(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3759410号
(P3759410)
(45)発行日　平成18年3月22日(2006.3.22)　(24)登録日　平成18年1月13日(2006.1.13)

(51)Int.Cl.　F I
G06F 9/54 (2006.01)　G06F 9/46 480Z

請求項の数 11 (全 27 頁)

(21)出願番号	特願2000-562827(P2000-562827)
(86)(22)出願日	平成11年7月21日(1999.7.21)
(65)公表番号	特表2002-521765(P2002-521765A)
(43)公表日	平成14年7月16日(2002.7.16)
(86)国際出願番号	PCT/US1999/016540
(87)国際公開番号	WO2000/007101
(87)国際公開日	平成12年2月10日(2000.2.10)
審査請求日	平成13年1月25日(2001.1.25)
(31)優先権主張番号	09/127,167
(32)優先日	平成10年7月29日(1998.7.29)
(33)優先権主張国	米国(US)

(73)特許権者　591067923
ユニシス　コーポレイシヨン
ＵＮＩＳＹＳ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国　ペンシルバニア州　１９４２４　ブルーベル，　ピイー．オー．ボツクス　５００　タウンシツプ　ライン　アンド　ユニオン　ミーテイング　ローズ（番地なし）
(74)代理人　100064746
弁理士　深見　久郎
(74)代理人　100085132
弁理士　森田　俊雄
(74)代理人　100083703
弁理士　仲村　義平

最終頁に続く

(54)【発明の名称】クラスタ化された計算処理環境において実行する分散ネットワークアプリケーションの管理用リクエストを処理するための方法および装置

(57)【特許請求の範囲】

【請求項１】

定義されたマスタノードを介した集中管理を必要とする分散ネットワークアプリケーションが集中管理をサポートしないクラスタ化された計算処理環境のノード（１０、１２）上で実行することを可能にするための方法であって、前記方法は、

クラスタ化された計算処理環境からの管理用リクエストをその親ノードで受取るステップと、

親ノードが分散ネットワークアプリケーションのための指定されたマスタノードであるかどうかを決定するステップと、

もし親ノードが指定されたマスタノードでなければ親ノードから指定されたマスタノードへ管理用リクエストを経路制御するステップとを含む、方法。

【請求項２】

前記経路制御するステップは、

親ノードとマスタノードとの間に接続を与える名前付きパイプのインスタンスを生成するステップと、

名前付きパイプを介して親ノードからマスタノードへ管理用リクエストを送るステップとを含む、請求項１に記載の方法。

【請求項３】

名前付きパイプを介してマスタノードから親ノードへ管理用リクエストに対する応答を送り返すステップをさらに含む、請求項２に記載の方法。

【請求項４】
名前付きパイプを介してマスタノードで管理用リクエストを受取るステップと、
分散ネットワークアプリケーションの管理アプリケーションプログラミングインターフェイス（ＡＰＩ）を呼出して、指定されたマスタノードによるリクエストの処理を起動するステップとをさらに含む、請求項１に記載の方法。
【請求項５】
コンピュータ読出可能媒体上で実現されるサーバプログラム（４２）が記録されたコンピュータ読出可能媒体であって、サーバプログラムは、定義されたマスタノードを介した集中管理を必要とする分散ネットワークアプリケーションが集中管理をサポートしないクラスタ化された計算処理環境のノード（１０、１２）上で実行することを可能にするプログラムコードを含み、前記サーバプログラムのプログラムコードは、クラスタ化された計算処理環境の各ノード上で実行されると、前記ノードのいずれかに、以下のステップ：
クラスタ化された計算処理環境からの管理用リクエストをそのノードで受取るステップを含み、そのノードは親ノードを定義し、
親ノードが分散ネットワークアプリケーションのための指定されたマスタノードであるかどうかを決定するステップと、
もし親ノードが指定されたマスタノードでなければ親ノードから指定されたマスタノードへ管理用リクエストを経路制御するステップとを含むステップを実行させる、サーバプログラムが記録されたコンピュータ読出可能媒体。
【請求項６】
プログラムコードは、親ノードとマスタノードとの間に接続を与える名前付きパイプのインスタンスを生成し、名前付きパイプを介して親ノードからマスタノードへ管理用リクエストを送ることによって、ノードに前記経路制御するステップを実行させる、請求項５に記載のサーバプログラムが記録されたコンピュータ読出可能媒体。
【請求項７】
プログラムコードはさらに、マスタノードに、名前付きパイプを介してマスタノードから親ノードへ管理用リクエストに対する応答を送り返させる、請求項６に記載のサーバプログラムが記録されたコンピュータ読出可能媒体。
【請求項８】
定義されたマスタノードを介した集中管理を必要とする分散ネットワークアプリケーションが集中管理をサポートしないクラスタ化された計算処理環境のノード（１０、１２）上で実行することを可能にするサーバプログラムが記録されたコンピュータ読出可能媒体であって、サーバプログラムは、各ノード上でクラスタ化された計算処理環境からの管理用リクエストをそのノードで傍受し、もし親ノードが指定されたマスタノードでなければ親ノードから指定されたマスタノードへ管理用リクエストを経路制御する、サーバプログラムが記録されたコンピュータ読出可能媒体。
【請求項９】
サーバプログラムは、前記管理用リクエストを経路制御する前に親ノードが分散ネットワークアプリケーションのための指定されたマスタノードであるかどうかを決定する、請求項８に記載のサーバプログラムが記録されたコンピュータ読出可能媒体。
【請求項１０】
サーバプログラムは、親ノードとマスタノードとの間に接続を与える名前付きパイプのインスタンスを生成し、次に名前付きパイプを介して親ノードからマスタノードへ管理用リクエストを送ることによって管理用リクエストを指定されたマスタに経路制御する、請求項９に記載のサーバプログラムが記録されたコンピュータ読出可能媒体。
【請求項１１】
サーバプログラムはさらに、名前付きパイプを介してマスタノードから親ノードへ管理用リクエストに対する応答を送り返す、請求項８に記載のサーバプログラムが記録されたコンピュータ読出可能媒体。
【発明の詳細な説明】

【０００１】
【著作権表示】

【０００２】
【背景】
発明分野
本発明は、クラスタ化された計算処理環境における分散ネットワークアプリケーションに向けられ、より特定的には、そのような環境における分散アプリケーションの管理用リクエストを処理するための方法および装置に向けられる。
【０００３】
先行技術の説明
分散ネットワークアプリケーションは、しばしば、分散アプリケーション全体の管理を行なうマスタマシンの概念を有する。この場合には、管理は、構成要素をオンラインにしたり、構成要素をオフラインにしたり、または個々の構成要素のステータスをチェックしたりすることを含む。複数のコンピュータがクラスタ内であわせて動作して全体の可用性を向上させることを可能にするソフトウェアアーキテクチャには、これらの分散ネットワークアプリケーションによって必要とされる集中管理の種類をサポートしないものがある。そのようなクラスタ化された計算処理環境の一例は、Ghormley, D.P.らの「ＧＬＵＮＩＸ：ワークステーションのネットワークのためのグローバル層Ｕｎｉｘ（“A global layer Unix for a Network of Workstations”）」Software Practice & Experience, GB, John Wiley & Sons Ltd. Chichester, Vol.28 no. 9, 929-961頁, XP000765510 ISSN:0038-0644に記載されている。しかしながら、Microsoft（登録商標）Cluster Server（ＭＳＣＳ）のアーキテクチャなどの、他のクラスタ化された計算処理環境は、そのような分散ネットワークアプリケーションの集中管理モデルに適さない。たとえば、ＭＳＣＳでは、クラスタの各ノード上のカスタムリソースＤＬＬが、構成要素をオンラインにしたり、構成要素をオフラインにしたり、または構成要素のステータスをチェックしたりするなどの、そのノードのための管理操作を実現しなければならない。したがって、管理操作（たとえば、オフライン、オンラインなど）が所与のノード上で行なわれるとき、ＭＳＣＳは、そのノード上のリソースＤＬＬを呼出して操作を行なう。これは、そのような操作のすべてがクラスタ内の単一のマスタノードによって起動および制御される必要がある、すなわち、たとえば、ノードがオンラインまたはオフラインにされるべきならば、その操作は分散アプリケーションのためのマスタノードによって起動および実行されるであろう多くの分散ネットワークの集中管理モデルに反するものである。より具体的な例として、たとえば、BEA System, Inc.のTuxedo（登録商標）トランザクションマネージャおよびメッセージングミドルウェアのために書かれた分散アプリケーションが、クラスタ化された計算処理環境においてMicrosoftのCluster Server（ＭＳＣＳ）ソフトウェアの制御下で実行されるとき、この問題が生じる。
【０００４】
Ａ．Tuxedo
Tuxedo環境では、１つ以上の論理的マシンをあわせてグループ化してタキシードドメインを規定することができる。１個の論理的マシンが、タキシードサーバマシンを表わし、しばしば物理的コンピュータシステム（すなわち、ネットワークの１つのノード）に対応するが、いつもそうとは限らない。いくつかの場合、２個以上の論理的マシンが単一ノード上に定義可能である。
【０００５】
各論理的マシンは、ＢＢＬプロセス、ブリッジプロセス、およびおそらくＤＢＢＬプロセスを含む、タキシードアドミンサーバを含む。各論理的マシンはまた、タキシードＴリッスンプロセスを有する。ＢＢＬプロセスは、ユーザ書込のサーバを監視し制御する。ある

論理的マシン上のブリッジプロセスは、Ｔリッスンプロセスを別の論理的マシンに接続する。ドメイン内の論理的マシンの１つがマスタとして指定される；マスタにはバックアップが構成されてもよい。ＤＢＢＬは、マスタとして動作する論理的マシン上で実行される。ＤＢＢＬは、さまざまなＢＢＬを調整し制御する、すなわち、それは、構成要素をオンラインにしたり、構成要素をオフラインにしたり、または個々の構成要素のステータスをチェックしたりすることを含む、ドメインのための管理を行なう。

【０００６】

論理的マシン上のユーザ書込のサーバは、１つ以上のタキシードサーバグループにグループ化される。タキシードサーバグループがトランザクションをサポートする場合、トランザクションの状態は、ＴＬＯＧにログされる。各論理的マシンは、それ自身の独自のＴＬＯＧを有する。他の状態および構成情報は、バイナリＴＵＸＣＯＮＦＩＧファイルに記憶される。各論理的マシンは、それ自身のＴＵＸＣＯＮＦＩＧのコピーを有するが、ＤＢＢＬはさまざまなＴＵＸＣＯＮＦＩＧファイルの同期をとる。

【０００７】

Ｂ．Microsoft Cluster Server（ＭＳＣＳ）

Microsoft（登録商標）Windows NT（登録商標）,Enterprise Editionの現在のリリースでは、クラスタは、可用性の高いシステムとして動作する２つの接続されたシステム（ふつうノードと呼ばれる）のことをいう。これらの２つのシステムは、共有ＳＣＳＩバス上に１つ以上の物理的ディスクドライブを有する；共有ＳＣＳＩバス上の各ディスクはいずれのシステムによってもアクセス可能であるが、同時には１つのシステムによってのみ可能である。クラスタ内のシステムの１つに障害が発生すると、他のシステムが共有ディスクの制御を獲得し、障害が発生したシステムが止めたところをひろってサービスを与え続ける。Windows NTの将来のリリースは、３つ以上のノードでクラスタをサポートするものと見込まれる。

【０００８】

ＭＳＣＳは、クラスタの各ノード上で実行するクラスタサービスによって制御される。クラスタサービスは、１つ以上のリソースモニタを生じる。リソースモニタは、１つ以上のリソースを監視する。リソースとは、ＭＳＣＳによって監視および制御可能であるエンティティである。リソースモニタは、リソースＤＬＬ内のエントリポイント（ＭＳＣＳリソースＡＰＩ内に定義される）を呼出して、特定のリソースを監視および制御する。特に、リソースモニタは、適切なときにリソースＤＬＬ内のエントリポイントを呼出して、リソースの状態をチェックしたり、リソースをオンラインにしたり、またはリソースをオフラインにする。したがって、リソースＤＬＬは、リソースをオンラインにしたりリソースをオフラインにしたりするのに必要とされる動作－Tuxedo環境においてはタキシードマスタによって行なわれなければならない管理操作を実現することに注目されたい。あるノードから別のノードへリソースを移すことは、第１のノード上でリソースをオフラインにし、第２のノード上でリソースをオンラインにすることによっておこる。

【０００９】

ＭＳＣＳは、いくつかのリソースタイプを定義するリソースＤＬＬ（clusres.dll）を含む。以下は、clusres.dllによってサポートされる関連のリソースタイプのリストである：

物理的ディスク－共有ＳＣＳＩバス上に位置する物理的ディスクドライブを制御する。

【００１０】

ＩＰアドレス－あるノードまたは他のノードに動的に割当可能であるＩＰアドレスを定義する。

【００１１】

ネットワーク名－ＩＰアドレスリソースのための記号名を定義する。

汎用サービス－いかなるWindows NTサービスをも制御する。

【００１２】

汎用アプリケーション－正しく振舞うWindows NTコンソールアプリケーションプログラム

を制御する。

【００１３】

加えて、第三者の開発者が、リソースＡＰＩに従うリソースＤＬＬを開発および登録することによってカスタムリソースタイプを生成することができる。

【００１４】

ＭＳＣＳリソースは、あわせてリソースグループにグループ化可能である。リソースグループは、フェールオーバの基本ユニットである；すなわち、所与の時点では特定のグループ内のあらゆるリソースが同じノード上で実行する。リソースグループ内のリソースは、オンライン操作およびオフライン操作の相対順序を制御する依存性を有し得る。

【００１５】

典型的には、リソースグループは、１つ以上の物理的ディスクリソースと、ＩＰアドレスリソースと、ネットワーク名リソースと、汎用サービスリソース、汎用アプリケーションリソース、および／またはカスタムリソースタイプなどの、サーバアプリケーションを表わす１つ以上の付加的なリソースとを含む。それ自身のＩＰアドレスリソースおよびネットワーク名リソースを有するリソースグループは、仮想サーバとして知られている。

【００１６】

外部のクライアントには、１個の仮想サーバが独自のサーバコンピュータとしてＴＣＰ／ＩＰクライアント／サーバタイプのアプリケーションを実行しているように見える。実際には、ＭＳＣＳクラスタの単一のノード上で実行するいくつかの仮想サーバがあり、各々が異なったＩＰアドレスを有するだろう。さらに、仮想サーバは、ＭＳＣＳクラスタのあるノードから別のノードへ移ることができ、これはクライアントには見えない（ただし、一時的割込またはサービスの減速がある）。

【００１７】

図１は、各々がそれぞれの仮想サーバ１４、１６を実行する２つのノード１０、１２を含む例示的なクラスタを示す。各ノード上で実行するクラスタサービス１８は、たとえば、各仮想サーバ１４、１６のリソースモニタ（図示せず）への管理用リクエストをイニシエートすることを含め、クラスタを制御する。各ノード１０、１２は、たとえば、ＳＣＳＩバスなどの共有バス２４を介して複数個のディスク２０に接続される。各ノード１０、１２はまた、それぞれのネットワークインターフェイスを介してローカルエリアネットワーク（ＬＡＮ）２２に接続される。

【００１８】

ここで図２を参照して、クラスタの第２のノード１２に障害が発生した場合、ＭＳＣＳは、障害が発生した仮想サーバ１６をクラスタの第１のノード１０上で起動する。これがフェールオーバである。この時点で、仮想サーバ１４、１６はどちらも、第１のノード上で実行している。クライアントプログラムは引続き正常に実行する（ただし、性能が落ちる可能性がある）。

【００１９】

クラスタの第２のノード１２が正常動作を再開すると、ＭＳＣＳは、フェールオーバした仮想サーバ１６を第１のノード１０上でオフラインにする。次に、ＭＳＣＳは、この仮想サーバ１６を第２のノード１２上で再びオンラインにする。これがフェールバックである。この時点で、図１に示される構成が再開し、すなわち各ノードは１個の仮想サーバ上で実行している。

【００２０】

Ｃ．クラスタ化された環境における分散ネットワークアプリケーションの配備

Tuxedo環境において実行するよう設計されたアプリケーションなどの、分散ネットワークアプリケーションは、ＭＳＣＳなどのクラスタ化された計算処理環境内で実行するよう設定可能である。たとえば、図１を参照して、２つの論理的マシンを備えるタキシードドメインは、２つのＭＳＣＳ仮想サーバ１４、１６を備えるよう構成可能である。正常動作中は、１個の仮想サーバ（およびしたがって１個の論理的マシン）がクラスタのノード１０、１２の各々上で実行する。しかしながら、前に例示したように、管理用リクエストがTu

xedo環境とＭＳＣＳ環境とでは異なって処理されるという点で問題が生じる。Tuxedoアーキテクチャでは、構成要素をオンラインにしたりオフラインにしたりすることを含む、管理用リクエストは、指定されたマスタノードによって行なわれる。反対に、ＭＳＣＳ環境では、管理用リクエストは、指定されたマスタノードによってではなく、操作が向けられるノード上のリソースＤＬＬによって行なわれる。本発明は、この課題に対する解決を提供する。

【００２１】

【発明の概要】

本発明は、アプリケーションがクラスタ化により達成される増大した可用性を利用できるように、マスタノードを介した集中管理を必要とする分散ネットワークアプリケーション（たとえば、Tuxedoベースのアプリケーション）がそのような集中管理をサポートしないクラスタ化された環境（たとえば、ＭＳＣＳ）内に配備されることを可能にするための方法および装置に向けられる。より特定的には、本発明の方法および装置は、クラスタ化された計算処理環境内で実行する分散ネットワークアプリケーションの管理用リクエスト（たとえばオンライン、オフラインなど）を、リクエストが発するノードから、そのクラスタ内の分散ネットワークアプリケーションのためのマスタとして動作しているノードへ経路制御する。たとえば、ある具体的な実施例では、この発明は、ＭＳＣＳリソースＤＬＬからＭＳＣＳで制御されたクラスタ内で実行するTuxedoアプリケーションのマスタマシンへTuxedoベースの管理用リクエストを経路制御する。

【００２２】

好ましい実施例では、管理用リクエストの経路制御は、クラスタの各ノード上にインストールされる、パイプサーバと以下呼ばれる複数個のサーバプログラムによって達成される。所与のノードのパイプサーバは、名前付きパイプのインスタンスを生成し、これを介して、他のノード上のパイプサーバを含む、クライアントプログラムは、管理用リクエストを含むメッセージをそのノードに送ることができる。パイプサーバは、受取ったメッセージを、分散ネットワークアプリケーションの管理ＡＰＩに転送する。

【００２３】

本発明のさらなる特徴および利点は、以下に明らかとなるであろう。

前述の概要、および好ましい実施例の以下の詳細な説明は、添付の図面と関連付けて読まれるとよりよく理解される。この発明を例示する目的のために、現在のところ好ましい実施例が図面に示されるが、この発明は開示される特定の方法および手段に限られるものでないことが理解される。

【００２４】

【好ましい実施例の詳細な説明】

Ｉ．概観

本発明は、アプリケーションがクラスタ化により達成される増大した可用性を利用できるように、マスタノードを介した集中管理を必要とする分散ネットワークアプリケーションがそのような集中管理をサポートしないクラスタ化された環境内に配備されることを可能にするための方法および装置に向けられる。より特定的には、本発明の方法および装置は、クラスタ化された計算処理環境内で実行する分散ネットワークアプリケーションの管理用リクエスト（たとえば、オンライン、オフラインなど）を、リクエストが発するノードから、そのクラスタ内の分散ネットワークアプリケーションのためのマスタとして動作するノードへ経路制御する。本発明は、フロッピィディスク、ＣＤ－ＲＯＭ、ハードドライブ、またはいかなる他の機械で読出可能な記憶媒体などの有形の媒体に実現されるプログラムコード（すなわち、命令）の形をとってもよく、プログラムコードがコンピュータなどのマシンによってロードされ実行されると、マシンがこの発明を実施するための装置になる。本発明はまた、電気的配線もしくはケーブルによって、光ファイバを介して、または伝送のいかなる他の形によるなどの、何らかの伝送媒体によって伝送されるプログラムコードの形で実施可能であり、プログラムコードが、コンピュータなどのマシンによって受取られロードされ実行されると、マシンがこの発明を実施するための装置になる。汎用

プロセッサ上で実現されると、プログラムコードは、プロセッサと組合さって、特定の論理回路と同じように動作する独特の装置を提供する。

【００２５】

本発明に従うと、アプリケーションがマスタノードを介した集中管理を必要とするような、複数個のノードを含むクラスタ化された計算処理環境内で実行する分散ネットワークアプリケーションの管理用リクエストは、リクエストが発するノードから、分散ネットワークアプリケーションのためのマスタとして動作するノードへ経路制御される。次に、リクエストの成功または失敗を示す応答が、マスタノードから、リクエストを発したノードまで送り返される。管理用リクエストの経路制御は、クラスタの各ノード上にインストールされる、パイプサーバと以下に呼ばれる複数個のサーバプログラムによって達成される。所与のノードのパイプサーバは、名前付きパイプのインスタンスを生成し、これを介して、他のノード上のパイプサーバを含む、クライアントプログラムは、管理用リクエストを含むメッセージをそのノードに送ることができる。パイプサーバは、受取ったメッセージを、分散ネットワークアプリケーションの管理ＡＰＩに転送する。

【００２６】

図３Ａおよび図３Ｂは、本発明の方法の好ましい実施例を例示する流れ図を含む。図示のとおり、ステップ６０で、クラスタ化された環境は、管理用リクエスト（たとえば、分散ネットワークアプリケーションの構成要素をオンラインにするか、またはこれをオフラインにするためのリクエスト）を親ノードで発行する。次に、ステップ６４で、分散ネットワークアプリケーションの構成要素は、親ノード上で実行するパイプサーバに名前付きパイプを開き、ステップ６６で、名前付きパイプを介してパイプサーバにリクエストメッセージを送る。

【００２７】

ステップ６８で、親ノード上のパイプサーバは、親ノードが分散ネットワークアプリケーションのマスタノードとして指定されるかどうかを決定する。もしそうならば、ステップ７０で、リクエストはローカルに処理される。

【００２８】

ステップ６８で、親ノードが指定されたマスタノードでないことが決定された場合、ステップ７２で、パイプサーバは、そのマスタノードがどのマスタノードであるかを決定する。次に、ステップ７４で、親ノード上のパイプサーバは、指定されたマスタノード上のパイプサーバに名前付きパイプを開く。ステップ７６で、リクエストメッセージは、名前付きパイプを介してマスタノード上のパイプサーバに転送される。

【００２９】

ステップ７８で、マスタノード上のパイプサーバは、これへの名前付きパイプ接続を介してメッセージを受取る。ステップ８０で、マスタノード上のパイプサーバは、分散ネットワークアプリケーションの管理ＡＰＩを呼出して、メッセージ内に含まれるリクエストの処理を開始する。ステップ８２（図３Ｂ）で、マスタノード上のパイプサーバは、分散ネットワークアプリケーションからリクエスト処理の結果（たとえば、成功または失敗）を獲得し、ステップ８４で、同じ名前付きパイプを介して親ノードのパイプサーバに応答を送り返す。ステップ８８で、親ノード上のパイプサーバは、親ノード上のクラスタ環境に応答を転送し、処理を完了する。

【００３０】

II．例示的アプリケーション

図４は、本発明の方法を実施するための装置の好ましい実施例と、（集中管理をサポートしない）ＭＳＣＳによって制御されかつ複数個のノードを含むクラスタ化された計算処理環境内に配備される（マスタノードを介した集中管理を必要とする）Tuxedo環境のために書かれた分散ネットワークアプリケーションを含む環境における本発明の例示的アプリケーションとの両方を例示するブロック図である。以下により詳細に記載されるように、タキシードアプリケーションの管理用リクエストは、リクエストが発するノードのリソースＤＬＬからタキシードアプリケーションのためのマスタとして動作するノードへ経路制御

される。しかしながら、本発明は、Tuxedoベースのアプリケーションおよび／またはＭＳＣＳ制御のコンピュータクラスタでの使用に限られるものでなく、分散ネットワークアプリケーションがマスタノードを介した集中管理を必要とするが、アプリケーションが配備されるクラスタ化された環境がそのような集中管理をサポートしないようないかなるシステムにおいても実施可能である。

【００３１】

以下に記載される実施例に従うと、Tuxedoベースの分散ネットワークアプリケーションの各論理的マシンはそれぞれのＭＳＣＳ仮想サーバに対応付けられている。この例では、各論理的マシンは、Oracle（登録商標）データベースを制御する。さらなる背景として、Oracle Corporationから入手可能なデータベースソフトウェアに含まれる無料のアドオンプロダクトーOracle（登録商標）FailSafeーが、オラクルデータベースと呼ばれるカスタムリソースタイプを定義するリソースＤＬＬを提供する。Oracle Corporationは、Oracle FailSafe Managerと呼ばれる管理用ツールも提供して、ＭＳＣＳ環境のためにOracleリソースを生成および管理する。

【００３２】

図４は、例示の構成の各仮想サーバ／論理的マシンの機能構成要素を例示する。図示のとおり、各論理的マシンごとに、以下のリソースを含むＭＳＣＳリソースグループが生成される：

物理的ディスク３０－共有ＳＣＳＩバス（図示せず）上の１つ以上の物理的ディスク３０－これらのディスクは、タキシード論理的マシンによって使用されるＴＬＯＧ、ＴＵＸＣＯＮＦＩＧ、およびオラクルデータベースを含む；

オラクルリソース３２－タキシード論理的マシンによって使用されるオラクルデータベースのOracle FailSafe Managerによって生成されるリソース；

タキシードＴリッスン３４；

タキシードアドミンサーバ３６；および

タキシードサーバグループ３８－タキシード構成のＧＲＯＵＰＳセクション内にこの論理的マシンのために構成される各サーバグループごとに１つのタキシードサーバグループリソースインスタンスがある。

【００３３】

各論理的マシンについてのリソースグループは、ローカルエリアネットワーク（図示せず）上のＩＰアドレスと、タキシード構成のＭＡＣＨＩＮＥＳセクション内で用いられるマシン名と一致するネットワーク名と、ＮＥＴＷＯＲＫセクションにおいて指定されるＮＬＳＡＤＤＲ値およびＮＡＤＤＲ値とを有する。本発明に従うと、各リソースグループはさらに、リソースＤＬＬ４０とパイプサーバ４２とを含み、これらはあわせて本発明の方法を実施するための装置を提供する。以下に記載されるように、リソースＤＬＬ４０は、クラスタ環境とそのノード上のタキシードアプリケーションの他の構成要素との間にインターフェイスを与えるタキシード分散ネットワークアプリケーションの構成要素を表わす。

【００３４】

この構成では、たとえば２個の論理的マシンを備えるタキシードドメインは、２個のＭＳＣＳ仮想サーバを備えるよう構成される。正常動作中は、１個の仮想サーバ（およびしたがって１個の論理的マシン）がクラスタのノードの各々上で実行する。クラスタの第１のノードに障害が発生した場合、ＭＳＣＳは、障害が発生した仮想サーバ（および対応する論理的マシン）をクラスタの第２のノード上で起動するーすなわちフェールオーバ状況である。両方の仮想サーバ（および両方の論理的マシン）が次に、第２のノード上で実行する。タキシードクライアントプログラムは引続き正常に実行する（ただし、性能が落ちる可能性がある）。クラスタの第１のノードが正常動作を再開すると、ＭＳＣＳは、第２のノード上で、フェールオーバした仮想サーバ（および対応する論理的マシン）をオフラインにする。次にＭＳＣＳは、第１のノード上でこの仮想サーバ（および対応する論理的マシン）をオンラインにする。これがフェールバック状況である。フェールバックの後、各ノードは再び、１個の仮想サーバ（およびしたがって１個の論理的マシン）を実行する。

以下に記載するとおり、本発明の方法は、これおよび他のシナリオに関連する管理用リクエストが、親ノードからタキシードドメインのためのマスタとして動作するノードへ経路制御されることを可能にする。

【００３５】

Ａ．リソースＤＬＬ４０

リソースＤＬＬ４０は、以下のリソースタイプ：タキシードサーバグループ、タキシードアドミンサーバ、パイプサーバ、およびタキシードＴリッスンをサポートする。各リソースタイプは、リソースＤＬＬ４０内のエントリポイント関数４４、４６、４８および５０のそれぞれの組によってサポートされる。タキシードサーバグループ・リソースタイプ４４は、タキシードサーバグループ３８の操作を監視しかつ制御するために用いられる。タキシードアドミンサーバ・リソースタイプは、タキシードアドミンサーバ３６の操作を監視しかつ制御するために用いられる。パイプサーバ・リソースタイプは、パイプサーバプロセス４２の操作を監視しかつ制御するために用いられる。タキシードＴリッスン・リソースタイプは、タキシードＴリッスンプロセス３４の操作を監視しかつ制御するために用いられる。

【００３６】

リソースＤＬＬは、リソースＤＬＬによってサポートされる各リソースタイプ（たとえば、リソースタイプ４４、４６、４８および５０）について実現されなければならないエントリポイント関数を特定するマイクロソフトリソースＡＰＩの仕様に従う。リソースモニタ２６は、これらのエントリポイント関数を呼出して、特定のリソース上の所与の関数を実行する。リソースＡＰＩはまた、リソースモニタ２６によって実現されリソースＤＬＬから呼出されるコールバック関数を特定する。リソースＤＬＬによって実現されなければならないエントリポイント関数のうち、２つが、タキシードアプリケーションなどの分散ネットワークアプリケーションがマスタノードによって集中的に実行されることを必要とし得る管理用リクエストのタイプの例である、すなわち、（１）オンライン、および（２）オフラインである。各リソースタイプごとに１個のオンライン関数と１個のオフライン関数とがある。オンラインの目的は、リソースの特定のインスタンスをオンラインにすることであり、オフラインの目的は、リソースの特定のインスタンスをオフラインにすることである。この発明に特に関連するのは、タキシードアドミンサーバ・リソースタイプ４６のオンライン関数とオフライン関数とである。これらの関数は、タキシードアドミンＡＰＩ内に対応する関数を有するが、これはタキシードアプリケーションの指定されたマスタによって実行されなければならない。したがって、ＭＳＣＳ環境のリソースＡＰＩは、リソースＤＬＬがエントリポイント関数を与えてリクエストが発するノードでオンラインリクエストおよびオフラインリクエストを処理することを必要とするが、タキシードアドミンＡＰＩは、そのようなリクエストが指定されたマスタノードによって集中的に実行されることを必要とする。これは、本発明が解決を提供する課題の特定的な例の１つである。

【００３７】

この発明に従うと、リソースＤＬＬ４０は、タキシードリソースを制御するためにタキシードアドミンＡＰＩと直接インターフェイスしない。代わりに、リソースＤＬＬは、パイプサーバＡＰＩ（以下に記載される）を用いて、リクエストをパイプサーバ４２に送る。パイプサーバ４２は次に、タキシードアドミンＡＰＩを用いてタキシードリソースを監視および制御する。したがって、複数個のノードを含むが集中管理をサポートしないクラスタ化された計算処理環境において、パイプサーバ４２は、分散ネットワークアプリケーションの管理用リクエストを、リクエストが発するノードからそのアプリケーションのためのマスタとして指定されるノードへ経路制御するための手段を提供する。

【００３８】

リソースＤＬＬ４０が同じノード上のタキシードアドミンＡＰＩと直接インターフェイスせず、代わりにそのノード上のパイプサーバ４２とインターフェイスする理由は、リソースＤＬＬ４０はネイティブクライアントとして動作不可能であるがパイプサーバ４２は可

能であるからである。タキシードアドミンＡＰＩコールを実行するクライアントプログラムは、ネイティブタキシードクライアントでなければならない。ネイティブクライアントとは、タキシード構成のＭＡＣＨＩＮＥＳセクション内に特定されるマシンの１つの上で実行するクライアントである。リソースＤＬＬ４０は、リソースモニタ（たとえば、リソースモニタ２６）のプロセス下で実行し、（ＷＩＮ３２ＡＰＩ関数GetComputerNameによって戻される）そのコンピュータ名は、それが実行しているクラスタノードの名前と一致する。しかしながら、タキシード構成は、対応する仮想サーバのネットワーク名をマシン名として用いる。したがって、リソースＤＬＬは、ネイティブクライアントとして動作できない。反対に、パイプサーバ４２は、GetComputerNameが呼出されたとき（クラスタノード名の代わりに）ネットワーク名が戻されるようにするＭＳＣＳ特徴で実行する。したがって、パイプサーバ４２は、ネイティブクライアントとして動作することができる。

【００３９】

Ｂ．パイプサーバ４２

本発明の好ましい実施例に従うと、名前付きパイプ機能を用いて、クライアントプロセス（たとえば、リソースＤＬＬ４０）とパイプサーバ４２との間、および／または２つの異なったノード上のパイプサーバ４２間でメッセージをやり取りする。したがって、名前付きパイプを用いて、同じコンピュータ上のプロセス間、またはネットワーク上の異なったコンピュータ上のプロセス間で双方向の通信が可能になる。この実施例では、Microsoft Windows NTオペレーティングシステムの名前付きパイプ機能が採用される。

【００４０】

パイプサーバ４２は、管理用リクエストを含むメッセージを名前付きパイプを介してプロセス間でやり取りするためのアプリケーション・プログラム・インターフェイス（ＡＰＩ）を与える。パイプサーバＡＰＩは、交換されるべきメッセージのフォーマットおよび、交換を行なうための方法を特定する。

【００４１】

１．パイプサーバＡＰＩ－メッセージタイプ

メッセージは、３２ビットのリクエストコードを含み、特定のリクエストメッセージの内容が後に続く。リクエストコードは、リクエストのタイプを特定する。オンラインリクエストおよびオフラインリクエストを処理するために、パイプサーバＡＰＩは、以下のリクエストコードを定義する：

【００４２】

【数１】

```
#define PIPE_REQ_MIB              1
#define PIPE_REQ_ADM_ONLINE       2
#define PIPE_REQ_ADM_OFFLINE      3
```

【００４３】

ＰＩＰＥ＿ＲＥＱ＿ＭＩＢは、マネジメント・インフォメーション・ベース（ＭＩＢ）にアクセスするタキシードアドミンリクエストの一般的な形を表わすメッセージタイプである。ＭＩＢは、タキシードのプログラムされた管理を行なうためにアクセスされる。リクエストコードがＰＩＰＥ＿ＲＥＱ＿ＭＩＢであれば、リクエストコードワードのすぐ後にタキシードＦＭＬ３２バッファが続く。ＦＭＬ３２は、フィールド化されたバッファのタイプである。フィールド化されたバッファは、フィールドと呼ばれる属性値対を含む。属性はフィールドの識別子であり、関連付けられた値は、フィールドのデータ内容を表わす。フィールド化されたバッファと関連付けられるデータのすべては独立している（すなわち、フィールド化されたバッファの外のデータへのポインタがない）。これは、フィールド化されたバッファが、独自のプロセス間で、異なったコンピュータ上で実行するプロセス間でさえ、伝送可能であることを意味する。タキシードは、ＦＭＬ３２バッファにデー

タを割当て、組立て、加算し、かつそこからデータを抽出するのに必要とされる関数ライブラリを提供する。タキシードアドミンＡＰＩは、ＭＩＢアクセスリクエストを実行するために必要とされるＦＭＬ３２リクエストバッファの特定のフォーマットを定義する。この実施例では、ＰＩＰＥ＿ＲＥＱ＿ＭＩＢリクエストタイプは、“ＧＥＴ”または“ＳＥＴ”のＴＡ＿ＯＰＥＲＡＴＩＯＮで、いかなる有効なＭＩＢアクセスリクエストをもサポートする。“ＧＥＴＮＥＸＴ”操作は、この実施例ではサポートされないが、他の実施例で可能である。

【００４４】

ＰＩＰＥ＿ＲＥＱ＿ＡＤＭ＿ＯＮＬＩＮＥおよびＰＩＰＥ＿ＲＥＱ＿ＡＤＭ＿ＯＦＦＬＩＮＥなどの、ＰＩＰＥ＿ＲＥＱ＿ＭＩＢ以外のすべてのリクエストコードについて、リクエストメッセージは、リクエストコードおよび、リクエストコードによって特定されるリクエストの特定のタイプに関連するいかなる他の情報をも含むデータ構造を含む。たとえば、以下のＣ言語の型定義が採用可能である：

【００４５】

【数２】

```
typedef struct _pipe_request_t {
      DWORD   requestCode;
      DWORD   packetRevision;
      DWORD   cmdInputSize;
      BOOL    returnCmdOutput;
      char    cmd[MAX_TUX_CMD_LEN+1];
      char    opts[MAX_TUX_OPTS_LEN+1];
      char    groupName[MAX_GROUPNAME_LEN+1];
} PIPEREQ;
typedef PIPEREQ         *LPPIPEREQ;
typedef const PIPEREQ   *LPCPIPEREQ;
```

【００４６】

requestCodeフィールドは、送られるリクエストメッセージのタイプを特定する特定のリクエストコードを含む。この実施例では、packetRevisionフィールドは、将来の使用のために確保され、単に０にセットされる。残りのフィールドは、この発明の一部ではないパイプサーバ４２の他の機能に関連する。

【００４７】

ＰＩＰＥ＿ＲＥＱ＿ＡＤＭ＿ＯＮＬＩＮＥのリクエストコードは、ＭＳＣＳのために、（パイプサーバが実行している仮想サーバに対応する）タキシードアドミンサーバをオンラインにするリクエストを表わす。この場合には、リクエスト構造内のすべての他のフィールドは、０にセットされる。ＰＩＰＥ＿ＲＥＱ＿ＡＤＭ＿ＯＦＦＬＩＮＥのリクエストコードは、ＭＳＣＳのために、（パイプサーバが実行している仮想サーバに対応する）タキシードアドミンサーバをオフラインにするリクエストを表わす。この場合もやはり、リクエスト構造内のすべての他のフィールドは、０にセットされる。

【００４８】

パイプサーバＡＰＩはまた、特定のリクエストの結果を表示するのに用いられる応答メッセージのフォーマットを特定する。リクエストコードがＰＩＰＥ＿ＲＥＱ＿ＭＩＢであった場合、応答メッセージはタキシードＦＭＬ３２バッファである。タキシードアドミンＡＰＩは、ＭＩＢアクセスリクエストを実行したとき戻されるＦＭＬ３２応答バッファの特定のフォーマットを定義する。ＰＩＰＥ＿ＲＥＱ＿ＭＩＢ以外のすべてのリクエストコードについて、応答メッセージフォーマットは、以下のＣの型定義によって定義可能である：

【００４９】

【数３】

```
typedef struct _pipe_reply_t {
      DWORD   packetRevision;
      DWORD   cmdOutputSize;
      int     status;                          // See statuses below:
#define PIPE_REP_GOOD_STATUS            0     // Success
#define PIPE_REP_ERROR_STATUS          -1     // General error
#define PIPE_REP_MASTER_UNAVAILABLE    -2     // Tuxedo Master was
                                              // unavailable, so
                                              // unable to accurately
                                              // determine
                                              // looks alive status.
#define PIPE_REP_NO_RESPONSE           -3     // No reply was received
due to
                                              // a broken pipe,
                                              timeout, etc.
} PIPEREP;
typedef PIPEREP         *LPPIPEREP;
typedef const PIPEREP   *LPCPIPEREP;
```

【００５０】

packetRevisionフィールドは、起こり得る将来の使用のために確保され、この実施例では常に０にセットされる。cmdOutputSizeフィールドは、この発明の一部ではないパイプサーバ４２の特徴に関する。statusフィールドは、リクエストの結果を示す値を保持する。statusフィールドは、パイプ切断エラーまたはタイムアウトのためにパイプサーバから応答が戻されない場合、ＰＩＰＥ＿ＲＥＰ＿ＮＯ＿ＲＥＳＰＯＮＳＥにセットされる。ＰＩＰＥ＿ＲＥＱ＿ＡＤＭ＿ＯＮＬＩＮＥリクエストに対して、statusフィールドは、タキシードアドミンサーバがうまくオンラインにされた場合にはＰＩＰＥ＿ＲＥＰ＿ＧＯＯＤ＿ＳＴＡＴＵＳにセットされ、または失敗の場合にはＰＩＰＥ＿ＲＥＰ＿ＥＲＲＯＲ＿ＳＴＡＴＵＳにセットされる。ＰＩＰＥ＿ＲＥＱ＿ＡＤＭ＿ＯＦＦＬＩＮＥリクエストに対して、statusフィールドは、タキシードアドミンサーバがうまくオフラインにされた場合にはＰＩＰＥ＿ＲＥＰ＿ＧＯＯＤ＿ＳＴＡＴＵＳにセットされ、または失敗の場合にはＰＩＰＥ＿ＲＥＰ＿ＥＲＲＯＲ＿ＳＴＡＴＵＳにセットされる。ＰＩＰＥ＿ＲＥＰ＿ＭＡＳＴＥＲ＿ＵＮＡＶＡＩＬＡＢＬＥは、この発明の一部ではない特徴に関連して用いられるステータスコードである。

【００５１】

２．パイプサーバＡＰＩ－アクセス関数

上述したように、この実施例では、Windows NT名前付きパイプは、クライアントプロセス（たとえば、リソースＤＬＬ４０）とパイプサーバ４２との間、または異なったノード上の２つのパイプサーバ４２間でメッセージをやり取りするために用いられる。所与のノード上のパイプサーバ４２からサービスを要求するために、リソースＤＬＬ４０などのクライアントプロセス、または別のノード上のパイプサーバ４２は、以下のリスティング１に示されるステップを実行する：

リスティング１：

（ａ）　WIN32 CreateFile関数を呼出して、読出および書込の両方のために名前付きパイプを開き、

（ｂ）　WIN32 SetNamedPipeHandleState関数を呼出して、（バイトストリームモードとは反対の）メッセージモードに名前付きパイプをセットし、

（ｃ）　WIN32 WriteFile関数を呼出して、リクエストバッファをサーバに送り、

（ｄ）　WIN32 ReadFile関数を呼出して、サーバから応答バッファを読出し、

（ｅ）　WIN32 CloseHandle関数を呼出して、名前付きパイプを閉じる。

【００５２】

割込を見込んでWindows NT Overlapped I/Oがリスティング１のステップ（ｃ）および（ｄ）に用いられてもよい。

【００５３】

所与のノード上のクライアントによって、その同じノード上または異なったノード上のパ

イプサーバ４２にメッセージを送るために用いられる名前付きパイプの名前（クライアント側の名前）は、以下の形をとる：
＼＼serverComputerName＼pipe＼TUXPIPESERVER.serverComputerName
ただし、serverComputerNameは、（ＷＩＮ３２GetComputerName関数によって戻される）パイプサーバが実行しているコンピュータの名前である。ＭＳＣＳ環境では、これは、パイプサーバ４２が実行しているＭＳＣＳ仮想サーバのネットワーク名値である。

【００５４】

名前付きパイプのインスタンスを生成するとき、パイプサーバ４２によって用いられる名前付きパイプの名前（サーバ側の名前）は、以下の形をとる：
＼＼.＼pipe＼TUXPIPESERVER.serverComputerName
なお、サーバが名前付きパイプを生成するとき、名前の最初にコンピュータ名の代わりにピリオド（.）の使用が必要とされる。したがって、名前付きパイプの名前のserverComputerName部分は、２つ以上の仮想サーバが同じ物理的コンピュータ上で実行している場合を処理するよう名前付きパイプの名前の最後の部分に特定される。

【００５５】

パイプサーバ４２は、デフォルトセキュリティを用いて名前付きパイプを生成する。パイプサーバ４２は、ＭＳＣＳサービスを起動するために用いられるWindows NTドメインアカウント下で実行する。これは、この同じアカウント（またはサーバシステムの管理者特権を備えるアカウント）を用いるクライアントのみが名前付きパイプを開くために必要とされるアクセス権を有することを意味する。すべての他のクライアントには、CreateFile関数でパイプのクライアント側を開くと、ＥＲＲＯＲ＿ＡＣＣＥＳＳ＿ＤＥＮＩＥＤエラーコード（５）が戻される。

【００５６】

クライアント側の処理を簡素化するために、上のステップ（ａ）から（ｅ）に記載される機能をカプセル化するアクセス関数が与えられる。関数の各々は、IntrPtrパラメータを有する。パイプサーバ４２にリクエストを送り応答を待つことは、時間のかかる動作であり得るので、IntrPtrパラメータを用いて、操作の割込を制御する。IntrPtrがＮＵＬＬであれば、要求された操作は割込可能でない。IntrPtrがＮＵＬＬでなければ、これは、以下に定義される構造を指す：

【００５７】

【数４】

```
// The structure type INTR is used to control interrupting long
running
// operations.
typedef struct _interrupt_t {
    LPVOID  workerPtr;
      // workerPtr is the pointer for this worker thread, which is
required
      // by the ClusWorkerCheckTerminate function, or NULL if
      // ClusWorkerCheckTerminate should not be called.
    time_t  stopTime;
      // stopTime is the time (as returned by the "time" function)
when
      // this operation should be interrupted, or 0 if there is no
time
      // limit.
} INTR;
typedef INTR    *LPINTR;
```

【００５８】

workerPtrフィールドがＮＵＬＬでなければ、これは、クラスタユーティリティ関数ClusWorkerCreateでワーカスレッドを生成するとき用いられるワーカ構造へのポインタである。クラスタユーティリティ関数は、ＭＳＣＳの一部として与えられる。ワーカスレッドの使用は、以下により詳細に記載される。workerPtrがＮＵＬＬでなければ、アクセス関数

は周期的に、クラスタユーティリティ関数ClusWorkerCheckTerminateを呼出して、ワーカスレッドが終了するよう要求されたかどうかを決定する。ClusWorkerCheckTerminateがＴＲＵＥを戻した場合、アクセス関数は、進行中の操作を打切り、エラーを戻す。

【００５９】

stopTimeフィールドが０でなければ、それは、Ｃのランタイム時間関数によって戻されるのと同じ形（すなわち、１９７０からの秒数）で、この操作のデッドライン時間を表わす。stopTimeが０でないとき、アクセス関数は周期的に、デッドライン時間が来たかどうかを見るようチェックする。もしそうであれば、アクセス関数は操作を打切り、エラーを戻す。

【００６０】

関数の各々は、numRetriesパラメータを有する。０が送られた場合、要求された操作は自動的には再試行されない。numRetriesが０でなく、応答がパイプサーバから受取られないようなエラーが生じた場合（たとえば、パイプサーバが実行していないためにパイプサーバを開くことができない）、アクセス関数は１秒スリープし、自動的に、numRetriesによって示される回数まで同じリクエストを再試行する。numRetries再試行を実行した後リクエストがなおもフェールするのであれば、アクセス関数はエラーを戻す。

【００６１】

これらの関数の各々はまた、serverComputerNameパラメータを有する。このパラメータは、（ＷＩＮ３２GetComputerName関数によって戻される）パイプサーバが実行しているコンピュータの名前である。ＭＳＣＳ環境では、これは、パイプサーバが実行しているＭＳＣＳ仮想サーバのネットワーク名値である。他の指定がない限り、このパラメータは指定されなければならず、ＮＵＬＬであってはならない。

【００６２】

以下のアクセス関数は、前述の、ＰＩＰＥ＿ＲＥＱ＿ＭＩＢ、ＰＩＰＥ＿ＲＥＱ＿ＡＤＭ＿ＯＮＬＩＮＥおよびＰＩＰＥ＿ＲＥＱ＿ＡＤＭ＿ＯＦＦＬＩＮＥメッセージタイプに対して用いられる。他の指定がなければ、関数の各々は、成功に対しては整数（）を戻し、またはエラーが生じたならば１を戻す。

【００６３】

getFromMib（）

目的：

タキシードＭＩＢから所与のオブジェクトクラスの１つ以上の属性値を取出すことである。

【００６４】

Function Call：

【００６５】

【数５】

```
int getFromMib(
  LPINTR IntrPtr,
  DWORD numRetries,
  LPCSTR serverComputerName,
  LPCSTR className,
  FLDID32 keyAttr,
  LPCVOID keyValue,
  ...
);
```

【００６６】

Parameters：

serverComputerName－ＮＵＬＬであってもよい。その場合、リクエストはパイプサーバに提出されず、代わりにリクエストは現在のプロセス内で実行される。

【００６７】
className－ＭＩＢクラス名を特定するために用いられる文字列である。
keyAttr－“GET key”として使用すべき属性を表わすＦＬＤＩＤ３２値である。（クラスがＴ＿ＤＯＭＡＩＮである場合のように）“GET key”がなければ、これをＢＡＤＦＬＤＩＤにセットする。
【００６８】
keyValue－“GET key”値へのボイドポインタである；この値は、keyAttrがＢＡＤＦＬＤＩＤであれば無視される。
【００６９】
getAttr1－取出すべき属性を表わすＦＬＤＩＤ３２値である。
getValue1－取出された属性値を保持するのに十分大きいバッファへのボイドポインタである。
【００７０】
可変数のさらなるgetAttrNパラメータおよびgetValueNパラメータがある。最後のgetAttrNは、ＢＡＤＦＬＤＩＤの値によって表示される。
【００７１】
setInMib（）
目的：
タキシードＭＩＢ内に所与のオブジェクトクラスのための１つ以上の属性値をセットする。
【００７２】
Function Call：
【００７３】
【数６】

```
int setInMib(
  LPINTR IntrPtr,
  DWORD numRetries,
  LPCSTR serverComputerName,
  long flags,
  LPCSTR className,
  FLDID32 keyAttr,
  LPCVOID keyValue,
  ...
);
```

【００７４】
Parameter：
serverComputerName－ＮＵＬＬであってもよい。その場合、リクエストはパイプサーバに提出されず、代わりにリクエストは現在のプロセス内で実行される。
【００７５】
className－ＭＩＢクラス名を特定するために用いられる文字列である。
flags－使用すべきＤＡ＿ＦＬＡＧＳ値である。
【００７６】
keyAttr－“SET key”として使用すべき属性を表わすＦＬＤＩＤ３２値である。（クラスがＴ＿ＤＯＭＡＩＮである場合のように）“SET key”がなければ、これをＢＡＤＦＬＤＩＤにセットする。
【００７７】
keyValue－“SET key”値へのボイドポインタである；この値は、keyAttrがＢＡＤＦＬＤＩＤであれば無視される。
【００７８】
setAttr1＝＞セットすべき属性を表わすＦＬＤＩＤ３２値である。

setValue1＝＞セットすべき属性値へのボイドポインタである。
【００７９】
可変数のさらなるsetAttrNパラメータおよびsetValueNパラメータがある。最後のsetAttrNは、ＢＡＤＦＬＤＩＤの値によって表示される。
【００８０】
processSimpleRequestViaPipe ()
目的：
ＰＩＰＥＲＥＱ構造体を組立て、これを、処理のために特定されたサーバコンピュータに送る。この関数は、この実施例では、ＰＩＰＥ＿ＲＥＱ＿ＡＤＭ＿ＯＮＬＩＮＥリクエストおよびＰＩＰＥ＿ＲＥＱ＿ＡＤＭ＿ＯＦＦＬＩＮＥリクエストに対して用いられる。
【００８１】
Function Call：
【００８２】
【数７】

```
int processSimpleRequestViaPipe(
  LPINTR IntrPtr,
  DWORD numRetries,
  LPCSTR serverComputerName,
  DWORD requestCode,
  LPCSTR groupName,
  BOOL logPipeNotFound,
  BOOL logBrokenPipe
);
```

【００８３】
Parameters：
requestCode－ＰＩＰＥＲＥＱ構造体に対するリクエストコードである。
【００８４】
groupName－この発明の一部でないリクエストタイプに対するgroupName値である。このフィールドは、ＰＩＰＥ＿ＲＥＱ＿ＡＤＭ＿ＯＮＬＩＮＥリクエストタイプおよびＰＩＰＥ＿ＲＥＱ＿ＡＤＭ＿ＯＦＦＬＩＮＥリクエストタイプに対してはＮＵＬＬである。
【００８５】
logPipeNotFound－パイプ・ノット・ファウンド・エラー（ＥＲＲＯＲ＿ＦＩＬＥ＿ＮＯＴ＿ＦＯＵＮＤ）がログされるべきであれば、ＴＲＵＥであるフラグである。
【００８６】
logBrokenPipe－パイプ切断エラー（ＥＲＲＯＲ＿ＢＲＯＫＥＮ＿ＰＩＰＥ）がログされるべきであれば、ＴＲＵＥであるフラグである。
【００８７】
出力：
この関数は、以下の値の１つを戻す：
ＰＩＰＥ＿ＲＥＰ＿ＧＯＯＤ＿ＳＴＡＴＵＳ　－成功。
【００８８】
ＰＩＰＥ＿ＲＥＰ＿ＥＲＲＯＲ＿ＳＴＡＴＵＳ　－一般的エラー。
【００８９】
ＰＩＰＥ＿ＲＥＰ＿ＭＡＳＴＥＲ＿ＵＮＡＶＡＩＬＡＢＬＥ　－タキシードマスタが利用不可能であり、ステータスを正確に判断することができなかった。このステータスコードは、この発明の一部でない特徴で使用される。
【００９０】
ＰＩＰＥ＿ＲＥＰ＿ＮＯ＿ＲＥＳＰＯＮＳＥ　－切断パイプ、タイムアウトなどのために応答が受取られなかった。

【００９１】
３．パイプサーバアーキテクチャ
パイプサーバは、ＷＩＮ３２コンソールアプリケーションとして実現される。スタートアップ時に、以下のものがコマンドラインパラメータとして送られる：
TUXCONFIG Path－サービスされるべきタキシード論理的マシンに対応するＴＵＸＣＯＮＦＩＧファイルのパスネーム全体である。このパラメータは必須である。
【００９２】
Trace Level－サーバのために活性化されるべきトレースのレベルである（オフに対しては０、またはトレース詳細のレベルを増大させるには１から４）。このパラメータはオプションである；省略されれば、トレーシングはオフにされる。
【００９３】
Number of Pipe Instances－生成されるパイプインスタンスの数である。このパラメータはオプションである；省略されれば、デフォルト値は１０である。このパラメータの最大値は６３である。名前付きパイプの生成は、以下により詳細に説明される。
【００９４】
Worker Thread Inactivity Timeout－特定のワーカスレッドが終了する前にそのワーカスレッドへのいかなるサービスリクエストもなしに経過しなければならない時間の量（ミリセカンド）である。このパラメータはオプションである；省略されれば、デフォルト値は６００００（すなわち、１分）である。ワークスレッドの使用は、以下により詳細に記載される。
【００９５】
パイプサーバ４２は、名前付きパイプの複数のインスタンスを生成可能である。インスタンスは、同じ名前を備えた名前付きパイプの独自のコピーである。各インスタンスを用いて、異なったクライアントとの接続を同時に維持することができる。
【００９６】
パイプサーバ４２のメインスレッドは、Number of Pipe Instancesパラメータによって特定されるとおり固定数のパイプインスタンスを生成する。メインスレッドは、Overlapped I/O操作を用いてパイプインスタンス上の接続操作を多重化する。Overlapped I/Oでは、接続操作は、WIN32 ConnectNamedPipe関数へのコールで起動される。もし操作が即座に完了しなければ、ConnectNamedPipe関数は、ＥＲＲＯＲ＿ＩＯ＿ＰＥＮＤＩＮＧのステータスを戻す；接続が完了すると、オーバラップした構造体内のイベントに信号が与えられる。
【００９７】
各パイプインスタンスは、以下のＣ言語構造によって表わされる：
【００９８】
【数８】

```
typedef struct {
    OVERLAPPED          connectOverlap;
    HANDLE              hPipeInst;
    CLUS_WORKER         worker;
    HANDLE              workerActivateEvent;
    BOOL volatile       workerActive;
    CRITICAL_SECTION    workerCriticalSection;
    DWORD               dwState;
#define CONNECTION_PENDING_STATE        0
#define PROCESSING_STATE                1
#define RECONNECTING_STATE              2
} PIPEINST, *LPPIPEINST;
```

【００９９】
ＯＶＥＲＬＡＰＰＥＤ、ＨＡＮＤＬＥ、ＢＯＯＬ、ＣＲＩＴＩＣＡＬ＿ＳＥＣＴＩＯＮお

よびＤＷＯＲＤは、Win32によって定義されるデータ型である。ＣＬＵＳ＿ＷＯＲＫＥＲは、ＭＳＣＳによって定義されるデータ型である。ＰＩＰＥＩＮＳＴ構造体のフィールドは、以下のように定義される：
connectOverlap－オーバラップした接続を行うために必要とされるオーバラップした構造体である。この構造体は、オーバラップした接続が完了したとき、またはワーカスレッドがサービスリクエストを完了したときを表示するよう用いられるイベントハンドルを含む。
【０１００】
hPipeInst－このパイプインスタンスのためのハンドルである。
worker－ＭＳＣＳクラスタユーティリティ関数の組によってワーカスレッドを制御するために用いられるワーカ構造体である。
【０１０１】
workerActivateEvent－処理すべきサービスリクエストがあるときワーカスレッドに通知するために用いられるイベントである。
【０１０２】
workerActive－ワーカスレッドが現在実行しているかどうかを表示するフラグである。
【０１０３】
workerCriticalSection－ワーカスレッドの起動および終了を調整するために用いられるクリティカルセクションである。クリティカルセクションは、Microsoft Windows NTによって定義されるオブジェクトである。
【０１０４】
dwState－パイプインスタンスの現在の状態である。
ＣＯＮＮＥＣＴＩＯＮ＿ＰＥＮＤＩＮＧ＿ＳＴＡＴＥは、ConnectNamedPipeが進行中であることを表わす。ＰＲＯＣＥＳＳＩＮＧ＿ＳＴＡＴＥは、クライアントと接続が確立され、パイプサーバ４２がワーカスレッド内のリクエストを処理する準備ができていることを表わす。ＲＥＣＯＮＮＥＣＴＩＮＧ＿ＳＴＡＴＥは、リクエストがワーカスレッドによって処理され、パイプサーバ４２が、クライアントから切断され、新しいクライアントとの新しい接続を開始する準備ができていることを意味する。
【０１０５】
パイプサーバ４２は、マルチスレッディングを用いて、それが生成する名前付きパイプを介して受取られた別個のリクエストの各々を処理する。そのようなスレッドは各々、ここではワーカスレッドと呼ばれる。より詳細には、クライアントがメインスレッド内の名前付きパイプインスタンスに接続した後、リクエストは別個のワーカスレッド内で処理される。これは、メインスレッドが即座に他のパイプインスタンスのサービスを再開することを可能にする。サービスリクエストは、完了するのに数秒かかることがある。したがって、メインスレッド内でサービスリクエストを実行することは、実際的でない。
【０１０６】
各パイプインスタンスは、対応するワーカスレッドを有する。しかしながら、システムリソースを最適化するために、ワーカスレッドは、リクエストが初めて特定のパイプインスタンスについて処理されるまで、起動されない。これは、１０個のパイプインスタンスがあって、クライアントからのサービスリクエストの同時レベルが６個を決して超えない場合、最後の４個のパイプインスタンスについてのワーカスレッドは決して起動されないということを意味する。
【０１０７】
ワーカスレッドが一旦起動されると、それはさらなるサービスリクエストを待ってループする。さらなるサービスリクエストが、パラメータWorker Thread Inactivity Timeoutによって定義されるタイムアウト時間期間内に特定のパイプインスタンスについて生じなければ、ワーカスレッドは終了する。ワーカスレッドは、必要なときに自動的に再起動される。
【０１０８】

４．メインスレッドおよびワーカスレッド処理
パイプサーバ４２のメインスレッドは、リスティング２に示す処理を実行する：
リスティング２：
１）　serverActiveフラグをTRUEに初期化する。（パイプサーバ４２を停止するリクエストがなされたとき、このフラグはFalseにセットされる）
２）　パイプサーバ４２コマンドラインパラメータを処理する。
【０１０９】
３）　パイプインスタンスを表わすＰＩＰＥＩＮＳＴ構造体のアレイ（パイプと呼ばれる）を割当てる。
【０１１０】
４）　ハンドル（hEventsと呼ばれる）の配列を割当てる。配列内のエントリの数は、パイプインスタンスの数よりも１つ大きい。
【０１１１】
５）　パイプサーバ４２の終了を制御するために用いられるイベントを生成する。このイベントハンドルは、hEvents配列の最終部分に記憶される。このイベントには、パイプサーバ４２を停止するリクエストがなされたとき信号が与えられる。
【０１１２】
６）　各パイプインスタンスについて：
ａ）　connectOverlap構造のためにイベントを生成し（かつhEventsにハンドルのコピーを記憶する）。
【０１１３】
ｂ）　ワーカ活性化のためにイベントを生成する。
ｃ）　ワーカ活性化のために使用されるクリティカルセクションを初期化する。
【０１１４】
ｄ）　パイプインスタンスを生成する。
ｅ）　WIN32関数ConnectNamedPipeを呼出してクライアントからの接続に対してリスニングを開始する。
【０１１５】
ｆ）　If ConnectNamedPipeがERROR__IO__PENDINGを戻した場合:
ｉ）　状態をCONNECTION__PENDING__STATEに変える。
【０１１６】
ｇ）　Else if ConnectNamedPipeがERROR__PIPE__CONNECTEDを戻した場合:
ｉ）　状態をPROCESSING__STATEに変える。
【０１１７】
７）　serverActiveフラグがTRUEである間ループする:
ａ）　hEvents配列内のハンドル上で待つWIN32関数WaitForMultipleObjectsを呼出す。これは、すべき何らかのワークを待つ。
【０１１８】
ｂ）　信号を与えられたイベントがhEvents内の最終イベントであれば、パイプサーバ４２は終了する。さもなくば、信号を与えられた特定のイベントは、特定のパイプインスタンスに対応する。
【０１１９】
ｃ）　If CONNECTION__PENDING__STATE:
ｉ）　WIN32関数GetOverlappedResultを呼出して、完了された、オーバラップしたConnectNamedPipe操作の結果を得る。
【０１２０】
ii）　PROCESSING__STATEに変える。
ｄ）　If PROCESSING__STATE:
ｉ）　connectOverlap構造内のイベントのためにWIN32関数ResetEventを呼出す。ワーカスレッドは、それがサービスリクエストの処理を完了したときこのイベントに信号を与え

る。
【０１２１】
ii)　このパイプインスタンスのためにActivateWorkerThread（下記）を呼出す。
【０１２２】
iii)　RECONNECTING__STATEに変える。
ｅ）　Else if RECONNECTING__STATE:
ｉ）　WIN32関数DisconnectNamedPipeを呼出す。
【０１２３】
ii)　WIN32関数ConnectNamedPipeを呼出して、クライアントからの接続に対してリスニングを開始する。
【０１２４】
iii)　If ConnectNamedPipeがERROR__IO__PENDINGを戻した場合:
（１）　状態をCONNECTION__PENDING__STATEに変える。
【０１２５】
iv)　Else if ConnectNamedPipeがERROR__PIPE__CONNECTEDを戻した場合:
（１）　状態をPROCESSING__STATEに変える。
【０１２６】
ActivateWorkerThread関数は、それがまだ実行していなければワーカスレッドを起動し、または既存のワーカスレッドを活性化する。入力パラメータは、ＰＩＰＥＩＮＳＴ構造へのポインタである。関数は、リスティング３に示すステップを実行する。
【０１２７】
リスティング３：
１）　workerCriticalSectionのためにWIN32関数EnterCriticalSectionを呼出す。
【０１２８】
２）　If workerActiveフラグがTRUE:
ａ）　workerActivateEventに信号を与える。
【０１２９】
ｂ）　workerCriticalSectionのためにWIN32関数LeaveCriticalSectionを呼出す。
【０１３０】
３）　Else:
ａ）　workerCriticalSectionのためにLeaveCriticalSectionを呼出す。
【０１３１】
ｂ）　クラスタユーティリティ関数ClusWorkerTerminateを呼出して、いかなる先行のワーカスレッドもが完全に終了されたことを確実にする。
【０１３２】
ｃ）　クラスタユーティリティ関数ClusWorkerCreateを呼出して、ワーカスレッド実行関数WorkerThreadFunction（下記）を生成する。
【０１３３】
ｄ）　workerActiveフラグをTRUEにセットする。
ｅ）　workerActivateEventに信号を与える。
【０１３４】
TerminateWorkerThread関数は、サーバのシャットダウン中に呼出され、ワーカスレッドの終了を強制する。入力パラメータは、ＰＩＰＥＩＮＳＴ構造体へのポインタである。serverActiveフラグが既にＦＡＬＳＥにセットされていると仮定して、関数はリスティング４に示すステップを実行する。
【０１３５】
リスティング４：
１）　workerActivateEventに信号を与えて、ワーカスレッドが待っていないということを確かめる。
【０１３６】

２）　クラスタユーティリティ関数ClusWorkerTerminateを呼出す。
WorkerThreadFunctionは、クラスタユーティリティ関数ClusWorkerCreateが呼出されたとき、新しいワーカスレッド内で呼出される。WorkerThreadFunction関数は、リスティング５に示すステップを実行する。
【０１３７】
リスティング５：
１）　workerActivateEvent上で待つWIN32関数WaitForSingleObjectを呼出す。タイムアウトの長さは、サーバパラメータWorker Thread Inactivity Timeoutによって与えられる値である。
【０１３８】
２）　If waitForSingleObjectがタイムアウトした場合:
ａ）　workerCriticalSectionのためにEnterCriticalSectionを呼出す。
【０１３９】
ｂ）　WorkerActivateEvent上で待つWaitForSingleObjectを再び呼出す。タイムアウトの長さは０であり、即時にリターンする。
【０１４０】
ｃ）　If waitForSingleObjectが再びタイムアウトした場合：
ｉ）　workerActiveフラグをFALSEにセットする。
【０１４１】
ii）　workerCriticalSectionのためにLeaveCriticalSectionを呼出す。
iii）　リクエストおよび応答のために割当てられるＦＭＬ３２バッファを解放する。
【０１４２】
iv）　WorkerThreadFunctionからリターンし、これはワーカスレッドを終わらせる。
【０１４３】
ｄ）　workerCriticalSectionのためにLeaveCriticalSectionを呼出す。
３）　If serverActive flagがFALSE:
ａ）　WorkerThreadFunctionからリターンし、これがワーカスレッドを終了させる。
【０１４４】
４）　名前付きパイプインスタンスから一時バッファにリクエストを読出す。
５）　If リクエストコードがPIPE__PEQ__MIB:
ａ）　まだ割当てられていなければ、リクエストおよび応答のためにＦＭＬ３２バッファを割当てる。
【０１４５】
ｂ）　（第２のワードで開始する）一時バッファをリクエストＦＭＬ３２バッファにコピーする。
【０１４６】
ｃ）　ＭＩＢリクエストを処理する（以下により詳細に記載する）。
ｄ）　応答ＦＭＬ３２バッファを名前付きパイプインスタンスに書込む。
【０１４７】
ｅ）　ＦＭＬ３２リクエストおよび応答バッファを再使用のためにクリアする。
【０１４８】
６）　Else：
ａ）　一時バッファをＰＩＰＥＲＥＱ構造体にキャストする。
【０１４９】
ｂ）　requestCodeフィールドに従ってリクエストを処理する（以下により詳細に記載する）。
【０１５０】
ｃ）　ＰＩＰＥＲＥＱ構造体を名前付きパイプインスタンスに書込む。
７）　クライアントが処理を完了しパイプのクライアント側を閉じるのを待つことだけを目的に名前付きパイプインスタンスから読出す。クライアントがパイプを閉じると、この

読出は、ERROR__BROKEN__PIPEエラーを受取る。

【０１５１】

８）　connectOverlap構造体内のイベントに信号を与えて、処理が完了したことを表示する。

【０１５２】

５．エラーロギング

すべてのエラーログは、タキシードＵＬＯＧに書込まれる。以下の条件がログされる：ａ）　ＷＩＮ３２ＡＰＩ関数から戻されるすべての予期せぬエラー；（ｂ）クラスタＡＰＩ関数から戻されるすべての予期せぬエラー；および（ｃ）タキシードアドミンＡＰＩ関数から戻されるすべての予期せぬエラー。

【０１５３】

６．詳細な操作

図５Ａ～図５Ｂは、図４に例示される例示のアプリケーションにおけるこの発明の操作のより詳細な例示をする流れ図を含む。特に、図５Ａ～図５Ｂは、ＯＮＬＩＮＥリクエストの処理を例示する。

【０１５４】

ステップ１００で、リソースモニタ２６は、リソースＤＬＬ４０内のタキシードアドミンサーバ・リソースタイプのＯＮＬＩＮＥエントリポイント関数を呼出す。ＯＮＬＩＮＥエントリポイント関数は、新しいスレッドを起動して、リクエストを処理し、ＥＲＲＯＲ__ＩＯ__ＰＥＮＤＩＮＧステータスをリソースモニタ２６に戻す。

【０１５５】

次に、ステップ１０１で、リソースＤＬＬ４０は、processSimpleRequestViaPipe（）関数を呼出し、ＰＩＰＥ__ＲＥＱ__ＡＤＭ__ＯＮＬＩＮＥリクエストコードを渡す。ProcessSimpleRequestViaPipe（）関数は、適切なリクエストメッセージを構成し（すなわち、ＰＩＰＥ__ＲＥＱ__ＡＤＭ__ＯＮＬＩＮＥのリクエストコードでＰＩＰＥＲＥＱ構造体を組立てる）、このノード（すなわち、親ノード）上のパイプサーバ４２のために名前付きパイプの正しい名前を構成する。ステップ１０２で、リソースＤＬＬ４０は、上のListing1に示す初期メッセージ交換ステップを実行する。

【０１５６】

ステップ１０４で、パイプサーバ４２のメインスレッドは、リソースＤＬＬ４０から接続を受取り、ワーカスレッドを活性化してリクエストを処理する。ワーカスレッドは、ＰＩＰＥ__ＲＥＱ__ＡＤＭ__ＯＮＬＩＮＥリクエストコードを認識し、リクエストの処理を開始する。ステップ１０６で、ワーカスレッドは、現在のノード（すなわち親ノード）がタキシードアプリケーションのための指定されたマスタかどうかを判断する。具体的には、ワーカスレッドは、タキシードアプリケーションのＴ__ＤＯＭＡＩＮオブジェクトからＴＡ__ＭＡＳＴＥＲ属性を獲得するgetMasterIds（）関数（図示せず）を呼出す。ＴＡ__ＭＡＳＴＥＲ属性は、タキシードアプリケーションの現在のところ指定されているマスタの論理的マシンＩＤ（ＬＭＩＤ）を特定する。マスタのＬＭＩＤを使用して、マスタの物理的マシンＩＤ（ＰＭＩＤ）は、タキシードＴ__ＭＡＣＨＩＮＥオブジェクトのＴＡ__ＰＭＩＤ属性から獲得可能である。次に、ワーカスレッドは、Win３２関数GetComputerNameを呼出すgetLmid（）関数（図示せず）を呼出して、親ノードの物理的マシンＩＤ（ＰＭＩＤ）を決定する。それは次に、ＰＭＩＤに対応するタキシードＴ__ＭＡＣＨＩＮＥオブジェクトからＴＡ__ＬＭＩＤ属性を獲得する。リクエストはこの例ではＯＮＬＩＮＥリクエストであるので、ワーカスレッドは、現在の状態がＩＮＡＣＴＩＶＥであるということをＴ__ＭＡＣＨＩＮＥオブジェクトのＴＡ__ＳＴＡＴＥ属性からベリファイする。

【０１５７】

ステップ１０６で、親ノードがマスタノードであるということが、ＬＭＩＤおよびＰＭＩＤから決定されると、ＰＩＰＥ__ＲＥＱ__ＡＤＭ__ＯＮＬＩＮＥリクエストは、ステップ１０８でローカルに処理される。そのような場合には、ワーカスレッドはまず、バックアップマスタノードが指定されているかどうかを決定する。もしそうなら、パイプサーバは

、バックアップマスタ上のパイプサーバにリクエストを送り、バックアップマスタにマスタとして自身を再指定するよう要請する。もしこれが成功であれば、ワーカスレッドは次に、名前付きパイプ接続を介して新しいマスタ上のパイプサーバにオンラインリクエストを転送して、それが、親ノード（これはたとえば、失敗後に再起動しようとしているかもしれない）をオンラインにし直すためにリクエストを処理できるようにする。バックアップマスタがなければ、親ノード上のパイプサーバは、タキシードアドミンＡＰＩを直接呼出して、そのノード上でタキシードドメインをオンラインにする。

【０１５８】

ステップ１０６で、親ノードがマスタノードでないことが決定されれば、制御はステップ１１０に移る。ステップ１１０で、ワーカスレッドは、パイプサーバアクセス関数setInMIB（）を呼出して、ＦＭＬ３２バッファが親ノードのＬＭＩＤと関連付けられたＴ＿ＭＡＣＨＩＮＥオブジェクト内のＴＡ＿ＳＴＡＴＥ属性をＡＣＴＩＶＥにセットするリクエストを含むＰＩＰＥ＿ＲＥＱ＿ＭＩＢメッセージを生成する（これは、ＯＮＬＩＮＥリクエストがタキシードアドミンＡＰＩを介してなされる態様である）。setInMIB（）関数は次に、マスタノード上のパイプサーバのために名前付きパイプの正しい名前を構成し、初期メッセージ交換ステップ（上記Listing１参照）を実行して、メッセージをマスタノード上のパイプサーバに転送する。

【０１５９】

ステップ１１２で、マスタノード上のパイプサーバ４２のメインスレッドは、親ノード上のパイプサーバ４２から接続を受取り、自分自身のワーカスレッドを活性化してＰＩＰＥ＿ＲＥＱ＿ＭＩＢリクエストを処理する。ステップ１１４で、ワーカスレッドは、ＰＩＰＥ＿ＲＥＱ＿ＭＩＢリクエストを認識し、タキシードアドミンＡＰＩに適切なコールを行なってリクエストを処理させ応答を受取る。具体的には、この例では、ワーカスレッドは、タキシードアドミン関数tpinit（）を呼出してタキシードアプリケーションに接続し、次にタキシードアドミンＡＰＩ関数tpacall（）とそれに続くtpgetreply（）を呼出してそれぞれ、リクエストを提出して処理させ、応答を受取る。

【０１６０】

次にステップ１１６で、（成功または失敗を表示する）応答ＦＭＬ３２バッファは、同じ名前付きパイプ接続を介して親ノードに送り返される。ステップ１１８で、親ノード上のsetInMIB（）関数は、名前付きパイプから応答を読出し、パイプを閉じる。ステップ１２０で、親ノード上のワーカスレッドは次に、名前付きパイプ接続を介して親ノード上のリソースＤＬＬ４０に応答ＰＩＰＥＲＥＰメッセージを書込む。最後に、ステップ１２２で、リソースＤＬＬ４０のＯＮＬＩＮＥエントリポイント関数は、リソースモニタ２６の適切なコールバック関数を呼出して、ＯＮＬＩＮＥリクエストのステータスを表示する。ＯＦＦＬＩＮＥリクエストなどの、タキシードアプリケーションのための指定されたマスタノードによって実行されなければならない他の管理用リクエストの処理は、上記と同様である。

【０１６１】

「Microsoft」、「Windows」および「Windows NT」は、Microsoft Corporationの登録商標である。「BEA」および「TUXEDO」は、BEA Systems,Inc.の登録商標である。「Oracle」は、Oracle Corporationの登録商標である。前に例示したように、この発明は、アプリケーションがクラスタ化により達成される増大した可用性を利用できるように、マスタノードを介した集中管理を必要とする分散ネットワークアプリケーションがそのような集中管理をサポートしないクラスタ化された環境内に配備されることを可能にするための方法に向けられる。広い発明の概念から逸脱することなしに上述した実施例に変更がされ得ることが理解される。たとえば、パイプサーバ間の接続の好ましい形は、Windows NTの名前付きパイプであるが、プロセス間通信の他の形が採用可能である。加えて、前述したように、この発明を採用してTuxedoベースのアプリケーションがＭＳＣＳ制御のコンピュータクラスタ内で実行することを可能にするような特定の実施例が記載されたが、この発明は、Tuxedoベースのアプリケーションおよび／またはＭＳＣＳ制御のコンピュータクラスタ

での使用に限られるものでなく、分散ネットワークアプリケーションがマスタノードを介した集中管理を必要とするが、アプリケーションが配備されているクラスタ化された環境がそのような集中管理をサポートしないようないかなるシステムにも採用可能である。したがって、この発明は、開示される特定の実施例に限られるものでなく、前掲の特許請求の範囲によって定義されるようなこの発明の精神および範囲内にあるすべての変形を含むものと意図される。

【図面の簡単な説明】

【図１】　２つのノードを含む先行技術のコンピュータクラスタの操作を例示するブロック図である。

【図２】　図１のクラスタの操作のフェールオーバモードを例示するブロック図である。

【図３Ａ】　この発明の方法の好ましい実施例を例示する流れ図である。

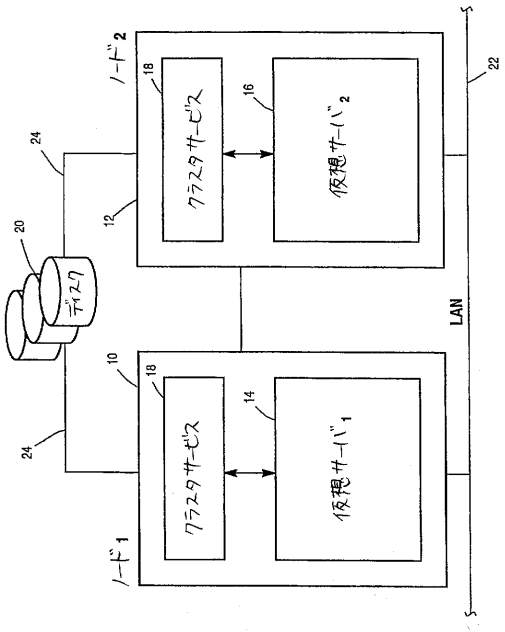
【図３Ｂ】　この発明の方法の好ましい実施例を例示する流れ図である。

【図４】　この発明の方法を実施するための装置の好ましい実施例、およびタキシード環境のために書かれかつＭＳＣＳによって制御される計算クラスタ化された計算処理環境内に配備される分散ネットワークアプリケーションを含む環境におけるこの発明の例示的アプリケーションの両方を例示するブロック図である。

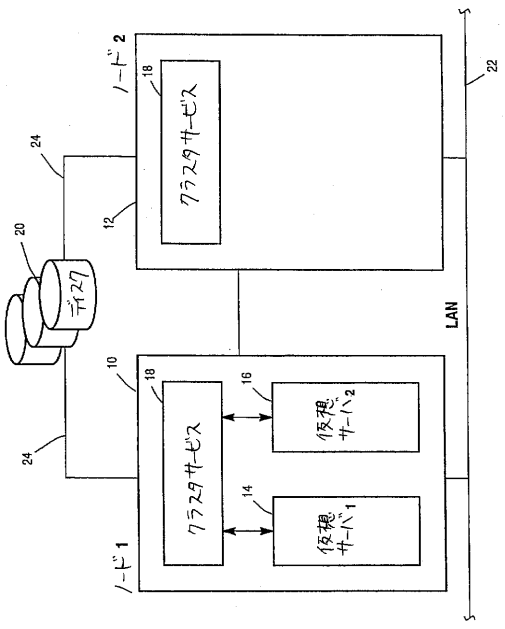
【図５Ａ】　図４の例示的環境におけるこの発明の操作のさらなる詳細を例示する流れ図である。

【図５Ｂ】　図４の例示的環境におけるこの発明の操作のさらなる詳細を例示する流れ図である。

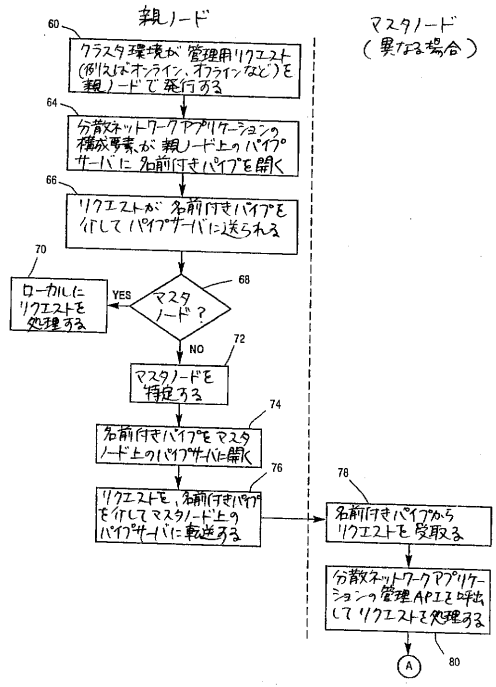
【図１】



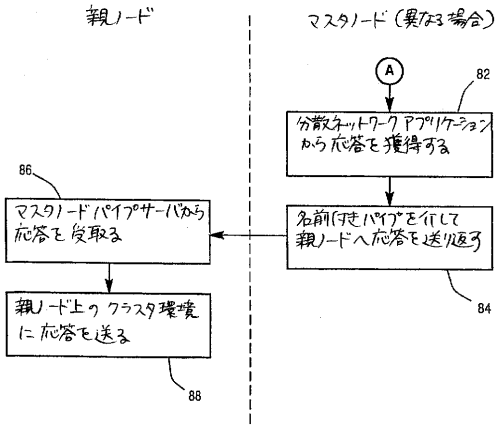
【図２】



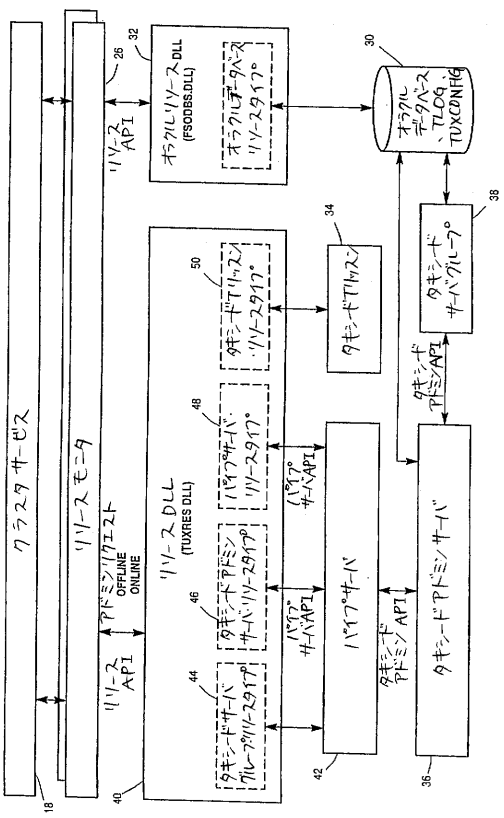
【図３Ａ】



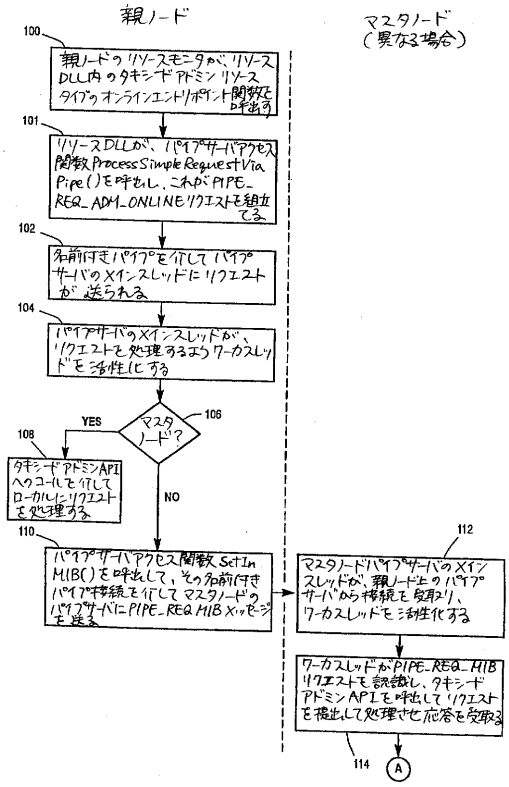
【図３Ｂ】



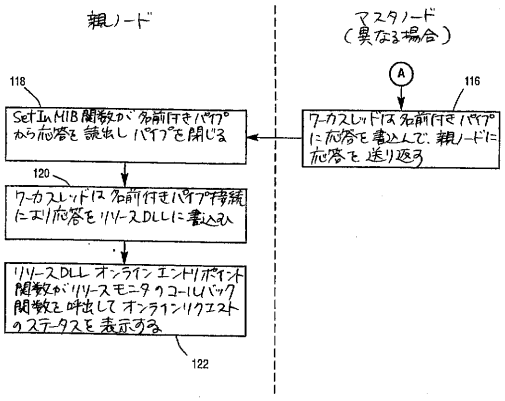
【図４】



【図５Ａ】



【図５Ｂ】


親ノード
マスタノード
（異なる場合）
A
118
116
SetIaMIB関数が名前付きパイプから応答を読出しパイプを閉じる
ワーカスレッドは名前付きパイプに応答を書込んで、親ノードに応答を送り返す
120
ワーカスレッドは名前付きパイプ接続により応答をリソースDLLに書込み
リソースDLL オンライン エントリポイント関数がリソースモニタのコールバック関数を呼出してオンラインリクエストのステータスを表示する
122

フロントページの続き

(74)代理人 100091409
弁理士 伊藤 英彦
(74)代理人 100096781
弁理士 堀井 豊
(74)代理人 100096792
弁理士 森下 八郎
(72)発明者 バーデウ，スティーブン・エイ
アメリカ合衆国、１９６４０ ペンシルバニア州、フェニックスビル、デイビッズ・ラン、１００８

審査官 殿川 雅也

(56)参考文献 特開昭６４－００８７４２（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 9/46 - 9/54