のコンテンツは依然として、Ｐｔの下で使用可能なままである。通常の状況下では、ＴＶＯＬの読取りは、その領域が無効である場合にしか満足されることはない。したがってＳＴＶＯＬの場合は、すべての領域は当初、無効としてマークされる。ＳＴＶＯＬからある領域を読み取る場合には、その読取りは、タイムスタンプT_0と共にＴＶＯＬへとリダイレクトされる。この読取りでは、（T_1の）現在のコンテンツではなく、T_0の領域のコンテンツを読み取る。

【００８０】

ＳＴＶＯＬに対する書込みが発生した場合は、まず、それに対応するブロックが、（タイムスタンプT_0を伴う）ＴＶＯＬからフェッチされ、次いで、ＳＴＶＯＬに書き込まれる。次いで、この領域は、それ自体のコピーがＳＴＶＯＬ内にあり、ＴＶＯＬにリダイレクトされる必要はないので、有効なものとしてマークされる。全体の領域を書き込む場合は、これをさらに最適化する。かかる場合には、フェッチングの部分を省略する。

【００８１】

図１２は、一実施形態による、スペース最適化時相スナップショットを時相ボリュームから導出する様子を示している。この元の時相ボリュームは、時相ウィンドウ$<T_a, T_b>$を有するＴＶＯＬである。ある時相スナップショットＳＴＶＯＬが、T_nにおいて作成された場合は、その時相ウィンドウは、$<T_n, T_n>$となる。ただし、$T_a <= T_n <= T_b$である。スナップショットの作成オペレーションではまず、プレックス（P_2、P_xなど）を作成し、次いで、キャッシュ・オブジェクトとスナップショット・ボリュームＳＴＶＯＬを作成する。次いで、ＳＴＶＯＬのすべての領域を「無効」にセットする。一実施形態では、ＳＴＶＯＬに対する入出力を、次のように進める。

・　時間T_xでのＳＴＶＯＬに対する時相読取りは、

その領域が、無効である場合には、ＴＶＯＬに対するT_nの時相読取りを実施し、データを返却する。T_nは、スナップショットが作成された時間である。

その領域が、有効であり、$T_x > T_n$の場合には、ＳＴＶＯＬに対するT_xの時相読取りを実施する。ただし、T_xは、ＳＴＶＯＬ上で領域が変更された時間であり、その領域における最新のコピーとなる。そうでない場合には、ＴＶＯＬの時相読取りを実施する。

・　時間T_xでのＳＴＶＯＬに対する時相書込みは、

その領域が、無効である場合には、T_nの時相読取りを使用して、その領域についての古いデータをＴＶＯＬから取得し、新しいデータを古いデータ上にオーバーレイし、時相書込みおよびタイムスタンプT_x（$T_x > T_n$）を使用して、ＳＴＶＯＬへの書込みを行う。マップにおいてその領域を有効にセットし、ＳＴＶＯＬの時相ウィンドウを、$<T_n, T_x>$に変更する。

その領域が、有効である場合には、タイムスタンプT_xを用いてＳＴＶＯＬ自体に対する時相書込みを行い、ＳＴＶＯＬの時相ウィンドウを、$<T_n, T_x>$に変更する。

【００８２】

ポイント・イン・タイム・イメージであるスナップショットとは異なり、スライスは、元の時相ボリュームをある期間だけ切り取ったものである。したがって、時相ウィンドウ$<T_a, T_b>$を伴う時相ボリュームのスライスは、T_iからT_jの履歴のサブセットを保持する。ただし、$T_i >= T_a$かつ$T_j <= T_b$である。

【００８３】

一実施形態では、時相ボリュームのスライスを作成することは、スナップショットを作成するのと同様に行う。まず、割付けが行われるが、これには、プレックスとキャッシュ・オブジェクトを作成し、最終的に、ボリュームＳＴＶＯＬ（スナップショット時相ボリューム、または時相スナップショット）を作成することを要する。次いで、マップ内のすべての領域が無効としてマークされ、スライスの時相ウィンドウが、$<T_i, T_j>$にセットされる。一実施形態では、ＳＴＶＯＬに対する入出力を、次のように進める。

・　時間T_xでのＳＴＶＯＬに対する時相読取りは、

その領域が、無効としてマークされている場合には、タイムスタンプT_xを用いてＴＶＯＬに対する時相読取りを実施する。

その領域が、変更されたものとしてマークされており、$T_x > T_i$である場合には、ＳＴＶＯＬに対するT_xの時相読取りを実施する。ただし、T_xは、ＳＴＶＯＬ上で領域が変更された時間であり、その領域における最新のコピーとなる。そうでない場合には、ＴＶＯＬの時相読取りを実施する。

・　時間T_xでのＳＴＶＯＬに対する時相書込みは、

その領域が、無効としてマークされている場合には、T_iの時相読取りを使用して、その領域についての古いデータをＴＶＯＬから取得し、新しいデータを古いデータ上にオー

バーレイし、時相書込みとタイムスタンプT_x（$T_x > T_j$）を使用して、ＳＴＶＯＬへの書込みを行う。マップにおいてその領域を有効にセットし、ＳＴＶＯＬの時相ウィンドウを、$< T_i, T_x >$に変更する。

その領域が、有効なものとしてマークされている場合には、タイムスタンプT_xを用いてＳＴＶＯＬ自体に対する時相書込みを行い、ＳＴＶＯＬの時相ウィンドウを、$< T_i, T_x >$に変更する。

【００８４】

一実施形態では、時相ボリューム上のデータの履歴がボリューム自体に格納されているので、ボリュームを、その過去のある状態に戻す。時相ボリューム（ＴＶＯＬ）が、時相ウィンドウ$< T_a, T_b >$を有し、ユーザが、その時相ボリュームの状態を時間T_kにおける状態に復元することを望んでおり、$T_a <= T_k <= T_b$である場合には、一実施形態において、このプロセスを次の方法で実施する。

・　ユーザが履歴を削除することを望む場合には、

ＴＶＯＬと有効とされているあらゆる領域を対象とするマップを参照し、T_kより大きいタイムスタンプを有するすべての領域のコピーを削除する（これは基本的に、データ・ブロックをフリー・プールに戻すことを意味する）。

・　ユーザが履歴を保持することを望む場合には、

ＴＶＯＬと有効とされているあらゆる領域を対象とするマップを参照し、（キャッシュ・オブジェクト上での）時間T_kにおける領域のコピーのオフセットを発見し、たとえばそれがB_xであれば、時間T_rにおける領域のコピーのオフセットをB_xにセットする。ただし、T_rはその復元が開始された時間である。

【００８５】

一実施形態は、時相ボリュームをそのポイント・イン・タイム・イメージの１つから復元するためのメカニズムを備えている。このプロセスは、前段で説明したプロセスと同様に進める。ＴＶＯＬ用のマップを使用する代わりに、スナップショット用のマップを使用する。しかし、この種の復元には、スナップショットが独立して変更されている可能性もあるので、データ移動が必要なこともある。一実施形態は、時相ボリュームをそのスライスの１つから復元するためのメカニズムを備えている。スライスは、変更されていることも、変更されていないこともある。この復元では、履歴の一部または全部を変更することができ、複雑なデータ移動が必要なこともある。

【００８６】

以上、入出力と関連するオペレーションを時相ボリュームに対して行うためのインターフェイスに関する例示的な諸実施形態を、説明してきた。以下では、先に説明したような一般的なインターフェイスではなく、限定的なものでもない、アプリケーションから見て有益である例示的なユーザ・レベルのインターフェイスを説明する。

【００８７】

アプリケーションは、ボリュームの領域の変更を途切れなく追跡することを望む場合、タイムスタンプを供給する必要があり、ボリュームに書込みを行うための時相書込みインターフェイスを使用する必要がある。このため、履歴の維持にしか関心のないアプリケーションについては、大きなオーバーヘッドが生まれる可能性がある。以下で説明する例示的なインターフェイスは、ユーザが時相ボリューム（ｄｅｖ　ｎａｍｅ）への書込み中にタイムスタンプを指定することを必要としない。以下は、タイムスタンプを必要としない時相ボリュームへの書込みのための例示的なＡＰＩである。

```
int  temporal_sys_write  (char          *dev_name,
                          void          *buffer,
                          unsigned  int  offset,
                          unsigned  int  length);
```

【００８８】

読取りを行うときは、ユーザは、時相ボリュームを読み取るための以下の例示的なＡＰＩに示すように、時間列（たとえば「２００３年１月２０日　午後２時１０分」）を使用

してタイムスタンプを指定する。

```
void temporal_sys_read (char          *dev_name,
                        char          *time_string,
                        unsigned int   offset,
                        unsigned int   length);
```

【００８９】

データベースに関しては、ブロックに対する入出力を行うことが望まれることもある。一実施形態は、データベース（またはその他のアプリケーション）が使用できる、時相ボリュームとのブロック・レベルのＡＰＩを提供する。一実施形態では、以下の例示的なインターフェイスを、ブロック・レベルのＡＰＩとして提供する。

```
void temporal_block_read (char          *dev_name,
                          unsigned int   block_offset);

void temporal_block_period_read (char     *dev name,
                          timestamp_t    period_start,
                          timestamp_t    period_end,
                          unsigned int   block_offset,
                          unsigned int   *copy_num);

int temporal block write (char*       dev_name,
                          void        *buffer,
                          timestamp_t  timestamp,
                          unsigned int block_offset);

int temporal_block_info (char         *dev_name,
                          unsigned int block_offset,
                          timestamp_t  *min_timestamp,
                          timestamp_t  *max_timestamp,
                          timestamp_t  **change_timestamps);
```

【００９０】

アプリケーションは、先に説明したインターフェイスを使用して、また一実施形態では、１つまたは複数のボリューム・マネージャ・ライブラリ（たとえばＶｘＶＭライブラリ）を使用して、時間特性にアクセスするが、提供されたインターフェイスまたはＡＰＩが適切でなく、インターフェイスをそれ自体のニーズに合わせることを望むこともある。たとえば、データベースは問合せ処理のためのブロック抽出により関心があることもあり、ファイル・システムはエクステントを用いてその仕事を行うこともあり、レポジトリはすべてのオブジェクトを抽出することもある。一実施形態は、アプリケーションがよりその個々のニーズに特有のエージェントを書くことを可能にし得る、エージェント・ベースのフレームワークを提供する。

【００９１】

本明細書では、エージェントは、アプリケーションの代わりに時相ボリュームと相互作用するブローカであり、アプリケーションの（一部または全部の）セマンティクスをサポートしている。エージェントは、アプリケーションが時相ボリュームと容易に相互作用できるようにすることが好ましい。さらに、エージェントは、アプリケーションがデータを生データのままではなく、必要とするフォーマットで抽出することを可能にする。これは、時相ボリュームには何らの影響もないが、アプリケーションには有利なことがある。

【００９２】

一実施形態では、エージェントは、そのアプリケーションのニーズに適した任意のインターフェイスを、アプリケーション側にエクスポートする。エージェントは、ボリューム側では、時相ボリュームのインフラストラクチャによって提供されたインターフェイス、

たとえば先に説明したインターフェイスを使用する。エージェントは、ライブラリ、ユーティリティ、あるいはスクリプトの形さえとる。エージェントは、アプリケーションへのフック（特定の呼出しのトラッピングなど）として書くことができ、あるいは、アプリケーションと密接に統合することもできる。一実施形態では、時相ボリュームは、標準ライブラリとコマンド・セットを提供することができ、それらを使用してエージェントを書くことができる。

【００９３】

エージェントは、たとえば時相オペレーションをサポートしていないが、本明細書に記載の時相ボリュームを使用した時相オペレーションをサポートすることを望む既存のシステム内で使用することができる。以下は、エージェントを使用するいくつかの例示的なシナリオである。

【００９４】

第１の例示的なシナリオでは、いくつかのデータベースは、時相オペレーションまたは時相データをサポートすることができないが、時相ボリュームを使用することを望む可能性がある。これらのデータベースを対象として、データベースを作成し、１つまたは複数のデータベース・エージェントを使用して、時間次元がこれらのデータベースに関連付けられるように、オペレーションを再ベクトル化する。このようにして、時相でない既存のデータベースは、インターフェイスの働きをするデータベース・エージェント層を１つ（または複数）の時相ボリュームに追加して、時相オペレーションをサポートする。

【００９５】

第２の例示的なシナリオでは、レポジトリは、エージェントを時相ボリュームとのインターフェイスとして使用して、それ自体のデータ・オブジェクトを時間的に格納し検索する。オブジェクトに変更がある場合には、その変更要求を、古いオブジェクトがあったのと全く同じ位置にそのオブジェクトを書き込む、オブジェクト・エージェントへと経路変更する。このレポジトリは、時相ボリューム上で作成されるので、変更のあったブロックだけが、バージョン化される。オブジェクト・エージェントは、特定のタイムスタンプ中にオブジェクトのバージョンを抽出している間には、その適切なタイムスタンプを、ボリュームの読取りに追加し、したがって、索引付けまたは抽出のパフォーマンスが低下することなく、オブジェクトの過去のバージョンが返却される。

【００９６】

一実施形態では、先に説明した時相ウィンドウのパラメータおよび／または時相ボリュームの周期型チェックポインティング機能を、周期型レプリケーションを実施する際に使用する。周期型レプリケーションでは、変更の発生に伴ってそれを送り届けるのではなく、周期的に１組のボリュームの複製を行う。これは、それだけに限らないが、（同期レプリケーションと比較して）入出力の待ち時間が短縮されること、および（非同期レプリケーションと異なり）ログが不要であることを含めて、複数の利点を有する。時相ウィンドウは、レプリケーションの期間よりも若干広く設定することができ、したがって、その期間内に発生した変更だけが格納される。全体の領域のコピーではなく、変更だけをこの領域に送ることによって、レプリケーションをさらに最適化することができる。これらの変更は、データの最新のコピーと最後のレプリケーションの時点（最後の期間の終了時点）におけるデータのコピーを使用して計算する。

【００９７】

時相アプリケーションは、極めて膨大な量のデータを生成することができ、これらのデータは、長期間にわたり時相ボリューム内で保持することを求められることもある。より古い履歴をオフラインで取得し、最新の履歴をオンラインで維持することは、たとえばユーザが、変更は失いたくないが、最近の履歴を使用したいという場合に有益である。かかる場合は、時相ウィンドウは、「無限」とされる。これにより、過去におけるデータのすべての状態を時相ボリューム内にもつことになり、そのため、大容量のストレージが必要になる。一実施形態では、アクセス頻度の低いデータおよび／または履歴を、アプリケーションがオンラインの状態にある間でも透過的に他のメディア（たとえば、テープまたは

光デバイス）に移動させるために、ＨＳＭ（階層型ストレージ管理）を使用する。

【００９８】

一実施形態では、時相ボリュームをバックアップ・ユーティリティと統合する。バックアップ・ユーティリティは、差分バックアップ技術を使用して、ファイルに加えられた変更だけを格納することができ、それにより、バックアップを作成するのに必要なバンド幅が最適化される。ネットワーク・バックアップ・ユーティリティは、バックアップを中央のバックアップ・サーバ（必ずしも単一である必要はない）に格納することができ、それにより、多数のアプリケーション・クライアントが、１つのバックアップ・インスタンスだけを共用することが可能になる。時相ボリュームを使用すると、この差分保存を論理デバイス・レベルで自動的に行うことができ、したがって、ファイルの差分を計算する必要はなく、明示的にそれを格納する必要もない。時相ボリュームは、複数のクライアントを対象とする差分のバージョニングに使用することもできる。複数のユーザが独立してファイルを変更した場合でも、それらのバージョンを同じ時相ボリューム内に格納することができ、それにより、後のファイルの各バージョンの抽出が、円滑に進められる。

【００９９】

一実施形態では、時相ウィンドウを用いると、バックアップ・ユーティリティがクライアント側のファイルへの変更を追跡することを可能にする。１つの変更の時相ウィンドウは、途切れなく変更を追跡する際に使用する。元のコンテンツは、（必ずしも）サーバからフェッチする必要はなく、その代わりに、データの以前の状態を対象とする時相読取りを使用して検索するので、差分の取得を、最適化する。一実施形態では、この差分をクライアント側で作成する。

【０１００】

用語解説

階層型ストレージ管理（ＨＳＭ）：ポリシー・ベースの自動データ移行、自動バックアップ、圧縮、セキュリティ、アーカイビングなど、企業規模のストレージ管理サービスを提供する１組のソフトウェアおよび／またはハードウェアである。すべてのＨＳＭソリューションにおいて最も一般的な機能は、データ移行、すなわちデータをそのアクセス・パターンに基づいて、テープ、ＣＤ、ＷＯＲＭドライブなど、より廉価なストレージまたはニアライン・ストレージにオフロードする能力である。ＨＳＭは、そのアクセスがあったときは、かかるデータを透過的に呼び出す。

【０１０１】

ミラー、プレックス：ボリュームのデータの論理マッピングまたはコピーである。ボリューム内には、１つまたは複数のプレックスが存在することがある。個々のミラーはそれぞれ、完全なコピーを有している必要はないが、物理メディア上の何らかのエラーが原因で、またはそのミラーのジオメトリの特性の故に、データがマッピングされていない穴が存在することもある。これを、スパース・プレックスと呼ぶこともある。しかし、完全なプレックスが、ボリューム内に存在している必要がある。

【０１０２】

元のボリューム：１つまたは複数のミラーが、スナップショットまたはデタッチのオペレーションの結果としてそこから分離される（またはされた）ボリュームである。

【０１０３】

レプリケーション：１次サイト上のデータ・ボリュームと、地理的に離れた２次サイト上のそれ自体のイメージとの間の一貫性を維持するプロセスである。３つのモードまたはレプリケーションがある。すなわち、同期型、非同期型、周期型がある。同期型レプリケーションは、２次サイトも更新されたときにだけ書込みを返却する。非同期型レプリケーションは、変更をログの形で記録し、書込みを直ちに返却し、その後（バックグラウンドで）２次サイトの更新を行う。周期型レプリケーションは、一貫性を維持するために、増分変更部分を２次サイトに転送する。

【０１０４】

復元：ある論理的な破損の後に、元のボリュームをその以前のポイント・イン・タイム

・イメージの１つに復帰させる、ボリューム・マネージャのオペレーションである。
【０１０５】
再同期化：コンテンツの古いプレックス(stale plex)をボリュームのコンテンツと同期させ、あるいはボリュームを（インスタント・スナップショットを用いて）別のボリュームと同期させる、ボリューム・マネージャのオペレーションである。
【０１０６】
スナップショット・ボリューム：元のボリュームのポイント・イン・タイム・イメージである。
【０１０７】
スナップショット：スナップショット・ボリュームを作成するボリューム・マネージャのオペレーションである。
【０１０８】
ボリューム：物理ブロック・ストレージ・デバイス（ディスク）のように振る舞うが、任意複雑形状の、ことによると冗長な内部ジオメトリを有する、ボリューム・マネージャの疑似デバイスである。ボリュームは最終的に、その論理スペースを実際のディスク・ドライブ上の物理スペースにマッピングする（ボリューム・マネージャから物理ディスクとして見えるものは、別のドライバまたはストレージ・アレイによって作り出された疑似デバイスであるかもしれないが）。
【０１０９】
ＶｘＦＳ：ＶＥＲＩＴＡＳ　Ｆｉｌｅ　Ｓｙｓｔｅｍである。これは、ロールバックとリドゥを可能にすることによって、ファイルに関するインテントまたはファイルに対して行われたオペレーションが保護される、ジャーナリング・ファイル・システムである。ＶｘＦＳは、クローニングも可能にするが、これは、ファイル・システムのポイント・イン・タイム・イメージを取得するのと等価である。
【０１１０】
チェックポインティング：データ・ボリュームの論理的なポイント・イン・タイム・イメージを取得するプロセスである。
【０１１１】
無効領域：アプリケーションによって書込みが依然として行われておらず、したがっていかなる有効なアプリケーション・データも含んでいないデータ・ボリュームの領域である。
【０１１２】
有効領域：アプリケーションにとって有効なデータを含んでいるデータ・ボリュームの領域である。
【０１１３】
結論
これらの様々な実施形態はさらに、前段の記載に従って、搬送媒体に実装された命令および／またはデータを受信し、送信し、または格納することを含む。一般的にいえば、搬送媒体としては、磁気または光メディアなどのストレージ・メディアまたはメモリ・メディア、たとえばディスクまたはＣＤ－ＲＯＭ、ＲＡＭ（たとえば、ＳＤＲＡＭ、ＤＤＲＳＤＲＡＭ、ＲＤＲＡＭ、ＳＲＡＭなど）やＲＯＭなどの揮発性または不揮発性メディア、ならびにネットワークおよび／または無線リンクなどの通信メディアを介して運搬される電気信号、電磁気信号、デジタル信号などの伝送メディアまたは信号を、挙げることができる。
【０１１４】
添付の図面に示し本明細書において説明した様々な方法は、それらの例示的な諸実施形態である。これらの方法は、ソフトウェア、ハードウェア、またはそれらの組合せで実装する。方法の順序は、変更してよく、また、様々な要素を追加し、順序を入れ替え、組合せ、省略し、修正するなどしてもよい。
【０１１５】

本開示の利益を有する当業者には自明であるように、様々な修正および変更を加える。本発明は、かかる修正および変更をすべて包含するものであり、したがって、上記の記載は、限定的な意味にではなく、例示的なものと見なされるべきである。

【図面の簡単な説明】

【０１１６】

【図１】一実施形態による、時相ボリュームにおける時相オペレーションを管理する時相ボリューム・マネージャの図である。

【図２】一実施形態による、アプリケーション・エージェントを使用して、時相ボリューム・マネージャとインターフェイスするアプリケーションの図である。

【図３】一実施形態による、入出力制御型チェックポインティングを使用して、時相ボリュームを論理デバイス・レベルで管理するための方法の流れ図である。

【図４】一実施形態による、アプリケーション制御型チェックポインティングを使用して、時相ボリュームを論理デバイス・レベルで管理するための方法の流れ図である。

【図５】一実施形態による、周期型チェックポインティングを使用して、時相ボリュームを論理デバイス・レベルで管理するための方法の流れ図である。

【図６】一実施形態による、時相ボリュームのスライス・イン・タイム・イメージを生成するための方法の流れ図である。

【図７】一実施形態による、時相ボリュームのポイント・イン・タイム・イメージを生成するための方法の流れ図である。

【図８】一実施形態による例示的なキャッシュ・オブジェクトの図である。

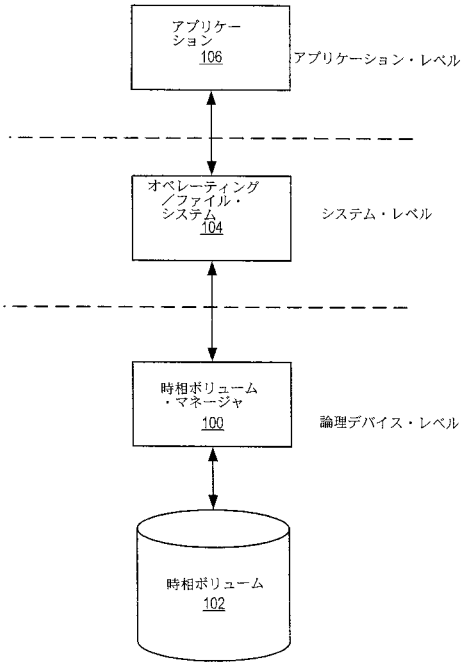
【図９】一実施形態による時相ボリュームＴＶＯＬの構成図である。

【図１０】一実施形態による、時相ボリューム上のデータ・ブロックの変更を処理する様子を示し、さらに、所与の時点でのキャッシュ・オブジェクト下の構造を示す図である。

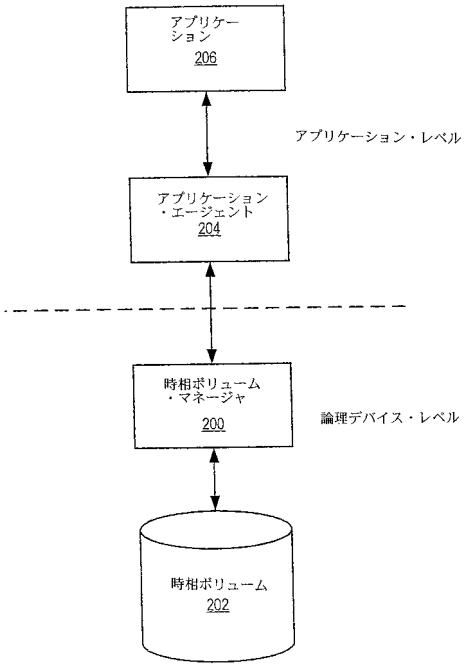
【図１１】一実施形態による、非時相スナップショットの図である。

【図１２】一実施形態による、スペース最適化時相スナップショットを時相ボリュームから検索する様子を示す図である。

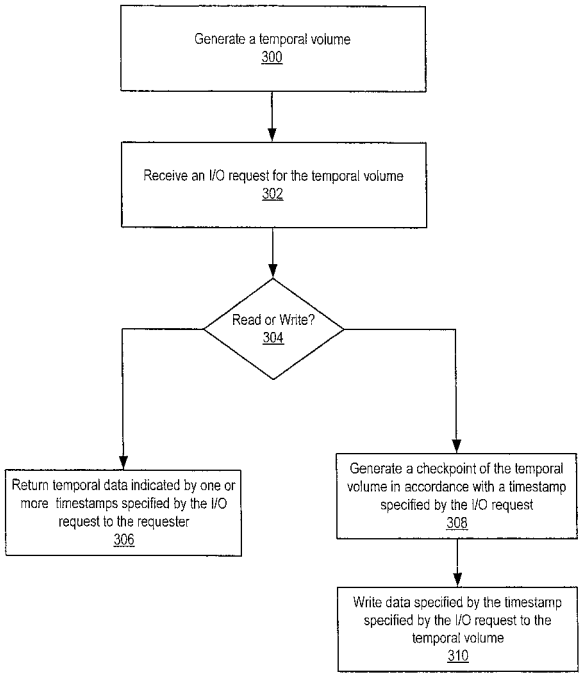
【図１】



【図２】

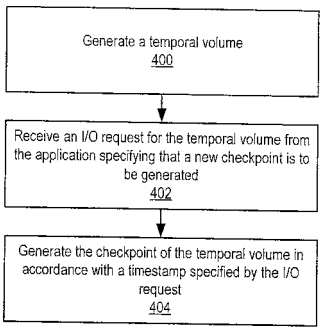


【図３】



３００　時相ボリュームを生成する
３０２　時相ボリュームを対象とする入出力要求を受信する
３０４　読取りか書込みか
３０６　入出力要求の指定する１つまたは複数のタイムスタンプで指示された時相データを要求元に返却する
３０８　入出力要求の指定するタイムスタンプに従って、時相ボリュームのチェックポイントを生成する
３１０　入出力要求の指定するタイムスタンプで指定されたデータを時相ボリュームに書き込む

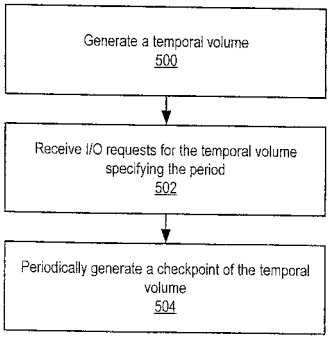
【図４】



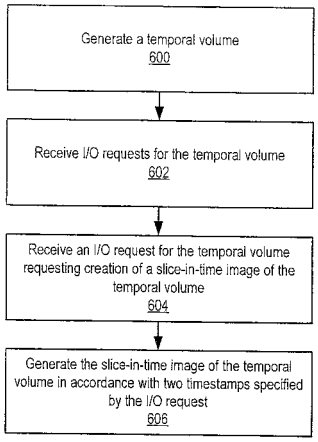
４００　時相ボリュームを生成する
４０２　時相ボリュームを対象とした、新しいチェックポイントを生成するよう指定する入出力要求をアプリケーションから受信する
４０４　入出力要求の指定するタイムスタンプに従って、時相ボリュームのチェックポイントを生成する

【図５】

５００　時相ボリュームを生成する
５０２　時相ボリュームを対象とした期間を指定する入出力要求を受信する
５０４　周期的に時相ボリュームのチェックポイントを生成する



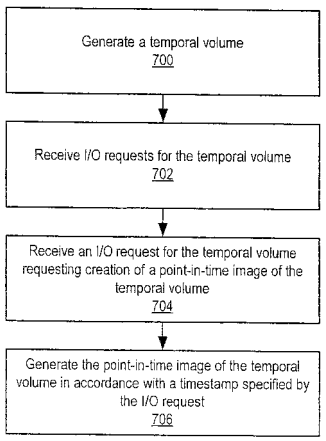
【図６】



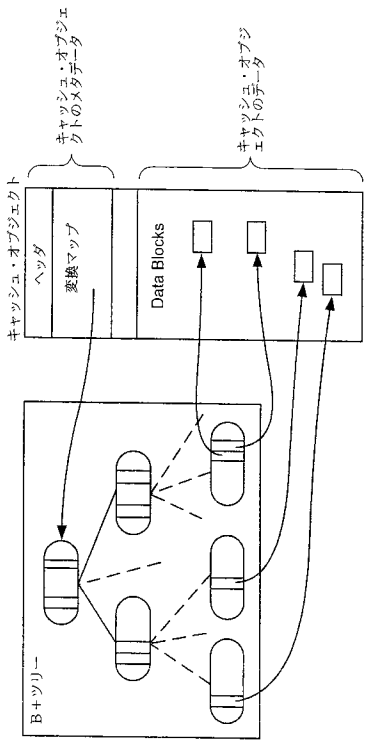
６００　時相ボリュームを生成する
６０２　時相ボリュームを対象とした入出力要求を受信する
６０４　時相ボリュームを対象とした時相ボリュームのスライス・イン・タイム・イメージを作成するよう要求する入出力要求を受信する
６０６　入出力要求の指定する２つのタイムスタンプに従って、時相ボリュームのスライス・イン・タイム・イメージを生成する

【図７】

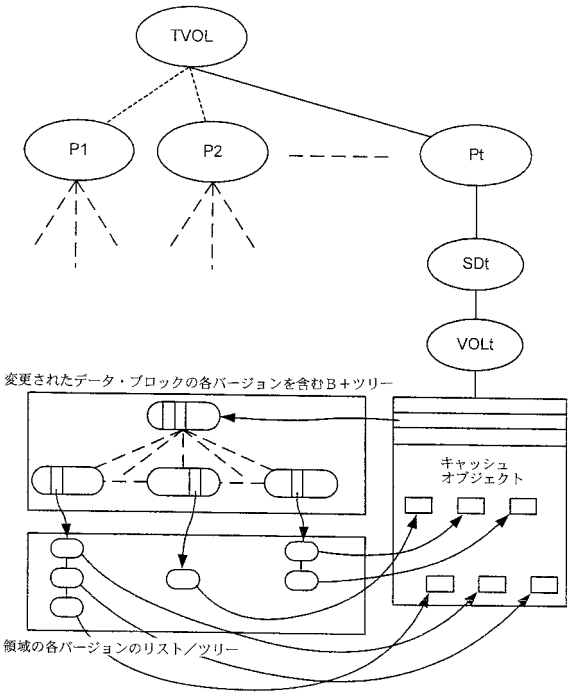
７００　時相ボリュームを生成する
７０２　時相ボリュームを対象とする入出力要求を受信する
７０４　時相ボリュームを対象とした、時相ボリュームのポイント・イン・タイム・イメージを作成するよう要求する入出力要求を受信する
７０６　入出力要求の指定するタイムスタンプに従って、時相ボリュームのポイント・イン・タイム・イメージを生成する



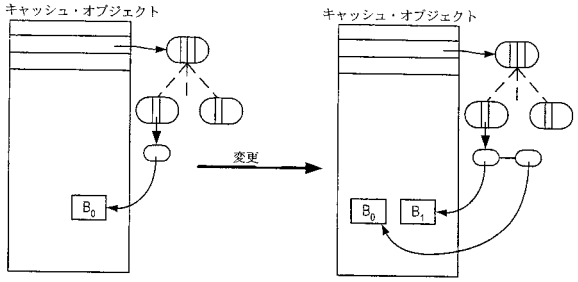
【図８】



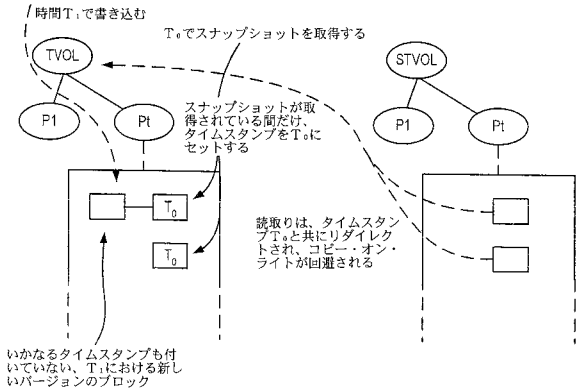
【図９】



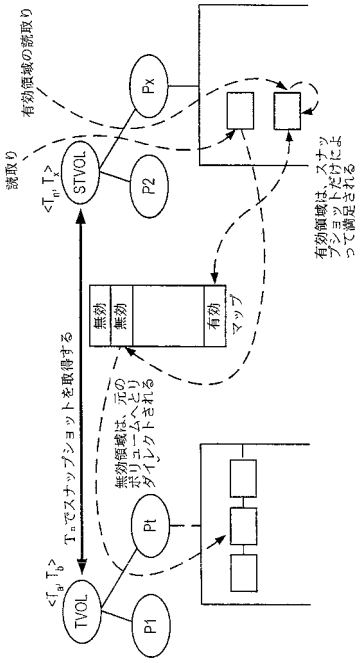
【図１０】


キャッシュ・オブジェクト
キャッシュ・オブジェクト
変更
B0
B0
B1

【図１１】


時間T1で書き込む
T0でスナップショットを取得する
TVOL
P1
Pt
STVOL
P1
Pt
スナップショットが取得されている間だけ、タイムスタンプをT0にセットする
T0
T0
読取りは、タイムスタンプT0と共にリダイレクトされ、コピー・オン・ライトが回避される
いかなるタイムスタンプも付いていない、T1における新しいバージョンのブロック

【図１２】


有効領域の読取り
Px
読取り
<Tn, Tx>
STVOL
P2
有効領域は、スナップショットだけによって満足される
無効
無効
有効
マップ
Tnでスナップショットを取得する
無効領域は、元のボリュームへとリダイレクトされる
Pt
<Tn, T0>
TVOL
P1

フロントページの続き

(72)発明者　ケクレ，アナンド・エイ
インド国・プーン　４１１０４５・バナー・エス　ナンバー　７４・コスモス　ガーデンズ・１０
(72)発明者　パンシュブッデ，アンカール
インド国・ナグプール　４４０００９・ハヌマン　ナガー・ヴィーナ・１９９

合議体
審判長　水野　恵雄
審判官　稲葉　和生
審判官　衣川　裕史

(56)参考文献　特開２００１－１５９９９３（ＪＰ，Ａ）
特開２００２－１５７１５６（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F3/06
G06F12/00
G06F12/16