(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4842279号
(P4842279)
(45)発行日　平成23年12月21日(2011.12.21)　(24)登録日　平成23年10月14日(2011.10.14)

(51)Int.Cl.　F I
*G06F 12/00 (2006.01)*　G06F 12/00 513A
G06F 12/00 535Z

請求項の数 13　(全 34 頁)

| | | | |
|---|---|---|---|
| (21)出願番号 | 特願2007-546744 (P2007-546744) | (73)特許権者 | 502303739 |
| (86)(22)出願日 | 平成17年12月6日 (2005.12.6) | | オラクル・インターナショナル・コーポレイション |
| (65)公表番号 | 特表2008-524707 (P2008-524707A) | | アメリカ合衆国、９４０６５　カリフォルニア州、レッドウッド・ショアーズ、オラクル・パークウェイ、５００ |
| (43)公表日 | 平成20年7月10日 (2008.7.10) | (74)代理人 | 100064746 |
| (86)国際出願番号 | PCT/US2005/044134 | | 弁理士　深見　久郎 |
| (87)国際公開番号 | WO2006/065587 | (74)代理人 | 100085132 |
| (87)国際公開日 | 平成18年6月22日 (2006.6.22) | | 弁理士　森田　俊雄 |
| 審査請求日 | 平成20年12月2日 (2008.12.2) | (74)代理人 | 100083703 |
| (31)優先権主張番号 | 11/014,354 | | 弁理士　仲村　義平 |
| (32)優先日 | 平成16年12月16日 (2004.12.16) | (74)代理人 | 100096781 |
| (33)優先権主張国 | 米国(US) | | 弁理士　堀井　豊 |

最終頁に続く

(54)【発明の名称】データベースサーバによるファイル操作を実行するためのインフラストラクチャ

(57)【特許請求の範囲】
【請求項１】
装置によって実現される方法であって、
リソース上のファイルシステムプロトコルによって規定されるファイルシステム操作の実行を要求するための要求をリクエスタからデータベースサーバが受取るステップを備え、前記要求は、前記要求に関連付けられる状態情報を識別する状態識別データを含み、前記方法はさらに、
前記要求に関連付けられ、リクエスタによって前記リソース上で以前に実行された状態保持操作を指定する状態情報を、前記データベースサーバにおけるルックアップ機構を使用して、前記データベースサーバが検索するステップを備え、
前記ルックアップ機構は、前記状態識別データを前記状態情報に関連付ける特定のデータを備え、
前記ルックアップ機構を使用することは、前記特定のデータと前記状態識別データとを使用して前記状態情報をルックアップすることを備え、前記方法はさらに、
少なくとも一部には前記状態情報に基づいて、ファイルシステム操作の実行を要求する前記要求を前記データベースサーバが処理するステップと、
前記要求の処理に応答して、ファイルシステム操作がリソース上で実行された後のリソースの状態を指定するようにルックアップ機構における状態情報を更新し、更新された状態情報を識別しかつリクエスタに関連付けられる第２の状態識別データを作成するステップとを備え、更新された状態情報も、リクエスタによって前記リソース上で以前に実行さ

れた状態保持操作を指定し、前記方法はさらに、
前記リクエスタに前記第２の状態識別データを伝送するステップを備える、方法。
【請求項２】
前記要求に関連付けられる前記状態情報を検索する前記ステップは、
前記状態情報を検索するために前記ルックアップ機構においてキーの値として前記状態識別データを使用することを備える、請求項１に記載の方法。
【請求項３】
前記ルックアップ機構は、ｂ－ツリーおよびハッシュテーブルからなる群から選択される１つのメンバーである、請求項２に記載の方法。
【請求項４】
前記第２の状態情報は、前記要求の処理に応答して前記リクエスタによって開かれたファイルを指定する、請求項１に記載の方法。
【請求項５】
前記第２の状態情報は、前記要求の処理に応答して前記リクエスタに付与されたファイル上の新しいロックを指定し、前記第２の状態情報は、前記リクエスタに以前に付与されたいずれのロックも指定する、請求項１に記載の方法。
【請求項６】
前記新しいロックは、前記ファイルの指定されたバイト範囲をカバーし、前記指定されたバイト範囲は前記ファイルのすべてには及ばない、請求項５に記載の方法。
【請求項７】
前記要求は第２の要求であって、前記方法はさらに、
前記第２の要求を受取る前に、リクエスタのためのクライアント識別子を確立するための第１の要求を前記データベースサーバにおいて受取るステップと、
前記リクエスタに前記クライアント識別子を伝送するステップとを備え、
前記第２の要求は前記クライアント識別子を含む、請求項１に記載の方法。
【請求項８】
前記要求は特定のリクエスタを識別し、前記方法はさらに、
前記要求に関連付けられる前記状態情報を検索する前記ステップの前に、要求を発行したリクエスタが実際に要求において識別される特定のリクエスタであるかどうかを判断するステップを備える、請求項１に記載の方法。
【請求項９】
前記要求はリクエスタによって発行され、前記方法はさらに、
前記要求によって要求された特定の操作を実行する前に、前記リクエスタが前記特定の操作を実行するのに十分な許可レベルを有しているかどうかを判断するステップを備える、請求項１に記載の方法。
【請求項１０】
前記要求はリクエスタによって発行され、前記要求を処理する前記ステップはさらに、
前記リクエスタを指定するように、前記データベースサーバに維持されるリクエスタデータを更新するステップを備え、前記リクエスタデータはファイルシステム操作を発行するために登録されるリクエスタを識別する、請求項１に記載の方法。
【請求項１１】
前記要求はリクエスタによって発行され、前記ファイルシステム操作はファイルを開くための操作であって、前記要求を処理する前記ステップはさらに、
前記リクエスタが前記ファイルを開いたことを反映するように、前記データベースサーバに維持されるファイルデータを更新するステップを備える、請求項１に記載の方法。
【請求項１２】
前記要求はリクエスタによって発行され、前記ファイルシステム操作はファイルの一部をロックするための操作であって、前記要求を処理する前記ステップはさらに、
前記リクエスタがロックした前記ファイルの前記一部を反映するように、前記データベースサーバに維持されるロックデータを更新するステップを備える、請求項１に記載の方

法。
【請求項１３】
１つ以上のプロセッサによって実行されるときに、請求項１から１２のいずれかに記載の方法を１つ以上のプロセッサに実行させる、命令の１つ以上のシーケンスを記憶する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】
発明の分野
この発明は、データベース管理システムにおけるファイル操作の実行に関する。
【背景技術】
【０００２】
背景
データは、データベースおよびファイルサーバなどの多くのタイプの格納機構に格納され得る。各々の格納機構は典型的にはその独自のアクセス手段を有する。たとえば、データベース上で操作を実行するためにＳＱＬプロトコルが典型的に使用され、ファイルシステム上で操作を実行するためにＮＦＳプロトコルが典型的に使用される。ＳＱＬプロトコルは、データベースに格納されたデータにアクセスし、そのデータを操作するためのＡＮＳＩ規格である。ＮＦＳプロトコルは、ネットワーク全体にわたってファイル上でのファイル操作の実行をサポートする分散型ファイルシステムプロトコルである。ＮＦＳは、ＵＮＩＸ（登録商標）ホスト間でファイルを共有するための周知の規格である。ＮＦＳプロトコルでは、ファイルシステム操作は、特定のファイルを識別する識別子であるファイルハンドルを使用してファイル上で実行される。ＲＦＣ３０１０に指定されるＮＦＳの現行バージョン、すなわちバージョン４は、バージョン３以上に、セキュリティの向上および状態保持操作の実行などのさらなる機能性をサポートする。
【０００３】
現在のところ、データベース管理システムは、ＮＦＳプロトコルを使用したデータベース内のデータアイテムへのアクセスをサポートしていない。したがって、ユーザがデータにアクセスしたいときには、ユーザは、データにアクセスする適切な態様について判断するためにどのタイプの格納機構がデータを格納しているかを確かめなければならない。たとえば、ＮＦＳプロトコルを使用できるかどうかを判断するために、ユーザは、データがデータベースにまたはファイルシステムにリレーショナルに格納されているかどうかを判断しなければならない。多くの状況において、どの格納機構にデータが実際に格納されているかを判断することはユーザにとって厄介なことであろう。
【０００４】
さらに、さまざまな理由のために、できる限り多くの種類のデータを単一の格納機構に格納することが望ましい。たとえば、データを格納するために使用される異なるタイプの格納機構の数を最小限にすることは、格納機構を維持するために必要なリソースの量を低減する傾向がある。さらに、多くの種類のデータをデータベースなどの中心の位置に格納することによって、使い勝手のよさおよびセキュリティが促進する。なぜなら、データが複数の機構に格納されない場合には各々が異なったセキュリティが実現される可能性があるためである。
【発明の開示】
【課題を解決するための手段】
【０００５】
その結果、データベース管理システム内でファイルシステム操作を実行するためのアプローチが望ましい。このセクションに記載されるアプローチは、追求され得るであろうアプローチであるが、必ずしも以前に考えられたまたは追求されたアプローチではない。したがって、特に指示がない限り、このセクションに記載されるアプローチはいずれもこのセクションにそれらを包含することによってのみ先行技術としての資格を得ると想定され

るべきではない。

【０００６】

この発明の実施例は限定としてではなく一例として添付の図面の図に示されており、図中、同様の参照数字は同様の要素を指す。

【発明を実施するための最良の形態】

【０００７】

詳細な説明

以下の説明では、説明の目的で、この発明の実施例の完全な理解をもたらすために多数の具体的な詳細について説明する。しかしながら、この発明の実施例はこれらの具体的な詳細がない状態で実施され得ることは明白である。他の例では、本明細書に記載されるこの発明の実施例を不必要に曖昧にすることを回避するために、周知の構造および装置はブロック図の形態で示される。

【０００８】

機能の概要

データベースに格納されたデータ上で、状態保持ファイルシステム操作などの状態保持操作を実行するための要求をデータベースサーバが処理できるフレームワークが提示される。「状態保持操作（stateful operation）」とは、（１）セッション内で要求される操作、および（２）そのセッションにおいて以前に実行された動作であったことを何らかの態様で考慮に入れる操作である。ある特定の操作の実行は、状態保持操作の実行に影響を及ぼす可能性がある。たとえば、データベース操作を実行した結果が、状態保持操作をうまく実行するために必要であるかもしれない。ＮＦＳを使用して実行された大半のファイルシステム操作は、状態保持操作である。状態保持ファイルシステム操作は、データベースサーバによって実行されると、１つ以上のデータベーストランザクションに及ぶ可能性がある。

【０００９】

実施例では、要求がデータベースシステムにおいて受取られる。この要求はたとえば、ＮＦＳプロトコルを使用して状態保持操作を実行するための要求であってもよい。この要求は状態識別データを含む可能性があり、状態識別データは、要求に関連付けられる状態情報を識別するデータである。以下でさらに詳細に記載される状態情報は、任意のセッションにおいてリクエスタによってリソース上で以前に実行されたいかなる動作も記述する情報である。たとえば、いくつかの状態保持操作が異なるセッションにおいてリクエスタによってリソース上で実行される場合、そのリソースについての状態情報は、実行された状態保持操作を反映するリソースの状態を記述するであろう。

【００１０】

状態保持操作がリソース上で実行されると、状態保持操作の実行が以前に実行された他の状態保持操作を反映するように、リソースに関連付けられる状態情報が検索される。要求に関連付けられる状態情報は、要求内に含まれる状態識別データに基づいてデータベースシステム内で検索される。要求は次いで、少なくとも一部には状態情報に基づいて処理される。

【００１１】

この発明の実施例は有利に、ファイル、リレーショナルデータおよびオブジェクト－リレーショナルデータなどのデータベース管理システムによって維持されるいかなるデータにもアクセスするためにデータベース管理システムにおいてファイルシステム操作を処理することを提供する。本明細書に記載されるフレームワークによって、有利に、ＮＦＳなどの状態保持プロトコルに一致する要求がデータベースサーバにおいて処理されることができる。この発明の実施例は主にＮＦＳプロトコルを使用して実現される要求の処理に関して説明されるが、任意の状態保持プロトコルまたは状態不保持（stateless）プロトコルを処理するためにフレームワークが使用されてもよい。この発明の実施例は、バージョン４または以後に開発された任意のバージョンを含むＮＦＳプロトコルに一致する要求を処理するために使用され得る。

【００１２】
アーキテクチャの概要
図１は、この発明の実施例に従ってファイルシステム操作を実行するための要求を処理できるシステム１００のブロック図である。システム１００は、クライアント１１０と、データベース管理システム（database management system）（ＤＢＭＳ）１２０と、通信リンク１３０とを含む。クライアント１１０のユーザは、１つ以上のファイルシステム操作の実行を指定する要求をＤＢＭＳ１２０に発行できる。説明の目的で、要求がバージョン４などのＮＦＳのバージョンに一致する例が与えられるものとする。
【００１３】
クライアント１１０は、ＤＢＭＳ１２０に要求を発行できる任意の媒体または機構によって実現されてもよい。クライアント１１０は状態保持要求をＤＢＭＳ１２０に発行できる。本明細書において使用されるように、「状態保持要求」とは、状態保持操作の実行のための要求である。典型的には、状態保持要求はＮＦＳなどの状態保持プロトコルを使用して発行される。クライアント１１０の非限定的な例示的な例は、通信リンク１３０にアクセス可能な装置で実行するアプリケーションを含む。説明を容易にするために図１では１つのクライアントしか示されていないが、システム１００は、各々が通信リンク１３０によってＤＢＭＳ１２０と通信する任意の数のクライアント１１０を含んでもよい。
【００１４】
クライアント１１０は、複数の要求を同時に発行できる媒体または機構によって実現されてもよい。たとえば、クライアント１１０は装置で実行するアプリケーションに対応してもよく、このアプリケーションは、各々がＤＢＭＳ１２０に要求を伝送できる複数のプロセスで構成されてもよい。したがって、混乱を回避するために、「リクエスタ」という用語はＤＢＭＳ１２０に要求を発行する任意の主体（entity）を指すように本明細書において使用される。このように、リクエスタは、クライアント１１０、クライアント１１０で実行するプロセス、またはクライアント１１０によって生成されるプロセスに対応し得る。
【００１５】
ＤＢＭＳ１２０は、電子データの格納および検索を容易にするソフトウェアシステムである。ＤＢＭＳ１２０はデータベースサーバ１２２と、データベース１２４とからなる。データベースサーバ１２２は、データベース１２４に維持されるファイル上で、ファイル操作を実行するための要求などの任意の状態保持要求をデータベースサーバ１２２が処理できるようにするフレームワークを使用して実現される。
【００１６】
データベースサーバ１２２は、マルチプロセスのシングルスレッド環境において実現されてもよく、マルチスレッドサーバとしてエミュレートされる。各々が作業を実行できるプロセスのプールがデータベースサーバ１２２にある。データベースサーバ１２２が要求を受取ると、データベースサーバ１２２は、受取られた要求を処理するためにプロセスのプールにおける任意のプロセスを割当てることができる。マルチプロセスのシングルスレッド環境においてデータベースサーバ１２２を実現することによって、データベースサーバ１２２は多数のクライアント１１０をサポートするように大きさを決めることが可能である。
【００１７】
通信リンク１３０は、クライアント１１０とＤＢＭＳ１２０との間のデータの交換をもたらす任意の媒体または機構によって実現されてもよい。通信リンク１３０の例は、ローカルエリアネットワーク（Local Area Network）（ＬＡＮ）、広域ネットワーク（Wide Area Network）（ＷＡＮ）、イーサネット（登録商標）もしくはインターネットなどのネットワーク、または１つ以上の地上波リンク、衛星リンクまたは無線リンクを含むが、これらに限定されない。
【００１８】
フレームワーク

図２は、この発明の実施例に従うデータベースサーバ１２２の機能コンポーネントのブロック図である。上で説明したように、データベースサーバ１２２は、データベース１２４に維持されるファイル上で状態保持要求をデータベースサーバ１２２が処理できるようにするフレームワーク２００を使用して実現される。さらに、同一のフレームワーク２００によって、データベースサーバ１２２はデータベース１２４に維持されるデータ上で、ＨＴＴＰまたはＦＴＰプロトコルで実現される要求などの状態不保持要求を処理することが可能であろう。さらに、以下で説明するように、フレームワーク２００は、新しい状態不保持プロトコルまたは状態保持プロトコルをサポートするためまたはフレームワーク２００によってサポートされる既存のプロトコルに新しい機能性を追加するために、さらなるコンポーネントを含むように構成されてもよい。

【００１９】

データベースサーバ１２２のフレームワーク２００における各コンポーネントについて以下で説明し、その後、「フレームワークを使用したファイル操作の処理」と題されるセクションにおいて、フレームワーク２００を使用した例示的な状態保持要求の処理について説明するものとする。

【００２０】

フレームワーク２００は、状態保持要求または状態不保持要求が必要とする追加の機能性をもたらす、図２に示されないさらなるコンポーネントで実現されてもよい。たとえば、拡張部２３４は、フレームワーク２００にプラグインされ得るコンポーネントを参照させ、このコンポーネントによって、フレームワーク２００は新しい状態不保持プロトコルまたは状態保持プロトコルをサポートでき、またはフレームワーク２００によってサポートされる既存のプロトコルに新しい機能性を追加できる。拡張コンポーネント２３４をフレームワーク２００にプラグインするために、適切なときに適切な情報を用いて拡張コンポーネント２３４を呼び出すようにプロトコルインタープリタ２１０が構成される。

【００２１】

プロトコルインタープリタ

プロトコルインタープリタ２１０は、通信リンク１３０によってＤＢＭＳ１２０に送られたパケットを受取る。プロトコルインタープリタ２１０は、以下に記載するように、通信リンク１３０によってクライアント１１０からパケットを受取ることができ、そのパケットを処理できる任意のソフトウェアまたはハードウェアコンポーネントを使用して実現されてもよい。プロトコルインタープリタ２１０は、パケットを受取ると、そのパケットに関連付けられるパケットタイプを識別し、そのパケットタイプのパケットを読取るように構成されるコンポーネントにパケットを送る。たとえば、パケットのヘッダを調べることによって、パケットがＮＦＳ要求を含むことをプロトコルインタープリタ２１０が判断する場合、プロトコルインタープリタ２１０はパケットをＮＦＳパケットリーダ２２４に送る。ＮＦＳ要求を含むパケットがＮＦＳパケットリーダ２２４によって読取られた後、ＮＦＳパケットリーダ２２４は、パケット内に指定される個々のファイルシステム操作についての情報をプロトコルインタープリタ２１０に送り返し、さらに処理する。

【００２２】

プロトコルインタープリタ２１０はルックアップ機構２１２を含む。ルックアップ機構２１２は、ＤＢＭＳ１２０のリクエスタについての状態情報を格納できる任意のソフトウェアまたはハードウェアコンポーネントを使用して実現されてもよい。ルックアップ機構は、揮発性記憶装置に状態情報を格納する場合もあれば、状態情報の検索を容易にするＢ－ツリーおよびハッシュテーブルなどの任意の機構を使用して実現される場合もある。ルックアップ機構２１２の例示的な実施例は、「状態情報の維持」と題されるセクションにおいて以下でさらに詳細に提示される。

【００２３】

プロトコルインタープリタ２１０は、プロトコルインタープリタ２１０によって受取られたパケットによって要求される操作を処理するように構成される。プロトコルインタープリタ２１０は、受取られたパケットによって要求される操作を実行するように構成され

る場合もあれば、以下でさらに詳細に説明するように、プロトコルインタープリタ２１０は、プロトコルインタープリタ２１０によって受取られたパケットによって要求される操作を実行するためにフレームワーク２００の１つ以上のコンポーネントと通信する場合もある。

【００２４】

エクスポータ

エクスポータ２２０は、エクスポート操作を実行できる任意のソフトウェアまたはハードウェアコンポーネントを使用して実現されてもよい。エクスポートによって、まるでディレクトリツリーがサーバにある代わりにディレクトリツリーがリクエスタにあるかのようにリクエスタがディレクトリツリーの一部を見ることが可能である。

【００２５】

実施例では、フレームワーク２００がエクスポート操作をうまく実行した後、フレームワーク２００は、（ａ）どのディレクトリフォルダがリクエスタにエクスポートされるかを識別する情報、ならびに（ｂ）エクスポートされたディレクトリフォルダへの読取アクセスおよび／または書込アクセスをリクエスタが有しているかどうかを識別する情報をエクスポート操作のリクエスタに伝送する。リクエスタがエクスポート操作を介してディレクトリフォルダへのアクセスを受取ると、リクエスタは、リクエスタがアクセスを有するディレクトリフォルダの、任意の子ディレクトリフォルダを含むすべての内容を見ることができる。

【００２６】

実施例では、エクスポータ２２０は、（ａ）どのリクエスタがエクスポートされたディレクトリフォルダであったか、および（ｂ）いずれのエクスポートされたディレクトリフォルダにも関連付けられるアクセス許可についての情報を維持できる。ディレクトリフォルダは、特定のクライアント１１０にエクスポートされる（たとえば、ディレクトリフォルダを特定のＩＰアドレスまたはドメイン名にエクスポートする）場合もあれば、１つ以上のクライアントにエクスポートされる場合もあり、たとえばディレクトリフォルダは、ディレクトリフォルダをＩＰマスクにエクスポートすることによって関連する装置の群にエクスポートされてもよい。

【００２７】

リソースロッカー

リソースロッカー２２２は、リソースをロックできる任意のソフトウェアまたはハードウェアコンポーネントを使用して実現されてもよい。実施例では、リソースロッカー２２２は、データベース１２４に格納されたファイル上でバイト範囲ロッキングを実行するように構成される。

【００２８】

リソース上で実行されるのにロックが必要であるとき、リソースロッカー２２２はロックを実行する。ファイルベースのロックを付与するための要求が実行される際、リソースロッカー２２２はルックアップ機構２１２によって維持される情報を更新できる。ファイルベースのロックについて以下でさらに詳細に説明する。

【００２９】

たとえば、一実施例の場合、プロトコルインタープリタ２１０は、ファイル上でファイルベースのロックの付与を要求するファイルシステム操作を実行するようにリソースロッカー２２２に指示してもよい。リソースロッカー２２２は、ファイルベースのロックが付与され得るかどうか、および要求されたファイルベースのロックが付与され得るかどうかを最初に判断するためにＢ－ツリーにアクセスでき、次いでリソースロッカー２２２は、ファイルベースのロックがファイル上に付与されたことを反映するために１つ以上のＢ－ツリーを更新できる。リソースロッカー２２２がアクセスし得るまたは更新し得る特定のＢ－ツリーについて以下でさらに詳細に説明する。

【００３０】

パケットリーダ

フレームワーク２００はいくつかのパケットリーダを含む。各パケットリーダは、特定のプロトコルに一致するパケットから情報を読取るように設計される。たとえば、フレームワーク２００は、ＮＦＳパケットリーダ２２４と、ＦＴＰパケットリーダ２２６と、ＨＴＴＰパケットリーダ２２８とを含む。

【００３１】

ＮＦＳパケットリーダ２２４は、ＮＦＳプロトコルに一致するパケットを読取ることができ、そのパケットを解釈できる任意のソフトウェアまたはハードウェアコンポーネントを使用して実現されてもよい。このようなパケットは、１つの操作または多くの操作を要求する場合がある。２つ以上のファイルシステム操作を要求するパケットは、「複合要求」と称される。ＮＦＳパケットリーダ２２４は、パケットに指定された第１の操作を読取り、その操作を識別するデータをプロトコルインタープリタ２１０に戻すように構成される。プロトコルインタープリタ２１０はその後、一旦以前の操作が処理されるとＮＦＳパケットリーダ２２４に別の操作をパケットから読取らせることができる。

【００３２】

ＦＴＰパケットリーダ２２６は、ＦＴＰ要求を含むパケットを読取ることができ、そのパケットを解釈できる任意のソフトウェアまたはハードウェアコンポーネントを使用して実現されてもよい。ＦＴＰパケットリーダ２２６は、ＦＴＰパケット内に含まれるＦＴＰ操作情報を読取り、そのＦＴＰ操作情報を解釈するように構成され、その後、処理のためにＦＴＰ操作情報をプロトコルインタープリタ２１０に通信するように構成される。

【００３３】

ＨＴＴＰパケットリーダ２２８は、ＨＴＴＰ要求を含むパケットを読取ることができ、そのパケットを解釈できる任意のソフトウェアまたはハードウェアコンポーネントを使用して実現されてもよい。ＨＴＴＰパケットリーダ２２６は、ＨＴＴＰパケット内に含まれるＨＴＴＰ操作情報を読取り、そのＨＴＴＰ操作情報を解釈するように構成され、その後、処理のためにＨＴＴＰ操作情報をプロトコルインタープリタ２１０に通信するように構成される。

【００３４】

図２は３つの異なるタイプのパケットタイプ、すなわちＮＦＳ、ＦＴＰおよびＨＴＴＰパケットのためのパケットリーダを示しているが、他の実施例はさらなるタイプのパケットのためのさらなるパケットリーダを含んでもよい。この態様では、フレームワークは任意の状態不保持プロトコルまたは状態保持プロトコルを読取ることができるコンポーネントを含んでもよい。

【００３５】

特権ベリファイヤ

特権ベリファイヤ２３０は、特定のリクエスタが特定のファイルシステム操作を実行するのに十分な許可レベルを有しているかどうかを検証できる任意のソフトウェアまたはハードウェアコンポーネントを使用して実現されてもよい。プロトコルインタープリタ２１０は、プロトコルインタープリタ２１０がファイルシステム操作を実行するたびに特定のリクエスタが特定のファイルシステム操作を実行するのに十分な許可レベルを有しているかどうかを判断するように特権ベリファイヤ２３０に指示できる。特定のユーザが特定のファイルシステム操作を実行するのに十分な許可レベルを有しているかどうかの判断について、図３のステップ３１８を参照して以下でさらに詳細に説明する。

【００３６】

オーソライザ

オーソライザ２３２は、プロトコルインタープリタ２１０によって受取られた特定の要求を発行したリクエスタが実際に特定の要求において識別される同一のリクエスタであるかどうかを判断できる任意のソフトウェアまたはハードウェアコンポーネントを使用して実現されてもよい。このように、リクエスタの識別は、要求に指定された任意の操作を実行する前にオーソライザ２３２によって検証され得る。プロトコルインタープリタ２１０は、プロトコルインタープリタ２１０が要求を受取るたびに、プロトコルインタープリタ

２１０によって受取られた特定の要求を発行したリクエスタが実際に特定の要求において識別される同一のリクエスタであるかどうかを判断するようにオーソライザ２３２に指示できる。特定の要求が特定のクライアント１１０によって発行されたかどうかの判断について、ステップ３１６を参照して以下でさらに詳細に説明する。

【００３７】

状態情報の維持

ＮＦＳプロトコルでは、ファイルシステム操作は、「開かれた」がまだ「閉じられていない」ファイル上で実行される。リクエスタは、リクエスタがファイル上で他のファイルシステム操作を実行し得る前に、ＯＰＥＮファイルシステム操作の実行を要求して、ファイルを開く。リクエスタがファイル上ですべての所望のファイルシステム操作を実行した後、リクエスタはＣＬＯＳＥファイルシステム操作の実行を要求して、ファイルを閉じる。

【００３８】

データベースサーバによって実行されるファイルシステム操作は、１つ以上のデータベーストランザクションに及ぶ可能性がある。その結果、各々がファイルの状態を変更する１つ以上のデータベーストランザクションが、ファイルを開くときとファイルを閉じるときとの間にファイル上で実行され得る。

【００３９】

ＮＦＳが状態保持プロトコルであるので、状態保持要求を処理するときにフレームワーク２００が状態情報を維持する必要がある。状態情報は、任意のセッションにおいてリクエスタによってリソース上で以前に実行されたいずれの動作も記述する情報である。一実施例によれば、状態情報は、リクエスタが開いた各ファイルごとに維持される。たとえば、リクエスタがファイルＡおよびファイルＢを開く場合、リクエスタはファイルＡについての状態情報の第１の組とファイルＢについての状態情報の第２の組に関連付けられるであろう。

【００４０】

状態情報は、リクエスタが（ａ）ファイルを開くもしくは閉じるとき、または（ｂ）開いたファイル上で新しいロックを得るときにいつでも割当てられるまたは更新される。したがって、リクエスタが（ａ）ファイルを開くもしくは閉じる、または（ｂ）開いたファイル上で新しいロックを得るときはいつも、ファイル上で実行される状態保持操作を反映するように状態情報が更新される。

【００４１】

リクエスタに関連付けられる状態情報は、ファイルが開かれて以来リクエスタによってファイル上で実行されたすべての状態保持操作を反映する。たとえば、リクエスタが最初にファイルを開くときに、状態情報Ａが割当てられてもよい。その後、同一のリクエスタがファイル上でロックを得る場合、状態情報Ａは無効になり、新しい状態情報Ｂが割当てられる。状態情報Ｂが両方のロックを反映していることおよびファイルがリクエスタによって開かれるという事実に注目されたい。その後、同一のリクエスタがファイル上で第２のロックを得る場合、状態情報Ｂは無効になり、新しい状態情報Ｃが割当てられる。状態情報Ｃが両方のロックを反映していることおよびファイルがリクエスタによって開かれるという事実に注目されたい。リクエスタがファイルを閉じるとき、そのリクエスタについて、すなわちそのファイルについての状態情報はもはや維持される必要がない。

【００４２】

リクエスター－ファイル間の関係の状態の追跡

状態識別データは、通信の際に参照されるファイルの現在の状態を参照するために、クライアント１１０とデータベースサーバ１２２との間で交換される通信に付随し得る。リクエスタがファイルを開くとき、状態識別データがフレームワーク２００によって作成される。状態識別データは、リクエスタが開いた特定のファイルに関して特定のリクエスタに関連付けられる状態情報を識別する。

【００４３】

開いたファイルの状態を追跡するために、新しく作成された状態識別データがリクエスタに戻される。たとえば、リクエスタＸＹＺがファイルＡＢＣを開くための要求を発行すると想定されたい。フレームワーク２００は新しく開かれるファイルＡＢＣに関連付けられる状態情報を記述する状態識別データを生成し、この状態識別データをリクエスタＸＹＺに戻す。

【００４４】

開いたファイル上でファイルシステム操作を実行するための要求をリクエスタがデータベースサーバ１２２に伝送するとき、この要求はリクエスタに以前に伝送された任意の状態識別データを含む。たとえば、状態識別データは、ファイルが開かれることに応答してリクエスタに以前に伝送されたかもしれない。この態様で、要求はファイルに関連付けられる状態情報を識別する。たとえば、リクエスタＸＹＺがファイルＡＢＣ上でロックのための要求を伝送する場合、この要求は、データベースサーバ１２２がファイルＡＢＣ上でＯＰＥＮファイルシステム操作を実行することに応答してリクエスタＸＹＺに以前に送られた状態識別情報を含むことになる。フレームワーク２００は、ルックアップ機構２１２を使用して対応する状態情報を検索するために、要求に含まれる状態識別を使用してもよい。

【００４５】

したがって、上述のように、フレームワーク２００はある特定の状態保持ファイルシステム操作の実行に応答して状態識別データを生成し、生成された状態識別データはファイルシステム操作のリクエスタに伝送される。その後、リクエスタは、要求の中に状態識別データを含むことによって同一ファイル上で追加の状態保持ファイルシステム操作を実行でき、これによって、フレームワーク２００は状態識別データを使用してファイルについての状態情報を検索することが可能である。

【００４６】

ファイルシステム操作が開いたファイル上で実行されると、ファイルに関連付けられる状態情報はファイルの新しい操作状態を反映するように更新される。更新された状態情報を参照するために新しい状態識別データが作成される。その後、フレームワーク２００はこの新しい状態識別データをリクエスタに伝送する。このように、状態識別データの１つの組だけがリクエスタとフレームワーク２００との間で交換される。フレームワーク２００から伝送された状態識別データは、フレームワークが状態保持ファイルシステム操作をうまく実行した後、状態保持ファイルシステム操作の対象であったリソースに関連付けられる最も最近の状態情報を識別する。

【００４７】

次のセクションにおいて説明するように、フレームワーク２００は、ルックアップ機構２１２に状態情報を格納でき、状態識別データを使用して、ルックアップ機構２１２に格納された状態情報を検索できる。

【００４８】

状態情報の維持

一実施例によれば、状態情報はルックアップ機構２１２を使用して維持される。一実施例では、ルックアップ機構２１２は複数のＢ－ツリーを使用して実現される。複数のＢ－ツリーは、状態保持ファイルシステム操作の要求を処理する際に使用される状態情報を格納する。たとえば、複数のＢ－ツリーはリクエスタデータ、ファイルデータおよびロックデータを格納してもよい。リクエスタデータとは、ファイルシステム操作を発行するために登録されるリクエスタを識別するデータである。ファイルデータとは、どのファイルがどのリクエスタによって開かれたかを識別するデータである。ロックデータとは、どのファイル上のどのロックがどのリクエスタに付与されたかを識別するデータである。

【００４９】

一実施例では、複数のＢ－ツリーは、「client Ｂ－ツリー」と、「client_exists Ｂ－ツリー」と、「requestor Ｂ－ツリー」と、「open_files Ｂ－ツリー」と、「opens Ｂ－ツリー」と、「locks_requestor Ｂ－ツリー」と、「granted_locks Ｂ－ツリー」と

を含む。これらのＢ－ツリーの各々は、状態情報を格納でき、以下でさらに詳細に説明するものとする。

【００５０】

この発明の他の実施例は、Ｂ－ツリーの異なる組を使用してルックアップ機構２１２を実現してもよい。たとえば、上述のいくつかのＢ－ツリー、たとえばclient_exists Ｂ－ツリーは、他のＢ－ツリーにも格納される情報を格納するため、上述のすべてのＢ－ツリーがルックアップ機構２１２のある特定の実現例に必要ではないかもしれない。しかしながら、第１の状況では第１のキーを使用してより効率的に情報にアクセスでき、第２の状況では第２のキーを使用してより効率的に情報にアクセスできるので、２つ以上のＢ－ツリーに同一のまたは同様の情報を格納することが有利であろう。

【００５１】

この発明の他の実施例では、ルックアップ機構２１２は複数のＢ－ツリーの代わりに複数のハッシュテーブルを使用して実現されてもよい。ルックアップ機構２１２を実現する複数のハッシュテーブルは、以下に記載されるものと同様の情報を格納する。ルックアップ機構２１２を実現するために他の機構もこの発明の他の実施例によって利用されてもよい。

【００５２】

client Ｂ－ツリー

client Ｂ－ツリーとは、クライアントについての情報を維持するＢ－ツリーである。フレームワーク２００に登録した各クライアント１１０は、client Ｂ－ツリー内での検索項目に反映されることになる。以下でさらに詳細に説明するように、クライアント１１０はクライアント識別子を確立するための要求を発行することによってフレームワーク２００に登録する。client Ｂ－ツリーのキーは、データベースサーバによってクライアントに以前に割当てられたクライアント識別子である。クライアント識別子は、フレームワーク２００に登録された特定のクライアント１１０を一意に識別する。client Ｂ－ツリーの各ノードは、クライアント識別子およびクライアントのネットワークアドレスなどのクライアントによって与えられる識別子を含む特定のクライアントについての情報を格納する。

【００５３】

client_exists Ｂ－ツリー

client Ｂ－ツリーと同様に、client_exists Ｂ－ツリーはクライアントについての情報を維持する。client Ｂ－ツリーおよびclient_exists Ｂ－ツリーは両方、クライアントについての情報を維持するが、client－exists Ｂ－ツリーのキーはクライアントによって与えられる識別子である。クライアントによって与えられる識別子はたとえばクライアントのネットワークアドレスであってもよい。

【００５４】

client_exists Ｂ－ツリーは、クライアントによって与えられる識別子に基づいて、特定のクライアント１１０がフレームワーク２００に登録したかどうかを判断するために使用され得る。client_exists Ｂ－ツリーの各検索項目も、クライアント識別子およびクライアントによって与えられる識別子を含む特定のクライアントについての情報を格納する。

【００５５】

requestor Ｂ－ツリー

requestor Ｂ－ツリーとは、リクエスタについての情報を維持するＢ－ツリーである。requestor Ｂ－ツリーのキーは、リクエスタに関連付けられるクライアント識別子、およびリクエスタを一意に識別するリクエスタ識別子の両方を反映する。requestor Ｂ－ツリーは、特定のクライアント１１０に関連付けられるすべてのリクエスタを判断するために使用されることができ、ＯＰＥＮファイルシステム操作の処理中または操作不能になったクライアントを回復させるときに必要とされるかもしれない。

【００５６】

requestor Ｂ－ツリーの各検索項目はリクエスタについての情報を格納する。たとえば、特定のリクエスタに対応するrequestor Ｂ－ツリーの検索項目は、どのクライアントがリクエスタに関連付けられるか、いつリクエスタからの最後の通信が受取られたか、どのファイルをリクエスタが開いたか、およびどのような状態情報がリクエスタに関連付けられるかについての情報を格納してもよい。

【００５７】

open_files Ｂ－ツリー

open_files Ｂ－ツリーとは、開かれたファイルについての情報を維持するＢ－ツリーである。open_files Ｂ－ツリーのキーはファイルのファイルハンドルである。open_files Ｂ－ツリーは、特定のファイル上でファイルシステム操作を実行することが可能であるかどうかを判断するために使用され得る。open_files Ｂ－ツリーの各検索項目は開いたファイルについての情報を格納できる。このような情報はたとえば、開いたファイル上のファイルベースのロックの数、開いたファイル上のファイルベースのロックのタイプ、どのような状態識別データが、開いたファイルに関連付けられる状態情報を識別するか、開いたファイルについてのオブジェクト識別子を含んでもよい。

【００５８】

opens Ｂ－ツリー

opens Ｂ－ツリーとは、開かれたファイルについての情報を維持するＢ－ツリーである。opens Ｂ－ツリーのキーは状態識別データである。opens Ｂ－ツリーを横断することによって、opens Ｂ－ツリーへのキーとして使用される状態識別データによって識別された状態情報に関連付けられる、開いたファイルについての情報を位置付けることができる。

【００５９】

たとえば、クライアントが特定のファイルを開いたと想定されたい。クライアントについて維持される状態情報は、クライアントが特定のファイルを開いたことを示すことになる。状態情報は状態識別データの組に割当てられることになる。状態識別データは、opens Ｂ－ツリーを横断して、特定のファイルが開いていることを示す検索項目を見つけるために使用され得る。

【００６０】

opens Ｂ－ツリーの各検索項目は、開いたファイルに関連付けられる状態情報を識別する状態識別データ、開いているファイルを開いたリクエスタ、ファイルが読取または書込のために開かれたのかどうか、開いたファイルが修正されたかどうか、および開いているファイルを開いたリクエスタ以外の他のリクエスタに対して読取または書込が拒否されたかどうかなどの開いたファイルについての情報を格納する。

【００６１】

ファイルを開くために、状態識別データが生成されて、開いたファイルを識別する。状態識別データは、（ａ）ファイルを開くように要求したリクエスタに伝送され、（ｂ）ファイルが開かれたことを反映するように項目をopens Ｂ－ツリーに追加するために使用される。

【００６２】

locks_requestor Ｂ－ツリー

locks_requestor Ｂ－ツリーとは、ロックリクエスタについての情報を維持するＢ－ツリーである。locks_requestors Ｂ－ツリーへのキーは状態識別データである。locks Ｂ－ツリーの各検索項目は、クライアント識別子、リクエスタ識別子およびロックオーナー識別子などのロックのリクエスタについての情報を含む。ロックオーナー識別子は、ロックを付与される特定のリクエスタを一意に識別する。クライアント識別子およびリクエスタ識別子はフレームワーク２００によって割当てられ、ロックオーナー識別子はリクエスタによって供給される。

【００６３】

granted_locks Ｂ－ツリー

granted_locks Ｂ－ツリーとは、付与されたロックについての情報を維持するＢ－ツリ

ーである。granted_locks Ｂ－ツリーへのキーはファイルハンドルである。granted_locks Ｂ－ツリーは、どのファイルベースのロックが、もしあれば、特定のファイル上に付与されたかをすばやく判断するために使用され得る。

【００６４】

プロトコルインタープリタ２１０が特定のロックの付与を要求するファイルシステム操作を実行するようにリソースロッカー２２２に指示すると、リソースロッカー２２２はルックアップ機構２１２の１つ以上のＢ－ツリーにアクセスし得る。例示するために、プロトコルインタープリタ２１０がファイル上に特定のロックを付与するための要求を受取り、その後、プロトコルインタープリタ２１０がファイルシステム操作を処理するようにリソースロッカー２２２に指示すると想定されたい。リソースロッカー２２２は最初に、granted locks Ｂ－ツリーにアクセスすることによって矛盾するロックが既にファイル上に付与されたかどうかを判断できる。リソースロッカー２２２は、ファイルシステム操作によって識別されるファイルのファイルハンドルを使用して、granted locks Ｂ－ツリーを横断できる。granted locks Ｂ－ツリーにおける項目がファイルハンドルのために存在する場合、その項目を考査することによって、リソースロッカー２２２は矛盾するロックが既にファイル上に付与されたかどうかを知らされることになる。

【００６５】

矛盾するロックがまだファイル上に付与されていないことをリソースロッカー２２２が判断する場合、リソースロッカー２２２は、（ａ）リソースの新しい状態を識別するために新しい状態識別データを生成でき、（ｂ）要求されたロックの付与を反映するように項目をgranted_locks Ｂ－ツリーに追加できる。リソースロッカー２２２は、リソースについての新しく生成された新しい状態識別データを使用して新しい項目をgranted_locks Ｂ－ツリーに追加でき、その後、以前の状態識別データが参照したlocks Ｂ－ツリーにおける以前の項目を削除できる。locks Ｂ－ツリーにおける新しい項目はリソース上で実行されたすべての以前の状態保持操作への参照を含むので、以前の状態識別データが参照した項目を格納する必要がない。

【００６６】

フレームワークを使用したファイル操作の処理

図３は、この発明の実施例に従うファイルシステム操作を処理するためのステップを示すフローチャートである。図３のステップを実行することによって、状態保持ＮＦＳ操作などの状態保持操作がＤＢＭＳ１２０によって実行されることができる。

【００６７】

概して、フレームワークはフレームワークが実行する操作についての状態情報を維持する。状態保持操作を実行すると、フレームワークは操作の状態に対応する状態識別データを渡してリクエスタに戻す。状態保持操作のための後続の要求の際に、リクエスタは状態識別データをフレームワークに送り返す。フレームワークは次いで、状態識別データをキーとして使用して、その後続の要求の際に操作に該当する状態情報を識別する。

【００６８】

フレームワークによって生成されるクライアント識別子の獲得

図３を参照して、最初にステップ３１０において、リクエスタのためのクライアント識別子を確立するための第１の要求がデータベースサーバにおいて受取られる。ステップ３１０は、通信リンク１３０によってクライアント１１０によって送られた、第１の要求を含むパケットを受取るプロトコルインタープリタ２１０によって実行され得る。

【００６９】

プロトコルインタープリタ２１０は、さまざまなパケットタイプのパケットを受取ることができる。プロトコルインタープリタ２１０は受取られたパケットのパケットタイプを識別するように構成されるが、プロトコルインタープリタ２１０は各々のパケットタイプを読取るように構成される必要はない。プロトコルインタープリタ２１０は、たとえばパケットのヘッダ内に含まれる情報を調べることによって、受取られたパケットのパケットタイプを判断できる。一旦プロトコルインタープリタ２１０が受取られたパケットのパケ

ットタイプを判断すると、プロトコルインタープリタ２１０はそのパケットタイプのパケットの読取を担当するコンポーネントにパケットを送る。

【００７０】

説明の目的で、ステップ３１０において受取られたパケットはリクエスタのためのクライアント識別子を確立するための要求を含むＮＦＳパケットであると想定される。クライアント識別子の確立はＮＦＳ操作である。これらの状況下で、プロトコルインタープリタはＮＦＳパケットリーダ２２４にパケットを送り、パケットを読取ることになる。ＮＦＳパケットリーダ２２４はパケットを読取り、そのパケットを解釈し、要求されるファイルシステム操作を識別する（すなわち、クライアント識別子を確立する）データをプロトコルインタープリタ２１０に送り返す。

【００７１】

ファイルシステム操作を識別するデータを受取った後、プロトコルインタープリタ２１０はファイルシステム操作を処理する。この例では、プロトコルインタープリタ２１０はクライアント識別子を確立するための要求を処理する。要求の処理の一部として、プロトコルインタープリタ２１０はたとえば、（ａ）クライアント識別子がリクエスタのためにまだ確立されているかどうかを判断するため、および（ｂ）クライアント識別子がリクエスタのためにまだ確立されていない場合には、どのようなクライアント識別子がリクエスタに関連付けられるべきであるかを判断するためにルックアップ機構２１２を調べてもよい。

【００７２】

実施例では、データベースサーバは、クライアント識別子が特定のリクエスタのために確立されたどうかを判断するために、（クライアントのネットワークアドレスなどの）クライアントによって与えられる識別子に基づいてclient_exists Ｂ－ツリーを横断できる。クライアント識別子がリクエスタのために確立されていない場合、データベースサーバはクライアントのためのクライアント識別子を生成できる。クライアントのためのクライアント識別子を生成した後、データベースサーバは、リクエスタに割当てられた新しいクライアント識別子についての情報を格納するために検索項目をclient Ｂ－ツリーおよびclient_exists Ｂ－ツリーに追加できる。

【００７３】

ステップ３１０を実行した後、処理はステップ３１２に進む。ステップ３１２では、ステップ３１０において上で確立されたクライアント識別子がリクエスタに伝送される。ステップ３１２は、クライアント識別子を含む通信を通信リンク１３０によってリクエスタに伝送するプロトコルインタープリタ２１０によって実行され得る。実施例では、リクエスタは、クライアント識別子を検証するためにデータベースサーバ１２２とさらなる通信を交換することによって、受取られたクライアント識別子をデータベースサーバ１２２を用いて検証し得る。ステップ３１２を実行した後、処理はステップ３１４に進む。

【００７４】

複合要求の受取り

ステップ３１４では、ファイルシステム操作を実行するための第２の要求が受取られる。ステップ３１４は、通信リンク１３０によってクライアント１１０によって送られた、第２の要求を含むパケットを受取るプロトコルインタープリタ２１０によって実行され得る。第２の要求はクライアント識別子を含む。

【００７５】

複合要求の処理を示すために、ステップ３１４において受取られた第２の要求が２つ以上のファイルシステム操作を含む複合要求であると想定されたい。複合要求に指定されたファイルシステム操作は、フレームワーク２００によってシーケンシャルに処理される。

【００７６】

状態保持ファイルシステム操作要求の処理を示すために、第２の要求に指定された第１のファイルシステム操作が、リクエスタによって以前に開かれたファイル上のファイルベースのロックのための要求であるとさらに想定されたい。フレームワーク２００がファイ

ルを開いた後、フレームワーク２００は、（ａ）開かれたファイルに関連付けられる状態情報を識別する状態識別データを生成し、（ｂ）その状態識別データをリクエスタに伝送する。したがって、ステップ３１４において受取られた要求が開いたファイル上でファイルシステム操作を実行するための要求である場合、ステップ３１４において受取られた要求は、リクエスタに以前に送られた状態識別データを含む。この例では、状態識別データによって、フレームワーク２００はファイルベースのロックのための要求の対象である、ファイルに関連付けられる状態情報を参照できることになる。

【００７７】

ステップ３１４の要求がファイルシステム操作要求を含むことをプロトコルインタープリタ２１０が判断した後、プロトコルインタープリタ２１０は、パケットを読取るために、ステップ３１４の要求を含むパケットをＮＦＳパケットリーダ２２４に送ることができる。その後、ＮＦＳパケットリーダ２２４は、パケットに指定された（以下で「現在の」ファイルシステム操作と称される）第１の未処理のファイルシステム操作についての情報をプロトコルインタープリタ２１０に伝送する。以下でさらに詳細に記載するように、現在のファイルシステム操作が処理された後、フレームワーク２００はパケットに指定されたさらなる未処理のファイルシステム操作を処理することになる。

【００７８】

要求の、セッションへの割当て

一旦プロトコルインタープリタ２１０が複合要求に指定された現在のファイルシステム操作についての情報をＮＦＳパケットリーダ２２４から受取ると、プロトコルインタープリタ２１０は現在のファイルシステム操作をデータベースセッションに割当てる。データベースセッションのプールから選択され得る割当てられたデータベースセッションは、複合要求内に含まれるファイルシステム操作をデータベースサーバが処理することになるセッションである。状態情報がセッションとは別個に維持される（上で説明したように、状態情報はルックアップ機構２１２に維持される）ので、現在のファイルシステム操作をサービスするために任意のセッションがデータベースセッションのプールから選択されてもよい。ステップ３１４を実行した後、処理はステップ３１６に進む。

【００７９】

クライアントの認証

ステップ３１６では、ステップ３１４において受取られた要求が、要求内に含まれるクライアント識別子によって識別されたクライアントによって発行されたかどうかについて判断がなされる。実施例では、要求が受取られるたびに、リクエスタの識別を確認するために要求を認証する。ステップ３１６は、オーソライザ２３２に要求を認証させるためにオーソライザ２３２と通信するプロトコルインタープリタ２１０によって実行され得る。オーソライザ２３２は、要求内に含まれるクライアント識別子を認証プロセスの際に使用できる。オーソライザ２３２がステップ３１４において受取られた要求を認証した後、オーソライザ２３２は認証プロセスの結果をプロトコルインタープリタ２１０に通信する。オーソライザ２３２は、ケルベロス（Kerberos）、ＬＩＰＫＥＹおよびＳＰＫＭ－３を含む標準的な認証ライブラリならびにプロトコルを使用してリクエスタを認証してもよい。

【００８０】

ステップ３１４において受取られた要求がオーソライザ２３２によって認証されない場合、プロトコルインタープリタ２１０は、（ステップ３１４において受取られた）第２の要求を送ったリクエスタに通信を送って、第２の要求が認証されなかったことをリクエスタに知らせる。一旦第２の要求が認証されると、処理はステップ３１８に進む。

【００８１】

要求された操作が許可されるかどうかの判断

ステップ３１８において、リクエスタが現在のファイルシステム操作を実行するのに十分な許可レベルを有しているかどうかについて判断がなされる。ステップ３１８は、リクエスタが現在のファイルシステム操作を実行するのに十分な許可レベルを有しているかどうかを特権ベリファイヤ２３０に検証させるために特権ベリファイヤ２３０と通信するプ

ロトコルインタープリタ２１０によって実行され得る。

【００８２】

実施例では、リクエスタが各リクエスタごとにアクセス制御リストを使用して指定されたファイルシステム操作を実行するのに十分な許可レベルを有しているかどうかを特権ベリファイヤ２３０が判断する。特権ベリファイヤ２３０は、各リクエスタごとにアクセス制御リストを維持する。各々のアクセス制御リストは、アクセス制御項目（access control entries）（ＡＣＥｓ）のリストを含む。各ＡＣＥは、リクエスタが特定の特権を付与されるか、または拒否されるかを識別する。

【００８３】

例示するために、リクエスタ１２３４が特権Ａおよび特権Ｂを必要とするファイルシステム操作を実行するための要求を発行したと想定されたい。特権ベリファイヤ２３０は、リクエスタ１２３４のためのＡＣＥのリストを維持する。特権ベリファイヤ２３０は、アクセス制御リストに指定されたＡＣＥをシーケンシャルに処理する。リクエスタ１２３４のためのアクセス制御リストが、リクエスタ１２３４が許可Ａを付与されたことを示した第１のＡＣＥと、リクエスタ１２３４が許可Ｂを付与されたことを示した第２のＡＣＥと、リクエスタ１２３４が許可Ａを拒否されたことを示した第３のＡＣＥとを含んでいた場合、特権ベリファイヤ２３０は、リクエスタ１２３４が要求されたファイルシステム操作を実行するのに十分な許可レベルを有していると判断することになる。なぜなら、判断がなされ得るまで特権ベリファイヤ２３０はアクセス制御リストにおけるＡＣＥをシーケンシャルに処理することになるためである。したがって、一旦特権ベリファイヤ２３０がリクエスタ１２３４のためのアクセス制御リストにおける第２のＡＣＥを読取ると、特権ベリファイヤ２３０は、リクエスタ１２３４が要求されたファイルシステム操作を実行するのに十分な許可レベルを有しているかどうかについて判断でき、特権ベリファイヤ２３０はアクセス制御リストの残りを読取ることはない。ステップ３１８を実行した後、処理はステップ３２０に進む。

【００８４】

適切な状態情報の位置付け

ステップ３２０では、現在のファイルシステム操作の実行が状態情報を必要とする場合、第２の要求内に含まれる状態識別データに基づいて適切な状態情報が検索される。状態識別データは以前に割当てられ、リクエスタに通信された可能性がある。たとえば、リクエスタは以前にファイルを開いたかもしれず、または以前にファイル上にロックを付与したかもしれない。ステップ３２０において検索される状態情報は、要求が複合要求である場合、現在のファイルシステム操作に関連付けられてもよい。ステップ３２０は、ルックアップ機構２１２を使用して状態情報を検索するプロトコルインタープリタ２１０によって実行され得る。ステップ３２０において検索された状態情報は、現在のファイルシステム操作を実行するのに必要な任意の状態情報を含む。ステップ３２０を処理した後、処理はステップ３２２に進む。

【００８５】

要求されたファイルシステム操作の実行

ステップ３２２では、現在のファイルシステム操作が、適切な状態情報に基づいて、選択されたデータベースセッション内で処理される。一実施例では、ステップ３２２はプロトコルインタープリタ２１０自体によって実行されてもよい。別の実施例では、プロトコルインタープリタ２１０は、フレームワーク２００の他のコンポーネントに現在のファイルシステム操作を実行させるために他のコンポーネントと通信してもよい。現在のファイルシステム操作が処理された後、処理はステップ３２４に進む。

【００８６】

状態情報の更新

ステップ３２２では、ファイルシステム操作がセッションにおいて実行される。セッションによって使用される状態は、ファイルシステム操作の実行によって変わる。この例では、そのセッションの状態を表わす状態情報は「更新された状態情報」と称されるものと

する。更新された状態情報は、現在のファイルシステム操作の処理から生じた状態の変更を反映する。たとえば、更新された状態情報は、ファイルシステム操作の対象であるファイルが開かれたかどうか、およびファイル上に任意のロックが付与されたかどうかを反映する。したがって、現在のファイルシステム操作がファイルに対して実行された後、更新された状態情報はファイルの現在の状態を反映する。

【００８７】

ステップ３２４では、現在のファイルシステム操作に関連付けられる、更新された状態情報を反映するように、ルックアップ機構２１２内に格納された情報が更新される。実施例では、ルックアップ機構２１２を構成する１つ以上のＢ－ツリーがセッションの新しい状態を示すように更新される。ルックアップ機構２１２を構成するＢ－ツリーは、（ａ）更新された状態情報を識別するために新しい状態識別データを生成することによって、および（ｂ）更新された状態情報を反映するように項目をルックアップ機構２１２の適切なＢ－ツリーに更新または追加することによって更新され得る。

【００８８】

たとえば、ステップ３２２では、ステップ３２２において処理された現在のファイルシステム操作が特定のファイルの最初の１００バイト上でファイルベースのロックを実行するための操作であったと想定されたい。リソースロッカー２２２は最初に、granted locks Ｂ－ツリーにアクセスすることによって矛盾するロックがファイル上に既に付与されたかどうかを判断できる。リソースロッカー２２２は、現在のファイルシステム操作において識別されたファイルのファイルハンドルを使用して、granted locks Ｂ－ツリーを横断できる。granted locks Ｂ－ツリーにおける項目がファイルハンドルのために存在する場合、その項目を考査することによって、リソースロッカー２２２は矛盾するロックがファイル上に既に付与されたかどうかを知らされることになる。

【００８９】

矛盾するロックがファイル上にまだ付与されていないことをリソースロッカー２２２が判断する場合、リソースロッカー２２２は、（ａ）リソースの新しい状態を識別するために新しい状態識別データを生成し、（ｂ）要求されたロックの付与を反映するように項目をgranted locks Ｂ－ツリーに追加する。具体的には、リソースロッカー２２２は、リソースのための新しく生成された新しい状態識別データを使用して新しい項目をgranted_locks Ｂ－ツリーに追加でき、その後、以前の状態識別データが参照したlocks Ｂ－ツリーにおける以前の項目を削除できる。granted_locks Ｂ－ツリーにおける新しい項目は、リソース上に付与された任意の以前のロックに加えて、ファイルの最初の１００バイト上に付与されたファイルベースのロックへの参照を含むので、以前の状態識別データが参照した項目を格納する必要がない。

【００９０】

ステップ３２４を実行した後、処理はステップ３２６に進む。

複合要求に指定された操作を介する反復

各要求は、実行されるべき１つ以上のファイルシステム操作を指定する複合要求であってもよい。ステップ３２６では、ステップ３１４において受取られた要求が複合要求であり、複合要求に指定されたさらなる未処理のファイルシステム操作がある場合、処理はステップ３１８に進み、ステップ３１４の第２の要求に指定された次の未処理のファイルシステム操作が「現在のファイルシステム操作」になる。この態様で、複合要求に指定された各々のファイルシステム操作はフレームワーク２００によってシーケンシャルに処理される。

【００９１】

ステップ３１４の第２の要求に指定されたすべてのファイルシステム操作が処理された後、処理はステップ３２８に進む。

【００９２】

結果および改訂された状態識別子の、リクエスタへの提供

ステップ３２８では、ステップ３１４の要求に指定されたすべてのファイルシステム操

作を実行した結果が通信でリクエスタに伝送される。通信は、うまく実行されたファイルシステム操作の対象である特定のリソースに割当てられた状態情報を識別する任意の状態識別データを含んでもよい。ステップ３２８の実行は、状態保持ファイルシステム操作の実行に応答して生成される任意の状態識別データとともに、複合要求の各々のファイルシステム操作を処理した結果をリクエスタに送るプロトコルインタープリタ２１０によって実行され得る。たとえば、リクエスタが以前に開いたファイル上の特定の範囲のバイトに読取－書込ロックが付与されることをリクエスタが要求した場合、プロトコルインタープリタ２１０は、リソースの新しい状態を識別する新しい状態識別データを含む、すなわち、特定のファイル上の特定の範囲のバイトに読取－書込ロックが付与された通信をリクエスタに送ることによってステップ３２８を実行してもよい。なお、新しい状態識別情報は、状態不保持ファイルシステム操作のうまくいった処理に応答するのではなく、状態保持ファイルシステム操作のうまくいった処理に応答してリクエスタに伝送される。

【００９３】

ＮＦＳプロトコルでは、複合要求に指定された複数のファイルシステム操作を処理した結果が単一の通信でリクエスタに伝送され得る。したがって、ステップ３２８においてリクエスタに伝送された状態識別データは、複合要求に指定された各々のうまく実行された状態保持ファイルシステム操作ごとに状態識別情報を含む通信によって単一の通信で送られることができる。

【００９４】

フレームワーク２００が複合要求における特定のファイルシステム操作を処理できない場合、単一の通信がリクエスタに伝送される。通信は、（ａ）任意の新しい状態識別情報を含む、処理された複合要求に指定されたファイルシステム操作を処理した結果、および（ｂ）どのファイルシステム操作が実行できない可能性があるかを示す情報を記述する情報を含む。

【００９５】

フレームワークを使用した状態不保持トランザクションの処理

フレームワーク２００は、状態不保持ファイルシステム操作または状態不保持プロトコルに一致する要求などの状態不保持要求も処理できる。プロトコルインタープリタ２１０が状態不保持要求を含むパケットを受取ると、プロトコルインタープリタ２１０はパケットを読取りかつ解釈するためにパケットをコンポーネントに伝送できる。たとえば、プロトコルインタープリタ２１０はＦＴＰ要求を含むパケットをＦＴＰパケットリーダ２２６に送り、プロトコルインタープリタ２１０はＨＴＴＰ要求を含むパケットをＨＴＴＰパケットリーダ２２８に送る。

【００９６】

状態不保持要求を読取りかつ解釈した後、ＦＴＰパケットリーダ２２６およびＨＴＴＰパケットリーダ２２８は、状態不保持要求を識別する情報をプロトコルインタープリタ２１０に伝送する。プロトコルインタープリタ２１０は、次に、状態不保持要求を実行する場合もあれば、状態不保持要求を実行するためにフレームワーク２１０の別のコンポーネントと通信する場合もあり、たとえばリソースをロックするのにリソースロッカー２２２が必要であるかもしれない。要求が状態不保持であるので、一旦要求がうまく実行されると、状態情報を要求に割当てる必要がない。

【００９７】

ファイルシステム操作とデータベーストランザクションとの間の関係

クライアントがファイルに書込みたいとき、クライアントはＯＰＥＮファイルシステム操作、次いで複数の書込ファイルシステム操作、次いでＣＬＯＳＥファイルシステム操作の実行を要求し得る。このセクションの目的のために、単一のファイルシステム操作とは、ＯＰＥＮファイルシステム操作から始まって対応するＣＬＯＳＥファイルシステム操作までの複数のＮＦＳ操作を指す。単一のファイルシステム操作を実行するために、１つ以上のデータベーストランザクションが実行されるようにするのにデータベースサーバ１２２が必要であるかもしれない。１つ以上のデータベーストランザクションの各々は、ファ

イルシステム操作が実行される前にコミットされる。したがって、特定のデータベーストランザクションによってデータベース１２４に加えられる変更は、ファイルシステム操作の実行がうまくいくものであるかどうかがわかる前にコミットされる。

【００９８】

したがって、次のいくつかのセクションにおいて以下でさらに詳細に説明するように、リソースを見たいリクエスタは、（ａ）任意のコミットされたデータベーストランザクションを現在のところ反映しているリソースのバージョン、または（ｂ）完了したファイルシステム操作のみを反映し、まだ完了していないファイルシステム操作に対応する任意のコミットされたデータベーストランザクションを反映しないリソースのバージョンのいずれかを見ることを予期するであろう。

【００９９】

オープンコミットされた変更

リクエスタは同一のリソース上で独立してＯＰＥＮコマンドおよびＣＬＯＳＥコマンドを発行できる。したがって、たとえＣＬＯＳＥコマンドが１つのリクエスタに対してファイルを閉じる場合であっても、ファイルはすべてのリクエスタに対して依然として閉じられない可能性がある。「最後のクローズ」という用語は、結果的にすべてのリクエスタに対してファイルが閉じられることになるＣＬＯＳＥファイルシステム操作を指す。したがって、１つ以上のリクエスタによって現在開かれているいずれのリソースも最後のクローズをリソース上で実行させていない。

【０１００】

各々がファイルの状態を変更する複数のデータベーストランザクションは、ファイルが開かれたときと最後のクローズのときとの間に実行され得る。ファイル上で実行される変更は、ファイル上での最後のクローズが実行される前にコミットされ得る。（１）データベースにおいてコミットされたが、（２）最後のクローズをもたなかったファイルを伴う変更が、本明細書において「オープンコミットされた変更」と称される。

【０１０１】

一貫性のないクライアント

最後のクローズがリソース上で実行されておらず、リクエスタがリソースを得るための要求を送るとき、リクエスタが受取るべきリソースの状態は、リクエスタに関連付けられるクライアントのタイプに依存する。「一貫性のないクライアント」とは、リソースの「現在の状態」を見ることを予期するクライアントである。この文脈では、リソースの現在の状態は、リソースに加えられる任意のオープンコミットされた変更を含むが、リソースに加えられる任意のコミットされない変更を含まない。

【０１０２】

たとえば、リソースが最初に開かれて以来２つのデータベースのコミットされたトランザクションがリソースの状態を変更しており、最後のクローズがリソース上で実行されていない場合、リソースのための要求を発行する一貫性のないクライアントは、２つのデータベーストランザクションによって加えられる変更を反映するリソースの状態を見ることを予期する。ＮＦＳ、ＦＴＰまたはＨＴＴＰプロトコルを使用してＤＢＭＳ１２０にアクセスするクライアントは、一貫性のないクライアントの一例である。一貫性のないクライアントに関連付けられるリクエスタは一貫性のないリクエスタであり、すなわち、リクエスタはリソースの現在の状態を見ることを予期することになる。

【０１０３】

一貫性のあるクライアント

一貫性のあるクライアントとは、任意のオープンコミットされた変更を見ることを許されないクライアントである。むしろ、一貫性のあるクライアントは、（ａ）リソースが開かれたが閉じられていない場合にリソースが開かれる前、または（ｂ）最後のクローズがリソース上で実行された後のいずれかにリソースに加えられたコミットされた変更のみを見る。たとえば、リソースが開かれたが、最後のクローズがリソース上で実行されていないと想定されたい。リソースへのアクセスを要求する一貫性のあるクライアントは、ＯＰ

ＥＮ操作の実行の直前にリソースの状態を見ることを予期する。

【０１０４】

したがって、リソースが開かれ、最後のクローズが実行されていないとき以来２つのコミットされたデータベーストランザクションがリソースの状態を変更した場合、リソースのための要求を発行する一貫性のあるクライアントは、２つのトランザクションによって加えられる変更を反映しないリソースの状態を見ることを予期する。説明を容易にするために、一貫性のあるクライアントが見なければならないリソースの状態は、リソースの「クローズコミットされた」バージョンと称されるものとする。

【０１０５】

ＳＱＬプロトコルを使用してＤＢＭＳ１２０にアクセスするクライアントは、一貫性のあるクライアントの一例である。一貫性のあるクライアントに関連付けられるいずれのリクエスタも一貫性のあるリクエスタであり、すなわち、リクエスタはクローズコミットされた状態でリソースの状態を見ることを予期することになる。

【０１０６】

さらに例示するために、以下のファイルシステム操作および時点が以下の順序で起こる。

【０１０７】

（１） 時間ｔ１
（２） リクエスタ１がファイルｆ１を開く
（３） リクエスタ１が変更をファイルｆ１にコミットする
（４） 時間ｔ２
（５） リクエスタ２がファイルｆ１を開く
（６） リクエスタ２が変更をファイルｆ１にコミットする
（７） 時間ｔ３
（８） リクエスタ１がファイルｆ１を閉じる
（９） 時間ｔ４
（１０） リクエスタ２がファイルｆ１を閉じる
（１１） 時間ｔ５

時間ｔ３において、ファイルｆ１の一貫性のあるバージョンは時間ｔ１におけるファイルであり、ファイルの一貫性のないバージョンは時間ｔ３におけるファイルである。時間ｔ４において、ファイルｆ１の一貫性のあるバージョンは時間ｔ１におけるファイルであり、ファイルの一貫性のないバージョンは時間ｔ４におけるファイルである。時間ｔ５において、ファイルｆ１の一貫性のあるバージョンは時間ｔ５におけるファイルであり、ファイルの一貫性のないバージョンは時間ｔ５におけるファイルである。一貫性のあるクライアントがリソースの以前の状態を見ることを予期するので、その状態は最後のクローズがリソース上で実行されるまで保存されなければならない。

【０１０８】

クローズコミットされたバージョンの再構築

フレームワーク２００が一貫性のあるリクエスタおよび一貫性のないリクエスタをサポートするために、フレームワーク２００は異なるタイプのロック、すなわちデータベースロックおよびファイルベースのロックを利用する。データベースロックとはデータベース操作の実行に応答して得られるロックであり、データベース操作がうまく完了した（コミットした）ときにデータベースロックは解除される。ファイルベースのロックとはＯＰＥＮファイルシステム操作の実行に応答して得られるロックであり、ＣＬＯＳＥファイルシステム操作が実行されるときにファイルベースのロックは解除される。

【０１０９】

図４は、この発明の実施例に従うデータベースロックおよびファイルベースのロックを使用する機能ステップを示すフローチャートである。ステップ４１０では、リクエスタは特定のリソースを伴う操作を要求する。ステップ４１０は、通信リンク１３０によってデータベースサーバ１２２に要求を送るクライアント１１０によって実行され得る。ステッ

プ４１０を実行した後、処理はステップ４１２に進む。
【０１１０】
ステップ４１２では、リクエスタのリクエスタタイプについて判断がなされる。ステップ４１２はデータベースサーバ１２２によって実行され得る。リクエスタタイプに基づいて、データベースサーバ１２２は特定のリソースのどのバージョンをリクエスタに送るかを判断する。リクエスタが一貫性のないリクエスタである場合、データベースサーバ１２２は特定のリソースの現在のバージョンを送る。しかしながら、リクエスタが一貫性のあるリクエスタである場合には、データベースサーバ１２２は特定のリソースのより古いバージョン、すなわちリソースのクローズコミットされたバージョンを送る。
【０１１１】
リクエスタタイプの判断は、要求が一致するプロトコルのタイプに基づいて実行されてもよい。要求がＳＱＬプロトコルに一致する場合、リクエスタは一貫性のあるリクエスタである。しかしながら、要求がＮＦＳ、ＦＴＰまたはＨＴＴＰプロトコルに一致する場合、リクエスタは一貫性のないリクエスタである。ステップ４１２を実行した後、処理はステップ４１４に進む。
【０１１２】
ステップ４１４では、要求された操作を実行するために、特定のリソース上の第１のロックが得られる。第１のロックは、ファイルベースのロックなどの第１のタイプのロックである。ステップ４１４を実行した後、処理はステップ４１６に進む。
【０１１３】
ステップ４１６では、要求された操作が必要とする各々のデータベース操作を実行するために、第２のロックが得られる。第２のロックは、データベースロックなどの第２のタイプのロックである。
【０１１４】
実施例では、特定のリソースの状態を変更する任意のデータベース操作を実行する前に、リソースの一時的なコピーがデータベース１２４に格納される。ファイルベースのロックが特定のリソース上に付与されたとき、特定のリソースに対する変更は、実際のリソース自体ではなくリソースの一時的なコピーに反映される。リソースの元のバージョンが修正されないままであるので、元のバージョンは一貫性のあるリクエスタをサービスする際にデータベースサーバ１２２によって使用されることができる。コミットされたデータベース操作によってリソースに加えられたすべての変更を一時的なコピーが反映するので、データベースサーバ１２２は一貫性のないリクエスタをサービスする際にリソースの一時的なコピーを使用できる。ステップ４１６を実行した後、処理はステップ４１８に進む。
【０１１５】
ステップ４１８では、データベースロックは対応するデータベース操作がうまく完了したことに応答して解除される。操作がデータベースシステムによって実行されるとき、データベースシステムは、操作を実行するために使用されるトランザクションをコミットし、操作中に修正されたすべてのリソース上に保持されるデータベースロックを解除する。要求された操作が必要とするすべてのデータベース操作が実行された後、処理はステップ４２０に進む。
【０１１６】
ステップ４２０では、ファイルシステム操作がうまく完了したことに応答して、ファイルベースのロックが解除される。具体的には、最後のクローズがリソース上で実行されるときに、リソース上のファイルベースのロックが解除され、リソースの一時的なコピーがリソースの現在のバージョンとして確立されることができる。一時的なコピーは、たとえば一時的なコピーを元のコピーの上にコピーし、次いで一時的なコピーを削除することによって、現在のバージョンとして確立されることができる。
【０１１７】
ファイルシステム操作が実行された後、リソースの一貫性のないバージョンおよびリソースのクローズコミットされたバージョンは同一のものである。その結果、一貫性のある

リクエスタおよび一貫性のないリクエスタは両方、リソースが再び開かれるまでリソースの元のバージョンを使用してサービスされることができる。

【０１１８】

図４のステップを実行することによって、データベースサーバ１２２が一貫性のあるリクエスタおよび一貫性のないリクエスタの両方をサービスできるようにするためにファイルベースのロックおよびデータベースロックが使用され得る。ファイルベースのロックがリソース上で維持されるとき、ＯＰＥＮファイルシステム操作の実行の前のリソースの状態が維持され、このようにして、データベースサーバ１２２が一貫性のあるリクエスタをサービスすることが可能になる。

【０１１９】

同時アクセスの管理

複数のリクエスタが同一のリソースを伴う操作を実行しているとき、ファイルベースのロックの使用は等しく有利である。たとえば、複数のリクエスタは各々が同一ファイル上でファイルシステム操作を実行するための要求を発行してもよい。２つ以上のリクエスタがファイルを開いてもよく、２つ以上のリクエスタがリソースの状態に変更を加えてもよい。

【０１２０】

例示するために、第１のリクエスタがファイルを開き、かつファイルに変更を加えたと想定されたい。第２のリクエスタが同一ファイルのバージョンのための要求をデータベースサーバ１２２に送るとき、データベースサーバ１２２は第２のリクエスタのリクエスタタイプを判断する。第２のリクエスタが一貫性のあるリクエスタである場合、データベースサーバ１２２は、ファイルが開かれて以来第１のリクエスタによってファイルに加えられたいかなる変更も反映しないファイルのバージョンをもたらす。第２のリクエスタが一貫性のないリクエスタである場合、データベースサーバ１２２は、ファイルが開かれて以来第１のリクエスタによってファイルに加えられた変更を反映するファイルのバージョンをもたらす。

【０１２１】

リソースがファイルベースのロックの対象でありながらいかにデータベースサーバがリソースの状態を維持できるかについてのさらなる情報が、「トランザクションセマンティックスの実行」と題されるセクションにおいて以下に記載される。

【０１２２】

トランザクションセマンティックスの実行

リソースがＯＰＥＮファイルシステム操作の対象であった時点でリソースの以前のバージョンについての情報を維持することが有利である多数の理由が存在する。第１に、上で説明したように、リソースがＯＰＥＮファイルシステム操作の対象であったが、最後のクローズの対象ではなかった時点でリソースの以前のバージョンを維持することによって、データベースサーバ１２２は一貫性のあるリクエスタからリソースのための要求をサービスすることが可能である。第２に、リソースの以前のバージョンを維持することによって、データベースサーバは以前のバージョンにリソースを戻すことが可能である。（ａ）リクエスタがリソースの誤ったバージョンを作成するとき、（ｂ）リクエスタがスキーマと互換性のないスキーマベースのリソースのバージョンを作成するとき、または（ｃ）複数のリクエスタによってリソース上で実行された変更が互いに互換性がないときなどのさまざまな状況において、以前のバージョンにリソースを戻すことが必要である場合がある。

【０１２３】

重要なことに、以前の状態にリソースを戻すためにリソースから除去される必要がある変更は、コミットされた変更を含み得る。その結果、コミットされないトランザクションによって加えられる変更を除去するためにデータベースシステムによって使用される従来のアンドゥ機構は、必要な戻しを実行するのに十分ではない。

【０１２４】

たとえ以前の状態からリソースの状態を変更した、コミットされたデータベーストラン

ザクションが実行されたとしても、この発明の実施例によって、有利に、リソースは以前の状態に戻されることが可能である。この発明の実施例によれば、コミットされたデータベーストランザクションによって１つ以上の変更がリソースに加えられる。コミットされたデータベーストランザクションがリソースの状態を変更した後、コミットされたデータベーストランザクションによって変更が加えられる前の状態にリソースを戻すための要求が受取られる。たとえば、クライアント１１０は、ファイルのクローズコミットされたバージョンなどの特定の時点より前の状態に特定のファイルを戻すための要求をデータベースサーバ１２２に発行してもよい。

【０１２５】

要求の受取に応答して、ファイルが開かれた時点などの特定の時点より前の状態にリソースが戻される。リソースを戻す際に、リソースの現在の状態は、コミットされたデータベーストランザクションによってファイルに加えられた変更を反映することを中止する。以前の状態にリソースを戻すための技術について次のセクションにおいてさらに詳細に説明する。

【０１２６】

リソース戻し技術

特定の時点より前の状態にリソースを戻すためにさまざまな技術が使用され得る。使用される特定の技術は、たとえばリソースがスキーマベースのリソースであるかまたはスキーマベースでないリソースであるかに依存するかもしれない。スキーマベースのリソースとは、定義されたスキーマに一致するリソースである。たとえば、所与のスキーマに一致する購入発注文書はスキーマベースのリソースの一例である。スキーマベースでないリソースとは、スキーマベースのリソースではないあらゆるリソースである。

【０１２７】

分解された形態でのリソースの格納

スキーマベースのリソースは、リソース全体を一緒に格納することによって、たとえばデータベーステーブルのlob列にＸＭＬ文書を格納することによって、構築された形態で格納されることができる。代替的には、スキーマベースのリソースを含む要素を別個に格納することによって、分解された形態でスキーマベースのリソースを格納することが有利であろう。たとえば、ＸＭＬ文書の個々のＸＭＬタグおよびそれらの関連付けられるデータを記述するデータは、データベーステーブルの列に格納されてもよい。スキーマベースのリソースの要素が別個に格納されるので、スキーマベースのリソースの要素は、スキーマベースのリソースが読取られる前に再構築される必要があるかもしれない。

【０１２８】

図５は、分解された形態でスキーマベースのリソースを格納するための機構を示すリソーステーブルを示す。図５のテーブルは参照列５０４を含む。スキーマベースのリソースを記述するデータは、リソーステーブルに格納される場合もあれば、リソーステーブルによって参照される場合もある。たとえば、リソーステーブルの参照列５０４は、スキーマベースのリソースに関するデータが格納される別のテーブル、すなわちＸＭＬタイプテーブル５１０を識別するポインタ５０６を含む。ＸＭＬタイプテーブル５１０はそれ自体が、スキーマベースのリソースの他のデータ要素を格納する１つ以上の他のテーブルを参照できる。たとえば、ＸＭＬタイプテーブル５１０は、入れ子テーブル５２０への参照５１２とともに示される。

【０１２９】

ＸＭＬタイプテーブル５１０および任意の入れ子テーブル５０２は、スキーマベースのリソースの要素についてのデータを格納する。リクエスタがスキーマベースのリソースの最初の１００バイトを読取りたいときには、リソースはその要求をサービスするために再構築されなければならない。なぜなら、ＸＭＬタイプテーブル５１０は、スキーマベースのリソースの各データ要素がどのバイトに現われるかを記述する情報を格納しないためである。その結果、データがスキーマベースのリソースから読取られるとき、スキーマベースのリソースは再構築されなければならず、ＸＭＬ lob列５０２に格納されなければなら

ない。リクエスタがスキーマベースのリソースの最初の１００バイトを読取りたい場合、このような要求は、ＸＭＬ lob列５０２に格納された、再構築されたリソースの最初の１００バイトを読取ることによってデータベースサーバ１２２によって容易に実行されることができる。

【０１３０】

以下でさらに詳細に説明するように、後続の操作はＸＭＬ lob列５０２に格納されたリソースの再構築されたコピー上で実行されることができ、ＸＭＬタイプテーブル５１０および任意の入れ子テーブル５２０に格納されたリソースの分解された要素をそのままの状態にする。

【０１３１】

スキーマベースのリソースの戻し

一実施例によれば、スキーマベースのリソースは「以前のバージョン情報」に基づいて戻される。図５は、この発明の実施例に従うスキーマベースのリソースについての以前のバージョン情報を格納するシステムのブロック図である。以前のバージョン情報はＸＭＬタイプテーブル５１０および任意の入れ子テーブル５２０に維持されることができるのに対して、スキーマベースのリソースに加えられる変更は、最後のクローズがスキーマベースのリソース上で実行されるまで、ＸＭＬ lob列５０２に格納されたリソースの再構築されたコピー上で実行されることができる。

【０１３２】

この発明の実施例では、ファイルベースのロックがリソース上に付与されると、リソースの状態を変更し得るデータベース操作を実行する直前に、スキーマベースのリソースの構築されたコピーが作成される。たとえば、スキーマベースのリソースの構築されたコピーはＸＭＬ lob列５０２において作成および格納されてもよい。

【０１３３】

その後、リソースの構築されたコピー（ＸＭＬ lob列５０２に格納されたリソースのコピー）はリソースの現在のバージョンとして扱われ、データベース操作が必要とする変更がリソースの構築されたコピー（ＸＭＬ lob列５０２に格納されたリソースのコピー）に加えられる。事実上、ＸＭＬ lob列５０２におけるリソースのコピーは、リソースの不正なバージョンのキャッシュになる。なお、スキーマベースのリソースの分解されたバージョンは依然としてＸＭＬタイプテーブル５１０に維持される。

【０１３４】

リソースの分解されたコピーにスキーマベースのリソースを戻すために、ＸＭＬ lob列５０２に格納されたリソースのコピーが削除される。その後、ＸＭＬタイプテーブル５１０および任意の入れ子テーブル５２０に格納されたリソースの分解されたバージョンが、ＸＭＬタイプテーブル５１０に格納された、構築されたコピーの代わりにリソースの現在のバージョンとして扱われる。

【０１３５】

ＣＬＯＳＥファイルシステム操作がリソース上で実行されるとき、ＸＭＬタイプテーブル５１０に格納されたリソースの分解されたコピーに加えられる変更は、ＸＭＬ lob列５０２に格納されたリソースの構築されたコピーを反映するように、ＸＭＬタイプテーブル５１０および任意の入れ子テーブル５２０に格納されたリソースの分解されたバージョンを変更することによって永久的なものにされることができる。

【０１３６】

スキーマベースでないリソースを戻すためのスナップショット時間の使用

図６Ａおよび図６Ｂは、この発明の実施例に従うスキーマベースでないリソースについての以前のバージョン情報を格納するブロック図である。図６Ａおよび図６Ｂは、スキーマベースでないリソースについての以前のバージョン情報を格納するための３つの異なるアプローチを説明するために使用するものとする。

【０１３７】

第１のアプローチによれば、図６Ａに示されるように、リソーステーブル６００はＬＯ

Ｂ列６０２にスキーマベースでないリソースを格納する。このアプローチでは、ＯＰＥＮファイルシステム操作がリソース上で実行されるとき、スナップショット時間がリソーステーブル６００の列６０４に格納される。スナップショット時間は、ＯＰＥＮファイルシステム操作がリソース上で実行される直前の論理的な時間を示す。

【０１３８】

１つ以上のデータベーストランザクションが変更をリソースにコミットした後、データベーストランザクションは「アンドゥ」されないかもしれないが、リソースはスナップショット時間以来リソースに関連付けられるアンドゥ情報を使用してスナップショット時間時点の状態に戻されることができる。アンドゥ情報とは、実行されたが、コミットされていないデータベーストランザクションを「巻き戻す（roll back）」またはアンドゥするために使用され得る、ＤＢＭＳ１２０によって維持される情報を指す。

【０１３９】

スナップショット時間およびアンドゥ情報は、リソースの状態を変更させて、スナップショット時間時のリソースの状態を反映するようにリソースに変更の組を適用するために使用される。一旦リソースがスナップショット時間時のリソースの状態を反映するように戻されると、スナップショット時間はリソーステーブル６００の列６０４から除去される。

【０１４０】

実施例では、リソースの状態を変更させて、スナップショット時間時のリソースの状態を反映するようにリソースに変更の組を適用するために「フラッシュバッククエリー」が使用されてもよい。フラッシュバッククエリーを実行するための技術は、２００３年４月３０日に出願された「フラッシュバックデータベース（Flashback Database）」と題される米国特許出願連続番号第１０／４２７，５１１号に記載され、これはまるで本明細書に十分に説明されているかのように全文が引用によって援用される。

【０１４１】

スキーマベースでないリソースを戻すためのキャッシュ列の使用

第２のアプローチによれば、図６Ｂに示されるように、リソーステーブル６５０はＬＯＢ列６５２にスキーマベースでないリソースを格納する。このアプローチでは、ＯＰＥＮファイルシステム操作がリソース上で実行されると、リソースのコピーはリソーステーブル６５０の列６５４に格納される。列６５４は「キャッシュ列」として使用される。具体的には、列６５４に格納されるリソースのコピーは、リソースの現在のバージョンとして扱われる。データベーストランザクションがリソースに変更を引起こすとき、その変更は列６５２に格納された元のリソースの代わりに列６５４に格納されたリソースのコピーに加えられる。

【０１４２】

ＣＬＯＳＥファイルシステム操作がリソース上で実行される場合には、６５４に格納されたリソースのコピーが列６５２に格納され得るので、元のリソースはコミットされたデータベース操作によってリソースに加えられるいかなる変更も反映することになる。ＣＬＯＳＥファイルシステム操作が実行されるまで、列６５２に格納されたリソースの現在の値はＯＰＥＮファイルシステム操作の実行の直前のリソースの状態を反映する。したがって、ＯＰＥＮファイルシステム操作の実行の直前のリソースの状態にリソースを戻す必要がある場合、生じる必要があるリソーステーブル６５０に対する唯一の変更は、列６５４に格納されたリソースのコピーを除去することである。最後のクローズがリソース上で実行される前に、一貫性のないリクエスタは列６５４におけるリソースのコピーを見ることができ、一貫性のあるリクエスタは列６５２に格納されたリソースを見ることができる。

【０１４３】

ハイブリッドアプローチ

格納空間の制約のために、ある特定の時間よりも古いアンドゥ情報は典型的には、より新しいアンドゥ情報によって上書きされる。その結果、スナップショット時間を使用して戻しを実行すること（すなわち第１のアプローチ）は必ずしも実現可能ではない。しかし

ながら、アンドゥ情報が利用可能であるときには、スナップショット時間ベースの戻しがキャッシュ列の戻し（すなわち第２の戻し）よりも好ましいであろう。

【０１４４】

その結果、第３の（ハイブリッド）アプローチでは、リソースが戻される必要があるかもしれないときにリソースについてのアンドゥ情報が利用可能でないかもしれないということをデータベースサーバ１２２が判断しない限り、上述のスナップショットベースのアプローチが実行される。リソースが戻される必要があるかもしれないときにリソースについてのアンドゥ情報が利用可能でないかもしれないということをデータベースサーバ１２２が判断する場合、上述のキャッシュ列アプローチが次いで実行される。

【０１４５】

データベースサーバ１２２は、データベースサーバ１２２によってアンドゥ情報を維持する時間の量が時間の構成可能な量未満である場合に、リソースが戻される必要があるかもしれないときにリソースについてのアンドゥ情報が利用可能でないかもしれないということを判断できる。

【０１４６】

一貫性のチェック

一実施例によれば、ファイルが閉じられるときに修正されたファイルの一貫性がチェックされ、それ以上の待ち状態のＯＰＥＮファイルシステム操作はない。たとえば、スキーマベースのリソースが確実にスキーマの規則に一致するようにするためにスキーマベースのリソースがチェックされ得る。スキーマベースのリソースが対応するスキーマに一致しない場合には、リソースは開かれたときにリソースの状態に戻され得る。

【０１４７】

上述のように、リソースが付与されたファイルベースのロックの対象であり、リクエスタが前の状態にリソースを戻すための要求を発行するか、またはリソースが一貫性のチェックの役に立たない場合に、リソースは上述のように前の状態に戻され得る。ファイルベースのロックのさらなる詳細および利点について以下に提示するものとする。

【０１４８】

ファイルベースのロック

ファイルベースのロックによって、データベースサーバ１２２はデータベース１２４に維持されるファイル上でファイルシステム操作を実行できる。リソースロッカー２２２は、データベース１２４に格納されたリソース上でファイルシステムロックを管理できる。ファイルベースのロックの挙動は、３つの重要な局面において、ＨＴＴＰなどの状態不保持プロトコルのために使用される他のロックとは異なる。

【０１４９】

第１に、ファイルベースのロックは、単にリソース全体の代わりにリソースの一部上に付与されてもよい。特に、ファイルベースのロックはリソース上のバイトの範囲に付与されてもよい。したがって、単一のファイルは複数のファイルベースのロックの対象であってもよく、各々のファイルベースのロックはファイルの異なるバイト範囲をカバーする。

【０１５０】

第２に、ファイルベースのロックはリースベースであり、これは、一旦特定のファイルベースのロックがリクエスタに付与されると、特定のロックは第１の期間付与され、第１の期間が切れた後、その特定のロックは期限切れになることを意味する。しかしながら、リクエスタによって受取られる通信はいずれも、第２の期間に特定のロックを新しくする。したがって、ファイルシステムロックが期限切れになる前にリクエスタがデータベースサーバ１２２と通信する限り、リクエスタは絶えずファイルベースのロックを新しくすることができる。

【０１５１】

一旦特定のファイルシステムロックが期限切れになると、特定のロックがもはや付与されないことを反映するようにルックアップ機構２１２が更新される。ルックアップ機構２１２内に維持されるデータは、リクエスタによって要求される各々のロックが確実にまだ

有効であるようにするために周期的にチェックされることができる。

【０１５２】

特定のリクエスタが以前に付与された別のロックと矛盾するロックを要求するときには、以前に付与されたロックが確実にまだ有効であるようにするために、以前に付与されたロックをチェックできる。以前に付与されたロックがもはや有効でない場合には、ロックが無効であることを反映するように、ルックアップ機構２１２に格納された情報が更新される（たとえば、ロックについての情報が削除され得る）。さらに、特定のクライアントに付与されたすべてのロックは、特定のクライアントが期限切れになったときに解除される。実施例では、クライアントが最後にフレームワーク２００と通信して以来構成可能な量の時間が経過した後、クライアントは期限切れになる可能性がある。したがって、以前に付与されたロックが、付与されるように要求されるロックと矛盾する場合に、クライアントがまだ有効であることを検証するために、以前に付与されたロックに関連付けられるクライアントをチェックできる。クライアントが有効でない場合には、以前に付与されたロックが解除され、付与されるように要求されるロックが実行され得る。この発明の実施例では、特定のクライアントが期限切れになったかどうかの判断はclient Ｂ－ツリーをチェックすることによって実行されることができる。

【０１５３】

状態不保持プロトコルロックと比較したファイルベースのロックの第３の相違点は、読取アクセスのみを提供するファイルベースのロックがないことである。その代わりに、ファイルベースのロックが読取アクセスを付与する程度に、ファイルベースのロックは読取－書込アクセスも付与する。

【０１５４】

この発明の実施例では、ファイルベースのロックは、リソース全体をカバーする第１の組と、リソースのバイトの範囲などのリソースの一部をカバーする第２の組とを含む。図７は、この発明の実施例に従うさまざまなタイプのファイルベースのロックおよびそれらの互換性を示すテーブルである。図７に示されるさまざまなファイルベースのロックの各々について以下で簡単に説明することにする。

【０１５５】

バイト－読取－書込ファイルベースのロックは、リソースの一部上のロックである。バイト－読取－書込ファイルベースのロックは、リソース上のバイトの範囲への読取アクセスおよび書込アクセスを付与するために使用されることができる。

【０１５６】

バイト－書込ファイルベースのロックは、リソースの一部上のロックである。バイト－書込ファイルベースのロックは、リソース上のバイトの範囲への書込アクセスを付与するために使用されることができる。

【０１５７】

拒否－読取ファイルベースのロックは、リソース全体上のロックである。拒否－読取ファイルベースのロックは、拒否－読取ロックを付与したリクエスタ以外の任意のリクエスタに対してリソースへの読取アクセスを拒否するために使用されることができる。

【０１５８】

拒否－書込ファイルベースのロックは、リソース全体上のロックである。拒否－書込ファイルベースのロックは、拒否－書込ロックを付与したリクエスタ以外の任意のリクエスタに対してリソースへの書込アクセスを拒否するために使用されることができる。

【０１５９】

ファイルベースのロックは、ＷｅｂＤＡＶロックなどのロック共有ロックまたはロック排他的ロックと互換性がない。図７は、さまざまなファイルベースのロックの互換性を記載する。特定のファイルベースのロックが以前に付与された別のロックと互換性がないときには、ファイルベースのロックは付与されないことになる。したがって、バイト－読取－書込ロックおよびバイト－書込ロックの範囲が矛盾しない場合に、既にバイト－書込ロックをリソース上に付与させたリソース上にバイト－読取－書込ロックを付与できる。し

かしながら、既にバイト－書込ロックをリソース上に付与させたリソース上に拒否－読取ロックを付与することはできない。

【０１６０】

リアルアプリケーションクラスタにおけるファイルベースのロック

データベース１２２は、オラクル・コーポレイションのＲＡＣ　１０ｇオプションを使用するなど、リアルアプリケーションクラスタ（Real Application Cluster）（ＲＡＣ）において実現されてもよい。ＲＡＣ環境では、ファイルベースのロックがリソース上に付与されるとき、どのデータベースサーバがリソース上でファイルベースのロックを付与したかを記述するデータがデータベース１２４に格納されなければならない。

【０１６１】

たとえば、データベースに格納されたリソースは、（ａ）ファイルベースのロックがリソース上に付与されたことを示すフラグおよび（ｂ）リソース上でファイルベースのロックを付与したデータベースサーバを識別する情報に関連付けられてもよい。ルックアップ機構２１２は、付与されたファイルベースのロックについてのデータをメモリに維持する。付与されたファイルベースのロックについての情報がＲＡＣインスタンスにおける他のノードの目に見える場合には、メモリに格納された情報は、永続的に格納されなければならないか、またはデータの一貫性を維持する態様でＲＡＣの他のノードに運ぶことができなければならない。ルックアップ機構２１２に格納された情報が、ＲＡＣがあるデータベースサーバ以外のＲＡＣの他のデータベースサーバの目に見えない場合には、第１のデータベースサーバによって付与された任意のファイルベースのロックは第２のデータベースサーバのファイルベースのロックと矛盾する可能性があるだろう。

【０１６２】

データベースサーバ１２２によって利用される上述のファイルベースのロックによって、データベースサーバ１２２はデータベース１２４によって維持されるファイル上で、要求されるＮＦＳ操作などの状態保持要求を処理することが可能である。その結果、データベース１２２が上述のファイルシステム操作のロックを利用できるときに、データの一貫性を保つ態様でＮＦＳプロトコルを使用して、データベース１２４に格納されたファイルにアクセスできる。

【０１６３】

機構の実現

クライアント１１０、データベースサーバ１２２およびデータベース１２４は各々が実施例に従ってコンピュータシステム上で実現されてもよい。図８は、この発明の実施例が実現され得るコンピュータシステム８００を示すブロック図である。コンピュータシステム８００は、情報を通信するためのバス８０２または他の通信機構と、バス８０２に結合されて情報を処理するためのプロセッサ８０４とを含む。コンピュータシステム８００は、バス８０２に結合されて、プロセッサ８０４によって実行されるべき情報および命令を格納するための、ランダムアクセスメモリ（random access memory）（ＲＡＭ）または他の動的記憶装置などのメインメモリ８０６も含む。メインメモリ８０６は、プロセッサ８０４によって実行されるべき命令の実行中に一時的な変数または他の中間情報を格納するためにも使用されることができる。コンピュータシステム８００はさらに、バス８０２に結合されて、プロセッサ８０４についての静的な情報および命令を格納するためのリードオンリーメモリ（read only memory）（ＲＯＭ）８０８または他の静的記憶装置を含む。磁気ディスクまたは光学ディスクなどの記憶装置８１０は、情報および命令を格納するために設けられ、バス８０２に結合される。

【０１６４】

コンピュータシステム８００は、情報をコンピュータユーザに表示するための、陰極線管（cathode ray tube）（ＣＲＴ）などのディスプレイ８１２にバス８０２を介して結合されてもよい。英数字キーおよび他のキーを含む入力装置８１４は、バス８０２に結合されて、情報およびコマンド選択をプロセッサ８０４に通信するためのものである。別のタイプのユーザ入力装置は、方向情報およびコマンド選択をプロセッサ８０４に通信するた

め、およびディスプレイ８１２上でカーソルの動きを制御するための、マウス、トラックボールまたはカーソル方向キーなどのカーソル制御装置８１６である。この入力装置は典型的には２つの軸、すなわち第１の軸（たとえばｘ）および第２の軸（たとえばｙ）において２自由度を有し、これによって、この装置は平面において位置を指定することが可能である。

【０１６５】

この発明は、本明細書に記載される技術を実現するためのコンピュータシステム８００の使用に関連する。この発明の一実施例によれば、それらの技術は、プロセッサ８０４がメインメモリ８０６に含まれる１つ以上の命令の１つ以上のシーケンスを実行することに応答してコンピュータシステム８００によって実行される。このような命令は、記憶装置８１０などの別の装置可読媒体からメインメモリ８０６に読取られてもよい。メインメモリ８０６に含まれる命令のシーケンスの実行は、本明細書に記載されるプロセスステップをプロセッサ８０４に実行させる。代替的な実施例では、この発明を実現するためにソフトウェア命令の代わりにまたはソフトウェア命令と組合せられてハードワイヤード回路が使用されてもよい。したがって、この発明の実施例はハードウェア回路およびソフトウェアの任意の特定の組合せに限定されない。

【０１６６】

本明細書において使用される「装置可読媒体」という用語は、具体的な態様で装置に動作させるデータの提供に参画する任意の媒体を指す。コンピュータシステム８００を使用して実現される実施例では、さまざまな装置可読媒体はたとえばプロセッサ８０４に命令を与えて実行することに関与する。このような媒体は、不揮発性媒体、揮発性媒体および伝送媒体を含むがこれらに限定されない多くの形態を取り得る。不揮発性媒体はたとえば、記憶装置８１０などの光学ディスクまたは磁気ディスクを含む。揮発性媒体は、メインメモリ８０６などの動的メモリを含む。伝送媒体は、バス８０２を構成するワイヤを含む同軸ケーブル、銅製ワイヤおよび光ファイバを含む。伝送媒体は、電波および赤外線データ通信中に発生するものなどの音波または光波の形態も取り得る。

【０１６７】

装置可読媒体の一般的な形態はたとえば、フロッピー（登録商標）ディスク、フレキシブルディスク、ハードディスク、磁気テープもしくは他の磁気媒体、ＣＤ－ＲＯＭ、他の光学媒体、パンチカード、紙テープ、穴のパターンを有する他の物理的な媒体、ＲＡＭ、ＰＲＯＭおよびＥＰＲＯＭ、ＦＬＡＳＨ－ＥＰＲＯＭ、他のメモリチップもしくはカートリッジ、以下に記載される搬送波、またはコンピュータが読取ることができる他の媒体を含む。

【０１６８】

装置可読媒体のさまざまな形態は、１つ以上の命令の１つ以上のシーケンスをプロセッサ８０４に搬送して実行することに関与し得る。たとえば、命令は最初にリモートコンピュータの磁気ディスクに搬送されてもよい。リモートコンピュータは命令をその動的メモリにロードでき、モデムを使用して電話線によって命令を送ることができる。コンピュータシステム８００にローカルなモデムは、電話線に沿ってデータを受取ることができ、データを赤外線信号に変換するために赤外線送信機を使用できる。赤外線検出器は赤外線信号で搬送されたデータを受取ることができ、適切な回路はデータをバス８０２に置くことができる。バス８０２はデータをメインメモリ８０６に搬送し、メインメモリ８０６から、プロセッサ８０４は命令を検索および実行する。メインメモリ８０６によって受取られた命令は、プロセッサ８０４による実行の前または後に、任意に記憶装置８１０に格納されてもよい。

【０１６９】

コンピュータシステム８００は、バス８０２に結合された通信インターフェイス８１８も含む。通信インターフェイス８１８は、ローカルネットワーク８２２に接続されたネットワークリンク８２０に結合して２方向のデータ通信をもたらす。たとえば、通信インターフェイス８１８は、対応するタイプの電話線にデータ通信接続をもたらすための総合デ

ジタル通信網（integrated services digital network）（ＩＳＤＮ）カードまたはモデムであってもよい。別の例として、通信インターフェイス８１８は、互換性のあるＬＡＮにデータ通信接続をもたらすためのローカルエリアネットワーク（ＬＡＮ）カードであってもよい。無線リンクも実現されてもよい。いかなるこのような実現例においても、通信インターフェイス８１８は、さまざまなタイプの情報を表わすデジタルデータストリームを搬送する電気信号、電磁気信号または光信号を送受信する。

【０１７０】

ネットワークリンク８２０は典型的には、１つ以上のネットワークを介するデータ通信を他のデータ装置にもたらす。たとえば、ネットワークリンク８２０はローカルネットワーク８２２を介する接続をホストコンピュータ８２４にもたらす場合もあれば、インターネットサービスプロバイダ（Internet Service Provider）（ＩＳＰ）８２６によって操作されるデータ装置にもたらす場合もある。ＩＳＰ８２６は次に、現在一般的に「インターネット」８２８と称される世界的なパケットデータ通信ネットワークを介してデータ通信サービスをもたらす。ローカルネットワーク８２２およびインターネット８２８は両方、デジタルデータストリームを搬送する電気信号、電磁気信号または光信号を使用する。デジタルデータをコンピュータシステム８００におよびコンピュータシステム８００から搬送する、さまざまなネットワークを介する信号およびネットワークリンク８２０に沿って通信インターフェイス８１８を介する信号は、情報を運ぶ搬送波の例示的な形態である。

【０１７１】

コンピュータシステム８００は、ネットワーク、ネットワークリンク８２０および通信インターフェイス８１８を介して、メッセージを送ることができ、プログラムコードを含むデータを受取ることができる。インターネットの例では、サーバ８３０が、インターネット８２８、ＩＳＰ８２６、ローカルネットワーク８２２および通信インターフェイス８１８を介して、アプリケーションプログラムのための要求されたコードを伝送するかもしれない。

【０１７２】

受取られたコードは、受取られたときにプロセッサ８０４によって実行されてもよく、および／または以後の実行のために記憶装置８１０または他の不揮発性記憶装置に格納されてもよい。この態様で、コンピュータシステム８００は搬送波の形態でアプリケーションコードを得ることができる。

【０１７３】

先の明細書では、この発明の実施例は実現例ごとに異なる可能性がある多数の具体的な詳細を参照しながら記載されてきた。したがって、何がこの発明であるかおよび何がこの発明であるように出願人によって意図されるかを単におよび排他的に指示するものは一組の特許請求の範囲である。一組の特許請求の範囲は、このような特許請求の範囲が生じる具体的な形態でこの出願に由来し、いかなるその後の訂正も含む。このような特許請求の範囲に含まれる用語について本明細書において明らかに説明する定義はいずれも、特許請求の範囲において使用されるような用語の意味を決定するものとする。それ故に、特許請求の範囲に明らかに記載されない限定、要素、特性、特徴、利点または属性は決してこのような特許請求の範囲を限定すべきではない。したがって、明細書および図面は限定的な意味ではなく例示的な意味で考えられるべきである。

【図面の簡単な説明】

【０１７４】

【図１】この発明の実施例に従って状態保持プロトコルの状態で実現される要求を処理できるシステムのブロック図である。

【図２】この発明の実施例に従うデータベースサーバの機能コンポーネントのブロック図である。

【図３】この発明の実施例に従うファイル操作を処理する機能ステップを示すフローチャートである。

【図４】この発明の実施例に従うデータベースロックおよびファイルベースのロックを使用する機能ステップを示すフローチャートである。
【図５】この発明の実施例に従うスキーマベースのリソースについての以前のバージョン情報を格納するブロック図である。
【図６Ａ】この発明の実施例に従うスキーマベースでないリソースについての以前のバージョン情報を格納するブロック図である。
【図６Ｂ】この発明の実施例に従うスキーマベースでないリソースについての以前のバージョン情報を格納するブロック図である。
【図７】この発明の実施例に従うさまざまなタイプのファイルベースのロックおよびそれらの互換性を示すテーブルである。
【図８】この発明の実施例が実現され得るコンピュータシステムを示すブロック図である。

【図１】

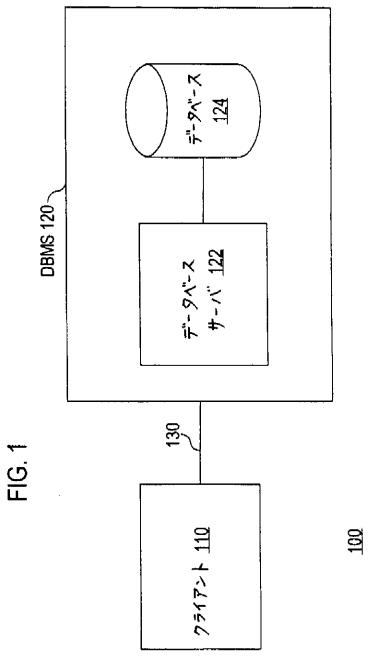



FIG. 1

【図２】

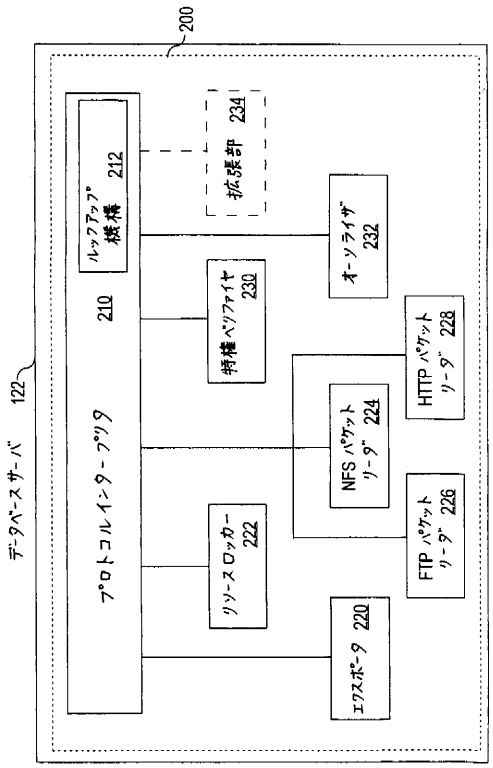
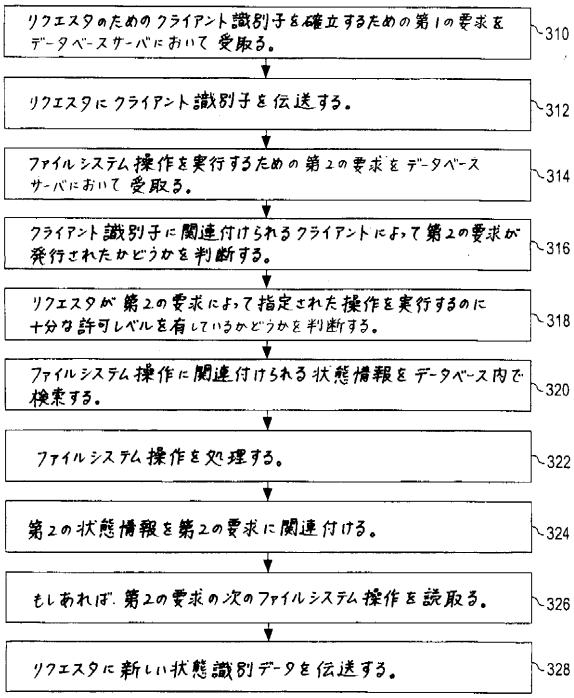



FIG. 2

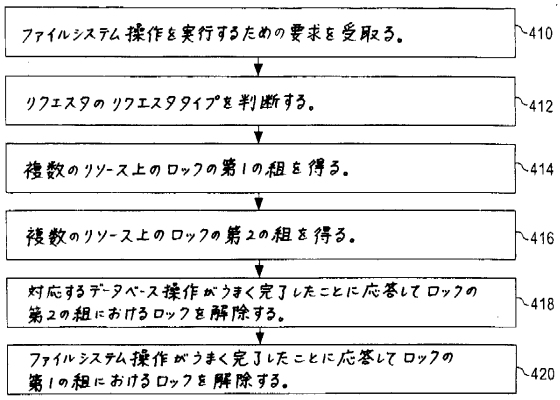
【図３】

FIG. 3



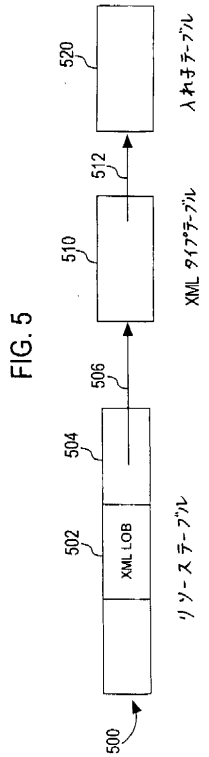

【図４】

FIG. 4



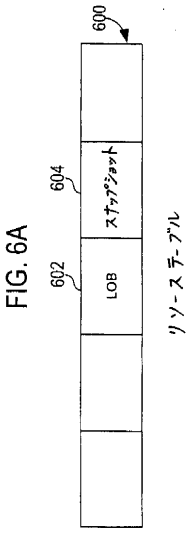

【図５】

FIG. 5



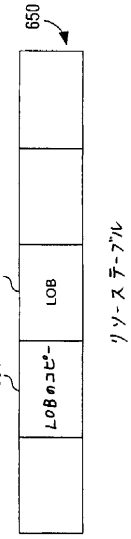

【図６Ａ】

FIG. 6A



【図６Ｂ】

FIG. 6B




【図７】

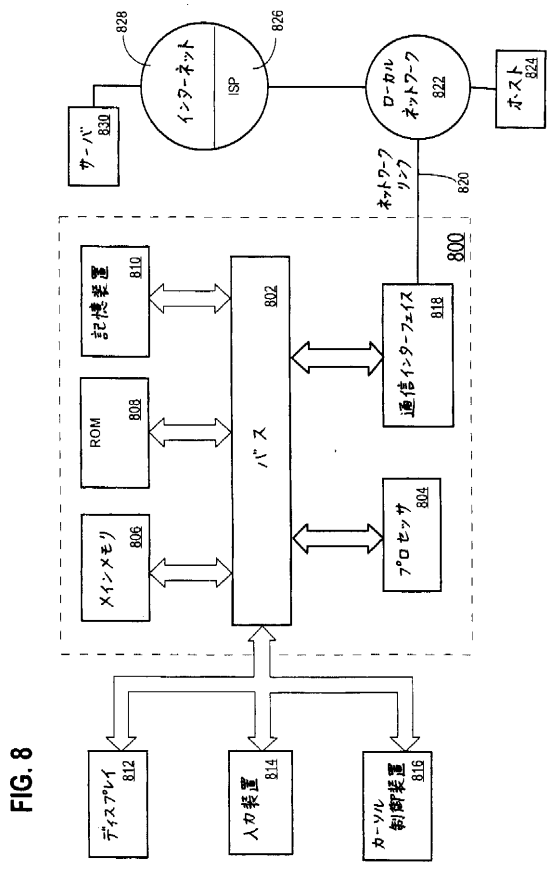
FIG. 7

さまざまなタイプのファイルシステム操作ロックの互換性

| | バイト－読取－書込 | バイト－書込 | 拒否－読取 | 拒否－書込 |
|---|---|---|---|---|
| バイト－読取－書込 | 矛盾する範囲がなければ | 矛盾する範囲がなければ | YES | YES |
| バイト－書込 | 矛盾する範囲がなければ | 矛盾する範囲がなければ | YES | YES |
| 拒否－読取 | YES | YES | YES | YES |
| 拒否－書込 | YES | YES | YES | YES |

【図８】

FIG. 8



フロントページの続き

(74)代理人 100098316
弁理士 野田 久登
(74)代理人 100109162
弁理士 酒井 將行
(74)代理人 100111246
弁理士 荒川 伸夫
(72)発明者 ジェーン，ナミット
アメリカ合衆国、９５０５４ カリフォルニア州、サンタ・クララ、ジアネラ・ストリート、２２３４
(72)発明者 アガルワル，ニプン
アメリカ合衆国、９５０５４ カリフォルニア州、サンタ・クララ、チーニー・ストリート、４７６８
(72)発明者 セドラー，エリック
アメリカ合衆国、９４１３１ カリフォルニア州、サンフランシスコ、シーザー・チャベス・ストリート、４２７０
(72)発明者 イディキュラ，サム
アメリカ合衆国、９５１１７ カリフォルニア州、サン・ノゼ、キーリー・ブルーバード、５５０、アパートメント・３８
(72)発明者 パンナラ，シャム
アメリカ合衆国、９４５３６ カリフォルニア州、フリーモント、ランチョ・アロヨ・パークウェイ、４０５、ナンバー・２８１

審査官 田川 泰宏

(56)参考文献 国際公開第０４／０９７６８０（WO，Ａ１）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 12/00